SiTex Graphics

AIR

User Manual

by Scott Iverson

© 2001-2014 SiTex Graphics, Inc.

Table of Contents

1 Getting Started

Welcome to Air, SiTex Graphics' advanced 3D rendering software for visual effects, design, and visualization.

Air accepts files in RIB (RenderMan® Interface Bytestream) format. Many popular modeling and animation programs can generate RIB output either natively or with the aid of a plugin.

The following plugins are available from the SiTex Graphics web site (www.sitexgraphics.com):

Air Stream plugin for Maya
RhinoAir for Rhino 4 or Rhino 5
SketchAir for SketchUp

The following applications support Air natively or via a 3rd party plugin:

Massive
Houdini (from Side Effects Software)
CineMan for Cinema 4D

Other compatible products include:

CityEngine from Procedural, Inc.
The RealFlow Rendering Toolkit for the RealFlow fluid and dynamics software

On Windows systems Air ships with Air Space, a standalone user interface for Air. More information on Air Space can be found in the separate Air Space User Manual.

Air can also be used to render a scene file from a command prompt or using the Air Control tool included with Air.

Air Feature Overview

Air is an advanced 3D graphics renderer with a unique architecture and extensive features designed for the rapid production of high-quality images. Air is a *hybrid* renderer, combining the advantages of scanline rendering - fast rendering of complex scenes, motion blur, and depth of field - with the flexibility of on-demand ray tracing for accurate reflections, soft shadows, global illumination, and caustics.

Air supports a broad range of geometric primitives, including polygon meshes, trimmed NURBs, subdivision meshes, curves, particles, and implicit surfaces. All primitives are supported in their natural form; no pre-meshing is required. Air provides both bump mapping and true ("sub-pixel") displacement. 8-bit, 16-bit, and floating-point texture maps and environment maps are supported. High-dynamic range (HDR) images can be used to perform image-based lighting.

Air offers the flexibility of fully programmable shading using the industry-standard RenderMan® shading language. Air comes with a collection of shaders, and hundreds more are freely available. The Air distribution also includes Vshade, a Visual Tool for constructing shaders without programming.

Images rendered with Air may be displayed on the screen using the companion Air Show tool or written to a file in a variety of graphics file formats. Air can output multiple images from a single rendering for efficient multipass rendering. Air is capable of producing high-quality contours for cartoon rendering and illustration.

1.1 Installation

Installing Air under Windows

Run the installer to install Air on your machine. See below for instructions on installing your license file. If you are trying the Air demo, you do not need to install a license file.

Installing Air under Linux

- Unzip the distribution package to a directory. For example:

```
cd /usr/local
unzip airx19.zip
```

- Declare an AIRHOME environment variable to point to the directory containing the Air installation. In the above example, the directory would be `/usr/local/air`. For the bash shell:

```
export AIRHOME=/usr/local/air
```

  For tcsh or csh:

```
setenv AIRHOME /usr/local/air
```

- Add `$AIRHOME/bin` to the `PATH` environment variable

- Make both of the above changes to the startup file for your shell - `.bashrc` for bash, `.cshrc` for csh or tcsh.

Installing the RLM license file

When you purchase Air, you will be issued an RLM license file named `sitex.lic`. There are two options for installing the license file:

- Place the license file in the `bin` directory of your Air installation. Air will automatically look for a license file in that location.
- Save the license file to another location and create a SITEX_LICENSE environment variable with the full path to the license file.

The license file may be a node-locked license or a floating license. A node-locked license file will not have a HOST line, and the LICENSE line will include the string 'uncounted'. If you received a node-locked license, you are now ready to start using Air.

If you received a floating license, you will need to start the RLM license server. Here are brief instructions for starting the RLM server:

- Check the HOST line in the license file and make sure it contains the host name of the server. If you see a string such as "localhost" or "hostname", change it to the name of the server. You may also change the port number - last number on the HOST line. Don't change anything else in the file.

- Start the license server from a command shell. If the RLM license file is saved in the `bin` directory of the Air installation, cd to that directory and type:

```
rlm
```

  If the RLM license file was saved to a different location, start rlm using the -c option and a path to the license file. E.g.,

```
rlm -c c:\mylicenses\sitex.lic
```

The server should start and you should then be able to render.

Full documentation for the Reprise License Manager can be found in a separate document in the `doc` directory of the Air installation.

Network Installation

There are two common methods of organizing Air on a render farm:

Central Installation for Clients and Server

In this approach Air is installed once on a central file server, and the license server and each client machine reference the same installation.

To use this method, follow the installation instructions above for Windows or Linux to install Air on the central file server. Then for each client machine (including the license server):

- Add the bin directory of the Air home directory to the PATH
- Define an AIRHOME environment variable that points to the Air home directory on the server.
- If the RLM license file is not in the `bin` directory of the Air installation, create a SITEX_LICENSE environment variable with the port and host name of the license server in the format *port@hostname*. For example:

  2192@MYSERVER

  The port number is the last number on the HOST line in the RLM license file.

Separate Installation for Each Machine

In this approach Air is installed separately on each machine.

After installing Air, each client machine must be configured to find the license server by defining a `SITEX_LICENSE` environment variable with the port and host name of the license server in the format *port@hostname*. For example:

2192@MYSERVER

The port number is the last number on the HOST line in the RLM license file.

1.2 Gamma Correction

It is important to view rendered images with proper gamma correction for the display device. When rendering to the Air Show window, use the sRGB button to view a linear image with sRGB gamma correction. For more information on color conversion for Air, see the Color section of this document.

1.3 Rendering

The easiest way to render with Air is use one of the many plugins available for popular modeling and animation programs. A plugin will provide a render button or command that automatically exports a scene and starts an Air rendering process.

Alternatively, the Air Space user interface included with the Windows version of Air can be used to add materials and lights to a model for rendering with Air. See the separate Air Space User Manual for more information.

Command Line Rendering

A RIB scene file can be rendered by starting a standalone Air process from an MS-DOS prompt on Windows or a command shell on Linux.

The general syntax for invoking Air from the command line is:

```
air [options] RIBcommand(s) RIBfile(s)
```

Options, if present, must appear before any commands or filenames. Depending on the display settings in the file, AIR will either generate an image file or open a window on the screen. Any RIB command may be passed as an argument by enclosing it in braces {}. For example:

```
air {Format 320 240 1} myscene.rib
```

Air accepts multiple commands and/or RIB files. They are processed in order from left to right.

To abort rendering, press Control-C in the console window or click the close button on the framebuffer window (if displayed.)

If no file name is included in the command line, Air attempts to read input from stdin.

The Reference section contains a list of command line options. You can view a short list of command line options with

```
air -
```

The user note for Massive on command line rendering introduces several commonly used command line switches.

Under Windows the 64-bit version of Air can be invoked with

```
air64 [options] RIBcommand(s) RIBfile(s)
```

Air Control

AIR Control provides a graphical user interface for setting basic parameters and starting a rendering process with AIR.

See Also

Command Line Rendering of Massive Scenes
Command Line Switches
Linux Notes

1.4 Linux Notes

Libraries

The user-interface tools such as Air Show and Vshade rely on older versions of GTK, GDK, and related libraries. You may need to add the 32-bit version of these libraries to your Linux installation if they are not already present. You can see a full list of dependencies with, e.g.,

```
ldd $AIRHOME/bin/airshow
```

The `air` binaries and other command line tools do not depend on GTK, GDK or the related interface libraries.

1.5 Command Line Switches

You can view an abbreviated list of command line options by starting Air with an argument of -

```
air -
```

The following command line arguments are recognized by Air:

`-anim` *`nframes`* `(`*`starttime endtime`* `(`*`fractionopen`*`))`

Generates a sequence of *nframes* frame blocks with shutter times for animating time-varying RIB files.

`-at` *`name token value`*

Define a custom attribute with a string or float value (the equivalent of `Attribute "`*`name`*`" "`*`token`*`" [`*`value`*`]`).

`-bs` *`width`* `(`*`height`*`)`

Sets the size (in pixels) of the buckets or tiles used for rendering.

`-columns`

Use column bucket order.

`-crop` *`left right top bottom`*

Restricts rendering to a subrectangle. Each coordinate should lie in the range 0 to 1. This switch is equivalent to the `CropWindow` call with the same coordinates.

`-d`

Forces the rendered image to appear in Air Show

`-e` *`filename`*

Saves error messages and warnings to a file.

`-echoproc`

Echoes the output from any procedural RunPrograms to stdout

`-echorif`

Echoes the output from any RI filters to stdout

`-file` *`imagename`*

Forces image output to the specified file. This switch overrides any `Display` call in the scene.

`-frames` *`start end`*

Restricts rendering to frames numbered between start and end, inclusive. AIR 8 and later will also use this range for expanding a frame number range in a RIB file name.E.g.,

```
air -frames 19 21 scene#4f.rib
```

would render `scene0019.rib`, `scene0020.rib`, and `scene0021.rib`.

`-g` *`n`*

Sets the default gamma.

`-indirect`

Enable object visibility to indirect rays.

`-ipp`

Enabled the indirect diffuse prepass

`-mf` *`n`*

Sets the default value for `GeometricApproximation "motionfactor"`

`-mode rgb|rgba`

Selects the channels for output. This setting may be overridden by a subsequent `Display` call.

`-nodisp`

Ignore/suppress `Displacement` calls.

© 2001-2014 SiTex Graphics, Inc.

`-noerror`

Ignore/suppress all error and warning messages.

`-nosurf`

Ignore subsequent `Surface` calls. The surface shader used will be `defaultsurface` or the surface shader declared in an air.rib file in AIRHOME or the local directory.

`-nowarn`

Suppresses warning messages during rendering

`-opp`

Enable the occlusion prepass

`-opt` *`name token value`*

Define a custom option with a string or float value (the equivalent of `Option "`*`name`*`" "`*`token`*`" [`*`value`*`]`).

`-p` *`nthreads`* (or `-threads` *`nthreads`*)

Sets the number of processor threads to use. This option overrides any option that sets the number of threads in the RIB file. A value of 0 means use all detected processor cores (the default).

`-pri` *`priority`*

Sets the task priority for the renderer under Windows. Possible values are 0 (normal priority) or -1 (low priority). Rendering at low priority allows "foreground" applications such as a modeler to run unimpeded while Air renders in the background.

`-Progress`

Causes the renderer to print progress information to the standard output stream. Progress is reported as a string formated as:

```
R90000 xxx% mmmK  nnnK tttS
```

where *xxx* is the percentage complete, *mmm* is the current memory used in kilobytes, *nnn* is the peak memory used in kilobytes, and *ttt* is the elapsed time in seconds.

`-q8`

Set quantization to 8-bit for rgb and rgba output, as well as any channels specified in the -mode option.

`-q16`

Set quantization to 16-bit for rgb and rgba output, as well as any channels specified in the -mode option.

`-qfloat`

Set quantization to floating-point for rgb and rgba output, as well as any channels specified in the -mode option.

`-reflect`

Enable object visibility to reflection rays. Shorthand for `Attribute "visibility" "integer trace [1]`

`-res` *`width`* `(`*`height`*`)`

Sets the resolution of the rendered image. This option overrides any subsequent `Format` call in the RIB stream. If height is omitted, a square image is rendered.

`-rif "`*`rifiltername args`*`"`

Define a default RIB filter plugin for the scene.

`-rows`

Use row tile order.

`-samples` *`xsamples`* `(`*`ysamples`*`)`

Sets the size of the array of geometry samples per pixel. This option overrides any subsequent `PixelSamples` call.

`-shadows`

Enables object visibilty to shadow rays. Shorthand for `Attribute "visibility" "integer transmission" [1]`

`-showpath`

Print the full path name for each resource as it is loaded.

`-silent`

Suppresses the progress report when rendering.

`-spiral`

Use spiral tile order.

`-stats`

Print statistics for the rendered scene.

`-surf` *`shadername`*

Overrides all surface shaders with the provided shader. Shader parameters can be set by including a comma-separated list:

```
air -surf plastic,Kd=1,Ks=0 myscene.rib
```

Only float and string type parameters are supported.

`-step` *`n`*

Specifies an increment to use when rendering animation frames. For example, if step=2, every other frame will be renderered.

`-vhost` *`ipaddress`*

Connect to an instance of Vortex executing on the machine at *ipaddress*. Not available in the demo version.

See Also

Command Line Rendering of Massive Scenes

2 What's New

What's new in Air 13. (For information on prior releases, see the History section.)

- New SketchAir plug-in for SketchUp, available as a separate download from the SiTex Graphics web site

- Air Stream for Maya 2.0, available as a separate download from the SiTex Graphics web site

- New Retexture tool for interactive material tweaks. (Windows only) See the separate Retexture User Manual for more information.

- Physically Plausible Shading
  - New importance-based method for sampling reflections based on a specular BRDF allow reflections and specular highlights to be treated in a consistent manner, enabling a new set of physically plausible shaders.
  - New physically plausible shaders: SimpleMetal, SimplePlastic, VPhysicalMetal, VPhysicalPlastic

- Color
  - A new Color section of this document provides a simple unified discussion of the proper handling of color input and output values when rendering with Air.
  - Other topics dealing with color have been grouped under the new Color section.
  - A color can be specified as a solar spectrum
  - New option to have Air automatically convert raw texture images from sRGB color space to linear rgb:

```
ColorSpace "rawtexture" "sRGB"
```

- The option to apply the dominant illuminant and chromatic adaptation to rgb colors has a new syntax. Instead of the old:

  ```
  Option "render" "string currentcolorspace" "litRGB"
  ```

  use

  ```
  ColorSpace "current" "litRGB"
  ```

- Indirect Lighting
  - New radiosity cache can greatly accelerate indirect lighting
  - New option to force Air to wait until an indirect or occlusion prepass has finished rendering before rendering the final pass:

    ```
    Option "render" "syncprepass" [1]
    ```

    This option provides a solution in cases where a bucket has not finished computing in the prepass when it is rendered in the final pass. Enabling this option will in general increase rendering time, since it forces all threads to wait until the last thread finishes the prepass. The option is enabled by default.

- Occlusion
  - New optimizations improve performance when using a cache for complex scenes with many threads

- Output
  - Most Air surface shaders have a new __diffuse_shadow output value that contains a shadow value based on a linear relationship between the shadowed and unshadowed diffuse values. This relationship is preserved through Air's filtering and compositing, unlike the old __shadow output variable which could exhibit artifacts along object edges and in regions of partial transparency. __diffuse_shadow is now the recommended shadow value to use in multipass rendering.
  - When the prefilterclamptoalpha option is enabled, output from any environment or imager shader is also clamped
  - Updated the Photoshop display driver to support the new __diffuse_shadow layer (with a subtract blend mode)

- Shaders
  - New spherelight light shader allows a point light to be used to mimic the effect of a spherical area light.
  - New spot_projector and distant_projector light shaders project a texture map like slide projector.
  - New FakeCarpet surface shader makes a rectangle look like a fuzzy carpet
  - New instCarpet instancer shader for growing carpet fibers on a rectangular patch
  - New VShadeCarpet surface shader to use with the above instCarpet shader
  - New VHexTile surface shader for hexagonal ceramic tiles
  - New UseBackground surface shader for computing reflections and shadows for compositing over a background image
  - New maxemission parameter for all light shaders that vary emitted intensity based on distance from the light source. Useful for limiting the maximum light intensity on near surfaces for lights that use a linear or quadric falloff of light intensity with distance.
  - Updated VFur and VHair shaders: new rootcolor and tipcolor parameters to work with Shave export from Maya. New ColorBias parameter for tweaking the interpolation between root and tip colors. The shaders now pass a valid normal to illuminance() to work better with occlusion and indirect lighting.

- Updated envPhysicalSky shader to include parameters to tune intensity based on ray type and visibility toggles for indirect and reflection rays
- Updated portallight shader to match the appearance of a window with indirect lighting
- Updated VBlinn and VPhong shaders to include __environment AOV and to support tracing light channels through reflections
- Updated Emitter surface shader with texture option and separate intensity controls for different ray types.
- Disabled the builtin background imager so the shader file is always run instead. This change solves an issue where the standard "background" color was not applied to rgb or rgba outputs contained in a multichannel file.
- Atmosphere, Interior, and Exterior shaders may now use shader networks

- Curves
  - New attributes to allow curves to be tapered at either or both ends

- Miscellaneous:
  - Settting the number of rendering threads to 0 can now be used to reset the thread count to the number of detected processor cores (the default)
  - The shading language `texture()` function can be used to generate an antialiased 2D noise pattern by supplying the special texture name "noise"
  - Volume rendering: ray marching now applies the view density and shadow density attributes
  - The -q16 and -qfloat command line options now apply the appropriate quantization when the -d or -file options are used. A new -q8 option forces quantization to 8-bit.
  - Air now defines a user option of type float with the current version number in the format

    major.minor

  This user option can be referenced in a conditional rib statement to optionally include or exclude features based a particular version of the renderer

- TweakAir
  - Support for opacity values attached to a primitive variable list
  - Fix for uniform primitive variable indexing when redrawing the camera view
  - Fix for varying/facevarying prim var indexing when re-tracing the camera view

- Shading Compiler
  - The `noise()` function may now specify the type of noise to use with an initial string. The options are the same as those for

    ```
    Option "noise" "string algorithm" ""
    ```

  - The error message for undefined var declared with "extern" now includes the variable name
  - Boolean variables can now be used as the target in `option()` or `attribute()`

- Air Show 8:
  - Support for the new __diffuse_shadow output variable in composite mode
  - New simple Print command in the File menu (Windows only)
  - Status bar text is no longer sometimes reversed under Windows 7

- Texture Conversion tools (mktex and mktexui)
  - New options to convert to and from sRGB color space
  - Unrecognized options are reported but no longer cause the conversion to fail
  - New check box in mktexui to convert all files in a directory

- Massive

- New user note on adding a background image

- Bug fixes:
  - Texture lookups when shading indirect rays
  - Fix in ray traced hider jitter computation
  - Tweak for trace quad intersection under 64-bit (explicitly use double precision for sqrt)
  - Fix for texture coordinate computation when shading triangular micropolygons
  - Fix for fragment shading in 64-bit Air for Windows (workaround compiler bug)
  - Fix for unfiltered texture read of images with fewer than 3 or more than 4 channels
  - Tweak for shading optimizer to provide derivs inside a loop with multiple iterations
  - Shader files that have been truncated are now reported as corrupt
  - Fix for rendering curves with "uniform width" as tubes or ribbons
  - Fix for memory use reporting to prevent erroneous extreme values due to thread interference
  - When Air is unable to save an occlusion file, the error message includes the file name

3 Color

Obtaining correct results from any renderer depends on the proper handling of color input and output values. This section describes how to properly convert colors when rendering with Air.

Linear and sRGB Color Spaces

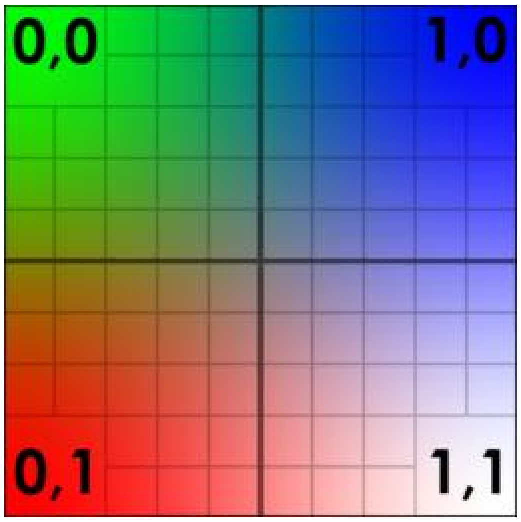
The computations performed by Air assume that color values are linear, so that say doubling the output color for a pixel doubles the intensity value. However, most monitors and printers display images in a non-linear color space that better matches the response characteristics of the human visual system. Obtaining linear color values for use with Air will typically require some conversion on the input and the output sides.

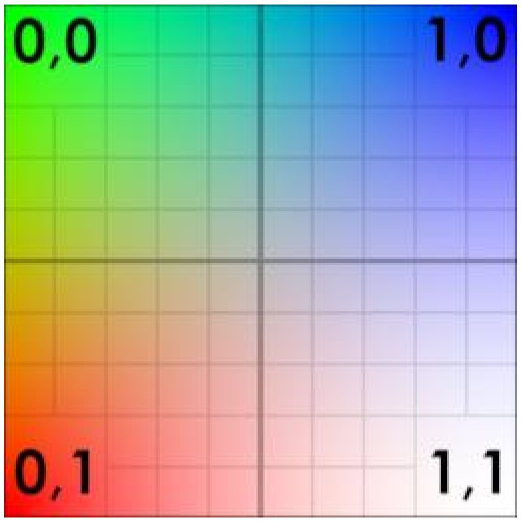
The non-linear aspect of the display color space is often represented as a mathematical function with a single parameter called *gamma*. Typical gamma values can range from 1.8 to 2.4 or more. In recent years most manufacturers of devices that display images - such as monitors or printers - or that create images - such as cameras and scanners - have adopted the sRGB color space as a standard for digital images. The sRGB color space is approximately equivalent to a gamma value of 2.2 - though that approximation is not very close at the low end of the intensity range.

Input Color Conversion

In order to provide Air with linear color values, color properties selected in a user interface may need to be converted from the display color space to a linear space. Your plugin or application should do this automatically, or have an option to do so.

Texture maps also need to be converted from a display color space if the images are not already linear. Here is a comparison of an image in a linear rgb space (left) and sRGB space (right):





Linear images will often appear too dark when viewed on a normal monitor without gamma correction.

The Air texture conversion tools - mktex and mktexui - can be used to convert an image from sRGB color space to linear rgb space. They can also be used to convert the image to an Air texture file that is more efficient for rendering.

Air 13 introduces a new option to automatically convert source texture files from sRGB color space. This option can be enabled by adding the following to a RIB file:

```
ColorSpace "rawtexture" "sRGB"
```

when enabled, Air will automatically perform an sRGB to linear conversion for images used as texture maps that have not been converted to an Air texture file already. This option does not apply to images used as environment maps.

Output Color Conversion

The output colors computed by Air are in a linear color space, and they will need to be converted to the display color space at some point in your pipeline. When that conversion takes place depends on the target device and purpose:

Rendering to Air Show

If you are rendering to Air Show for previewing or final rendering, you can use the sRGB display button in the Air Show toolbar to view a linear image in sRGB color space. If possible, select an option to render 16-bit or floating-point data to avoid banding artifacts that can appear when sRGB conversion is applied to an 8-bit image. This may be the only display correction you'll need. If you wish to save the image as it is displayed in sRGB space, use the Save as Displayed item in the File menu. To save the raw linear image, use the normal Save command.

Rendering to a file for later processing

When rendering to a file for later processing in an image editor or compositor, render linear images at 16-bit or float precision and apply sRGB conversion after any post-processing. This case includes any type of multipass rendering for later compositing.

Rendering a final image

A rendered image that is to be viewed or printed directly should have appropriate display conversion applied prior to being saved. Air provides two main ways to do that. The RIB standard defines an exposure setting that can be employed to apply a simple gamma factor, typically 2.2 for an sRGB display device. Your plugin or user interface should provide an option to set the gamma value for the rendered image.

For more accurate conversion to sRGB color space, the sRGB imager shader can be used to convert the computed rgb values to sRGB color space. Your user interface may or may not provide an option for using an imager shader.

High Dynamice Range (HDR) Images

The output values computed by Air may include values that outside the unit interval. The visual appearance of such high-dynamice range images can be improved by rendering at float precision and applying a tone mapping operator.

3.1 Spectral Color Definition

Air 10 introduces the ability to specify colors as a spectrum using spectral power distribution (SPD) files. SPD files define wave-length dependent reflectance or emission values as piece-wise linear curve connecting pairs of wavelength and intensity/reflectance values.

Use the following attribute to set the standard Color property with an SPD file:

```
Attribute "render" "string color_spd" "filename.spd"
```

A color parameter for a shader can be set via SPD by including a string parameter whose name is the same as the color parameter with a _spd suffix. For example,

```
LightSource "distantlight" 2 "string lightcolor_spd"
"light_CIE_D65_noon_daylight.spd"
```

The shader doesn't need to do anything with the lightcolor_spd parameter. Indeed the parameter doesn't even have to be part of the shader, just included in the RIB shader declaration.

The Air distribution includes a small collection of SPD reflectance and illumination files in the `spectra` directory. A search path for SPD files can be specified using:

```
Option "searchpath" "string spectrum" ""
```

Spectral data is converted to an RGB color for rendering.

Illuminant Color Temperature

Air 11 allows the spectral profile for an illuminant to be specified by entering a color temperature as the SPD file name value. E.g.,

```
LightSource "distantlight" 1 "string lightcolor_spd" "2400" # sunset
```

Solar Spectrum

Air 13 introduces a new capability to define a spectrum using a new builtin sun() function instead of an SPD file. There are two forms. The first is:

```
"sun(zenith=45,turbidity=5)"
```

zenith is an angle in degrees giving the sun's elevation measured from the vertical
turbidity is a the same value passed to the sunlight shader.

The second form is:

```
"sun(month=4,day=15,hour=12,timezone=-8,latitude=47,longitude=-
122,turbidity=5)"
```

The parameters have the same meaning as those in the sunlight light shader.

This new capability allows spectral prefiltering and chromatic adaptation to work with a sun light.

See Also

Spectral Prefiltering and Chromatic Adaptation

3.2 Spectral Prefiltering and Chromatic Adaptation

Spectral Prefiltering

In "Picture Perfect RGB Rendering Using Spectral Prefiltering and Sharp Color Primaries" Greg Ward and Elena Eydelberg-Vileshin describe a method of improving the accuracy of RGB rendering without the performance degradation associated with full spectral rendering. Their spectral prefiltering technique is based on a few simple observations:

1. Direct lighting results are the most important for overall image accuracy.
2. Most scenes have a single dominant illuminant. There may be many light sources, but most tend to share the same spectral profile.
3. Scenes with spectrally distinct light sources cannot be properly white balanced and will look "wrong" no matter how accurately the colors are computed.

These observations suggest the following prefiltering algorithm:

> "We apply a standard CIE formula to compute the reflected XYZ color of each surface under the dominant illuminant, then transform this to a white-balanced RGB color space for rendering and display. The dominant sources are then replaced by white sources of equal intensity, and other source colors are modified to account for this adaptation. By construction, the renderer gets the exact answer for the dominant direct component, and a reasonably close approximation for other sources and higher order components."

In Air spectral prefiltering is enabled by using the following option to specify the dominant illuminant for the scene:

```
Option "render" "string dominantilluminant_spd" ""
```

Lights with the same spectrum as the dominant illuminant should emit white light. Other lights will have their `lightcolor_spd` (if present) divided by the dominant illuminant to compensate for the material color premultiplication. Note that if all lights have the same spectral profile (and default white light color), adding the above option will effectively change the spectrum of all lights without requiring any change to the way the lights are declared in the RIB file.

Chromatic Adaptation

Once we start using real spectral data for rendering, we also need to take better account of how the human visual system processes that data. According to Ward and Eydelberg-Vileshin:

> "there is a strong tendency for viewers to discount the illuminant in their observations, and the color one sees depends strongly on the ambient lighting and the surround. For example, [our equation] might compute a yellow-orange color for a white patch under a tungsten illuminant, while a human observer would still call it 'white' if they were in a room lit by the same tungsten source."

To compensate for the viewer's chromatic adaptation, they suggest transforming color data from the color space of the dominant illuminant to a that of an illuminant corresponding to the display viewing conditions.

In Air chromatic adapation is enabled by defining an illuminant for the display viewing conditions with:

```
Option "render" "string displayilluminant_spd" ""
```

Chromatic adaptation also requires specification of a dominant illuminant (via the option mentioned above under Spectral Prefiltering). For images with an assumed sRGB color space, a D65 display illuminant (`$AIRHOME/lights/light_CIE_D65_noon_daylight`) is an appropriate choice.

SPD Data

Air 10 includes sample SPD data in a new `spectra` directory. The `lights` subdirectory includes SPD profiles for a number of standard CIE illuminants. The `colors` subdirectory contains SPD data for colors in the Macbeth colorchecker chart. The `metals` directory includes a few metallic profiles.

Material Reflectance

When a color value is specified using an SPD file, the effects of the dominant illuminant and chromatic adaptation are automatically included in the conversion to RGB.

By default the effects of DI and CA are not applied to colors specified as RGB values. Those effects can be included by changing the "current" color space for RGB colors to the special "litRGB" space:

```
ColorSpace "current" "litRGB"
```

When current color space is set to "litRGB", spectral prefiltering and chromatic adaptation are automatically applied to color attributes and shader parameters by first converting the RGB values to a spectral reflectance and then applying the spectrum-to-rgb conversion used for SPD files.

Shaders that allow the diffuse color to be modulated by a texture map can be modified to include the effects of spectral prefiltering and chromatic adaptation by converting the texture map color to "current" space:

```
color Ct = texture(...);
Ct = ctransform("rgb", "current",Ct);
```

When current space is "litRGB", the color is transformed as per the description above. With the default current space of "rgb", the above transform does not change the input color.

Care must be taken with metallic shaders that use the standard color attribute value to modulate reflections because the color used to modulate reflections should not include the effects of spectral prefiltering. To handle this situation, Air provides an option to query the raw SPD value for the standard Color attribute:

```
color spdrgb;
if (attribute("rawcolorspdrgb",spdrgb)==1) {...}
```

When an SPD file is used to specify the material color, a reflective metal shader should use the raw rgb value to modulate reflections.

See Also

Spectral Color Definition

3.3 Tone Mapping

Air 11 provides a basic tone-mapping operator, available as a display mode in Air Show and as an imager shader (ToneMap).

A tone mapping operator converts a high-dynamic range image into a low-dynamic range image for display. A tone-mapped image should normally be viewed with sRGB or custom gamma correction or an equivalent LUT.

The tone-mapping operator in Air Show and the ToneMap imager shader is based on the global operator described in "Photographic Tone Reproduction" by Erik Reinhard et al. It has three parameters:

Middle Luminance: the input luminance value corresponding to a mid-range grey
Middle Grey: the output value for the middel luminance (between 0 and 1)
Maximum Luminance: the input luminance value that should be mapped to 1.

The tone mapping operator is inspired by the zone system in photography. First, the image luminance is scaled linearly to map the Middle Luminance value to the Middle Grey output value. The dynamic range is then non-linearly compressed so that the maximum luminance value is mapped to 1. If the maximum luminance parameter value is less than the maximum luminance in the image, sections of the image will appear overexposed.

For speed Air Show and the ToneMap shader use an approximation that in some cases can produce output values that are still greater than 1. The `ClampToAlpha` parameter in the ToneMap imager shader can be used to ensure that all output values are less than or equal to 1.

4 Image Quality

High quality images require the elimination of *aliasing* - artifacts in an image due to undersampling some aspect of the scene. AIR provides extensive support for fine-tuning the sampling in a scene. The following three commands that determine overall image quality:

- The `PixelSamples` option determines how often the scene's geometry is sampled.
- The `PixelFilter` option determines how individual pixel samples are combined into the final pixel color.
- The `ShadingRate` attribute determines how often an object's shading is evaluated.

AIR provides independent control over the sampling of geometry and shading. This is an important capability: for many scenes rendering time is dominated by shading calculations. With AIR you can increase the sampling of a scene's geometry to smooth out jagged edges without increasing the number of shading calculations that are performed.

Topics

Geometry Sampling
Shader Sampling

4.1 Geometry Sampling

The frequency with which scene geometry is sampled is determined by the `PixelSamples` setting:

```
PixelSamples xsamp ysamp
```

which defines a `xsamp` by `ysamp` grid used to sample the scene at each pixel.

Use more pixel samples to smooth out jagged edges due to inadequate sampling of scene geometry. Because geometric sampling is independent of shader sampling, increasing the number of pixel samples will not usually result in a large increase in rendering time.

Sub-pixel Edge Mask

AIR provides another means of anti-aliasing jagged edges. If the following option is enabled:

```
Option "render" "integer edgemask" [1]
```

AIR will compute a high-resolution sub-pixel edge mask at each pixel sample to better approximate the area covered by each surface. The edge mask is particularly effective at anti-aliasing nearly horizontal and nearly vertical edges as well as capturing very small objects.

Using a sub-pixel edge mask can significantly slow down the renderer because it disables some of the optimizations normally used by AIR.

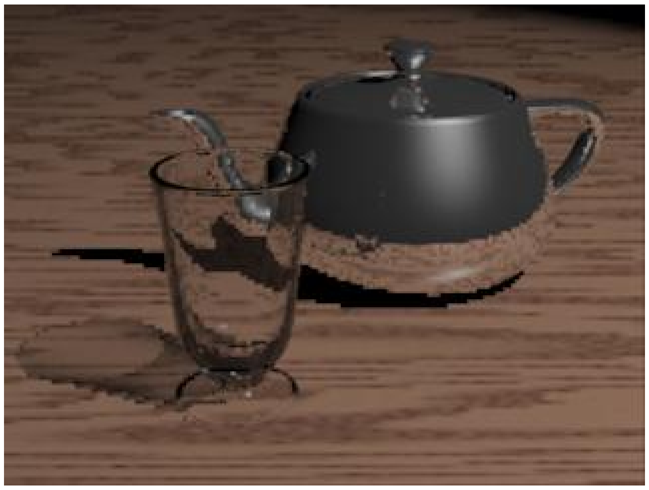
4.2 Shader Sampling

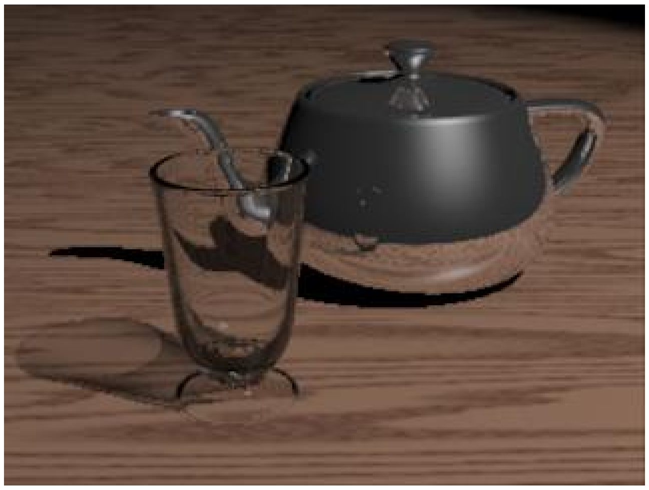
The frequency with which an object's shading is computed is determined by the shading rate attribute:

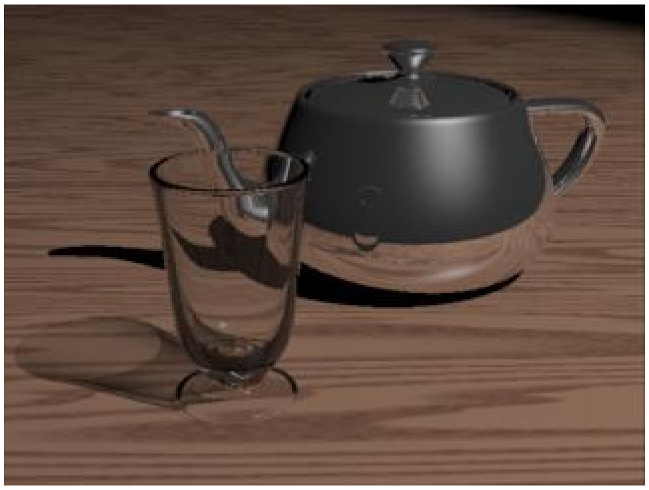
```
ShadingRate 1
```

The shading rate gives the area, in pixels, covered by a shading sample. If the shading rate is R, the renderer will compute 1/R shading samples per pixel. When R>1, shading samples are shared between adjacent pixels. When R<1, more than one sample is taken for each pixel.

The following images illustrate the effect of varying the shading rate:


ShadingRate 4 (0.25 shading samples per pixel)


ShadingRate 1 (1 shading sample per pixel)


ShadingRate 0.25 (4 shading samples per pixel)

When the shading rate is large and the number of samples is low (as in the top picture), the image appears blocky, since neighboring pixels share shading samples.

When the shading rate less than 1 (bottom image), more than one shading sample is taken per pixel, smoothing the shading. The bottom two images in the picture look similar, but close examination reveals that the shadow edges and reflections are smoothed in the image on the right.

The pictures also demonstrate that the rate at which shading is sampled is independent of the rate at which geometry is sampled. `PixelSamples` is 4 4 for all three images, and the geometric edges are smooth in all of them.

Understanding the difference between geometric sampling and shading sampling and the relation between the two is key to generating nice, anti-aliased images. In particular, increasing the number of pixel samples will not in itself alleviate aliasing in the shading. On the other hand, increasing pixel samples to deal with jagged edges doesn't incur any additional shading cost.

Micropolygon Shading

AIR 9 introduces a new method of shading surfaces, enabled with

```
Attribute "shade" "integer mode" [1]
```

The new mode breaks surfaces into small polygons (sometimes called micropolygons) for shading, with each micropolygon covering approximately the area (in pixels) specified by the shading rate.

This new mode may produce smoother images for highly reflective surfaces and may produce smoother results in slow motion animation.

4.3 Anti-Aliasing

The section describes how to eliminate common sources of aliasing in rendered images.

Jagged Edges

Try one or more of the following suggestions:

- Increase the number of `PixelSamples`
- Use a smooth `PixelFilter` with a wide filter width such as

  ```
  PixelFilter "gaussian" 2 2
  ```

- Enable the sub-pixel edge mask

  ```
  Option "render" "integer edgemask" [1]
  ```

Shading Aliasing

Shading can introduce aliasing from a procedural pattern, specular highlights, small bumps, shadow edges, and other sources. The simplest way to improve shading quality is to increase the number of shading samples by lowering the shading rate. For example, a shading rate of 0.5 will result in two shading samples per pixel, while a rate of 0.25 will yield 4 shading samples per pixel.

It is more efficient, however, to address the specific source of aliasing if possible. For example, if shadow edges are jagged, increasing the number of shadow samples will smooth out the shadows without requiring the entire shading computation to be performed multiple times.

Shadows

Both ray traced and shadow-mapped shadows can exhibit shading artifacts in the form of jagged edges or, for blurry shadows, excessive noise. In both cases increasing the number of shadow samples will improve shadow quality. Most light shaders provide parameter to set the number of shadow samples.

Reflections

Ray-traced reflections may exhibit artifacts in the form of jagged edges or excessive noise. As with shadows, increasing the number of samples or rays used will improve image quality. Most shaders with reflections include a parameter to setting the number of reflection rays.

4.4 Pixel Filters

Air supports all 5 pixel filters listed in the RenderMan Interface specification:

```
box
catmull-rom
gaussian
triangle
sinc
```

Air provides 3 additional filters

```
disk
mitchell
lanczos
```

Mitchell Filter

The mitchell filter is similar to the catmull-rom filter except that the mitchell filter has a finite support. Filter width should be 4 in each direction for the mitchell filter.

Lanczos Filter

The lanczos filter is like a sinc filter with a finite support. Filter width should be 6 in each direction for the lanczos filter.

4.5 Clamp to Alpha

Output values much greater than 1 can be difficult to adequately filter even with many pixel samples. To improve filtering of high-dynamic range images, Air will clamp output values to the current alpha value if the following option is enabled:

```
Option "render" "integer prefilterclamptoalpha" [1]
```

When this option is enabled, all output values will be less than or equal to 1.

4.6 Field Rendering

AIR 7 and later can render motion-blurred frames on "fields" if the following option is enabled:

```
Option "render" "integer fields" 1
```

Half the motion samples go to field 1, half to field 2, so the results are only fully correct for full frame motion blur. For best results pixel samples should be a power of 2 - 4 8 or 16 with xsamples=ysamples.

4.7 Hidden Surface Methods

Air supports several primary hidden surface methods. The method used for a given frame can be specified using the Hider command:

```
Hider "name" parameter list
```

All methods render the image one tile at a time. The following methods are supported:

Scanline

```
Hider "hidden"
```

The scanline method resolves hidden surfaces by scanning polygons and other primitives into a high-resolution buffer, one row at a time. Note that this algorithm is not the REYES algorithm used by some other renderers.

The scanline algorithm employs a raster-based shading cache to avoid re-shading at every pixel sample. This method also supports a sub-pixel edge mask for capturing sharp edges and small or thin objects.

Ray Tracing

```
Hider "raytrace"
```

Air 12 introduces a new primary hider using ray tracing. The ray tracing hider ignores the shading rate setting - each pixel sample is shaded separately, which will in general take more time than scanline rendering with the shading cache. For scenes using ray tracing heavily anyway, the performance penalty may be modest. Some new effects are possible with the trace hider such as shading at different time values for motion blur. Jittered sampling for motion blur and dof can be enabled with the following extra parameter:

```
Hider "raytrace" "jitter" 1
```

Deep Image

```
Hider "deepimage"
```

The deep image hider can store more than one depth sample at each pixel, which is useful for rendering deep shadow maps or deep images for compositing.

By default the deep image hider stores all surface samples at each pixel. When rendering deep images, the hider will remove samples that are occluded by samples closer to the camera when the following option is enabled:

```
Hider "deepimage" "culloccluded" [1]
```

See Also

Deep shadow maps
Deep images

5 Output

An AIR rendering process produces one or more raster images. Each image is passed to a user-specified display driver which may save the image to a file or display it in a window. AIR comes with display drivers for most common image formats. A current list of AIR display drivers can be found here.

A plugin or host application should provide an interface for specifying the output images rendered by AIR.

In a RIB file an output image is declared using the `Display` call which has the following syntax:

```
Display "filename" "drivername" "channels" parameter list
```

Here's an example:

```
Display "test.tif" "file" "rgba"
```

Driver Specification

The driver name is used to select the display driver. If the driver name is "file", AIR uses the file name extension to choose a display driver. When the driver name is "framebuffer", the image is sent to the AIR Show framebuffer instead of being saved to a file.

Channels

The RenderMan standard specifies 5 standard output channels: r, g, b, a, and z corresponding to the red, green, blue, alpha, and depth output values.

AIR also allows any output variable from a surface, displacement, or atmosphere shader to be saved to a file. Most AIR surface shaders provide a standard set of output variables described in more detail in the multipass rendering section. Each additional output value must have its type declared either using the Declare RIB statement or an "inline" type declaration in the Display call. For example:

```
Declare "__diffuse" "varying color"
Display "diffuse.tif" "file" "__diffuse"
```

Multiple channels may be specified for a single image with a comma-separated list:

```
Display "all.tif" "file" "rgba,color __diffuse,color __specular"
```

The display driver and image format must of course support more than 4 output channels of data. The list of display drivers indicates how many channels each driver supports.

Quantization and Image Data Precision

AIR generates all data as 32-bit floating point values. Depending on the file format used, the image data may need to be converted to 8-bit or 16-bit precision prior to being passed to the display driver. The output precision or quantization is set with the RIB `Quantize` command or by adding a "quantize" parameter to a `Display` call.

```
Quantize "channels" one min max dither
Display ... "float[4] quantize" [zero one min max]
```

Typical values for 8-bit precision are

```
Quantize "rgba" 255 0 255 0.5
Display "test.tif" "file" "rgba" "float[4] quantize" [0 255 0 255] "float
dither" [0.5]
```

Typical values for 16-bit precision are

```
Quantize "rgba" 65535 0 65535 0.5
Display "test.tif" "file" "rgba" "float[4] quantize" [0 65535 0 65535]
"float dither" [0.5]
```

For floating-point precision use the following values:

```
Quantize "rgba" 0 0 0 0
Display "test.tif" "file" "rgba" "float[4] quantize" [0 0 0 0] "float
dither" [0]
```

Multiple Output Images

Multiple images can be produced from a single rendering by providing additional `Display` calls with the file name prepended with a +. For example:

```
Display "mp_beauty.tif" "file" "rgba" "quantize" [0 255 0 255]
Display "+mp_diffuse.tif" "file" "varying color __diffuse" "quantize" [0
255 0 255]
Display "+mp_specular.tif" "file" "varying color __specular" "quantize"
[0 255 0 255]
```

Selecting Channel Components

Sometimes it may be helpful to save only a single channel from a color output variable. A single component can be specified by appending an index after the variable name in the `Display` call. For example, the following declaration creates a 3 channel file with the first (red) component of each output variable:

```
Declare "__specular" "varying color"
Declare "__diffuse_unshadowed" "varying color"
Declare "__shadow" "varying color"

Display "key.tif" "file"
"__diffuse_unshadowed[0],__shadow[0],__specular[0]"
```

Per-Frame Display Names

If a display file name contains the special formatting sequence `#nf`, AIR will automatically substitute the current frame number for the sequence. E.g.,

```
Display "test#4f.tif" "file" "rgb"
```

would become

```
Display "test0023.tif" "file" "rgb"
```

for frame 23. The frame number is taken from the `FrameBegin` statement in the RIB file.

5.1 Multipass Rendering

Multipass rendering is a method of improving the efficiency and flexibility of image creation by rendering different attributes of a scene as separate images. Rendering in passes allows the final image to be created by combining and fine-tuning individual passes in a compositing program in real time without the need to re-render.

Air can produce multiple images from a single render recording arbitrary output values created by a surface shader, permitting multipass rendering to be accomplished with a single invocation of the renderer. Most of the surface shaders included with Air have extra output variables containing a breakdown of the emitted color into the following components:

Name	Description
`__ambient`	ambient color
`__diffuse`	diffuse color
`__specular`	specular color
`__reflect`	reflected color
`__refract`	refracted color
`__environment`	reflected background color
`__incandescence`	self-illuminated color
`__constant`	unlit color

All the above variables are of type color. Most surface shaders only export a few of these variables. E.g., a surface without reflections obviously has no need to export a reflection output variable.

Many shaders further break down the diffuse component into 3 parts:

Name	Description
`__diffuse_unshadowed`	unshadowed direct lighting
`__diffuse_shadow`	shadowed portion of direct lighting
`__indirect`	indirect lighting

The 3 passes above can be combined to produce a diffuse layer in a compositing program:

```
__diffuse = (__diffuse_unshadowed - __diffuse_shadow) + __indirect
```

The `__diffuse_shadow` variable was introduced in Air 13. The above relationship is preserved through the per-pixel compositing and filtering performed by Air, and it is now the preferred method of breaking down the diffuse component.

Most Air surface shaders also provide a `__shadow output` variable that computes a shadow value based on the ratio between the shadowed and unshadowed direct lighting:

```
__shadow = 1 - (__diffuse - __indirect)/ __diffuse_unshadowed
```

and the corresponding equation for compositing is:

```
__diffuse = __diffuse_unshadowed * (1 - __shadow) + __indirect
```

However, the above relationship is not preserved through filtering and compositing (because the equation is not linear), and it may lead to artifacts along object edges and in regions with partial opacity.

Extra passes in a RIB File

To save an extra output variable, declare the variable and then add an additional `Display` to the renderer's output. For example, you could add the following to a scene file to record a diffuse pass:

```
Declare "__diffuse" "varying color"
Display "+pic_diffuse.tif" "file" "rgb"
  "quantize" [0 255 0 255]
```

By default additional output variables are saved as floating-point values. The optional `"quantize"` parameter to the `Display` call can be used to save 8-bit or 16-bit data instead. The `quantize` parameter consists of 4 floats - *zero*, *one*, *min* and *max*. The `Display` call above records 8-bit data. A `Display` call to save 16-bit data would look like:

```
Display "+pic_diffuse.tif" "file" "rgb"
  "quantize" [0 65535 0 65535]
```

Multiple variables may also be saved to a single file by providing a comma-separated list of output variables to a `Display` call. For example:

```
Declare "__diffuse" "varying color"
Declare "__specular" "varying color"
Declare "__reflection" "varying color"

Display "mp.psd" "file" "rgba,__diffuse,__specular,__reflection"
  "quantize" [0 255 0 255]
  "float[2] exposure" [1 1]
```

Only a few file formats support more than 4 channels of output. The Photoshop, TIFF, and OpenEXR drivers can all save additional channels.

Disabling Gamma Correction

Gamma correction should normally be disabled for an image being assembled in a compositing program. You can do this by adding an `Exposure` command at the beginning of a scene file:

```
Exposure 1 1
```

or by adding the following parameter to a `Display` call:

```
"exposure" [1 1]
```

Both set gain and gamma correction to 1.

Depth Pass

AIR has a special option for selecting the range of depth values to be saved as part of a multipass render:

```
Option "render" "float[2] zrange" [zmin zmax]
```

If *zmin* is not equal to *zmax*, z depth values are scaled so that z is 0 at *zmin* and 1 at *zmax*, and clamped to the range 0..1.

By default AIR filters depth values by taking the minimum depth value in each pixel. You can apply the same filtering used for color to depth values by setting the zfilter option:

```
Option "render" "string zfilter" "normal"
```

Example

A sample RIB file for rendering multiple passes can be found in

```
$AIRHOME/examples/multipass
```

5.2 Rendering in Layers

Rendering in layers refers to the process of rendering different elements or objects in a scene as separate images, which are used to assemble the final image in a 2D compositing or paint program.

Beginning with version 5, Air supports rendering multiple layers in a single rendering pass by allowing user selection of the objects that appear in each output image using groups.

Specify the objects visible in a given `Display` call by providing a `"string subset"` parameter with a list of groups:

```
Display "background.tif" "file" "rgb" "string subset" "ground tree"
```

Only objects in the listed groups will appear in the display image; other objects will be treated as matte objects for this image only.

A matte for a set of objects can be generated by rendering an alpha channel for the group:

```
Display "matte.tif" "file" "a" "string subset" "tree"
```

Air 11 and later allow a separate group to be specified for each output variable in a multichannel file by providing a comma-separated list of groups as the subset value. For example:

```
Display "all.tif" "file" "rgba,rgba,rgba" "string subset"
",terrain,agents"
```

An empty entry in the list of groups corresponds to no group subset. The above `Display` call would produce an image with three layers containing, respectively, all objects, terrain objects, and agent objects.

Examples

A sample RIB file for rendering layers in a single pass:

```
$AIRHOME/examples/multipass/multilayer.rib
```

A sample RIB that renders mattes for different elements:

```
$AIRHOME/examples/multipass/mattes.rib
```

5.3 Light Channels

Air 5.0 and later provide shaders and shading language functions for per-light output channels from surface shaders. Per-light channels can then be combined in a compositing or paint program to produce a final image.

The availability of per-light output depends on cooperating light and surface shaders. All Air lights provide a `__channel` output variable for per-light output. Set the `__channel` parameter to the channel number in which to store the light's output. Note that more than one light may share the same channel; each channel holds the sum of the illumination of all lights with that channel number. If you do not wish to store a per-channel value for a given light, set it's `__channel` parameter to -1.

Surface shaders provide per-light output as a `__lights` output variable which is an array of color values, one color for each channel. The surface shaders included with Air provide up to 10 channels of per-light output, with channel numbers 0 to 9.

To save light channels, first declare the `__lights` array:

```
Declare "__lights" "varying color[10]"
```

Each channel can then be referenced using an index into the `__lights` array in a `Display` call. E.g.,

```
  Display "reference.tif" "file" "rgba"
  Display "+light_0.tif" "file" "__lights[0]"
  Display "+light_1.tif" "file" "__lights[1]"
  Display "+light_2.tif" "file" "__lights[2]"
```

Per-light values should normally be saved with gamma correction disabled (gamma=1).

Detailed Per-Light Channels

Some shaders allow per-light output to be further broken down into diffuse, unshadowed, shadow, and specular components by providing additional output variables:

```
  Declare "__lights_diffuse" "varying color[10]"
  Declare "__lights_unshadowed" "varying color[10]"
  Declare "__lights_shadow" "varying color[10]"
  Declare "__lights_specular" "varying color[10]"

  Display "+diffuse_0.tif" "file" "__lights_diffuse[0]"
  Display "+unshadowed_0.tif" "file" "__lights_unshadowed[0]"
  Display "+shadow_0.tif" "file" "__lights_shadow[0]"
  Display "+specular_0.tif" "file" "__lights_specular[0]"
```

The VBlinn surface shader provides these additional output variables.

Tracking Light Channels through Reflections

Beginning with AIR 9 most AIR surface shaders with reflections allow light channels to include the effects of reflection and refraction. The AIR shaders look for the following user option to enable light tracking through reflections:

```
  Option "user" "float reflectlights" [1]
```

Note that light channel tracking must be explicitly supported by a surface shader.

When this feature is enabled, the `__reflect` and `__refract` output variables are no longer applicable to the composite. However, the light channels will not include the contribution of the background environment map for reflections (if any). The `__environment` output variable from the surface shader will contain that contribution.

Tracking Light Channels through Indirect Illumination

AIR 9 and later can track light channels through indirect diffuse light bounces. This capability is provided in the indirectchannels light shader, which can be substituted for the indirect light shader normally used for indirect diffuse illumination.

Shading Language Support for Light Channels

AIR enables shaders to efficiently provide per-light output values with special extensions to the standard diffuse, specular, phong, blinn, and brushedspecular illumination functions. These functions provide a "`channels`" output variable, an array of colors, containing the function result for each light, indexed by a light shader's `__channel` output variable. A "`nchannels`" output variable gives the number of valid channels. The diffuse() function also provides an "`unshadowedchannels`" output with unshadowed diffuse results for each channel.

Example

```
$AIRHOME/examples/multipass/perlightpasses.rib
```

5.4 Stereo and Multicamera Rendering

AIR 10 introduces the capability to render images from multiple cameras in a single rendering pass. Rendering stereo pairs is one possible application of this capability.

Stereo rendering requires two modifications to a typical RIB scene file:

1. Define an additional world-to-camera space transformation for the alternate viewpoint using the new Camera rib command.
2. Assign the alternate camera to at least one `Display` call with the new "`string camera`" extra parameter.

Here's a simple example:

```
Display "left.tif" "framebuffer" "rgba"
Display "+right.tif" "framebuffer" "rgba" "string camera" "right"

Projection "perspective" "fov" [ 11.6855 ]

TransformBegin
  ConcatTransform [ ... ] # transform for right view
  Camera "right"
TransformEnd

ConcatTransform [ ... ] # transform for left view

WorldBegin
```

A `Display` call with a `camera` parameter may appear before the definition of the camera transformation.

Each bucket is rendered once per camera, so with two cameras, each bucket will be rendered twice. Unlike some other stereo rendering implementations, view-dependent shading computations will be correct for every camera, and no shader modifications are required.

There should be at least one `Display` call for each camera, including the main or default camera. The current implementation assumes that all cameras have approximately the same view of the scene. Using cameras with very different points of view may produce unexpected results and degrade performance.

Air Show 5 has a new mode for viewing a stereo pair as an anaglyph image. When the stereo button is enabled, Air Show attempts to form a stereo pair with the current image and the following image; if that fails, Air Show tries the current image and the previous image. Stereo pairs must have the same image size, data format, and number of channels.

Use the Stereo Channels item in the Options menu to select which components of the anaglyph are taken from the first image. Once a stereo pair is formed, most display parameters will be synchronized for the pair.

5.5 Motion Vector Output

The global shader variable `dPdtime` stores the motion vector at the current shading location. Typically the raw vector needs to be converted to a form usable by a 2D application for post-process motion blur.

The Air distribution includes a ReelSmartMotion surface shader whose output is compatible with the ReelSmart Motion Blur plugin that illustrates this process. The source code for the shader can be found at `$AIRHOME/shaders/src/surface/ReelSmartMotion.sl`.

5.6 Deep Images

Air 11 introduces a new hider for saving "deep" image data. The deep image hider allows every fragment that contribute to a given pixel to be recorded to an output image.

Enable the new hider with:

```
Hider "deepimage"
```

When the deep image hider is enabled, the `PixelSamples` grid is set to 1x1, and the sub-pixel edge mask is enabled to generate sub-pixel coverage information.

The deepimage hider has one optional parameter:

```
Hider "deepimage" "boolean culloccluded" [1]
```

When `culloccluded` is enabled, Air omits fragments that are completely hidden by surfaces nearer the camera.

Air includes a deep image display driver that stores deep image data in a simple text file. Use the `.deeptxt` file name extension to select the deep image display driver:

```
Hider "deepimage" "culloccluded" [0]
Display "test.deeptxt" "file" "rgbaz,subpixelweight" "quantize" [0 0 0 0]
```

Deep image data can also be saved to any standard image format as a series of layers. Just include an extra "deeplayer" parameter in the Display declaration specifying the depth layer to record. E.g.,

```
Hider "deepimage" "culloccluded" [0]
Display "layer0.tif" "file" "rgbaz" "float depthlayer" [0]
Display "+layer1.tif" "file" "rgbaz" "float depthlayer" [1]
Display "+layer2.tif" "file" "rgbaz" "float depthlayer" [2]
```

An image with no transparent surfaces will have at most 16 fragments or layers per pixel.

Air provides two additional output variables for deep images:

- A `"float subpixelweight"` variable gives the contribution of the current fragment to the final pixel value. To reconstruct a final pixel value from a fragment list, take the sum of the product of each fragment and its sub-pixel weight.

- A `"float subpixelmask"` variable stores a 4x4 sub-pixel coverage mask as an integer between 0 and 65535. The sub-pixel mask can be used to accurately handle object edges when performing operations on deep images.

5.7 Display Drivers

The raster image created by Air is passed to a display driver, which may display it in a window or save it to disk. Air comes with the following display drivers:

File Format	Device Name	Extension	Data	# Channels
Windows Bitmap	`bmp`	`.bmp`	8-bit	3 or 4
OpenEXR	`exr`	`.exr`	float	arbitrary
Radiance	`hdr`	`.hdr`	float	3
JPEG	`jpeg`	`.jpg`	8-bit	3
Portable Network Graphics	`png`	`.png`	8-bit, 16-bit	1-4
Rich Pixel Format	`rpf`	`.rpf`	special	special
SGI Image Format	`sgi`	`.sgi`	8-bit, 16-bit	arbitrary
AIR Shadow Map	`shadow`	`.shd`	float	1
Targa	`tga`	`.tga`	8-bit	3 or 4
AIR Texture Map	`texture`	`.tx`	8-bit,16-bit,float	arbitrary
TIFF	`tiff`	`.tif`	8-bit,16-bit,float	arbitrary
Photoshop File Format	`psd`	`.psd`	8-bit, 16-bit	arbitrary
Softimage PIC	`pic`	`.pic`	8-bit	3 or 4
Deep Image Text File	`deeptxt`	`.deeptxt`	float	arbitrary
PTEX	`ptex`	`.ptx`	8-bit, 16-bit half, float	arbitrary

See Also

Output

5.7.1 bmp (Windows Bitmap)

Description: saves images in Windows Bitmap format

File extension: `.bmp`

Channels: single channel or 3 channels (rgb)

Data types: 8-bit unsigned integer

File data: uncompressed

Parameters:

`"float[2] resolution" [`*`x y`*`]`

Gives the resolution of the image in x and y in pixels per unit given by resolutionunit.

`"string resolutionunit" ["`*`unit`*`"]`

Gives the unit of measurement for the resolution of the image. Unit must be one of `inch` or `centimeter`.

5.7.2 deeptxt (Deep Image Text File)

Description: saves deep image data to a simple text file

File extension: `.deeptxt`

Channels: arbitrary number of channels, and arbitrary fragments per pixel

Data types: float

File data: uncompressed ASCII text

5.7.3 exr (OpenEXR)

Description: saves data in OpenEXR format

File extension: `.exr`

Channels: rgba,z

Data types: float

Parameters:

`"string exrpixeltype" ["half"]`

Defines the data type in which variables other than r,g,b,a,z are saved. Supported types are `"half"` (16-bit floating point) or `"float"` (32-bit IEEE floating point).

r,g,b, and alpha channel data is always saved at half precision. Depth data (z) is always stored at 32-bit float precision.

`"string exrcompression" ["zip"]`

Compression method: `none`, `rle`, `zip`, `zips`, `piz`, `piz12`

5.7.4 framebuffer

Description: displays an image in a window in the AIR Show display utility

Channels: any

Data types: 8-bit unsigned integer, 16-bit unsigned integer, floating point

Parameters:

`"integer[2] position" [x y]`

Specifies an offset in pixels for the upper left-hand corner of the display window.

`"string description" text`

Stores an optional description of the image.

5.7.5 hdr (Radiance Image Format)

Description: saves images in Radiance hdr (high-dynamic range) format

File extension: `.hdr`

Channels: 3 channels (rgb)

Data types: floating-point

File data: compressed

5.7.6 jpeg (JPEG Interchange File Format)

Description: saves images in JPEG File Interchange Format

File extension: `.jpg`

Channels: single channel or 3 channels (rgb)

Data types: 8-bit unsigned integer

File data: compressed (lossy)

Parameters:

`"float[2] resolution" [`*`x y`*`]`

Gives the resolution of the image in x and y in pixels per unit given by resolutionunit.

`"string resolutionunit" [`*`unit`*`]`

Gives the unit of measurement for the resolution of the image. Unit must be one of inch, meter, centimeter, or none. The default is none.

`"integer quality" [75]`

JPEG uses a lossy compression technique; the quality parameter determines how much error is allowed in the compressed image. The range is 0 to 100, with 100 providing the highest quality and the least compression.

5.7.7 pic (Softimage PIC)

Description: saves images in Softimage PIC format

File extension: `.pic`

Channels: 3 or 4

Data types: 8-bit unsigned integer

File data: uncompressed

5.7.8 png (Portable Network Graphics)

Description: saves images in Portable Network Graphics format

File extension: `.png`

Channels: single channel, single channel plus alpha, rgb, rgba

Data types: 8- or 16-bit unsigned integer

File data: compressed (lossless)

Parameters:

`"integer[2] origin" [`*`x y`*`]`

Specifies an offset in pixels for the upper left-hand corner of the image.

`"float[2] resolution" [`*`x y`*`]`

Gives the resolution of the image in x and y in pixels per unit given by resolutionunit

`"string resolutionunit" [`*`unit`*`]`

Gives the unit of measurement for the resolution of the image. Unit must be one of `"inch"`, `"meter"`, `"centimeter"`, or `"none"`. The default is none.

```
"string author" name
"string copyright" text
"string title" title
"string description" text
```

Each parameter takes a string with the appropriate information. By default none of these parameters are included in the output file.

5.7.9 psd (Photoshop)

Synopsis: saves images in Photoshop file format (including layer information)

File extension: `.psd`

Channels: up to 32 channels

Data types: 8-bit or 16-bit unsigned integer

File data: compressed or uncompressed

Parameters:

`"string compression"` *`type`*

Type is one of `"none"` or `"rle"`. By default 8-bit data is compressed using RLE compression. No compression is performed on 16-bit data.

Description:

The Photoshop display driver can write a simple Photoshop file with rgb or rgba information or a layered file with multiple output images ready for compositing in Photoshop or other applications.

Creating a Layered Photoshop File with AIR Control

- Start AIR Control and select the scene to be rendered
- Check **Save to File** and choose a file name with a `.psd` extension
- Set **Channels** to rgba and **Precision** to 16-bit (preferably) or 8-bit
- Set **Image gamma** to 1. (The composited image can always be gamma corrected in Photoshop.)
- Check **Extra Output Channels** and **in single image**
- Select the channels to save. Here are some common combinations

  Diffuse, Specular, Reflections
  Unshadowed, Shadows, Specular, Reflections (if the scene has no indirect lighting)
  Unshadowed, Shadows, Indirect, Specular, Reflections

- Click the green render button

You can also preview the layered image in AIR Show (by checking Display in AIR Show) and then save it as a psd file. Set the AIR Show viewing gamma to the same gamma you will later apply to the composited image (the AIR Show gamma doesn't affect the raw image data).

RIB Commands for Creating a Layered PSD File

```
Declare "__diffuse" "varying color"
Declare "__specular" "varying color"
Declare "__reflection" "varying color"

Exposure 1 1
Display "mp.psd" "file" "rgba,__diffuse,__specular,__reflection"
  "quantize" [0 255 0 255]
```

All layers should be of type color (except for AIR's special __toonline output variable which the driver recognizes as a single float).

5.7.10 ptx (PTEX)

Description: saves multiple images as a PTEX per-face texture file

File extension: `.ptx`

Channels: unlimited

Data types: 8-bit, 16-bit, half, float

File data: compressed

Parameters:

```
"float half" [1]
```

enables saving data as 16-bit floating point numbers ('half' precision). The display output quantization should also be set for float precision when saving half data.

`"float makemipmaps" [1]`

enables mip-map generation for the created PTEX file. Each face map will be stored at multiple resolutions for faster, higher-quality rendering.

See Also

PTEX: Per-Face Texture Mapping
Baking PTEX Textures with BakeAir

5.7.11 rpf (Rich Pixel Format)

Synopsis: saves data in Rich Pixel Format

File extension: `.rpf`

Channels: rgba,z,P,dPdtime,z,s,t

Data types: various

Parameters:

`"string author"` *`name`*

String identifying the "author" of the image.

`"string description" text`

Description of the image.

`"float bits" [8]`

By default R,G,B,A values will be saved as 8-bit values. To save RGBA data at 16-bit precision pass the bits parameter to the display call with a value of 16.

Description

RPF files are a useful way to pass extra information about a scene to a compositing package. In addition to the color and alpha information stored in most file formats, RPF files can also store depth values, texture coordinates, surface normals, motion vectors, and object or material identifiers. Because of the variety of output options available in the RPF driver, it is somewhat more complicated to set up and use than other display drivers. Please read the following carefully!

Every file using RPF output should have the following lines before any `Display` call:

```
Quantize "rgba" 0 0 0 0
Declare "__gBufferID" "constant float"
Declare "__mBufferID" "constant float"
```

The `Display` call must include `rgba` output. Additional variables can be included for storing extra information as a comma-separated list. Here's an example `Display` call that includes all optional output variables:

```
Display "test.rpf" "rpf"
```

```
"rgba,z,s,t,P,dPdtime,__gBufferID,__mBufferID"
```

Additional Output Channels

Depth (z)

By including "z" in the Display call, the z-depth of each pixel will be saved at floating-point precision. Saved values are the camera space z-values multiplied by -1.

Texture Coordinates (s,t)

Two-dimensional texture coordinates can be saved in the file by including "s,t" or "u,v" in the Display call. A custom pair of texture coordinates can also be saved by passing the variable names to the display driver. E.g.,

```
Display "rgba,u1,v1"
  "string xcoordname" "u1"
  "string ycoordname" "v1"
```

Custom texture coordinates must be assigned to an output variable in a surface shader in order to be saved.

Motion Vector (P,dPdtime)

Include P and dPdtime in the `Display` call to save a two-dimensional motion vector in the RPF file. Motion blur has to be enabled for the scene to produce meaningful values.

Material Identifier (__mBufferID)

This identifier will saves an 8-bit channel with a per-material number. The `__mBufferID` must be set by an output variable from a surface shader. Because the identifier may have been pre-scaled for other purposes, the RPF driver provides an optional `Display` parameter for re-scaling the input value:

```
"float materialidmultiplier" [1]
```

Object Identifier (__gBufferID)

This identifier will saves a 16-bit channel with a per-object number. The __gBufferID must be set by an output variable from a surface shader. Because the identifier may have been pre-scaled for other purposes, the RPF driver provides an optional `Display` parameter for re-scaling the input value:

```
"float objectidmultiplier" [255]
```

5.7.12 tga (Targa)

Description: saves images in Targa file format

File extension: `.tga`

Channels: rgb, rgba

Data types: 8-bit unsigned integer

File data: uncompressed

5.7.13 tiff (Tag Image File Format)

Description: saves images in Tag Image File Format

File extension: `.tif`

Channels: up to 32 channels

Data types: 8-, 16-, or 32-bit signed or unsigned integer; IEEE single-precision floating-point

File data: compressed or uncompressed

Parameters:

`"string description"` *`text`*

Stores an optional description of the image.

`"string compression"` *`type`*

Type is one of `"none"`, `"packbits"`, or `"lzw"`. By default no compression is used.

On Linux the tiff driver uses the libtiff library installed on the system, which may or may not support lzw compression. All modes are supported on Windows.

The default compression type can be set with an AIR_TIFF_COMPRESSION environment variable. E.g.,

```
AIR_TIFF_COMPRESSION=lzw
```

`"float[2] resolution" [`*`x y`*`]`

Gives the resolution of the image in x and y in pixels per unit given by resolutionunit.

`"string resolutionunit" ["`*`unit`*`"]`

Gives the unit of measurement for the resolution of the image. Unit must be one of `inch` or `centimeter`.

5.7.14 sgi (SGI Image Format)

Description: saves images in SGI image format

File extension: `.sgi`

Channels: any number

Data types: 8-bit unsigned integer, 16-bit unsigned integer

File data: uncompressed

Parameters:

`"string description"` *`text`*

Stores an optional description of the image.

5.7.15 shadow (AIR Shadow Map File)

Description: saves a depth channel as an ***AIR*** shadow map file

File extension: `.shd`

Channels: `z`

Data types: float

File data: compressed, tiled TIFF

Parameters:

`"float append" [1]`

Signals the driver to append the current shadow map to the given file. This option can be used to generate a cube-face shadow map or an occlusion map without creating temporary depth files.

`"float lerp" [1]`

Enables smooth interpolation of deep shadow map data when rendered.

5.7.16 texture (AIR Texture Map)

Description: saves images as AIR texture files (compressed, mip-mapped TIFFs)

File extension: `.tx`

Channels: up to 32 channels

Data types: 8-bit or 16-bit unsigned integer; float

File data: compressed

Parameters:

`"float tilesize"` *`size`*

Size for image tiles. The default size is 32.

`"string wrapmodes" "`*`Xwrap,Ywrap`*`"`

Sets the wrap modes in the X and Y directions, which determines how the renderer handles texture access outside the unit square. *Xwrap* and *Ywrap* should be one of `black`, `clamp`, or `periodic`. The default value is `"clamp,clamp"` - which clamps texture values at the boundary.

`"float[2] size" [`*`width height`*`]`

Explicitly specifies the size of the largest mip-map in the texture file. *Width* and *height* should be a power of 2 and will be rounded up to a power of 2 if they are not.

`"float incremental" [1]`

When this parameter value is included in the Display call, the driver expects data in scanline order, and writes out each tile as it is completed. This option minimizes the amount of temporary memory required when writing large images.

5.8 Custom Display Drivers

Display drivers for AIR are written as dynamically linked libraries on Windows or shared objects on Linux. AIR can use display drivers written for PRman, and AIR also has it's own display driver architecture documented below.

AIR Display Driver Structure

An AIR display driver must implement three functions, defined in the `SID.h` header file in `$AIRHOME\include`.

Required Functions

```
SID_OpenDisplay

__declspec (dllexport) int SID_OpenDisplay(SID_Info *di, int n,
SID_Parameter *p);
```

This function is called when the display is opened (usually at `WorldBegin`). The `SID_Info structure` provides the following information

```
typedef struct {
  void *globaldata;
  void *localdata;
  char *errmsg;
  int width, height;  /* image width and height */
  float par;          /* pixelaspectratio */
  float gamma;        /* gamma applied prior to final quantization */
  int nch;            /* number of output channels */
  int chtype;         /* channel type */
  char *filename;     /* filename passed to RiDisplay */
  char *modes;        /* modes passed to RiDisplay */
  char *software;     /* string describing software & version, e.g., "AIR
1.0.0" */
} SID_Info;
```

A display driver will normally allocate a structure to store its internal data and save a pointer to that structure in `localdata`. p points to an array of n parameters passed to `RiDisplay`. `SIDUtil.h` contains a few helper functions for processing parameters.

Any non-zero return value indicates failure. You can optionally return a pointer to an error message in errmsg.

```
SID_WriteRect

__declspec (dllexport) int SID_WriteRect(SID_Info *di, int xmin, int
xmaxplus1, int ymin, int ymaxplus1, void *data);
```

The write function is called repeatedly to pass image data to the display routine. di points to the same structure passed to the open display routine. xmin, xmaxplus1-1, ymin, and ymaxplus1-1 define the rectangle of image data. Currently image data is always passed as complete scanlines, in order from top to bottom. data is a pointer to the data for the rectangle, packed bytes in machine byte order. A non-zero return value should abort rendering, but this action is not presently implemented.

```
SID_CloseDisplay

__declspec (dllexport) void SID_CloseDisplay(SID_Info *di);
```

SID_CloseDisplay is called once to close the display and to allow the driver to free any internal resources. di points to the same structure passed to open display. This function may be called before all or any of the image has been passed to SID_WriteRect (if the user aborts the render, for example). It is up to the display driver to handle partially rendered images. The driver should either produce a valid image file (perhaps filling in missing image data) or delete an incomplete and invalid file.

Compiling a display driver

A display driver should be compiled as a DLL under Windows or a shared object on Linux. Exported function names must be in the DLL, not in a separate LIB file.

Making a driver available

To make a driver available for use by the renderer:

1. Place the driver in `$AIRHOME\displays`.

2. Add a new entry to the driver list in `$AIRHOME\displays\displays.txt`. A driver entry is four comma-separated fields:

```
devicename,extension,filename,prompt
```

`devicename`	name passed to RiDisplay to select this driver
`extension`	filename extension associated with this driver
`filename`	name of the driver file
`prompt`	description of this driver for dialog windows

A sample display driver is available by request.

6 Options

An option is a property of the scene as a whole (in contrast to an attribute which is a property of an individual object). A custom option can be added to a scene file using the `Option` command:

```
Option "optionname" "type parameter" [value]
```

Options must appear before the `WorldBegin` statement in a RIB file. Most plugins provide a mechanism for adding custom options to a scene.

The RenderMan® standard allows custom options to be defined for renderer-specific features. The following list enumerates the custom options supported by Air.

Shading

```
Option "render" "float shadingmultiplier" [1]
Option "render" "integer prmanspecular" [1]
Option "noise" "string algorithm" ["air"]
Option "limits" "integer shadingcache" [1024]
Option "limits" "integer texturememory" [10000]
Option "limits" "integer automapmemory" [10000]
```

Sampling

```
Option "render" "integer edgemask" [0]
Option "render" "integer fields" [1]
Option "render" "integer prefilterclamptoalpha" [1]
```

Ray Tracing

```
Option "trace" "integer maxdepth" [6]
Option "trace" "float minweight" [0.01]
Option "trace" "float reducesamples" [0.1]
Option "render" "boolean optimizetracememory" [1]
```

Geometric Representation

```
Option "limits" "integer splitpolyfaces" [25]
Option "limits" "integer splitmeshfaces" [25]
```

Runtime Control

```
Option "limits" "integer nthreads" [0]
Option "runtime" "integer priority" [0]
Option "render" "string bucketorder" ["rows"]
Option "limits" "integer[2] bucketsize" [32 32]
Option "runtime" "string verbosity" ["normal"]
```

Traced Reflections

```
Option "reflection" "color background" [0 0 0]
Option "reflection" "string envname" [""]
Option "reflection" "float envstrength" [1.0]
Option "reflection" "integer envsamples" [8]
Option "reflection" "float envblur" [0.0]
Option "reflection" "string envspace" ""
```

Indirect Illumination

```
Option "indirect" "integer environmentcache" [0]
Option "indirect" "integer maxbounce" [1]
Option "indirect" "string savefile" [""]
Option "indirect" "string seedfile" [""]

Option "indirect" "integer prepass" [0]
Option "indirect" "float prepassfactor" [0.5]
Option "indirect" "string prepasschannels" "__lights"
Option "indirect" "string prepassgroup" "list of groups"
Option "render" "syncprepass" [1]

Option "indirect" "color background" [0 0 0]
Option "indirect" "string background_spd" ""
Option "indirect" "string envname" [""]
Option "indirect" "float envstrength" [1.0]
Option "indirect" "integer envsamples" [4]
Option "indirect" "float envblur" [0.0]
Option "indirect" "string envspace" ""

Option "indirect" "moving" [0]
```

Occlusion

```
Option "occlusion" "string savefile" [""]
Option "occlusion" "string seedfile" [""]
Option "occlusion" "integer prepass" [0]
Option "occlusion" "float prepassfactor" [0.5]
```

Spectral Prefiltering and Chromatic Adaptation

```
Option "render" "string dominantilluminant_spd" ""
Option "render" "string displayilluminant_spd" ""
Option "render" "string currentcolorspace" "rgb"
```

Caustics

```
Option "caustic" "string savefile" [""]
Option "caustic" "string seedfile" [""]
```

Subsurface Scattering

```
Option "render" "float subsurfacetolerance" [0.005]
Option "render" "integer sssmultithread" [1]
Option "render" "integer sssautocache" [0]
Option "runtime" "integer sssinfo" [0]
```

Toon Rendering

```
Option "toon" "float maxinkwidth" [0.0]
Option "toon" "float fadethreshold" [0.0]
Option "toon" "float mininkwidth" [0.0]
Option "toon" "float inkwidthmultiplier" [1.0]
Option "toon" "string vectorexportfile" ""
```

Network Cache

```
Option "netcache" "string cachedir" ""
Option "netcache" "integer cachesize" [1000]
Option "netcache" "integer debug" [0]
```

Search Paths

```
Option "searchpath" "string dirmap" ""
Option "searchpath" "string dirmapzone" ("UNC" for Windows, "NFS" for
Linux)
Option "searchpath" "boolean dirmapdebug" [0]
Option "searchpath" "string texture" ["$AIRHOME/texture"]
Option "searchpath" "string archive" ["$AIRHOME/archives"]
Option "searchpath" "string shader"
["$AIRHOME/usershaders:$AIRHOME/shaders"]
Option "searchpath" "string display" ["$AIRHOME/displays"]
Option "searchpath" "string procedural" ["$AIRHOME/procedurals"]
Option "searchpath" "string spectrum" ["$AIRHOME/spectra"]
Option "searchpath" "string resource" [""]
```

6.1 Runtime Control

Multithreading

Air can use multiple rendering threads on machines with more than one processor to accelerate rendering. The number of threads can be set with a command line option:

```
air -p 2 myscene.rib
```

or a custom option in a scene file:

```
Option "limits" "integer nthreads" [2]
```

Each extra thread uses some additional memory.

The special value 0 may be used to indicate that all detected processing cores should be used, which is now the default value. In Air releases prior to version 10, the default number of threads was 1.

Process Priority

The priority of the rendering process can be set on Windows systems with

```
Option "runtime" "integer priority" [n]
```

Valid values are 0 (normal) and -1 (low). Rendering at low priority allows a foreground application such as a modeler to run unimpeded while Air renders in the background.

Tiled Rendering

Air renders a scene using small tiles or "buckets". Bucketed rendering can be very efficient for rendering large scenes. The size of the buckets can be set with

```
Option "limits" "bucketsize" [80 80]
```

Larger buckets render faster; smaller buckets use less memory.

The order in which buckets are rendered is determined by

```
Option "render" "string bucketorder" ["rows"]
```

The default `rows` order marches from top to bottom across the screen in a zig-zag pattern. To minimize memory use for an image that is wider than it is high, use `columns` order. `"spiral"` order will render from the center of the image outward, allowing the middle of the scene to be viewed sooner. Different orders usually have different peak memory requirements and different rendering times for a given scene.

Progress and Statistics Reporting

The following option controls progress and statistics reporting:

```
Option "runtime" "string verbosity" "normal"
```

Valid verbosity modes are:

`normal`	Progress report, no statistics
`stats`	Progress report and final stastics
`silent`	No progress report or statistics
`silentstats`	Statistics but no progress report

Identifier Retention

Air normally discards names to save memory and permit better sharing of attributes among primitives. Some shaders, however, require name information to work properly. When this option is enabled, Air will save identifier names so they can be queried by a shader.

```
Option "render" "integer savenames" [1]
```

A name can be assigned to a primitive with the following attribute:

```
Attribute "identifier" "string name" ["name"]
```

Memory Use

The maximum memory the renderer may use can be set with

```
Option "limits" "memory" [n]
```

gives the upper limit in megabytes. If memory use exceeds the limit, the rendering will be aborted. By default there is no upper limit.

6.2 Search Paths

```
Option "searchpath" "string texture" ["$AIRHOME/texture"]
Option "searchpath" "string archive" ["$AIRHOME/archives"]
Option "searchpath" "string shader"
["$AIRHOME/usershaders:$AIRHOME/shaders"]
Option "searchpath" "string display" ["$AIRHOME/displays"]
Option "searchpath" "string procedural" ["$AIRHOME/procedurals"]
Option "searchpath" "string spectrum" ["$AIRHOME/spectra"]
Option "searchpath" "string resource" [""]
```

These options set the search path for texture files, archive files, shader files, display drivers, and procedural primitives respectively. The resource search path is searched for any resource not found in its resource-specific search path.

A path is a colon-separated list of directories. The & character will be replaced by the current path. An environment variable can be included in a path by preceding its name with a $. For compatibility with other renderers and operating systems, forward slashes (/) should be used instead of backward slashes (\) to separate directory names. A directory name may be preceded by a drive letter followed by a colon; AIR will distinguish between a colon after a drive letter and a colon used to separate directories.

Directory Mapping

Air 11 introduces a new directory mapping option for automatically converting path names depending on the rendering machine's environment:

```
Option "searchpath" "string dirmap" "list of mappings"
```

Each mapping is defined by 3 strings within brackets:

```
[zone frompath topath]
```

The zone determines when the mapping is applied. The zone for the current rendering process can be set with:

```
Option "searchpath" "dirmapzone" "name"
```

The default zone is UNC under Windows and NFS under Linux.

When a directory mapping is active, Air checks the beginning of each file name for a match with the frompath value. If a match is found, the frompath prefix is replaced by topath. Here's an example:

```
Option "searchpath" "dirmap" "[NFS C:/ /usr/local/][UNC /usr/local/ C:/]"
```

With this mapping, a Windows machine would convert a reference such as

```
/usr/local/image.png
```

to

```
C:/image.png
```

As an aid to configuring directory mapping, some basic debugging information can be enabled with the following option:

```
Option "searchpath" "dirmapdebug" [1]
```

6.3 Simple RIB Filtering

AIR provides a simple mechanism for disabling specific commands in a RIB stream:

```
Option "render" "string commands" ["-cmd"]
Option "render" "string commands" ["+cmd"]
```

The first statement disables the specified command; the second statement re-enables it.

Example: To override the surface shader for every object in a RIB stream, one could use:

```
Surface "plastic"
Option "render" "commands" ["-Surface"]
```

See Also:

RIB Filters

6.4 Shading

Global Shading Rate Multiplier

The frequency with which a surface is shaded is determined by the shading rate attribute. The following option serves as a global multiplier for the shading rate property of all objects:

```
Option "render" "float shadingmultiplier" [1]
```

Values below one reduce the effective shading rate, increasing the number of shading samples per pixel for higher image quality at the expense of longer rendering time. The -sm command line option is shortcut for this option.

Shading Cache Size

To accelerate rendering AIR stores and reuses shading samples when possible. The maximum number of shading samples stored for a single polygon is given by

```
Option "limits" "integer shadingcache" [1024]
```

Normally the default works just fine. For scenes with many output variables and many objects blurred over large distances onscreen, the memory used by the shading cache can become a significant factor. The maximum memory used by the shading cache is reported in AIR's rendering statistics. Using a lower cache limit will reduce peak memory use at the expense of slower rendering.

Specular Highlight Algorithm

The following attribute can be used to select whether the standard shading language specular function uses the algorithm defined in the RenderMan Interface specification or an alternate algorithm that better matches the results of other renderers:

```
Option "render" "integer prmanspecular" [1]
```

Noise Algorithm

AIR 7 includes a new alternative noise algorithm:

```
Option "noise" "string algorithm" "perlin2"
```

The perlin algorithm uses higher-order interpolation for better results with displacement. The new algorithm is computed at double-precision for better precision at very low frequencies.

The default noise algorithm can be restored with

```
Option "noise" "string algorithm" "air"
```

6.5 Network Cache

AIR 6 and later can optionally create a local copy of texture maps stored on a network drive to speed texture access.

To enable local caching, provide a directory to use for the local cache with:

```
Option "netcache" "string cachedir" "cachedirectory"
```

The directory must exist. The cache size (in megabytes) can be set with

```
Option "netcache" "integer cachesize" [1000]
```

There is also a debugging option:

```
Option "netcache" "integer debug" [1]
```

In AIR 8 and later, 3D point maps, brick maps, and point clouds will also use a network cache if available.

6.6 File Management

These options apply to Linux only. Air will limit the number of open file descriptors if the following option is enabled:

```
Option "runtime" "boolean limitopenfiles" [1]
```

File descriptor management is currently limited to open texture files only.

The maximum number of open file descriptors can be set with:

```
Option "runtime" "texturefilepool" [512]
```

6.7 User-Defined Options

Arbitrary data can be stored in a frame as a "user" option:

```
Declare "bgenv" "string"
Option "user" "bgenv" "sky.tx"
```

Use the `option()` function to query user option values in a shader:

```
string bgmap="";
if (option("user:bgenv",bgmap)==1) {
}
```

Default User Options

Air, BakeAir, and TweakAir define a user option with the renderer name, `'air'`, `'bakeair'`, or `'tweakair'` respectively.

```
Option "user" "string renderer" "air"
```

Air 13 and later builds define a user option of type float with the current version number in the format

major.minor

These options may be useful in conditional RIB statements.

See Also

Attributes -> User-Defined Attributes

7 Attributes

Attributes are properties of a particular object, in contrast to options which apply to the scene as a whole. A custom attribute is assigned by placing an `Attribute` command in the scene file prior to the definition of an object:

```
Attribute "name" "type parameter" value
```

Most plugins provide a facility for assigning custom attributes.

The RenderMan® standard allows custom attributes to be defined for renderer-specific features. The following list enumerates the custom attributes supported by AIR. Each attribute links to its description in this manual.

Lighting

```
Attribute "light" "integer nsamples" [1]
Attribute "light" "float maxdist" [-1.0]
Attribute "render" "has_shadows" "yes"
Attribute "light" "integer automapsize" [1024]
Attribute "light" "float[4] automapscreenwindow" [0 1 0 1]
```

Visibility

```
Attribute "visibility" "integer camera" [1]
Attribute "visibility" "integer transmission" [0]
Attribute "shade" "string transmissionhitmode" ["primitive"]
Attribute "visibility" "integer trace" [0]
Attribute "visibility" "integer indirect" [0]
Attribute "visibility" "subset" [""]
```

Motion Blur

```
GeometricApproximation "motionfactor" [0]
Attribute "dbo" "integer motionsamples" [0]
Attribute "render" "float shutterscale" [1]
Attribute "render" "float shutteroffset" [0]
Attribute "render" "integer adaptivemotionblur" [1]
```

Ray Tracing

```
Attribute "trace" "float bias" [0.01]
Attribute "trace" "integer motionblur" [0]
Attribute "trace" "integer displacements" [1]
```

Displacement

```
Attribute "render" "integer truedisplacement" [0]
Attribute "displacementbound" "float sphere" [0.0]
Attribute "render" "integer normaldisplacement" [1]
Attribute "render" "integer triangledisplacement" [0]
Attribute "render" "integer smoothlevel" [1]
Attribute "render" "integer meshdisplacement" [1]
Attribute "render" "fastdisplacement" [1]
```

Indirect Illumination

```
Attribute "indirect" "float maxerror" [0.0]
Attribute "indirect" "float maxpixeldist" [20]
Attribute "indirect" "integer nsamples" [256]
Attribute "indirect" "string shading" ["shade"]
Attribute "indirect" "float strength" [1]
Attribute "indirect" "color averagecolor" [-1 -1 -1]
Attribute "indirect" "string averagecolor_spd" ""
Attribute "indirect" "float maxhitdist" [1000000000.0]
Attribute "indirect" "integer moving" [0]
Attribute "indirect" "integer reflective" [0]
Attribute "indirect" "integer adaptivesampling" [0]
Attribute "indirect" "integer prepass" [1]
```

Caustics

```
Attribute "caustic" "integer ngather" [75]
Attribute "caustic" "float maxpixeldist" [20]
Attribute "caustic" "color specularcolor" [0 0 0]
Attribute "caustic" "color refractioncolor" [0 0 0]
Attribute "caustic" "float refractionindex" [1]
Attribute "caustic" "float dispersion" [0]
```

Subsurface Scattering / Point Sets

```
Attribute "pointset" "string handle" ""
Attribute "pointset" "string cachemode" ""
```

Geometric Approximation

```
Attribute "limits" "integer divisionlevel" [16]
Attribute "divisions" "integer udivisions" [-1] "integer vdivisions" [0]
Attribute "render" "float trimborder" [0]
Attribute "curve" "string type" ["polyline"]
Attribute "point" "string type" ["default"]
Attribute "point" "integer clustermax" [5000]
Attribute "render" "string flatnessspace" "raster"
GeometricApproximation "edgelength" [0]
```

Toon Rendering

```
Attribute "toon" "integer id" [-1]
Attribute "toon" "color ink" [0 0 0]
Attribute "toon" "float inkwidth" [1]
Attribute "toon" "float zthreshold" [0.0]
Attribute "toon" "float silhouette" [0.5]
Attribute "toon" "integer frontback" [0]
Attribute "toon" "integer nakededges" [0]
Attribute "toon" "integer faceangle" [0]
```

Geometry Export

```
Attribute "render" "string meshfile" ""
Attribute "render" "integer makePref" [0]
Attribute "render" "integer makeNref" [0]
Attribute "render" "string meshspace" ["world"]
```

Miscellaneous

```
Attribute "render" "string color_spd" ""
Attribute "render" "integer runprogramthreads" [1]
Attribute "render" "string ptexcoordinates" "st"
```

7.1 Visibility

AIR provides several attributes that control where and how an object appears in a rendering.

Camera Visibility

By default all objects are visible from the camera and appear in the final rendered image.

```
Attribute "visibility" "integer camera" [1]
```

Set the camera visibility attribute to 0 to hide an object in the rendered image. Such a hidden object may still appear in reflections or cast shadows.

AIR's grouping capability can also be used to restrict objects that appear in a rendering to a list of groups:

```
Attribute "visibility" "subset" "list of names"
```

Only primitives in the supplied groups will be visible in the rendered image.

Visibility in Ray-Traced Reflections

In order for an object to appear in ray-traced reflections, it must be tagged with the following attribute:

```
  Attribute "visibility" "integer trace" [1]
```

By default all objects are invisible to reflection/trace rays.

The objects that appear in a particular reflection can be further customized by providing a subset of groups to the shading language functions trace(), environment(), and gather() used to cast reflection rays.

Visibility for Ray-Traced Shadows

Objects that should cast ray-traced shadows must be made visible to shadow rays with one of the following attributes:

```
  Attribute "visibility" "integer shadow" [1]
  Attribute "visibility" "integer transmission" [1]
```

By default all objects are invisible to shadow rays.

When a shadow ray strikes an object, AIR uses the following attribute to determine how the object affects the shadow ray:

```
  Attribute "shade" "string transmissionhitmode" "primitive"
```

For the default "primitive" mode AIR uses the object's opacity value at the intersection point as the transmission opacity. This shortcut saves rendering time by not evaluating the surface shader at the intersection point. For surface shaders that compute a complex opacity, the transmission hit mode should be set to "shader" to force evaluation of the surface shader for shadow rays. The shader transmission mode can be much, much slower than the primitive mode.

See Also

Ray Traced Shadows
Ray Traced Reflections

7.2 Sidedness

AIR renders all objects as polygons. For opaque objects, only those polygons facing the camera need to be rendered. For objects that are declared to be one-sided, AIR will remove back-facing polygons. By default all objects are two-sided, but making objects one-sided can considerably improve performance. Make an object one-sided with

```
  Sides 1
```

7.3 Grouping

AIR 2.3 and later supports grouping of objects for controlling visibility. Group membership is defined with

```
Attribute "grouping" "string membership" "list of names"
```

The list is a comma- or space-separated list of names. Membership can be defined relative to the current setting by prepending the list with a "+" to add groups or a "-" to subtract groups.

Groups provide a powerful mechanism for controlling object visbility in a scene, including:

- The visibility subset attribute can be used to define which objects are visible from the camera.
- Shadow casting groups can be used to precisely control which objects casts shadows from a particular light.
- Reflection groups restrict objects visible in traced reflections from a primitive
- A `Display` subset selects which objects appear in a particular output image, facilitating multilayer rendering and rendering of multiple mattes in a single render pass.

Shading Language Support for Groups

Certain ray-tracing functions allow ray hit tests to be restricted to objects in a set of groups. The `trace()`, `shadow()`, `environment()`, `occlusion()`, and `areashadow()` shading language functions accept an optional `"subset"` parameter that contains a list of groups. Only primitives in the subset will be tested for ray intersections.

7.4 User-Defined Attributes

Any variable can be declared and associated with a primitive as a "user" attribute:

```
Declare "fadecolor" "color"
Attribute "user" "fadecolor" [1 1 1]
```

User attributes can be queried in a shader with the `attribute()` function. E.g.,

```
color fadecolor=0;
attribute("user:fadecolor",fadecolor);
```

8 Primitives

Air supports all primitives defined in the RI Spec 3.2. Supported primitives include:

Polygons: convex and concave polygons with holes, single polygon or indexed meshes

Bilinear patches and patch meshes
Bicubic patches and patch meshes with arbitrary basis functions
NURB patches with trim curves

Catmull-Clark subdivision surfaces

Quadrics: spheres, tori, cones, cylinders, disks, hyperboloids, paraboloids

Points: point collections may be rendered as disks, spheres, patches, or particles
Curves: linear, cubic, and NURB curves can be rendered as polylines, ribbons, or tubes
Blobby blobby implicit surfaces

Procedurals: generate or load other primitives on-demand at render time

Additional Air-specific primitives:

Implicit surfaces based on grid data

Volume primitives

Procedure shader primitives

8.1 Approximation of Curved Surfaces

Adaptive Meshing

Air renders objects by converting them to polygons. By default Air automatically subdivides curved surfaces based on how they appear on the screen. The accuracy of the polygonal approximation can be controlled by giving a maximum deviation of the true surface from the polygon mesh using

```
GeometricApproximation "flatness" [flatness]
```

`Flatness` is given in units of pixels. The default value is 0.5

Air 11 adds a new criterion for tessellation density:

```
GeometricApproximation "edgelength" [0]
```

which gives the maximum length (in object space) for any edge in the resulting mesh. The default value of 0 disables this test, which is currently only supported for polygon meshes. This control may be useful for increasing mesh density when baking a mesh with BakeAir.

The automatic meshing routine for NURB surfaces repeatedly subdivides a surface until a surface fragment is flat or a maximum division level is reached:

```
Attribute "limits" "integer divisionlevel" [16]
```

The default value is adequate for most surfaces. A long or complex curved surface may require a higher value (indicated by sharp corners or gaps in the polygon mesh).

Static Meshing

Air 9 and later allow the flatness tolerance to be applied as an absolute tolerance in object space instead:

```
Attribute "render" "string flatnessspace" "object"
```

The default flatness space is "raster", the only other supported coordinate system. Using object space for flatness produces a tessellation for non-deforming objects that does not change with object or camera position. A constant tessellation may be useful for cases where changes in tessellation between frames cause unwanted visual artifacts.

Explicit Meshing

The subdivision of parametric curved surfaces such as NURBs, quadrics, and bicubic patches can be set explicitly using the following attribute:

```
  Attribute "divisions"
    "integer udivisions" [udivisions]
    "integer vdivisions" [vdivisions]
```

udivisions and *vdivisions* are the number of segments used to represent the object in the u- and v- parametric directions respectively. Curved primitives will be rendered using *udivisions* x *vdivisions* polygons. For example,

```
  Attribute "divisions" "udivisions" 12 "vdivisions" 6
  Sphere 1 -1 1 360
```

produces a sphere made of 72 polygons.

Curved primitives that share an edge should have the same number of divisions along the shared edge to prevent cracks from appearing.

Set *udivisions* to -1 to restore automatic subdivision.

8.2 Polygons

There are 4 RIB commands for defining polygonal geometry:

```
Polygon
PointsPolygons

GeneralPolygon
GeneralPointsPolygons
```

The first two commands should be used only for polygons that are convex and planar. The second two commands support concave polygons and polygons with holes at the expense of longer rendering times (since every polygon is checked for concavity).

Non-planar Quads

Many modeling packages allow the creation of nonplanar 4-sided polygons as part of a polygon mesh. Such geometry should be passed to AIR using the `GeneralPointsPolygons` command so that AIR can properly detect nonplanar quads and convert them to triangles where necessary. The threshold used to detect nonplanar quads can be set with

```
  Option "render" "float quadtolerance" [0.99]
```

Higher values force more quads to be split into triangles. The maximum value is 1.

Triangulation

The following attribute can be used to force AIR to convert all polygons in a mesh to triangles:

```
  Attribute "render" "integer triangles" [1]
```

Simplifying Mesh Data

In some cases the memory required for polygon meshes with facevarying data can be reduced by converting the facevarying data to vertex data with:

```
  Attribute "render" "integer simplifyfacedata" [1]
```

This optimization is often effective for Massive agents.

Mesh Displacement

AIR usually performs true displacment by splitting a mesh into individual polygons and then displacing each of the polygons independently on-demand. AIR 5 introduces a new option to displace an entire polygon mesh at once:

```
  Attribute "render" "integer meshdisplacement" [1]
```

Mesh displacement may take longer to render and use more memory during tessellation, but the resulting mesh will be smaller, and it will be continuous even if the driving displacement function is discontinuous. To help reduce memory with mesh displacement, AIR can automatically split a large mesh into smaller subregions when the following attribute is enabled:

```
  Attribute "render" "integer splitmesh" [1]
```

AIR processes the boundaries between regions so that continuity is preserved for the displaced surface. The maximum number of faces per region can be set with:

```
  Option "limits" "integer splitpolyfaces" [25]
```

Automatic Level of Detail

AIR 7 and later can automatically reduce the number of polygons in a polygon mesh based on an object's on-screen size and other factors. Automatic LOD applies only to polygon mesh primitives.

Enable automatic level of detail for an object with the following custom attribute:

```
  Attribute "render" "float autolod" [n]
```

where *n* gives an error tolerance when simplifying a model. Reasonable values to try are 0.1 to 0.3. A level of 0 disables automatic LOD.

Automatic LOD is also influenced by the "flatness" attribute:

```
  GeometricApproximation "flatness" 0.5
```

which gives the maximum difference in pixels between the original surface and the simplified surface. The default flatness value is 0.5. A value of 1 or 2 will produce smaller final meshes.

A reasonable set of values to start with are:

```
  Attribute "render" "float autolod" [0.2]
  GeometricApproximation "flatness" 1
```

8.3 Subdivision Surfaces

AIR supports Catmull-Clark subdivision surfaces with holes and creases.

Creases

Semi-sharp creases are implemented using a new "linear crease" subdivision algorithm, and AIR adds a new "linearcrease" tag to provide direct control of crease sharpness. The "linearcrease" tag uses the same syntax as the standard crease tag. Linear crease values should lie between 0 (no crease) and 1 (an infinitely sharp crease).

AIR automatically translates "crease" tags into "linearcrease" tags that produce approximately the same appearance, so AIR will work with existing plugins that export semi-sharp creases using the "crease" tag. But because AIR uses a different algorithm than Pixar's PRMan, the two renderers will not in general produce exactly the same results, though the general look should be similar.

Facevarying Mesh Data

AIR provides two methods of interpolating mesh data assigned as "facevarying" (i.e., one value per vertex per face). By default AIR will interpolate facevarying data linearly across the face of subdivision mesh. This linear interpolation can distort texture coordinates because the position vertices move in accordance with the subdivision rules while the texture coordinates move respond linearly. AIR provides an alternative interpolation that uses "vertex" interpolation for facevarying data whereever possible (whereever mesh data is continuous), with varying interpolation used only along seams between regions:

```
Attribute "render" "integer facevertex" [1]
```

Tessellation

By default AIR converts an entire subdivision surface to a polygon mesh for rendering in one step. AIR 5 and later can optionally split a subdivision surface into subregions for tessellation:

```
Attribute "render" "integer splitmesh" [1]
```

Tessellating by regions may reduce peak memory use and produce faster rendering in some cases. The following option gives the maximum number of faces per subregion:

```
Option "limits" "integer splitmeshfaces" [25]
```

When split mesh is enabled, AIR has another optimization that allows facevarying data to be simplied to varying or vertex data on submeshes where possible (ie, where a given vertex has the same values for each adjoining face):

```
Attribute "render" "integer simplifyfacedata" [1]
```

This optimization can reduce memory requirements significantly for objects with facevarying texture coordinates.

Mesh Displacement

AIR usually performs true displacment by first converting a primitive to a polygon mesh and then displacing each of the polygons. AIR 5 introduces a new option to displace an entire subdivision surface at once:

```
Attribute "render" "integer meshdisplacement" [1]
```

Mesh displacement may take longer to render and use more memory during tessellation, but the resulting mesh will be smaller, and it will be continuous even if the driving displacement function is discontinuous.

8.4 Points

The `Points` primitive is useful for efficiently rendering large numbers of tiny particle-like objects.

By default points are represented as tiny hexagons by the main scanline renderer, and this underlying geometric representation will be visible if a point is large. By default the ray tracer treats a point as a disk oriented perpendicular to the incoming ray.

Points as Patches for Sprites

Beginning with version 4 AIR can render `Points` primitives as bilinear patches which can be textured and used as sprites. Patch/sprite mode is enabled with

```
  Attribute "point" "string type" ["patch"]
```

Patches are oriented perpendicular to the camera, unless a per-particle normal `N` is provided in which case each point is oriented perpendicular to its normal vector. If normal vectors are provided, the primitive may also optionally contain a per-particle vector `xaxis` defining the horizontal orientation of the patch. When present the normal `N` and horizontal direction `xaxis` together precisely define the orientation of each patch.

The size of each patch is taken from the primitives varying `width` or `constantwidth` data. For non-square patches a Points primitive can include a constant or varying `patchaspectratio` parameter giving the width to height ratio for each patch. If the aspect ratio is less than 1, the patch width is scaled by the aspect ratio. If the aspect ratio is greater than 1, the patch height is divided by the aspect ratio.

Patches can be rotated by including a `patchangle` parameter, giving the rotation angle in degrees.

Any varying parameters attached to the `Points` primitive are converted to uniform parameters for the bilinear patches.

Points as Spheres

AIR 5 and later will render points as true spheres with

```
  Attribute "point" "string type" ["sphere"]
```

Points as Disks

AIR 8 and later can render points as disks with

```
  Attribute "point" "string type" ["disk"]
```

If a per-point normal `N` is provided in the point data, the disk will be oriented perpendicular to the normal vector. Otherwise, disks will be oriented perpendicular to the camera for scanline rendering, and perpendicular to the incoming ray direction when ray tracing.

Point Clusters

To make rendering more efficient AIR 8 and later can divide a large point primitive into smaller spacially coherent groups called clusters. The maximum number of points in a single cluster can be set with

```
  Attribute "point" "integer clustermax" [5000]
```

If the custermax attribute is set to 0, the point primitive will not be subdivided into clusters.

Shading Points

AIR includes a special particle surface shader for shading points that represent small particles for simulating phenomena like smoke or dust.

8.5 Curves

AIR supports the standard `Curves` primitive which can be used to define linear or cubic curves as well as NuCurves primitive (described below) for rendering NURB curves.

Cubic and NURB curves are converted to a view-dependent linear curve at render time based on the flatness attribute:

```
  GeometricApproximation "flatness" [0.5]
```

which gives the maximum deviation in pixels allowed in the linear approximation. Lower flatness values will produce more accurate curves which use more memory.

Curve Types

AIR supports rendering curves using 3 different representations based on the curve type attribute:

```
  Attribute "curve" "string type" [name]
```

Where the type name is one of `polyline`, `ribbon`, or `tube`.

Curves as Polylines

Curves are represented internally as a sequence of polyline segments. This representation of curves is designed for rendering large numbers of thin hair-like curves rapidly, and it assumes that curves are at most a few pixels wide on-screen. Wide curves may exhibit gaps between successive segments. The gaps can sometimes be closed by lowering the flatness criterion. Very wide curves should be rendered as ribbons (see next curve type).

Curves as Ribbons

The ribbon curve type stores each curve as a strip of rectangles or quads. The ribbon representation uses about twice as much memory as the polyine representation of curves.

Curves as Tubes

The tube curve type converts each curve to a generalized tube whose diameter is determined by the curve width. To maintain a consistent rotational orientation, the curve data should include a per-curve or per-vertex normal value N.

Curve Orientation

By default ribbon and polyline curves are oriented to face the incoming view direction. If a per-curve or per-vertex normal N is provided in the curve data, the curve will be oriented perpendicular to the normal vector. For simple nearly straight curves one normal vector per curve may be sufficient. For more complex curves, one normal per vertex should be provided to control the orientation along the entire curve length.

Shading Curves

When rendering hair or fur it is useful to shade the ribbon-like 3D curves to appear cylindrical to mimic the appearance of the corresponding natural fiber. AIR includes two surface shaders designed specifically to shade hair or fur, VHair and VFur. The different names aside, either shader may be used for either hair or fur.

Curve Tapering

Air 13 introduces new attributes that allow curves to be tapered at either or both ends:

```
Attribute "curve" "string tiptapertype" ["none"]
Attribute "curve" "float tiptaper" [0.5]
Attribute "curve" "string roottapertype" ["none"]
Attribute "curve" "float roottaper" [0.5]
```

For tube-shaped curves, tapering seals the end of the tube. For ribbon curves, tapering produces a leaf-like shape.

By default no tapering is applied. The `tiptaper` and `roottaper` values determine where along the curve the tapering begins, based on the `tiptapertype` or `roottapertype` selection:

`"width"`: taper value is multiplied by the curve width
`"length"`: taper value is multiplied by the curve length
`"absolute"`: taper value is treated as a fixed length

Set the taper type to `"none"` to disable tapering.

NuCurves

AIR implements a `NuCurves` primitive for rendering NURB curves in addition to the standard `Curves` primitive, which only supports linear and cubic curves. The RIB syntax for a `NuCurves` primitive is:

NuCurves [*nvertices*] [*order*] [*knot*] [*min*] [*max*] *parameterlist*

`nvertices`[i] gives the number of vertices in the i-th curve.

`order`[i] is the order of the i-th curve.

Each curve has (`order[i]+nvertices[i]`) knots, and all the knot vectors are concatenated into the knots array.

`min`[i] and `max`[i] define the range of evaluation of the i-th curve.

The parameterlist must contain at least position data (P or Pw). If a normal N is supplied, the orientation of the curve will adjust to remain perpendicular to the provided normal. Otherwise, the curve will be oriented to face the incoming ray direction.

Curve width is specified by either supplying a single `"constantwidth"` value or a varying `"width"` parameter with values for each curve.

Like other primitives arbitrary user defined values can be attached to a `NuCurves`. The number of values for each class of variable is:

constant	1
uniform	number of curves
varying	sum (nvertices[i]-order[i]+2)
vertex	total number of vertices

8.6 Blobbies and Dynamic Blob Ops

AIR supports all features of the Blobby implicit surface primitive except the repelling ground plane blob op.

Rendering of blobby primitives can be optimized by using a higher "flatness" criterion to reduce the density of the polygon mesh produced for rendering. Flatness is set with:

```
  GeometricApproximation "flatness" [0.5]
```

The default value is 0.5. Values of 0.7 or 1 can significantly reduce render times and memory use.

Threshold Tag

AIR supports a special `"constant float __threshold"` tag for Blobbies that sets the distance at which the surface is defined. The default value is 0.5 and the usable range is 0.1 to 0.9

Dynamic Blob Ops

In addition to the standard Blobby features, AIR allows users to define custom blob shapes with Dynamic Blob Ops. A DBO is written in a programming language such as C and compiled as a DLL on Windows or a shared object on Linux. The renderer automatically loads DBOs as they are needed by `Blobby` primitives.

DBO Structure

Each DBO must implement the following functions:

```
void ImplicitBound(State *s, float *bd,
                   int niarg, int *iarg,
                   int nfarg, float *farg,
                   int nsarg, char **sarg);
```

The `ImplicitBound` function returns a bounding box for the blob op in *bd. The bound is six floats in the order *minx maxx miny maxy minz maxz*. Arguments are those passed from the blob definition in the scene file (see below).

```
void ImplicitValue(State *s, float *result,
                   float *p,
                   int niarg, int *iarg,
                   int nfarg, float *farg,
                   int nsarg, char **sarg);
```

The `ImplicitValue` function returns a scaled distance from point *p to the blob in *result in the range [0..1]. For blending purposes the surface of the blob is assumed to be at a distance of 0.5. Points further away than a distance of 1 are not affected by the blob.

```
void ImplicitRange(State *s, float *rng,
                   float *bd,
                   int niarg, int *iarg,
                   int nfarg, float *farg,
                   int nsarg, char **sarg);
```

`ImplicitRange` returns in *rng the range of ImplicitValue results for all points in the bounding box bd. If the box is completely outside the influence of the blob, ImplicitRange should return [1 1].

A DBO may optionally have an `ImplicitFree` function that is called after each blobby has been processed:

```
void ImplicitFree(State *s);
```

Note that `ImplicitFree` may be called for a blobby even if the blobby does not contain references to the DBO.

Deformation motion blur is supported for DBOs that implement an ImplicitMotion function:

```
void ImplicitMotion(State *s, float *movec,
                    float *pt, float *times,
                    int niarg, int *iarg,
                    int nfarg, float *farg,
                    int nsarg, char **sarg);
```

`ImplicitMotion` returns a 3-element vector in *movec that represents the motion of point *pt over the time interval. The times parameter points to a 4-element array:

shutter open time, shutter close time, interval start time, interval end time

Deformation motion blur for a DBO must also be enabled in the RIB file with

```
Attribute "dbo" "integer motionsamples" [1]
```

If a DBO does not support deformation blur, it should not implement the ImplicitMotion function, and the above attribute should not be set.

The State structure contains two fields that uniquely identify the particular Blobby and the op within the blobby for which the DBO is being queried:

```
typedef struct {
  int BlobbyId;
  int OpId;
} State;
```

These values can be used as cache indices if the DBO employs a caching mechanism.

Using Dynamic Blob Ops in a Blobby

A dynamic blob op can be used like any other primitive blob in a `Blobby` object. DBOs have two required parameters and can have an arbitrary number of float, string, or integer parameters that are passed to the DBO functions.

```
9000 2 nameix transformix
9000 4 nameix transformix nfloat floatix
9000 6 nameix transformix nfloat floatix nstring stringix
9000 (7+nint) nameix transformix nfloat floatix nstring stringix nint
int_1...int_nint
```

nameix is an index into the string array for the name of the DBO. AIR will search the procedure search path for a DLL or shared object with the corresponding name.

transformix is an index into the float array for a matrix giving the blob-to-object space transformation.

floatix (if present) is the index in the float array of the first of the nfloat float parameters.

stringix (if present) is the index in the string array of the first of the nstring string parameters.

Example:

```
Blobby 2 [
  1001 0              # ellipse
  9000 4 0 16 1 32    # plane
  0 2 0 1             # add
 ] [
  1 0 0 0  0 1 0 0  0 0 1 0   4 0 .84 1
 10 0 0 0  0 3 0 0  0 0 1 0   0 0 0 1
  2
] [ "plane" ]
```

A sample DBO is included in the AIR distribution in:

`$AIRHOME/examples/primitives/blobbies`

8.7 Procedurals

Air supports the `RunProgram`, `DelayedReadArchive`, and `DynamicLoad` procedural primitives defined in the RenderMan standard. These procedural primitives allow Air to generate or load objects as they are needed during rendering instead of requiring all objects to be loaded before rendering begins. A procedural primitive is created with the `Procedural` RIB call:

```
Procedural "proctype" [arguments] [xmin xmax ymin ymax zmin zmax]
```

The last argument is an array of 6 floats defining a bounding box for all objects that will be created by the procedural primitive. When the renderer encounters the bounding volume of a procedural primitive, Air calls the corresponding procedural function to obtain the contents of the box. If the bounding volume is never encountered during rendering, the procedural primitive will never be called.

There are three standard Procedural primitives:

Procedural DelayedReadArchive

The DelayedReadArchive procedural (often abbreviated DRA) is the simplest procedural to use because it does not require any programming. A DRA takes the name of a RIB archive as its only argument. For example:

```
Procedural "DynamicLoad" ["teapot.rib"] [-3 3 -2 2 -3 3]
```

When the procedural is called it simply loads the contents of the specified archive.

Some plugins have a special RIB export mode that creates a separate archive for each object and main RIB file that references the object archives using DelayedReadArchive procedurals.

Procedural RunProgram

A RunProgram procedural allows an external helper program to be used to generate new objects during rendering. A sample RunProgram call looks like:

```
Procedural "RunProgram" ["maketree" "height 10 width 2 branches 5"] [-2 2
-2 2 0 10]
```

The first argument is the name of the helper program. Air automatically adds the `.exe` extension for programs under Windows. Air 8 and later recognize Python and Perl programs by their corresponding file name extensions and will automatically call the appropriate frontend program to execute those scripts. AIR searches the procedural search path for valid RunProgram procedurals.

The second argument is an arbitrary string that is meaningful to the helper program.

The helper program can be written in any programming language that can read from the standard input stream and write to the standard output stream. The helper program should be structured as a loop that waits for a line of input from the standard input stream.

The first time a RunProgram procedural is encountered, Air starts the help program as an external process connected to the rendering process using pipes.

Each time an instance of the procedural is encountered, Air sends the argument string from the instance to the helper program via the helper's standard input stream. Each argument string is preceded by a single number representing the approximate area of the procedural's bounding box on-screen (in pixels). This detail measure can be used to adapt the level of detail generated by the procedural to the size of the objects in the rendered image.

The renderer then waits for RIB commands to be sent to the helper program's standard output. The helper program must write a byte 255 to stdout when it is done processing the current instance. On some systems the helper program may need to flush the output stream to force the final byte to be sent to the renderer.

The Air distribution includes source code for sample RunProgram procedurals in C and Python in

```
$AIRHOME/examples/primitives/procedurals
```

Air 10 and later can execute multiple copies of a RunProgram procedural to accelerate multithreaded rendering. The maximum number of RunProgram instances is set with:

```
Attribute "render" "runprogramthreads" [1]
```

For the Massive RunProgram procedural, Air will automatically send the scene options string to multiple instances. The `-rpp` switch can be used to set the attribute value on the command line:

```
air -p 2 -rpp 2 myscene.rib
```

In Air 11 and later, the RunProgram procedural declaration may omit the bounding box, in which case the procedural is evaluated immediately.

Procedural DynamicLoad

A DynamicLoad procedural allows a shared object or dynamic link library to directly add objects to the scene by calling standard RI functions. A DynamicLoad procedural call is very similar to a RunProgram call:

```
Procedural "DynamicLoad" ["maketree" "height 10 width 2 branches 5"] [-2
2 -2 2 0 10]
```

The first argument is the name of the dynamic object (minus the .so or .dll extension).

The second argument is a string that is meaningful to the DynamicLoad procedural.

The Air distribution includes source code for sample DynamicLoad procedural in

```
$AIRHOME/examples/primitives/procedurals
```

Air's support for `DynamicLoad` procedurals makes use of a DLL (shared object on Linux) that acts as a bridge between Air and the `DynamicLoad` DLL/shared object. This mechanism requires that the `DynamicLoad` module be compiled with certain extra options. The `ri.h` header file in the `include` subdirectory of the Air distribution contains sample compile commands for Windows and Linux.

See Also:

Procedure shaders

8.8 Implicit Surfaces

AIR 6 introduces a new Implicit primitive for rendering grid data as an isosurface.

RIB syntax:

```
Implicit [minx maxx miny maxy minz maxz] [nx ny nz]
   "constant float __threshold" [1]
   "vertex float density" [nx*ny*nz values]
   parameter list
```

The only required parameter is a list of density values for the 3D grid. The density value at which the surface is created can be set by providing a constant `__threshold` parameter or by using an attribute:

```
Attribute "render" "float thresholdmultiplier" [1]
```

Additional user data can be included in the parameter list in the usual way. Vertex or varying variables should have nx*ny*nz values. Uniform or constant variables should have 1 value.

8.9 Trim Curves

Trim curves are supported, and they require Phong (per-pixel) shading.

"Cracking" between adjacent trimmed surfaces is a common problem when rendering NURB surfaces. AIR provides a method of patching cracks by allowing the border of a trimmed surface to be slightly expanded.

```
  Attribute "render" "trimborder" n
```

gives a distance in uv-space to expand the border of a trimmed NURB surface.

8.10 Instancing with Inline Archives

AIR supports an "inline archive" extension to the RI interface for efficient creation of multiple instances of the same RIB commands.

```
  ArchiveBegin "identifier"
  ArchiveEnd
  ArchiveInstance "identifier"
```

An inline archive block can be used to encapsulate a sequence of RIB commands in a single entity that can then be efficiently instanced multiple times. Primitives and shaders declared within an inline archive will share data across instances of the archive. RIB options are not currently supported in an inline archive.

See `$AIRHOME/examples/instances` for an example.

Freeing Instances

Once created an instance will persist as long as the renderer is running. If a given instance is no longer needed, it can be freed with

```
FreeArchive "identifier"
```

which allows AIR to free the memory used by the instance once all references have been processed.

9 Shader Guide

AIR uses small programs called *shaders* to perform shading and lighting calculations. This programmable shading capability is one of AIR's most powerful features: by using a suitable shader, almost any type of light or surface can be simulated.

Shader Types

AIR supports the following types of shaders:

surface - calculates the color and opacity at a point on a surface.

displacement - moves or "displaces" a surface to simulate topographical features such as bumps, wrinkles, or grooves.

volume - modifies the light reflected by a surface to account for atmospheric effects such as fog and smoke.

light - calculates the light emitted from a light source.

environment - shades rays that miss all objects in the scene

imager - modifies final pixel values after rendering is complete.

instancer - creates new geometry on-demand based on the underlying primitive

procedure - creates new geometry on-demand at render time

generic - shader type for components that may be used with different shader types

Using Shaders

If you are using a plugin for a modeling program, some shaders will be used automatically to simulate the lighting and materials present in the host program.

Most plugins also provide a way to attach custom surface, displacement, and atmosphere shaders to objects. AIR comes with a sizable collection of custom shaders, which are documented in the following sections of this user manual. The AIR shader library is located in the `shaders` directory of your AIR installation.

Because AIR supports the industry standard RenderMan shading language, you can also use any of the hundreds of freely available RenderMan shaders available on the web. New shaders first need to be compiled with the AIR shading compiler shaded before they can be used.

Each shader has a set of parameters that can be used to control a shader's appearance. E.g., a tile floor shader might have parameters for the tile size, tile color, and grout width. By altering the default values for a shader's parameters, you can create a variety of different looks from a single shader.

Creating Shaders

AIR shaders are written in the industry-standard RenderMan shading language.

AIR ships with a visual tool for creating shaders called Vshade. Even if you are not a programmer, you can easily create shaders with Vshade. See the separate Vshade documentation for tutorials and hints on building your own shaders.

Why do many AIR shader names begin with V?

A shader name prefixed with an upper case V indicates that the shader was created with the Vshade shader creator included with AIR. You can find the source code for these shaders in the `vshaders` subdirectory of your AIR installation. The source code can serve as a good starting point for building your own customized version of a shader.

See Also

Shading Language Extensions
Layered Shaders and Shader Networks
Common Shader Features
DarkTree Shaders

9.1 Common Shader Features

Topics:

Illumination Models
3D Patterns
2D Patterns
3D to 2D Projections
Coordinate Spaces

9.1.1 Illumination Models

An *illumination model* that describes what happens to light that strikes the surface: whether it is reflected, transmitted, absorbed or some combination of all three. Most AIR surface shaders incorporate one of the following illumination models:

Matte

A matte surface uniformly scatters incoming light with no preferred direction. The matte illumination model includes an Ambient parameter, which scales the contributes of ambient light sources, and a Diffuse parameter, which sets the intensity of the diffusely scattered light from non-ambient light sources.

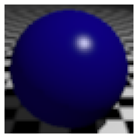
Plastic

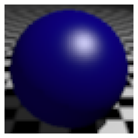
A plastic surface typically reflects some light diffusely - like a matte surface - and some light in a focused manner producing specular highlights. A plastic illumination model adds parameters for specular highlights to the base Ambient and Diffuse parameters of the Matte illumination model.

A Specular parameter sets the intensity of the specular highlight. To be physically plausbile, the sum of the diffuse and specular intensities should be less than or equal to 1:

```
Diffuse + Specular < 1
```

A Roughness or SpecularRoughness parameter determines the size of the highlight (which is inversely proportional to the roughness value):


0.05


0.2

Although most plastics have white highlights, a surface shader may provide a `SpecularColor` parameter for customizing the highlight color.

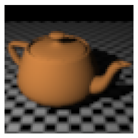
Metal

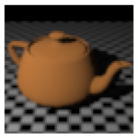
A polished metal surface typically reflects a high percentage of the incoming light coherently. A metal illumination module provides Ambient, Diffuse, and Specular parameters like the plastic model. The Diffuse strength should normally be much less than the Specular value. For metals the highlight color is usually the same as the diffuse color, unlike plastics which typically have white highlights regardless of the diffuse surface color.

Polished metals often exhibit coherent reflections. Most metal shaders do not apply Fresnel effects to metallic reflections (varying the reflection strength based on viewing angle) because the Fresnel phenomenon is much less noticable for metals. Reflections are typically tinted by the same color as the diffuse and specular components.

Clay

The clay illumination model is a more general type of matte illumination model. AIR's shaders use the model for rough surfaces developed by Michael Oren and Shree Nayar in their SIGGRAPH 94 paper. This model adds a diffuse roughness parameter to the basic matte illumination model. When the roughness value is 0, the clay model produces the same results as the matte illumination model. Larger values of roughness cause the surface to act more as retroreflector - reflecting light coming from the same direction as the viewer in a manner that does not depend on the orientation of the surface.


0.0

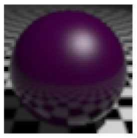

0.5

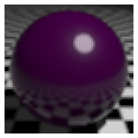
The general effect is to make objects look flatter compared to a simple matte illumination model

Ceramic

The ceramic illumination model is useful for glossy or wet surfaces. This model is similar to the plastic model with the addition of a Sharpness parameter that controls the edge of the specular highlight.

When sharpness is set to 0, the ceramic model produces the same results as the plastic model. When sharpness is 1, the highlight edge is maximally sharp.


0.0


0.8

9.1.2 3D Patterns

Solid textures define a pattern for points in 3D space. Solid textures are used to model materials such as wood, marble, and granite. An object shaded with a solid texture looks like it is "carved out" of the corresponding material.

In order to generate a 3D pattern, a shader must know the coordinates of the point being shaded. Solid texture shaders usually provide a `ShadingSpace` parameter for selecting the coordinate system in which shading calculations are performed. That coordinate system is referred to as the shading space for that shader. It will normally be `object`, `shader`, or `world` space. (Note that shader space is the coordinate system in which a shader is declared, which may be different from the shading space in which calculations are performed.)

Solid texture shaders also usually have a `PatternSize`, `FeatureSize`, or `ShadingFrequency` parameter that controls the overall size of the 3D pattern. `PatternSize` or `FeatureSize` give the size of one copy of the pattern in 3D space. `ShadingFrequency` sets the size indirectly by specifying how often the pattern repeats per unit interval.

9.1.3 2D Patterns

2D patterns include texture maps and procedurally generated patterns such as tiles, bricks, stripes, and dimples.

Each type of geometric primitive has a default mapping for the global s,t texture coordinates. Some shaders simply use these s and t global variables as the shading coordinates. Many are more flexible, allowing you to choose a projection for transforming a 3D point into 2D texture coordinates.

9.1.4 3D to 2D Projections

The shaders that are included with AIR recognize the following projection types:

`st`	Use the default s,t texture coordinates
`planar`	Use the x and y components of the point
`box`	Project the point onto a plane orthogonal to the x,y, or z axes based on the largest component of the surface normal. This projection is useful for walls, boxes, and other shapes with flat sides that are perpendicular to a coordinate axis.
`spherical`	Project the point onto a sphere centered at the origin with the poles on the z-axis. The "latitude" and "longitude" of the sphere are used as the texture coordinates.
`cylindrical`	Project the point onto a cylinder wrapped around the z axis. The first texture coordinate is the angle around the axis of the point (measured counter-clockwise from the x-z plane) scaled to the range 0-1. The second texture coordinate is the z component.

Projection Space

For projection types other than `st`, the coordinate system for the point used for projection may also often be chosen (just as the shading space for a solid texture is) using a `ProjectionSpace` or equivalent parameter.

Texture Coordinate Transformation

Shaders may also accept a transformation matrix to be applied to the point prior to projection. The transformation matrix can be used to scale, translate, rotate, or shear the texture pattern.

9.1.5 Coordinate Spaces

AIR supports hierarchical modeling, in which objects are positioned by applying a series of translation, rotation, scaling, and arbitrary 3D transformation operations. Each transformation defines a new coordinate system. AIR provides names for those systems or spaces that are commonly used for shading calculations:

`object`	Coordinate system in which a geometric primitive is defined.
`shader`	Coordinate system in which a shader is declared.
`world`	Coordinate system at WorldBegin, after the camera has been positioned and before any transformations for objects have been declared. (The base coordinate system for a scene.)
`camera`	Coordinate system of the camera. The camera is assumed to be at the origin, with the y axis pointing up, the z-axis pointing in, and the x-axis increasing to the right.
`current`	The space in which calculations are performed within shaders. For AIR current space is camera space, but current space may be different for other renderers.
`screen`	Coordinate space after perspective projection (with z coordinates scaled and offset to 0 at the near clipping plane and 1 at the far clipping plane.)
`NDC`	Normalized device coordinates. A 2D device-independent coordinate system defined on the screen, with x running from 0 to 1 left-to-right, and y increasing from 0 to 1 top-to-bottom.
`raster`	A 2D coordinate system where x and y are the pixel coordinates of a point after projection to the screen.

9.2 Displacement

Displacement shaders add small topographical features to a surface such as bumps, wrinkles, or dents. Your plugin should provide a means of assigning a displacement shader to an object. If not, you can add the following line before the object definition in the scene file:

```
Displacement "shadername"
```

The AIR distribution includes a number of displacement shaders, and you can write your own using AIR's Vshade shader creation tool.

Bump Mapping vs. True Displacement

AIR offers two methods of rendering displacements:

Bump mapping: Displacement is simulated by altering the surface normal. The surface itself does not move.

True displacement: The internal representation of the surface is displaced.

The two methods differ in cost and appearance. Bump mapping is faster and uses less memory, but silhouette edges will not exhibit displacement, and shadows will not reflect the displacement. True displacement is slower and potentially uses much more memory because the renderer must dice the surface down to sub-pixel-size polygons which are then displaced. However, truly displaced surfaces will have appropriate silhouettes and shadows.

True Displacement

By default displacement shaders only perform bump mapping. True displacement is turned on with:

```
Attribute "render" "integer truedisplacement" [1]
```

Truly displaced surfaces also need to have a displacement bound set, giving the maximum distance the surface will be displaced:

```
Attribute "displacementbound" "float sphere" [radius]
```

A plugin should provide controls for setting these attributes.

For displacement shaders included with AIR the above instructions are sufficient for true displacement. Some other displacement shaders also contain a "truedisplacement" parameter, which should be set to correspond with the `truedisplacement` attribute. For true displacement the shader parameter tells the shader to modify the global position variable P, and the attribute tells the renderer to move the surface point to the new position.

Displacement Methods

AIR has two methods of performing displacement. The default method takes the polygon mesh normally generated for rendering and displaces each facet by dividing it into one or more grids of micropolygons. The default method is normally fairly fast and distributes well over multiple processors. However, it has some drawbacks:

- If the initial tessellation of a curved surface such as a subdivision mesh is too coarse, the displacement may show artifacts of the initial tessellation.
- With some shaders small differences in the exact values used for displacement along a shared edge can lead to cracks.
- The default method can be slow if the initial polygon mesh is very dense, since each polygon is essentially processed a second time as a separate primitive.

To overcome these limitations AIR 5 & 6 introduced a new "mesh displacement" method that displaces an entire surface at once using a single dense mesh. Enable mesh displacement with:

```
Attribute "render" "integer meshdisplacement" [1]
```

Mesh displacement is supported for subdivision meshes, polygon meshes, and NURB surfaces. Mesh displacement can be slower than the default method and use more temporary storage. The final displaced mesh usually consumes less memory than the default displacement method.

For subdivision surfaces we recommend using the splitmesh attribute to improve the performance of mesh displacement.

Attributes and Options

The following attributes and options apply to true displacement:

```
Attribute "render" "integer normaldisplacement" [1]
```

This attribute tells the renderer if displacement takes place only in the direction of the surface normal - which is usually the case - and that information allows the renderer to employ an additional optimization.

```
Attribute "render" "integer triangledisplacement" [0]
```

When set to 1 this attribute forces the use of triangles instead of quads for displacement meshes, which may reduce artifacts at the cost of increased rendering times.

```
Option "limits" "integer displacement grid" [8]
```

Sets the maximum size of the grid of micropolygons for displacement.

```
Attribute "render" "integer smoothlevel" [1]
```

Enables additional smoothing of the normals computed for displaced surfaces when the argument value is greater than the default of 1. The range of valid levels is 1 to 6.

```
Attribute "render" "fastdisplacement" [1]
```

With the default value of 1, AIR uses a faster displacement method that does not retain the normal values computed in the shader. When set to 0, a slower displacement method is used that also allows bump mapping to be applied after the displacement.

Ray Tracing Displacements

Ray tracing truely displaced surfaces can use a lot of memory because AIR must retain all displaced geometry for the duration of the render. To save memory, a surface can be made to displace when it is rendered from the camera but not when it is ray traced. To disable displacement for a surface when it is ray traced, use:

```
Attribute "trace" "integer displacements" [0]
```

True Displacement Tips

True displacement can be expensive in terms of memory and rendering times. Here are some suggestions for making it somewhat less costly:

- If you do not need to ray-trace a displaced surface, make it invisible to shadow, reflection, and indirect rays. AIR can then discard the surface as soon as it has been rendered by the scanline renderer. This can have a large effect on memory usage.

- If you want a surface to appear in ray-traced reflections or participate in indirect lighting, disable true displacement when the object is traced with

  ```
  Attribute "trace" "integer displacements" [0]
  ```

- Use single-sided surfaces if possible.

- The dicing rate for true displacement is inversely proportional to the "flatness" attribute:

  ```
  GeometricApproximation "flatness" [0.5]
  ```

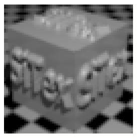
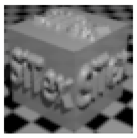
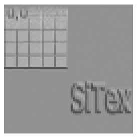
  Using values of 0.7 or 1.0 instead will reduce the micropolygon count by approximately a factor of 2 or 4 respectively.

- Polygons and subdivision meshes usually use less memory than NURB surfaces for true displacement.
- Use as simple a shader as possible. E.g., if you only need a simple bump map with standard texture coordinates, use the standard `bumpy` shader, which is faster than any custom shader.
- Compile the displacement shader to a DLL on Windows or a shared object on Linux.

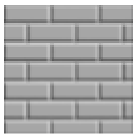
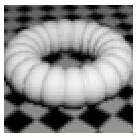
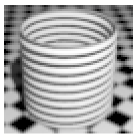
9.2.1 Displacement Shaders

Displacement shaders are used to add small topographical features to a surface such as bumps, wrinkles, or dents.

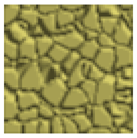
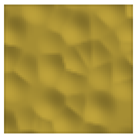
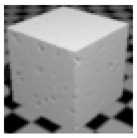
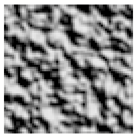
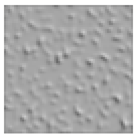
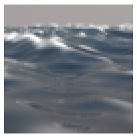
Texture-Mapped Displacement

	bumpy	Simple shader for displacement with a single texture map using standard texture coordinates
	VTexturedBump	Displacement with a single texture map, using standard texture coordinates or a 3D-to-2D projection, with control of bump profile
	VBumpRegions	Multiple regions with different bump maps

Regular Patterns

	VBrickGrooves	2D pattern of brick grooves
	VPillows	2D pattern of pillow-like bumps
	VRibbed	1D pattern of semi-circular bumps
	VThreads	2D pattern of screw threads

3D Irregular Patterns

	VDimples	pattern of rounded cells
	VCracks	cracks
	VDents	dents
	VHammered	hammered surface
	VNicks	nicked surface
	VRipples	ripple-like patterns
	VStuccoed	stucco
	OceanWaves	animated ocean waves

Normal Mapping

VNormalMap	apply normal map holding object-space or world-space normals
VTangentNormalMap	apply normals stored in tangent space

Additional Displacement Shaders

	DarkTreeDisplacement	displacement shader for DarkTree shaders

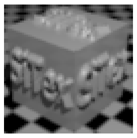
Displacement Output Variables

All Air displacement shaders provide the following output variables which may be queried by a surface or atmosphere shader:

`__bump`: the distance the surface was displaced at the current shading location

`__undisplaced_N`: the undisplaced normal vector at the current shading location

9.2.2 bumpy


simple displacement with a texture map

Parameter	Default value	Range	Description
Km	`1.0`		Displacement amplitude multiplier
texturename	`""`		Texture file name
repeatx	`1.0`		Copies of texture in X direction
repeaty	`1.0`		Copies of texture in Y direction
originx	`0.0`		Location of left texture edge
originy	`0.0`		Location of top texture edge

Description

The bumpy displacement shader uses a single texture map to displace a surface. The texture map is positioned using an object's standard texture coordinates.

See Also

VTexturedBump for displacement with a texture map with more controls

9.2.3 DarkTreeDisplacement

displace using a DarkTree shader

Parameter	Default value	Range	Description
DarkTreeShader	""		Name of Darktree shader
BumpMax	1.0		Bump multiplier
BumpMin	0		Bump at 0
Use2DCoordinates	0	0 or 1	Set to 1 for a 2D pattern using an object's standard texture coordinates
Texture2dSize	1 1		2D pattern width and height
Texture2dOrigin	0 0		2D pattern origin
TextureAngle	0		Rotation angle in degrees for 2D pattern
Pattern3dSize	1		Overall scale for 3D pattern
Pattern3dOrigin	0 0 0		3D pattern origin
Pattern3dSpace	"shader"	spaces	Coordinate system for 3D pattern
TimeScale	1.0		Multiplier for global time variable
FloatTweak1Name	""		Tweak name
FloatTweak1	0.5		Tweak value
FloatTweak2Name	""		Tweak name
FloatTweak2	0.5		Tweak value
FloatTweak3Name	""		Tweak name
FloatTweak3	0.5		Tweak value
FloatTweak4Name	""		Tweak name
FloatTweak4	0.5		Tweak value

Description

The DarkTreeDisplacement shader allows any DarkTree shader to be used as a displacement shader in AIR.

The DarkTreeDisplacement shader provides a number of tweak parameters that can be used to alter the tweaks defined for a particular shader. Some user interfaces, such as AIR Space, will automatically fill in the list of tweaks for a particular shader. For those that do not, the tweak names and values can be entered by hand.

2D or 3D Shading

The DarkTree shading pattern can be calculated in 2D or 3D. If the Use2DCoordinates parameter is set to 1, the base shading location will be (s,t,0), where s and t are the standard 2D texture coordinates for a primitive.

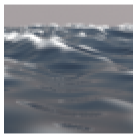
Animation

DarkTreeDisplacement uses the shading language time value (set with the `Shutter` RIB call) multiplied by the `TimeScale` parameter as the frame number input for DarkTree shaders. By varying the time value in the `Shutter` call, animated DarkTree shaders can be queried at different times.

See Also

DarkTree overview
DarkTreeSurface shader

9.2.4 OceanWaves


animated ocean waves

Parameter	Default value	Range	Description
BumpMax	1.0		Displacement amplitude multiplier
GridCountX	128		Grid size in X direction
GridCountY	128		Grid size in Y direction
SizeX	200		Nominal size of X dimension
SizeY	200		Nominal size of Y dimension
WindSpeed	5		Wind speed
WindAlign	8		Wave alignment with wind direction
WindDir	0		Wind direction as an angle in degrees
SmallWave	0.1		Small waves will be ignored
Dampen	0.75		Amount to dampen waves not aligned with the wind
Chop	1		Amount of chop to add to waves
FramesPerSecond	24		Value used to compute current time for animation
FoamEnabled	1	0 or 1	Whether foam needs to be simulated for the surface shader
RandomSeed	123		Base for random number generator

Description

The OceanWaves displacement shader simulates deep ocean waves using the method described in Jerry Tessendorf's 2004 paper "Simulating Ocean Water".

The waves are computed by evaluating a simulation on a discrete grid, which is mapped to the unit interval in texture space. X corresponds to the first texture coordinate, Y to the second coordinates. The GridCountX and GridCountY parameters give the grid size. A larger grid produces more detail but takes longer to compute.

SizeX and SizeY give the nominal size of the ocean area.

The waves can be tiled in X and Y.

WindSpeed gives the average wind speed. Higher values produce larger waves. WindDir can be used to change the wind direction by providing an angle in degrees about the vertical axis.

WindAlign controls how much the wind direction influences the wave directions by applying an exponent to the dot product of the wind direction and the direction of each wave. Larger values reduce the amplitude of waves not in the direction of the wind base on the Dampen factor.

Chop

The appearance of waves in windy conditions can be improved using the Chop parameter, which sharpens wave peaks and stretches troughs by shifting the surface laterally. Because the chop displacement shifts the surface in arbitrary directions, you'll need to tell Air that the displacement is not strictly in the direction of the surface normal by applying the following attribute:

```
Attribute "render" "normaldisplacement" [0]
```

Note that too much chop can tear the surface apart.

Foam

The waves simulation can compute a foam value which can be used by the companion OceanSurfaceWithFoam surface shader to add a foam effect to the waves. In order for the foam to appear in the proper location, the common parameters between the OceanWaves and OceanSurfaceWithFoam shaders must be set to the same values. If you do not need to render foam, set the FoamEnabled parameter to 0 to reduce computation time for the waves.

Animation

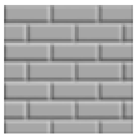
The waves will be animated based on the current frame number. The current time value is the current frame number divided by the FramesPerSecond value.

For speed the simulation is computed using the seawave dynamic shadeop (DSO) included with Air.

See Also

OceanSurfaceWithFoam surface shader

9.2.5 VBrickGrooves


brick grooves

Name	Default Value	Range	Description
GrooveDepth	`0.01`		Groove depth
BrickWidth	`0.25`		Brick width
BrickHeight	`0.08`		Brick height
MortarWidth	`0.01`		Mortar width
Stagger	`0.5`	`0.0-1.0`	Fraction to offset every other row
EdgeVary	`0.01`		Unevenness of brick edges
EdgeVaryFrequency	`4`		Frequency of brick edge unevenness
Projection	`"st"`	projections	Projection type
ProjectionSpace	`"world"`	spaces	Coordinate space for 3D to 2D projections
Transformation[16]	`1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1`		Transformation matrix
DisplacementSpace	`"shader"`	spaces	Coordinate space for displacement

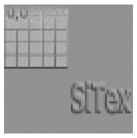
Description

The VBrickGrooves displacement shader provides a basic grooved brick pattern.

See Also

VBrick2D surface shader

9.2.6 VBumpRegions

 multiple regions with different bump maps

Parameter	Default value	Range	Description
BumpMax	`1`		Set this to the max of all displacements
DisplacementSpace	`"shader"`	spaces	Coordinate space for displacement

FirstBumpMap	""	Bump map file name
FirstBumpScale	1	Bump height multiplier
FirstBumpSizeXY	1 1	Width and height of bump map in texture coords
FirstBumpOriginXY	0 0	Position of top, left texture corner
FirstRegionSizeXY	1 1	Width and height of textured region
FirstRegionOriginXY	0 0	Top, left corner of textured region

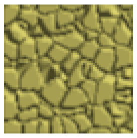
The shader includes 5 more texture regions with analogous parameters.

Description

The VBumpRegions displacement shaders allows up to six different regions on a surface to each receive a different bump-mapped displacement. Where regions overlap, the displace offsets are added together.

The BumpMax parameter does not affect the result of the shader, but some plugins may use it to automatically set the displacement bounds for true displacement. This parameter should be set to the maximum of all the BumpScale parameters (assuming no texture regions overlap).

9.2.7 VCracks

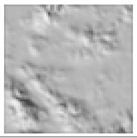

cracked surface

Parameter	Default value	Range	Description
BumpMax	0.1		Maximum depth of cracks
CrackWidth	0.2		Crack width
Jitter	1.0	0.0-1.0	Deviation from a regular grid
PositionVary	1.0		Variation in initial shading position
PatternSize	0.2		
PatternOrigin	0 0 0		
PatternSpace	"shader"	spaces	Coordinate space for shading calculations

Description

The VCracks displacement shader uses a cellular pattern to generate cracks in a surface.

9.2.8 VDents


dents

Parameter	Default value	Range	Description
BumpMax	0.02		Maximum depth of dents
Power	3	1-8	Exponent for sharpening bumps
LevelsOfDetail	6	1-8	Levels of detail in dents
PatternSize	0.3		Overall scale of pattern
PatternSpace	"shader"	spaces	Coordinate space for shading calculations

Description

The VDents displacement shader generates a pattern of dents using a turbulence function.

9.2.9 VDimples

pattern of rounded cells

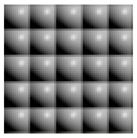
Parameter	Default value	Range	Description
BumpMax	0.1		Maximum height of cells
Jitter	1.0	0.0-1.0	Deviation from a regular grid pattern
Power	2	1-5	Power for cell shape
PatternSize	0.2		Pattern size
PatternOrigin	0 0 0		Pattern origin
PatternSpace	"shader"	spaces	Coordinate space for shading calculations

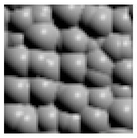
Description

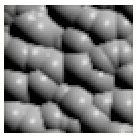
The VCellBumps displacement shader produces a 3D pattern of similarly sized but irregular rounded cells.

The overall size of the cells is set by the `CellWidth` parameter, with the `CellHeight` parameter giving the maximum displacement distance.

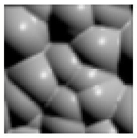
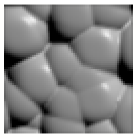
The `Jitter` parameter controls deviation from a regular grid:


0.0

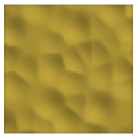

0.5


1.0

The `Power` parameter influences the profile and shape of the cells:

		
1	2	4

9.2.10 VHammered

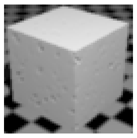

hammer dents

Parameter	Default value	Range	Description
BumpMax	0.02		Maximum depth of hammered dents
Jitter	1.0	0.01-1.0	Deviation from a regular grid pattern
PatternSize	0.2		Overall scale of pattern
PatternOrigin	0 0 0		Pattern offset
PatternSpace	"shader"	spaces	Coordinate space for shading calculations

Description

The VHammered displacement shader produces the effect of hammered metal. The `DentSize` parameter sets the overall scale of the pattern.

9.2.11 VNicks


nicks

Parameter	Default value	Range	Description
BumpMax	0.2		Maximum depth of nicks
Coverage	0.35	0.01-1.0	Fraction of surface covered with nicks
FilterWidth	0.25		Filter width multiplier
PatternSize	0.1		Overall scale of pattern
PatternOrigin	0 0 0		Pattern offset
PatternSpace	"shader"	spaces	Coordinate space for shading calculations

Description

The VNicks shader produces small nicks in a surface. The Coverage parameter gives the approximate fraction of the surface covered with nicks.

9.2.12 VNormalMap

Apply normal map with normals stored in object space or world space

TextureName	""		Texture name
TextureBlur	0.0	0.0-1.0	Texture blur
TextureSizeXY	1 1		Texture size in X and Y direction
TextureOriginXY	0 0		Location of left, top texture edge
TextureAngle	0		Texture rotation angle in degrees

See Also

VTangentNormalMap for normal maps stored in tangent space

9.2.13 VPillows

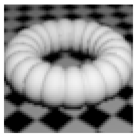
pillow pattern

Parameter	Default value	Range	Description
BumpMax	0.05		Maximum height of pillows
TextureSizeXY	0.1 0.1		Size of pillow in X and Y
TextureOriginXY	0 0		Pattern offset in X and Y
TextureAngle	0		Rotation in degrees
DisplacementSpace	"shader"	spaces	Coordinate space for displacement

Description

The VPillows shader produces a pillowed surface. The pattern is applied using standard texture coordinates.

9.2.14 VRibbed

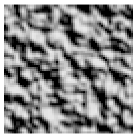

adds semi-circular bumps to a surface

Parameter	Default value	Range	Description
Height	0.1		Maximum height of ribs
Frequency	10.0		Number of ribs per texture unit
RotationAngle	0.0	0-180	Angle (in degrees) for rotating pattern
DisplacementSpace	"shader"	spaces	Coordinate space for displacement

Description

The VRibbed displacement shader adds ribs to a surface by displacing it using sem-circles. The pattern is applied using the object's standard texture coordinates. By default the pattern runs along the first texture coordinate. The `RotationAngle` parameter can be used to rotate the pattern on the surface.

9.2.15 VRipples


ripple-like bumps

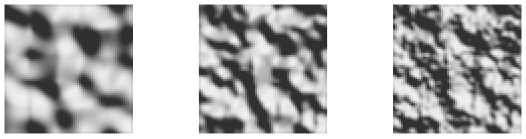
Parameter	Default value	Range	Description
BumpMax	0.1		Maximum ripple height
Roughness	0.5	0.01-1.0	Gain
LevelsOfDetail	2	1-7	Number of octaves of noise
FilterWidth	0.5		Filter width multiplier
PatternSize	0.2		Overall scale of pattern
PatternOrigin	0 0 0		Pattern offset
PatternSpace	"shader"	spaces	Coordinate space for shading calculations

Description

The VRipples displacement shader produces ripple-like bumps using several levels of noise patterns.

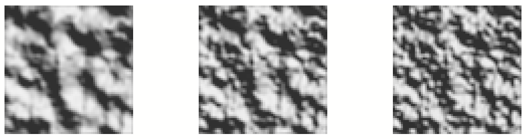
The ripples are computed as a 3D pattern with `PatternSize` determining the overall size of the ripples.

`LevelsOfDetail` sets the number of octaves or levels of noise that are combined to produce the ripples. The added detail provided by additional levels can be seen in the following examples:


1 2 3

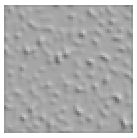
The `Roughness` parameter controls how much each level of noise contibutes to the final result. Lower roughness values produce a smoother final appearance. Higher numbers produce a "noisier" result.

Some examples with `LevelsOfDetail`=3:



This shader performs anti-aliasing by fading out the ripples as the region being shaded grows large. Lower `FilterWidth` values will allow more detail to appear (up to the limit set by `LevelsOfDetail`), but the additional detail may cause aliasing in animation.

9.2.16 VStuccoed

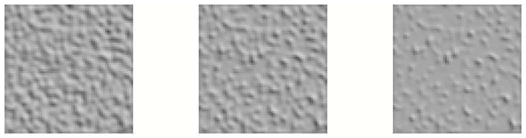

stucco

Parameter	Default value	Range	Description
BumpMax	0.04		Maximum height of bumps
Power	5	1-8	Exponent for sharpening bumps
PatternSize	0.05		Overall scale of pattern
PatternOrigin	0 0 0		Pattern offset
PatternSpace	"shader"	spaces	Coordinate space for shading calculations

Description

The VStuccoed displacement shader produces a stucco-like pattern using a single noise function raised to a power.

The `Power` parameter sharpens the pattern:



9.2.17 VTangentNormalMap

Apply normal map in tangent space

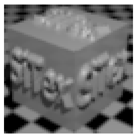
TextureName	""		Texture name
TextureBlur	0.0	0.0-1.0	Texture blur

TextureSizeXY	1 1		Texture size in X and Y direction
TextureOriginXY	0 0		Location of left, top texture edge
TextureAngle	0		Texture rotation angle in degrees

See Also

VNormalMap displacement shader for normals in object or world space

9.2.18 VTexturedBump


displacement using a texture map

Parameter	Default value	Range	Description
BumpMax	0.1		Maximum displacement amount
BumpMin	0.0		Minimum displacement amount
BumpSharpness	0.5	0.0-1.0	Angle (in degrees) for rotating pattern
BumpBias	0.5	0.0-1.0	Coordinate space for displacement
TextureName	""		Texture name
TextureBlur	0.0	0.0-1.0	Texture blur
TextureSizeXY	1 1		Texture size in X and Y direction
TextureOriginXY	0 0		Location of left, top texture edge
TextureAngle	0		Texture rotation angle in degrees
Projection	"st"	type	Projection to use for 2D coordinates
ProjectionSpace	"shader"	spaces	Coordinate space for projection
ProjectionTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection

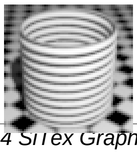
Description

The VTexturedBump displacement shader uses a texture map to displace a surface. The texture map may be placed using standard texture coordinates or any of the standard 3D-to2D projections.

See Also

bumpy displacement shader for very simple bump mapping
VBumpRegions for multiple bump maps per surface

9.2.19 VThreads


displacement shader for generating screw threads

Parameter	Default value	Range	Description
BumpMax	`0.05`		Maximum thread height or depth
DisplacementSpace	`"shader"`	spaces	Coordinate space for displacement
Frequency	`10`		Threads per unit interval
Start	`0`		Start position of threads along the vertical axis
End	`1`		End position along the vertical axis
FadeZone	`0.05`		Fade region near start and end
SwapXY	`0`		If 1, swap X and Y values

Description

The VThreads displacement shader generates a pattern of screw threads (typically along a cylinder).

9.3 Environments

Air 11 introduces a new environment shader type for shading rays that miss all objects in the scene.

There is one environment shader or shader network for the entire scene. The current environment shader is set with a new RIB Environment command:

```
Environment "shadername" parameter list
```

As an option, the `Environment` statement must appear before `WorldBegin` in the RIB stream. Additional environment shaders can be added to the current environment shader list by prepending the shader name with a "+".

Environment Shader Evaluation

The current environment shader is called automatically in the following cases:

- When a reflection or indirect ray exits the scene.
- For non-opaque pixels in the rendered image, with the ray type defined as a camera ray. If the scene also has an imager shader, the imager shader is executed after the environment shader at each pixel.

Shaders can also explicitly sample the current environment using the `environment()` call with the special map name `"environment"`. The optional "samples" parameter to the `environment()` function can be used to super-sample the environment shaders over a cone of directions (defined by the "blur" parameter). This brute-force super-sampling can be time-consuming if the environment shader is computationally intensive.

Environment shaders have access to the ray direction via the global variable `I` and the ray origin in the global variable `P`. An environment shader should set the output color `Ci` and output opacity `Oi` just like a surface shader.

The behavior of an environment shader can be customized for different ray types by querying the current ray type with the `rayinfo()` shading language function.

Environment Cache

Air 12 introduces a new option to automatically build an in-memory cache for environment queries from indirect rays:

```
 Option "indirect" "environmentcache" [1]
```

Using the cache produces much smoother results and is usually at least twice as fast for environment queries than rendering without. The only drawback is that the cache is based solely on the direction of the indirect rays. If an environment shader varies its computations based on position, that behavior cannot be accurately captured by the cache, and the cache should not be used in that case.

Past and Future

In older versions of Air the color returned by a reflection or indirect ray that exited the scene was determined by a set of custom options that provided a limited range of options (basically a fixed color or an environment map query). The new environment shader type provides more flexibility in defining the color returned by exiting rays as well as a unified interface for all ray types.

If no environment shader is defined, Air 11 reverts to the old custom options for indirect and reflection rays. In the future it is likely that these old options will be deprecated entirely.

9.3.1 envColor

simple environment shader returning a constant color

Parameter	Default Value	Range	Description
Color	1 1 1		base color
Intensity	1.0		intensity multiplier for all rays
ReflectionVisibility	1	0 or 1	whether the environment is visible to reflection rays
ReflectionIntensity	1.0		intensity multiplier for reflection rays
ReflectionTint	1 1 1		color multiplier for reflection rays
IndirectVisibility	1	0 or 1	whether the environment is visible to indirect rays
IndirectIntensity	1.0		intensity multiplier for indirect rays
IndirectTint	1 1 1		color multiplier for indirect rays
CameraVisibility	1	0 or 1	whether the environment is visible to camera rays
CameraIntensity	1.0		intensity multiplier for camera rays
CameraTint	1 1 1		color multiplier for camera rays

Description

The envColor environment shader returns a constant color value for each ray type. The returned color value can be the same for all ray types, or adjusted individually for each type of ray using the *Intensity and *Tint parameters.

Output only parameters

`color __background`	environment shader color result

9.3.2 envMap

environment shader with a common environment map for all rays

Parameter	Default Value	Range	Description
MapName	`""`		file name of environment map
MapIntensity	`1.0`		intensity multiplier for all rays
MapBlur	`0.01`		blur for map look up
MapSpace	`"world"`	spaces	
MapSamples	`4`		samples for environment map query
RotateX	`90`		rotation angle in degrees about the X axis
RotateY	`0`		rotation angle in degrees about the Y axis
RotateZ	`0`		rotation angle in degrees about the Z axis
ReflectionVisibility	`1`	0 or 1	whether the environment is visible to reflection rays
ReflectionIntensity	`1.0`		intensity multiplier for reflection rays
ReflectionAddBlur	`0.0`		increment to map blur for reflection rays
IndirectVisibility	`1`	0 or 1	whether the environment is visible to indirect rays
IndirectIntensity	`1.0`		intensity multiplier for indirect rays
IndirectAddBlur	`0.0`		increment to map blur for indirect rays
CameraVisibility	`1`	0 or 1	whether the environment is visible to camera rays
CameraIntensity	`1.0`		intensity multiplier for camera rays
CameraAddBlur	`0.0`		increment to map blur for camera rays

Description

The envMap environment shader uses an environment map to compute the color for a query ray.

Use the RotateX, RotateY, and RotateZ parameters to rotate the environment map with respect to the lookup coordinate space (MapSpace).

Output only parameters

`color __background`	environment shader color result

9.3.3 envPhysicalSky

environment shader with sky color based on position, time, and atmosphere

SkyIntensity	1.0		overall multiplier for output color
Turbidity	5	2-9	fraction of atmospheric scattering due to haze instead of molecules. Larger values model atmospheres with more large (dust) particles
DateMonth	4	1-12	month for sun position
DateDay	15	1-31	day of month for sun position
DateHour	12	0-24	24-hour value for sun position (local time)
LatitudeLongitude	47.45 -122.3		location of observer
TimeZone	-8		difference between local time and GMT
AnimateHour	0	0-24	if non-zero, the end hour for sun animation
AnimateTimeScale	1		multiplier for the global time value
OverrideZenith	-1		override for sun zenith angle (in degrees)
OverrideAzimuth	0		override for sun azimuth angle (in degrees)
CoordSysYisUp	1	0 or 1	if 1, the Y axis is used as the "up" direction; otherwise Z is up
CoordSysNorthAngle	0		north as an angle in degrees relative to the x-axis
CoordSysRightHanded	1	0 or 1	is the world space coordinate system right-handed?
ReflectionVisibility	1	0 or 1	whether the sky is visible in reflections
ReflectionIntensity	1.0		intensity multiplier for reflection rays
IndirectVisibility	1	0 or 1	whether the sky contributes to indirect rays
IndirectIntensity	1.0		intensity multiplier

Description

The envPhysicalSky shader computes a physically plausible sky function based on observer position, time, and atmospheric conditions.

The sun position can optionally be expicitly set using the OverrideZenith and OverrideAzimuth parameters.

Reference:

"A Practical Analytic Model for Daylight" by A. J. Preetham, Peter Shirley, and Brian Smits.

Output only parameters

`color __background`	environment shader color result

See Also

sunlight light shader

9.4 Generics

The generic shader type is compatible with all other shader types. Use generic shaders to build components for use with different shader types in shader networks.

genColor	shader for selecting a color
genOcclusion	shader for adding an occlusion output value to a rendering
genQueryBakedMap	shader for querying a baked map
genTextureUV	shader for querying a texture map using u,v coordinates
genUserAttributes	shader for querying user attributes
genUV	simple shader emitting the global u, v coordinates

9.4.1 genColor

generic shader for specifying a color value

Parameter	Default	Range	Description
Color	`0.5 0.5 0.5`		rgb color value
Color_spd	`""`		optional file name for spectral color profile
ColorName	`""`		optional named color in dictionary
DictionaryName	`"$AIRHOME/viztools/colors.txt"`		path for color dictionary

Description

The genColor shader allows a color to be specified using rgb values, a spectral profile, or a named color in a color dictionary.

Output only parameters

`color __OutColor`	output color

9.4.2 genOcclusion

generic shader for adding an occlusion output value to a render pass

Description

The genOcclusion shader performs a simple occlusion query using the default occlusion and indirect settings, providing output variables with the occlusion value and bent normal vector. Use this shader to add an occlusion output value to any rendering by appending the shader to the list of surface shaders. E.g.,

```
Surface "VPlastic" ...
Surface "+genOcclusion"
PointsPolygons ...
```

Output only parameters

`color __occlusion`	occlusion value
`vector __bent_normal`	average unoccluded direction (in world space)

See Also

Lighting -> Ambient Occlusion

9.4.3 genQueryBakedMap

generic shader for querying a baked map

Parameter	Default	Range	Description
AttributeName	`"user:bakedmap"`		user attribute to query for baked map name
DefaultColor	`0.5 0.5 0.5`		default color to emit if no baked map
Channel	`0`		first channel to query

Description

The genQueryBakedMap generic shader can be used to query a map baked using Air Stream.

Output only parameters

`color OutputColor`	output color
`float OutputRed`	red component
`float OutputGreen`	green component
`float OutputBlue`	blue component
`float OutputAlpha`	alpha component

9.4.4 genTextureUV

generic shader that queries a texture map using standard u,v coordinates

Parameter	Default	Range	Description
TextureName	`""`		texture map
DefaultColor	`0 0 0`		default color value
DefaultAlpha	`0`		default alpha value

Description

The genUV shader exports the standard u, v coordinates assigned to an object.

Output only parameters

`color Color`	emitted color
`float Alpha`	emitted alpha
`float Grey`	grey value (average of r,g,b)

9.4.5 genUserAttributes

generic shader for querying float and color user attributes

Parameter	Default	Range	Description
Float1Name	""		User attribute to query
Float1Default	0		Default value
Float2Name	""		User attribute to query
Float2Default	0		Default value
Float3Name	""		User attribute to query
Float3Default	0		Default value
Color1Name	""		User attribute to query
Color1Default	0		Default value
Color2Name	""		User attribute to query
Color2Default	0		Default value
Color3Name	""		User attribute to query
Color3Default	0		Default value

Output only parameters

`color OutputColor1`	output color 1
`color OutputColor2`	output color 2
`color OutputColor3`	output color 3
`float OutputFloat1`	output float 1
`float OutputFloat2`	output float 2
`float OutputFloat3`	output float 3

9.4.6 genUV

generic shader that exports the standard u,v coordinates of a surface

Parameter	Default	Range	Description
RepeatU	1		multiplier for U value
RepeatV	1		multiplier for V value

Description

The genUV shader exports the standard u, v coordinates assigned to an object.

Output only parameters

`float[2] OutUV`	u,v coordinates as an array of two floats
`float OutUVCoverage`	fraction covered (always 1)

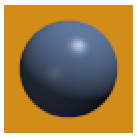
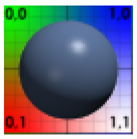
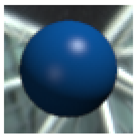
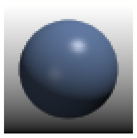
9.5 Imagers

Imager shaders modify the the final rendered image prior to output. An imager shader is specified in a RIB file with the `Imager` statement:

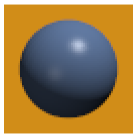
```
  Imager "background" "color background" [1 1 1]
```

The current imager shader is an option (i.e., there is one imager shader or shader network for the entire scene). The `Imager` statement must appear before `WorldBegin` in the RIB stream.

In Air 11 and later, multiple imagers can be specified by pre-pending the shader name of the second and later imagers with a "+". Multiple imager shaders will be executed in the order in which they were declared.

	background	constant background color
	LensFlareImager	adds lens flares to the output image
	sRGB	applies sRGB gamma correction
	ToneMap	tone-mapping for HDR images
	VBackDrop	texture-mapped background
	VBgEnvironment	environment-mapped background
	VBgGradient	2-color vertical background gradient
	VGreyscale	convert output color to grey level

9.5.1 background


sets the image background color

Parameter	Default Value	Range	Description
background	`0 0 0`		background color for image

Description

The background imager shader simply sets the color of all unrendered pixels to the color of the `background` parameter. The output alpha value is also set to 1.

Unlike all other shaders, the background shader is internal to AIR and cannot be modified.

9.5.2 LensFlareImager

imager shader that adds lens flare effects

Parameter	Default Value	Range	Description
Intensity	`1`		overall intensity scale for flare effects
BloomIntensity	`0.4`		Intensity multiplier for bloom effect
BloomRadius	`0.1`		Nominal radius of bloom spot
BloomPoints	`13`		Number of points in bloom star
BloomFalloff	`2`		Intensity falloff with distance
BloomTint	`1 1 1`		Bloom color
StarIntensity	`0.4`		Intensity multiplier for star effect
StarRadius	`0.3`		Nominal star radius
StarPoints	`50`		Number of points in star
StarFalloff	`3`		Intensity falloff with distance
StarTint	`1 1 1`		Star color
RingIntensity	`0.1`		Intensity multiplier for rings
RingRadius	`0.05`		Nominal ring radius
RingWidth	`0.07`		Ring width
RingTint	`1 .5 .5`		Ring color
FilterWidth	`2`		Filter width multiplier for antialiasing
MinLightIntensity	`0.01`		Minimum light intensity
Apply_sRGB	`0`	0 or 1	if non-zero, apply sRGB gamma correction
LightCategories	`"lensflare"`		Categories identifying lights that contribute to lens flare
RandomSeed	`137`		Seed number for random effects

Description

The LensFlareImager shader adds lens flare effects to an image. It is inspired by the lensflare shader in the "Advanced RenderMan" book. This version is an imager shader (not a surface shader) and differs substantially from the original in many other ways.

The flare effect has three independent components:

Bloom - a glow effect representing the visible light source
Star - a star shape with thin points
Ring - a single ring around the light source

Each effect is exported in a corresponding output variable (`__bloom`, `__star`, and `__ring`) for use in post-processing.

The flare is generating by looping over the lights in the scene. Only lights in the categories listed in the LightCategories parameter will be included. By default `LightCategories` is set to `"lensflare"`. Use light categories to exclude particular lights from the lens flare effect (including those with no physical location like the indirect light shader).

This effect works most reliably with point lights, since the `illuminance()` loop only returns lights that illuminate the current shading location. For spotlights, the camera location must be within the cone of illumination to generate a lens flare. For a spotlight or area light, you may wish to add an extra point light (that does not illuminate objects in the scene) to generate the lens flare for the the light.

Output only variables

`color __bloom`	Bloom color
`color __star`	Star color
`color __ring`	Ring color

9.5.3 sRGB

applies sRGB gamma correction to the output image

Parameter	Default Value	Range	Description
ClampToAlpha	1	0 or 1	if 1, output r g b components are clamped to the alpha value

Description

The sRGB imager shader applies sRGB gamma correction to the output r,g,b channels. When ClampToAlpha is 1, each component is clamped to the alpha value.

9.5.4 ToneMap

tone-mapping for high-dynamic range images

Parameter	Default Value	Range	Description
MaxLuminance	11.5		luminance corresponding to display value of 1
MiddleLuminance	0.18		luminance mapped to display middle grey
MIddleGrey	0.18		display middle grey value
PremultipliedAlpha	1	0 or 1	when not 0, input color is treated as pre-multiplied alpha
Apply_sRGB	0	0 or 1	whether to apply sRGB gamma correction to the final color
ClampToAlpha	1	0 or 1	whether to clamp the final r,g,b values to alpha
ModifyOutputColor	1	0 or 1	when zero, the imager output color is not modified (the tone map value is available in the __tonemap output variable)

Description

The ToneMap imager provides a simple tone-mapping operator for converting high-dynamic range (HDR) images to low-dynamic range for display.

The conversion is based on the global operator described in "Photographic Tone Reproduction" by Erik Reinhard et al.

The tone mapping operator is inspired by the zone system in photography. First, the luminance is scaled linearly to map the MiddleLuminance value to the MiddleGrey output value. The dynamic range is then compressed with

```
  new = lum * (1 + lum/MaxLuminance^2) /( 1+lum)
```

MaxLuminance gives the luminance value corresponding to a display value of 1. By lowering the MaxLuminance value you can allow sections of an image to appear overexposed.

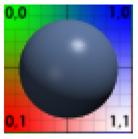
For speed this shader does not perform the rgb -> xyY -> rgb color space conversions described in the paper. Instead the r,g,b values are linearly scaled by the ratio of the output luminance from the tone mapping operator and original luminance. In some cases this shortcut can produce output values that are still greater than 1. If ClampToAlpha or Apply_sRGB is enabled, the final channel values will be less than or equal to 1.

The same tone mapping operator is available as a display option in Air Show.

Output only variables

`color __tonemap`	Tone-mapped color

9.5.5 VBackDrop



applies a texture map to the background

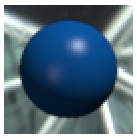
Parameter	Default Value	Range	Description
Intensity	1		Texture result multiplier
TextureName	""		Texture map for background
TextureBlur	0.0		Blur for texture lookup
RepeatX	1		Horizontal copies
RepeatY	1		Vertical copies
IsSequence	0	0 or 1	Is the map a sequence of frames?
FrameOffset	0		Offset added to current frame number to obtain sequence number for texture map

Description

The VBackDrop imager shader allows a texture map to be used as a background for the rendered image.

When IsSequence is set to 1, the texture map name is assumed to be one frame in a numbered sequence of images. The image for the current frame is chosen replacing the last number in the image name by the sum of the current frame number and the FrameOffset.

9.5.6 VBgEnvironment

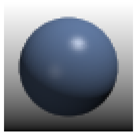

imager shader using an environment map for the background

Parameter	Default Value	Range	Description
Intensity	1.0		Map result intensity multiplier
EnvironmentName	""		Environment map for background
EnvironmentSpace	"world"	spaces	Coordinate space for environment map lookup
EnvironmentBlur	0.0	0.0-1.0	Environment map blur

Description

Use this imager shader to apply an environment map as the image background.

9.5.7 VBgGradient


2-color vertical gradient background

Parameter	Default Value	Range	Description
TopColor	`1 1 1`		Color at top of image
BottomColor	`0 0 0`		Color at bottom of image
Sharpness	`0.0`	`0.0-1.0`	Sharpening factor for transition
MidPoint	`0.5`	`0.0-1.0`	Location of half-way point in transition from top color to bottom color

Description

The VBgGradient imager shader provides a simple 2-color vertical background gradient.

9.5.8 VGreyscale

converts output color to grey scale

Parameter	Default Value	Range	Description
UseLuminance	`1`		whether to use luminance or HSV value

Description

The VGreyscale imager shader converts the output rgb color to grey scale. If UseLuminance is non-zero, the grey level is based on the color's luminance; otherwise the grey level is taken from the HSV value component.

9.6 Instancers

AIR 8 introduces a new instancer shader type for creating new geometry at render time based on properties of an existing primitive. As the name instancer implies, one motivation for the new shader type is the common need to convert a set of particles into a set of instances of some other geometry type. Instancer shaders make particle instancing easy (see the VInstanceArchive shader for an example). Because an instancer can be applied to any primitive type, an instancer can also be used to create primitives along a curve or "grow" primitives out of a surface.

An instancer shader is attached to a primitive just like a surface or displacement shader using the new `Instancer` RIB call:

```
  Instancer "VInstanceArchive" "string Archive" "teapot.rib"
```

As with true displacement, objects with an instancer shader must also give the renderer a bound on the region within which new primitives will be created. AIR currently uses the displacement bound attribute as the bound for an instancer as well:

```
  Attribute "displacementbound" "float sphere" [n]
```

The bound gives a distance used to expand the bounding box for the base primitive to encompass the region occupied by any primitives created by the instancer.

Writing Instancers

AIR 8 introduces two new shading language features to enable the creation of instancer shaders:

The `meshinfo()` function allows an instancer shader to query properties of the base primitive.
The `ribprintf()` statement allows an instancer to send a stream of RIB commands to the renderer to create new primitives.

The Vshade source code to all instancer shaders included with AIR can be found in `$AIRHOME/vshaders/Instancers`

9.6.1 instCarpet

instancer shader for growing carpet fibers on a rectangular patch

Parameter	Default	Range	Description
StrandCount	30		Strand count per unit length
StrandLength	0.2		Strand length
StrandWidthFraction	0.04		Strand width as a fraction of strand length
StrandTaper	1		Multiplier for the width of the curve at the curve tip. Set to 0 to taper the curve to a point.
StrandBaseVary	0.8		Variation in the base position of each strand
PolarAngle	40		Base rotation about the vertical axis
PolarVary	180		Variation in polar angle
PolarFrequency	20		Frequency of polar variation relative to the carpet rectangle
TiltMaxAngle	130		Maximum amount to bend or tilt the fibers, as an angle between the fiber tip and the vertical axis.
TiltPower	1		Exponent influencing the way fibers bend
Stiffness	0.4		Measure of fiber material stiffness
StiffnessVary	0.2		Variation in stiffness over the carpet
StiffnessFrequency	133		How often the stiffness varies relative to the carpet as a whole
StiffnessAtEdge	0.1		Stiffness near the edges of the carpet
EdgeWidth	3		Width of an edge region in strands
EdgeVary	0.5		Variation randomly added to the edge distance for each strand
Color	1 .96 .9		Base fiber color
ColorVary	0		Color variation per strand
ColorMapName	""		Optional color map
MaskName	""		Optional mask used to cut a shape out of the rectangle
MaskThreshold	0		Threshold for mask visibility
IndirectStrength	0.2		Diffuse reflectivity of strands for indirect rays
IndirectVisibility	0	0 or 1	Whether the strands are visible to indirect rays
IndirectSamples	0		If greater than 0, the number of rays to trace to compute indirect illumination
CurveType	"ribbon"	polyline ribbon tube	Geometric representation of the strand curves
ExportTextureCoordinates	1	0 or 1	Whether to export a texture coordinate pair for each strand
Seed	913		Seed value for random variation

The instCarpet instancer shader creates a carpet of fibers on a rectangular base polygon.

Carpet Shape

The carpet is modeled as a grid of carpet fibers. The geometric model of the fibers can be tuned using the following parameters:

StrandCount, StrandHeight, StrandWidthFraction, StrandTaper, StrandBaseVary

The overall density of the fibers is set with StrandCount, which gives the number of fibers per unit length. The diameter of a fiber is given by the StrandWidthFraction parameter as a fraction of the fiber length. The displacement bound for the base primitive should be set to the same value as StrandLength.

By default the carpet curves have the same width from root to tip. The fibers can be tapered by lowering the StrandTaper parameter, which gives the width at the curve tip relative to the width at the curve root.

StrandBaseVary varies the base position of each strand.

TiltMaxAngle, TiltPower, PolarAngle, PolarVary, PolarFrequency

TiltMaxAngle gives the maximum angle (in degrees) between the vertical direction and the tip of a fiber. TiltPower is an exponent applied to the tilt interpolation along the strand length. The direction of tilt can be changed using the StrandPolar parameters, which control a rotation in degrees about the vertical axis.

Stiffness, StiffnessVary, StiffnessFrequency, StiffnessAtEdge

Stiffness controls how each fiber bends or tilts away from the vertical. StiffnessVary and StiffnessFrequency control the variation in stiffness across the carpet. StiffnessAtEdge gives the stiffness value near the edges of the carpet.

EdgeWidth, EdgeVary

EdgeWidth defines a region near each edge as a strand count. If this value is greater than 0, the shader automatically bends the strands close to an edge so they tilt perpendicular to the edge and away from the carpet. EdgeVary varies the calculated distance-edge value to prevent artifacts from a regular pattern.

Color, ColorVary, ColorMapName

Color gives the base color for the carpet which can be randomly varied for each strand using ColorVary. If a ColorMapName is supplied, the color of each strand is taken from the color map, indexed using the texture coordinates of the base polygon. If the color varies per strand, the instancer attaches a separate color value to each strand.

MaskName, MaskThreshold

MaskName is an optional mask texture map which can be used to "cut" irregular shapes out of the base polygon. MaskThreshold gives a minimum value used to determine where the mask is opaque.

Carpet Shading

This instancer shader is designed to work with the companion VShadeCarpet surface shader, which should be assigned to the base polygon.

IndirectStrength, IndirectVisibility, IndirectSamples

By default the carpet fibers are not visible to indirect rays; set IndirectVisibility to 1 to enable visibility. The instancer shader automatically sets the indirect shading mode to constant for the carpet fibers, so if they are struck by an indirect ray, they simply return the fiber color multiplied by IndirectStrength. If IndirectSamples is greater than 0, it gives the number of rays to trace for indirect lighting. The instancer automatically sets the indirect:maxerror value to 0 to disable use of an indirect or occlusion cache.

ExportTextureCoordinates

When ExportTextureCoordinates is set to 1, each fiber inherits the texture coordinates of its root position on the base polygon.

See Also

VShadeCarpet surface shader
FakeCarpet surface shader

9.6.2 instMultipleArchives

particle instancer for multiple archives

Parameter	Default Value	Range	Description
ArchiveName	""		File name for one of the numbered archive files
ArchiveIndex	0		Index for archive file name
ArchiveIndexPrimVar	""		If specified, a primitive variable holding an index value for each vertex
Scale	1		Uniform scale applied to archive instances
ScaleByWidth	1	0 or 1	When non-zero, scale archive by point width
ApplyMotionBlur	0	0 or 1	When non-zero, apply translation motion blur
ApplyColor	0	0 or 1	When non-zero, apply vertex color to archive
XAngle	0		Rotation about X axis in degrees
XAnglePrimvar	""		Primitive variable with rotation angle for X axis
YAngle	0		Rotation about Y axis in degrees
YAnglePrimVar	""		Primitive variable with rotation angle for Y axis
ZAngle	0		Rotation about Z axis in degrees
ZAnglePrimVar	""		Primitive variable with rotation angle for Z axis
PrimVarToAttribute1	""		prim var to convert to user attribute
PrimVarToAttribute2	""		prim var to convert to user attribute
PrimVarToAttribute3	""		prim var to convert to user attribute
PrimVarToAttribute4	""		prim var to convert to user attribute
PrimVarToAttribute5	""		prim var to convert to user attribute
PrimVarToAttribute6	""		prim var to convert to user attribute

Use the instMultipleArchives instancer to create instances of a numbered sequence of archive files based on a particle system. The archives should be numbered sequentially, with no zero-padding.

The archive to use for each particle should be stored as a primitive variable, and that variable's name provided in the `ArchiveIndexPrimVar` parameter.

The shader can also optionally convert other primitive variables to user attributes assigned to the archives (where they can be queried by other shaders). The prim vars can have either 1 value (converted to a float user attribute) or 3 values (converted to a color user attribute).

See Also

VInstanceArchive instancer shader

9.6.3 MassiveAgents

adds Massive agents to a scene at render-time

Parameter	Default Value	Range	Description
AgentArchivePrefix	`""`		Prefix for per-frame agent rib archive generated by Massive (the file name up until the trailing frame number)
RenderPass	`"renderpass_Air"`		Name of the Massive render pass to use
MotionBlur	`0`	0 or 1	Set to 1 to enable motion blur
RotateX	`0`		Rotation angle in degrees about the X axis
Scale	`1`		Uniform scaling to apply to agents
FrameOffset	`0`		number added to the current scene frame number to obtain the index number for the agent archive
InheritTransform	`1`		When set to 1, the agents inherent the transformation applied to the base primitive
Groups	`"agents"`		List of groups in which to include the agents
ProceduralCount	`1`		Number of instances of the Massive procedural primitive to allow to run simultaneously
Procedural	`"run_program.exe"`		Name (optionally including full path) of the Massive procedural primitive (RunProgram version)

See Also:

Rendering Massive Agents in Rhino

9.6.4 VArrayOnCurve

creates an array of objects along a curve

Parameter	Default Value	Range	Description
Count	10		Number of instances along each curve
Start	0.2	0-1	Position of first instance along curve
End	0.9	0-1	Position of last instance along curve
Archive	""		File name of RIB archive
Scale	1.0		Uniform scale applied to archive
Offset	0 0 0		Translation applied to archive
RotateXYZ	0 0 0		Rotation in degrees about X, Y, Z axes
TwistStart	0		Twist angle at start position
TwistEnd	360		Twist angle at end position

9.6.5 VArrayOnSurface

instancer shader creating a 2D array of objects on a surface

Parameter	Default Value	Range	Description
CountX	10		Copies of archive in X direction
CountY	10		Copies of archive in Y direction
Archive	""		RIB archive file name
Scale	1		Uniform scale applied to archive instances
ScaleVary	0		Per-item variation in scaling
JitterX	0		Variation in X position
JitterY	0		Variation in Y position
Offset	0 0 0		Translation applied to archive
RotateXYZ	0 0 0		X, Y, Z rotation in degrees for archive
RotateVary	0 0 0		Per-item variation in rotation angles
ColorMap	""		Optional color map
Seed	593		Random seed for position variation

The VArrayOnSurface instancer creates instances of a RIB archive in a 2D array over a surface using standard texture coordinates.

If no Archive name is supplied, the shader places a small sphere at each instance location.

9.6.6 VExtrude

instancer that extrudes curves or points along a straight line

Parameter	Default Value	Range	Description
Distance	1		Distance to extrude
Direction	1 0 0		Extrusion direction

9.6.7 VGrowFur

grows short fur on a surface

Parameter	Default Value	Range	Description
Density	`200`		Max hairs per clump
Clumps	`100`		Clumps per surface
Width	`0.01`		Hair width
LengthMax	`0.2`		Max hair length
LengthMin	`0.15`		Min hair length
LengthMap	`""`		Optional map for hair length
LengthVaryFrequency	`5`		Frequency of length variation
TiltMin	`0`	`-90 to 90`	Min tilt angle in degrees
TiltMax	`20`	`-90 to 90`	Max tilt angle in degrees
TiltVaryFrequency	`22`		Frequency of tilt variation
Twist	`360`		Random twist angle per hair
FurColor	`.22 .15 .01`		Base fur color
FurRootColor	`0.5 0.5 0.5`		Fur root color multiplier
FurTipColor	`1 1 1`		Fur tip color multiplier
FurColorMap	`""`		Optional fur color map
FurDiffuse	`0.7`	`0-1`	Diffuse reflectance
FurDiffuse	`0.3`	`0-1`	Specular reflectance

Description

The VGrowFur instancer shader creates short fur on a surface. The fur is positioned using standard texture coordinates, so the underlying surface must have a well-defined set of standard texture coordinates.

The fur is created as a set of clumps. For each clump a single RIB Curves primitive is created with up to `Density` individual hairs. Each hair is represented as a simple linear line segment. The shader adjusts the number of hairs in a clump based on the area of the surface covered by the clump: smaller areas receive fewer hairs.

The shader assigns the VFur surface shader to the fur. The `FurRootColor`, `FurTipColor`, `FurDiffuse`, and `FurSpecular` values are passed to the corresponding VFur parameters.

9.6.8 VInstanceArchive

simple instancer creating copies of a RIB archive at particle locations

Parameter	Default Value	Range	Description
Archive	""		RIB archive file name
ArchiveIsSimple	1	0 or 1	When non-zero, archive is retained as an inline archive to accelerate instancing
Scale	1		Uniform scale applied to archive instances
MotionBlur	0	0 or 1	When non-zero, translation motion blur is applied
ApplyWidth	1	0 or 1	When non-zero, scale archive by point width
ApplyColor	1	0 or 1	When non-zero, apply vertex color to archive
XAngle	0		Rotation about X axis in degrees
XAnglePrimVar	""		Primitive variable to query for rotation angle
YAngle	0		Rotation about Y axis in degrees
YAnglePrimVar	""		Primitive variable to query for rotation angle
ZAngle	0		Rotation about Z axis in degrees
ZAnglePrimVar	""		Primitive variable to query for rotation angle
PrimVarToAttribute1	""		Primitive variable to convert to user attribute
PrimVarToAttribute2	""		Primitive variable to convert to user attribute
PrimVarToAttribute3	""		Primitive variable to convert to user attribute
PrimVarToAttribute4	""		Primitive variable to convert to user attribute
PrimVarToAttribute5	""		Primitive variable to convert to user attribute
PrimVarToAttribute6	""		Primitive variable to convert to user attribute

This simple shader creates a copy of the specifed RIB archive at every vertex position of the instancer base primitive.

In common usage the base primitive will be a points primitive, in which case the point cluster attribute can be used to force a large set of points to be subdivided into smaller sets before the instancer is executed. For example, tagging the base primitive with:

```
Attribute "point" "integer clustermax" [10]
```

would cause the instancer to be run on groups of at most 10 points.

The shader can optionally convert primitive variables to user attributes assigned to the archives (where they can be queried by other shaders). The prim vars can have either 1 value (converted to a float user attribute) or 3 values (converted to a color user attribute).

See Also

instMultipleArchives instancer shader

9.6.9 VInstanceTree

companion instancer shader for the VTree procedure shader

Parameter	Default Value	Range	Description
TreeArchive	""		RIB archive file name with VTree definition
TreeScale	1		Uniform scale applied to archive instances
TreeSeed	124		Base tree seed for random numbers
TreeQuality	1		Tree quality override when not 1
TreeHeightOverride	0		Tree height override if non-zero
TreeRotateX	0		Angle to rotate each tree about X axis
LeafColorOverride	0	0 or 1	Set to 1 to override average leaf color
LeafColor	0 .29 0		Average leaf color
LeafVaryHSL	0.1 0.1 0.1		Per-tree variation in average leaf color

The VInstanceTree instancer shader creates copies or instances of a VTree procedure shader definition at vertex locations of the base primitive (usually a collection of points). Each tree instance receives a unique random seed based on the `TreeSeed` value and the vertex index, so every tree will be unique.

Use `TreeScale` to adjust the size of tree instances to match the scale of your scene. The tree archives included with AIR are modeled in meters. For a scene modeled in centimenters, the appropriate `TreeScale` would be 100.

`TreeHeightOverride` can be used to set a specific tree height for an individual tree. The height should be in the same units as the tree archive.

The `TreeRotateX` parameter can be used to rotate a tree into the correct "up" position. The tree archives included with AIR assume the positive Y axis in world space is "up".

If the `LeafColorOverride` parameter is not 0, the average leaf color of each tree is taken from the `LeafColor` value with per-tree variation in hue, saturation, and lightness based on `LeafVaryHSL`. Note that the per-leaf color variation specified in the tree definition is still applied.

See Also:

VTree procedure shader

9.6.10 VShowBound

generates a wire frame bounding box for a primitive

Parameter	Default Value	Range	Description
LineWidth	0.1		Line width for wire frame box
LineColor	1 1 1		Line color
Print	0	0 or 1	If 1, prints the bound for each object to stdout

9.7 Lights

Light shaders are used to illuminate a scene.

Regular Lights

ambientlight	emits constant illumination regardless of surface orientation
distantlight	emits illumination in a single direction
pointlight	emits light in all directions from a single source point
spherelight	emits light like a spherical area light
spotlight	emits light in a cone of directions

Area Lights

arealight	shader for area lights using light supersampling
arealight2	shader for area lights using Air's fast areashadow() function
cardlight	emulates adding a constant shaded card
portallight	area light shader for windows, with environment map
texturedarealight	area light with optional texture map

Lights with Texture Maps

distant_projector	projects a texture using a distant light base
spot_projector	projects a texture using a spotlight
uberlight	multipurpose light with many controls

Special Lights

photometric_pointlight	point light source with IES support and physical illumination units
caustic	caustics
envlight	adds indirect illumination using ambient occlusion
indirect	adds indirect diffuse illumination ("GI" or "global illumination")
sunlight	sun light based on position, date/time & atmosphere

Lights for Baked Shadows and Illumination

Most light shaders have an extended version that allows shadows or illumination to be baked to 3D texture maps and re-used. The documentation for each light shader includes information on the "bake" version when appropriate.

See Also

Lighting

9.7.1 ambientlight

RenderMan standard ambient light shader

Parameter	Default Value	Range	Description
intensity	`1.0`		Light intensity
lightcolor	`1 1 1`		Light color

Description

The ambient light shader simply sets the emitted light value to the product of the `intensity` and `lightcolor` parameters.

9.7.2 arealight, arealight_baked

area light shader for general area lights

Parameter	Default Value	Range	Description
intensity	`1.0`		Light intensity
lightcolor	`1 1 1`		Light color
lightcolor_spd	`""`		Light color spectrum file
maxemission	`0`		If greater than 0, the maximum emitted light intensity (including falloff)
coneangle	`1.57`	`0.0-1.5`	Angle for emitted light range
falloff	`2`	`0, 1, 2`	Exponent for falloff in light intensity with distance
shadowname	`"raytrace"`		Name of shadow map or `raytrace` for traced shadows
shadowbias	`0.01`		Offset to prevent incorrect self-shadowing
shadowcolor	`0 0 0`		Color of shadows
shadowgroups	`""`		Optional groups for ray-traced shadows
shadowattribute	`"user:shadowgroups"`		Per-object shadowgroups override attribute name
__nondiffuse	`0`	`0, 1`	If 1 light is excluded from diffuse light
__nonspecular	`0`	`0, 1`	If 1 light will not produce highlights
__foglight	`1`	`0, 1`	If 1 light illuminates volumes
__channel	`-1`	`0-9`	Output channel

Description

The arealight shader can be applied to any geometric primitive to turn it into an area light source. The quality of the illumination and shadows is controlled with the light's nsamples attribute:

```
  Attribute "light" "integer nsamples" [n]
```

where *n* ranges between 1 and 256. A large number of samples produces smoother results.

For area light primitives that are simple and convex, such as a single polygon or a disk, the arealight2 shader can produce similar results much more quickly.

Light Intensity

The emitted light value is the product of the `intensity` and `lightcolor` parameters and an attenuation factor determined by the `falloff` parameter. The `falloff` parameter allows the light's intensity to diminish with distance from the light. A falloff value of 2 (which is physically accurate) reduces the incident light value by the square of the distance from the light:

```
  Cl = intensity * lightcolor / (Distance * Distance)
```

Although a falloff of 2 is physically correct, it can produce very large intensity values at locations close to the light source. The `maxemission` parameter can be used to set the maximum emitted intensity regardless of distance from the light source.

Alternatively, a falloff value of 0 produces no attenuation with distance, which may be easier to control.

Shadows

The `shadowgroups` parameter can be used to restrict shadow-casting to the specified list of groups. The shadow group can be overridden on a per-object basis by providing a user attribute:

```
  Attribute "user" "string shadowgroups" ["my custom groups"]
```

The `shadowattribute` parameter gives the name of the user attribute to check for a custom shadow casting-group. By default all lights look for a custom shadow set in the same user attribute. By providing a different user attribute for each light, one can precisely specify for each object and each light which objects are tested for shadows.

If the shadow group has the special value "null", no shadow rays will be traced and the object will not receive shadows.

Output only variables

`color __shadow`	Shadow value
`color __unshadowed_Cl`	Emitted illumination without shadows

Baked Shadows

AIR includes an extended version of the arealight shader - arealight_baked - with additional parameters for creating 3D shadow textures and using 2D or 3D baked shadows:

bakemap	""		Bake map file name
bakemode	""		Bake mode
bakemapblur	0		Blur for bake map query
bakemapfilterwidth	1		Filter width for bake map query
motionbound	0 0 0 0 0 0		bounding box for moving objects
motionspace	"world"	space	coordinate space of motionbound

See Baking Shadows and Illumination for more information. Note that the arealight_baked shader does not support the write or append bake modes. The arealight_baked shader does support the read mode for 3D shadow data baked with the arealight2_baked shader.

See Also

area lights
arealight2 shader
Baking Shadows and Illumination

9.7.3 arealight2, arealight2_baked

fast area light shader using AIR's areashadow function

Parameter	Default Value	Range	Description
intensity	1.0		Light intensity
lightcolor	1 1 1		Light color
lightcolor_spd	""		Light color spectrum file
maxemission	0		If greater than 0, the maximum emitted intensity, including falloff
samples	1	1-256	Samples for area light
falloff	2	0, 1, 2	Exponent for falloff in light intensity with distance
shadowbias	0.01		Offset to prevent incorrect self-shadowing
shadowcolor	0 0 0		Color of shadows
shadowgroups	""		Optional groups for ray-traced shadows
shadowattribute	"user:shadowgroups"		Per-object shadowgroups override attribute name
__nondiffuse	0	0, 1	If 1 light is excluded from diffuse light
__nonspecular	0	0, 1	If 1 light will not produce highlights
__foglight	1	0, 1	If 1 light illuminates volumes
__channel	-1	0-9	Output channel

Description

The arealight2 shader provides fast rendering for area lights with simple, convex geometry, such as a single polygon or disk. The quality of the area light illumination is determined by the shader samples parameter. The light's nsamples attribute should be set to 1 when the arealight2 shader is used.

```
Attribute "light" "integer nsamples" [1]
```

For complex area light geometry, use the arealight shader instead.

Light Intensity

The emitted light value is the product of the `intensity` and `lightcolor` parameters and an attenuation factor determined by the `falloff` parameter. The `falloff` parameter allows the light's intensity to diminish with distance from the light. A falloff value of 2 (which is physically accurate) reduces the incident light value by the square of the distance from the light:

```
Cl = intensity * lightcolor / (Distance * Distance)
```

Although a falloff of 2 is physically correct, it can produce very large intensity values at locations close to the light source. The `maxemission` parameter can be used to set the maximum emitted intensity regardless of distance from the light source.

Alternatively, a falloff value of 0 produces no attenuation with distance, which may be easier to control.

Shadows

The `shadowgroups` parameter can be used to restrict shadow-casting to the specified list of groups. The shadow group can be overridden on a per-object basis by providing a user attribute:

```
Attribute "user" "string shadowgroups" ["my custom groups"]
```

The `shadowattribute` parameter gives the name of the user attribute to check for a custom shadow casting-group. By default all lights look for a custom shadow set in the same user attribute. By providing a different user attribute for each light, one can precisely specify for each object and each light which objects are tested for shadows.

If the shadow group has the special value "null", no shadow rays will be traced and the object will not receive shadows.

Output only variables

`color __shadow`	Shadow value
`color __unshadowed_Cl`	Emitted illumination without shadows

Baked Shadows

AIR includes an extended version of the arealight2 shader - arealight2_baked - with additional parameters for creating 3D shadow textures and using 2D or 3D baked shadows:

bakemap	""		Bake map file name
bakemode	""		Bake mode
bakemapblur	0		Blur for bake map query
bakemapfilterwidth	1		Filter width for bake map query
motionbound	0 0 0 0 0 0		bounding box for moving objects
motionspace	"world"	space	coordinate space of motionbound

See Baking Shadows and Illumination for more information.

See Also

area lights
arealight shader
Baking Shadows and Illumination

9.7.4 cardlight

area light shader that emulates adding a constant shaded card

Parameter	Default Value	Range	Description
intensity	`1.0`		Light intensity
lightcolor	`1 1 1`		Light color
coneangle	`1.57`	`0.0-1.5`	Angle for emitted light range
falloff	`2`	`0, 1, 2`	Exponent for falloff in light intensity with distance
shadowname	`"raytrace"`		Name of shadow map or `raytrace` for traced shadows
shadowbias	`0.01`		Offset to prevent incorrect self-shadowing
shadowcolor	`0 0 0`		Color of shadows
shadowgroups	`""`		Optional groups for ray-traced shadows
shadowattribute	`"user:shadowgroups"`		Per-object shadowgroups override attribute name
__nondiffuse	`0`	`0, 1`	If 1 light is excluded from diffuse light
__nonspecular	`0`	`0, 1`	If 1 light will not produce highlights
__foglight	`1`	`0, 1`	If 1 light illuminates volumes
__channel	`-1`	`0-9`	Output channel

Description

The cardlight shader emulates the effect of adding a constant shaded diffuse emitter to a scene.

Output only variables

`color __shadow`	Shadow value
`color __unshadowed_Cl`	Emitted illumination without shadows

See Also

9.7.5 caustic, caustic_baked

light source returning caustics

Parameter	Default Value	Range	Description
intensity	`1.0`		Light intensity
lightcolor	`1 1 1`		Light color
__nonspecular	`1`	`0, 1`	If 1 light will not produce highlights
__foglight	`0`	`0, 1`	If 1 light illuminates volumes
__channel	`-1`	`0-9`	Output channel index

Description

The caustic light shader returns any visible caustics at the current shading location.

Baked Caustics

AIR includes an extended version of the caustic shader - caustic_baked - with additional parameters for baking caustics to a 3D texture map and using a 2D or 3D texture for caustics.

bakemap	`""`		Bake map file name
bakemode	`""`		Bake mode
bakemapblur	`0`		Blur for bake map query
bakemapfilterwidth	`1`		Filter width for bake map query
motionbound	`0 0 0 0 0 0`		bounding box for moving objects
motionspace	`"world"`	space	coordinate space of motionbound

See Baking Shadows and Illumination for more information.

See Also

Caustics
Baking Shadows and Illumination

9.7.6 distantlight, distantlight_baked

directional light shader

Parameter	Default Value	Range	Description
intensity	`1.0`		Light intensity
lightcolor	`1 1 1`		Light color
from	`0 0 0`		Source position of light
to	`0 0 1`		Target of light
shadowname	`""`		Name of shadow map or `raytrace` for traced shadows
shadowbias	`0.01`		Offset to prevent incorrect self-shadowing
shadowblur	`0.01`	`0.0-0.5`	Blur for shadows
shadowsamples	`1`	`1-256`	Number of rays for traced shadows
shadowcolor	`0 0 0`		Color of shadows
shadowgroups	`""`		Optional groups for ray-traced shadows
shadowattribute	`"user:shadowgroups"`		Per-object shadowgroups override attribute name
__nondiffuse	`0`	`0, 1`	If 1 light is excluded from diffuse light
__nonspecular	`0`	`0, 1`	If 1 light will not produce highlights
__foglight	`1`	`0, 1`	If 1 light illuminates volumes
__channel	`-1`	`0-9`	Output channel

Description

The distantlight shader emits light in a single direction defined by the vector `to-from`. The incident light at the point being shaded is simply the product of the `intensity` and `lightcolor` parameters.

Traced Shadows

If the shadowname parameter is set to the special value `"raytrace"`, shadows are generated by tracing shadow rays. The `shadowsamples` parameter specifies the number of shadow rays to trace. The `shadowblur` parameter gives an angle (in radians) to distribute the rays around the point being shaded. Adjust the `shadowbias` parameter to prevent incorrect self-shadowing.

The `shadowgroups` parameter can be used to restrict shadow-casting to the specified list of groups. The shadow group can be overridden on a per-object basis by providing a user attribute:

```
Attribute "user" "string shadowgroups" ["my custom groups"]
```

The `shadowattribute` parameter gives the name of the user attribute to check for a custom shadow casting-group. By default all lights look for a custom shadow set in the same user attribute. By providing a different user attribute for each light, one can precisely specify for each object and each light which objects are tested for shadows.

If the shadow group has the special value "null", no shadow rays will be traced and the object will not receive shadows.

Shadow-mapped Shadows

Assign the file name of a previously generated shadow map to the `shadowname` parameter to generate shadow-mapped shadows. The `shadowblur` parameter gives a blur value as a fraction of the shadow map size. AIR ignores the `shadowsamples` parameter for shadow-mapped shadows, but other renderers may use it to control the quality of mapped shadows. Adjust the `shadowbias` parameter to prevent incorrect self-shadowing.

Output only variables

`color __shadow`	Shadow value
`color __unshadowed_Cl`	Emitted illumination without shadows

Baked Shadows

AIR includes an extended version of the distantlight shader - distantlight_baked - with additional parameters for creating 3D shadow textures and using 2D or 3D baked shadows:

bakemap	`""`		Bake map file name
bakemode	`""`		Bake mode
bakemapblur	`0`		Blur for bake map query
bakemapfilterwidth	`1`		Filter width for bake map query
motionbound	`0 0 0 0 0 0`		bounding box for moving objects
motionspace	`"world"`	space	coordinate space of motionbound

See Baking Shadows and Illumination for more information.

9.7.7 distant_projector

directional light shader that projects a texture map like a slide projector

Parameter	Default Value	Range	Description
intensity	`1.0`		Light intensity
lightcolor	`1 1 1`		Light color
from	`0 0 0`		Source position of light
to	`0 0 1`		Target of light
shadowname	`""`		Name of shadow map or `raytrace` for traced shadows
shadowbias	`0.01`		Offset to prevent incorrect self-shadowing
shadowblur	`0.01`	`0.0-0.5`	Blur for shadows
shadowsamples	`1`	`1-256`	Number of rays for traced shadows
shadowcolor	`0 0 0`		Color of shadows
shadowgroups	`""`		Optional groups for ray-traced shadows
shadowattribute	`"user:shadowgroups"`		Per-object shadowgroups override attribute name
texturename	`""`		File name of texture map to project
textureblur	`0.0`		Blur for texture map
texturescale	`1.0`		Scale factor for texture
texturerotate	`0`		Rotation angle in degrees about the cone axis
textureonce	`0`	`0 or 1`	When set to 1, only one copy of the texture is displayed
textureup	`0 1 0`		Vertical axis used to orient the map projection
__nondiffuse	`0`	`0, 1`	If 1 light is excluded from diffuse light
__nonspecular	`0`	`0, 1`	If 1 light will not produce highlights
__foglight	`1`	`0, 1`	If 1 light illuminates volumes
__channel	`-1`	`0-9`	Output channel

Description

The distant_projector shader projects a texture map in the direction defined by the vector `to-from`. The incident light at the point being shaded is the product of the `intensity` and `lightcolor` parameters and the texture lookup result.

The texture map given by texturename parameter is mapped inside a square given by the texturescale parameter. The textureup parameter defines a vertical axis used to orient the map. The texturerotate parameter can be used to rotate the map.

By default only one copy of the map is displayed. You can change that behavior by setting the textureonce parameter to 0 and converting the texture map to an Air texture file using the Air texture conversion tools mktex or mktexui, selecting the desired wrap mode in the process.

Traced Shadows

If the shadowname parameter is set to the special value `"raytrace"`, shadows are generated by tracing shadow rays. The `shadowsamples` parameter specifies the number of shadow rays to trace.

The `shadowblur` parameter gives an angle (in radians) to distribute the rays around the point being shaded. Adjust the `shadowbias` parameter to prevent incorrect self-shadowing.

The `shadowgroups` parameter can be used to restrict shadow-casting to the specified list of groups. The shadow group can be overridden on a per-object basis by providing a user attribute:

```
Attribute "user" "string shadowgroups" ["my custom groups"]
```

The `shadowattribute` parameter gives the name of the user attribute to check for a custom shadow casting-group. By default all lights look for a custom shadow set in the same user attribute. By providing a different user attribute for each light, one can precisely specify for each object and each light which objects are tested for shadows.

If the shadow group has the special value "null", no shadow rays will be traced and the object will not receive shadows.

Shadow-mapped Shadows

Assign the file name of a previously generated shadow map to the `shadowname` parameter to generate shadow-mapped shadows. The `shadowblur` parameter gives a blur value as a fraction of the shadow map size. AIR ignores the `shadowsamples` parameter for shadow-mapped shadows, but other renderers may use it to control the quality of mapped shadows. Adjust the `shadowbias` parameter to prevent incorrect self-shadowing.

Output only variables

`color __shadow`	Shadow value
`color __unshadowed_Cl`	Emitted illumination without shadows

9.7.8 envlight, envlight_baked

environment light using ambient occlusion

Parameter	Default Value	Range	Description
intensity	`1.0`		Light intensity
lightcolor	`1 1 1`		Light color
sampleenvironment	`0`	0 or 1	When enabled, the scene environment is sampled for unoccluded rays and the envmap value is ignored
samplelikediffuse	`0`	0 or 1	When enabled, the environment is sampled using a lambert brdf, producing a result more similar to indirect diffuse lighting
envmap	`""`		Optional environment map for background
envspace	`"world"`	spaces	Coordinate space for environment map
envsamples	`64`		Samples for environment map lookup
envblur	`0.5`		Blur for environment map lookup
mapname	`"raytrace"`		`raytrace` or name of occlusion map
mapblur	`0.01`	`0.0-1.0`	Blur for occlusion map
mapbias	`0.01`		Bias for occlusion map
maxsolidangle	`0.05`		Max solid angle in radians for grouping of points in point-based occlusion
shadowcolor	`0 0 0`		Shadow color
shadowgroups	`""`		Optional list of groups to query for occlusion
__category	`"indirect"`		Light categories
__nonspecular	`1`	`0, 1`	If 1 light will not produce highlights
__foglight	`0`	`0, 1`	If 1 light illuminates volumes
__indirectlight	`1`	`0, 1`	Flag indicating indirect light status for multipass rendering
__channel	`-1`	`0-9`	Output channel

Description

The envlight shader simulates light incident from a surrounding environment with ambient occlusion. The incoming light can be a constant color or taken from an environment map specified with the `envmap` parameter.

Shadows are simulated using AIR's occlusion() function. When the `mapname` parameter is set to "raytrace", ray tracing is used to compute the ambient occlusion at each point. Otherwise, `mapname` is treated as the file name for an occlusion map with occlusion information.

Output only variables

`color __shadow`	Shadow value
`color __unshadowed_Cl`	Emitted illumination without shadows

Baked Shadows

AIR includes an extended version of the envlight shader - envlight_baked - with additional parameters for creating 3D shadow textures and using 2D or 3D baked shadows:

bakemap	`""`	Bake map file name
bakemode	`""`	Bake mode
bakemapblur	`0`	Blur for bake map query
bakemapfilterwidth	`1`	Filter width for bake map query

See Baking Shadows and Illumination for more information.

See Also

Ambient occlusion

9.7.9 indirect, indirect_baked

light source emitting indirect diffuse illumination

Parameter	Default Value	Range	Description
intensity	`1.0`		Light intensity
lightcolor	`1 1 1`		Light color
environmentmap	`""`		Optional environment map for background
environmentspace	`"world"`	spaces	Coordinate space for environment map
__category	`"indirect"`		Light categories
__nonspecular	`1`	`0, 1`	If 1 light will not produce highlights
__foglight	`0`	`0, 1`	If 1 light illuminates volumes
__indirectlight	`1`	`0, 1`	Flag indicating indirect light status for multipass rendering
__channel	`-1`	`0-9`	Output channel

Description

The indirect light shader encapsulates AIR's indirect diffuse illumination capability (sometimes referred to as "global illumination").

The emitted light value is the indirect light incident at the current shading location scaled by the `intensity` and `lightcolor` parameters, which can be used to "tweak" the indirect diffuse results.

The optional `environmentmap` parameter gives an environment map to query with indirect sample rays that miss all objects in the scene. A better way of setting the background environment map is with the following option:

```
Option "indirect" "string envname" [""]
```

Baked Illumination

AIR includes an extended version of the indirect shader - indirect_baked - with additional parameters for baking indirect illumination to a 3D texture map and using 2D or 3D texture maps with stored indirect values.

bakemap	""	Bake map file name
bakemode	""	Bake mode
bakemapblur	0	Blur for bake map query
bakemapfilterwidth	1	Filter width for bake map query

See Baking Shadows and Illumination for more information.

See Also

Indirect diffuse illumination
envlight shader

9.7.10 indirectchannels

light shader for indirect diffuse illumination with light channel tracking

Parameter	Default Value	Range	Description
intensity	1.0		Light intensity
lightcolor	1 1 1		Light color
environmentmap	""		Optional environment map for background
environmentspace	"world"	spaces	Coordinate space for environment map
__category	"indirect"		Light categories
__nonspecular	1	0, 1	If 1 light will not produce highlights
__foglight	0	0, 1	If 1 light illuminates volumes
__indirectlight	1	0, 1	Flag indicating indirect light status for multipass rendering
__channel	-1	0-9	Output channel
__bgchannel	-1	0-9	Light channel for indirect bg value

Output only variables

`color __background`	Background environment map contribution
`color[10] __channels`	Per-light output channels

Description

The indirectchannels light shader tracks light channels through indirect diffuse illumination bounces. The resulting __channels output value is automatically picked up by the standard `diffuse()` function and incorporated into its light channel output variables.

The __bgchannel parameter can be used to store the contribution of the background environment map in an otherwise unused light channel so the bg contribution can also be traced through reflections.

See Also

indirect light shader
Light channels

9.7.11 massive_envlight

environment light using ambient occlusion without the occlusion cache, designed for use with Massive

Parameter	Default Value	Range	Description
intensity	1.0		Light intensity
lightcolor	1 1 1		Light color
samples	256	1-1024	Rays to cast to estimate occlusion
maxhitdist	500		Maximum distance to search for occluding objects
adaptivesampling	-1	-1, 0, or 1	Whether to use adaptive sampling (0=no, 1=yes, -1=inherit from adaptivesampling attribute)
shadowbias	0.1		Offset to prevent incorrect self-shadowing
coneangle	1.57		Half angle in radians of the sample cone
envmap	""		Optional environment map contributing background illumination
envspace	"world"	spaces	Coordinate space for environment map
envblur	0.5		Blur for environment map lookup
envsamples	64		Number of samples for map lookup
rotate_x	0		map rotation angle in degress about the x axis
rotate_y	0		map rotation angle in degress about the y axis
rotate_z	0		map rotation angle in degress about the z axis
shadowcolor	0 0 0		Shadow color
__category	"indirect"		Light categories
__nonspecular	1	0, 1	If 1 light will not produce highlights
__foglight	0	0, 1	If 1 light illuminates volumes
__indirectlight	1	0, 1	Flag indicating indirect light status for multipass rendering
__channel	-1	0-9	Output channel

Description

The massive_envlight shader simulates light incident from a surrounding environment with ambient occlusion. The incoming light can be a constant color or taken from an environment map specified with the `envmap` parameter. This shader differs from the envlight shader in providing additional parameters for explicitly specifying the number of occlusion samples and maximum hit distance.

Shadows are computed by tracing rays at each point using AIR's occlusion() function.

Output only variables

`color __shadow`	Shadow value
`color __unshadowed_Cl`	Emitted illumination without shadows

See Also

Ambient occlusion
envlight
Ambient occlusion in Massive

9.7.12 massive_indirect

light source emitting indirect diffuse illumination, with extra controls for use with Massive

Parameter	Default Value	Range	Description
intensity	`1.0`		Light intensity
lightcolor	`1 1 1`		Light color
samples	`256`	`1-1024`	Number of rays to trace for each sample
maxhitdist	`500`		Max distance to search for nearby surfaces
adaptivesampling	`-1`	-1, 0 or 1	Whether to use adaptive sampling
environmentmap	`""`		Optional environment map for background
environmentspace	`"world"`	spaces	Coordinate space for environment map
__category	`"indirect"`		Light categories
__nonspecular	`1`	`0, 1`	If 1 light will not produce highlights
__foglight	`0`	`0, 1`	If 1 light illuminates volumes
__indirectlight	`1`	`0, 1`	Flag indicating indirect light status for multipass rendering
__channel	`-1`	`0-9`	Output channel

Description

The massive_indirect light shader encapsulates Air's indirect diffuse illumination capability (sometimes referred to as "global illumination").

The emitted light value is the indirect light incident at the current shading location scaled by the `intensity` and `lightcolor` parameters, which can be used to "tweak" the indirect diffuse results.

This light shader is designed for computing indirect illumination results without using an irradiance cache. The samples, maxhitdist, and adaptivesampling parameters provide basic controls over GI quality.

Massive users will also need to assign a custom options rib to the scene with the following attribute to enable object visibility to indirect rays:

```
Attribute "visibility" "indirect" 1
```

Users may also wish to set the number of indirect bounces with:

```
Option "indirect" "maxbounce" [n]
```

By default only 1 bounce of indirect illumination is computed.

See Also

massive_envlight light shader
indirect light shader
Indirect Lighting

9.7.13 photometric_pointlight

point light source with support for IES light profiles and physically-based illumination values

Parameter	Default Value	Range	Description
intensity	`1.0`		Light intensity
intensityunit	`"none"`	`none,candela,lumen,watt`	Units for light intensity
lightcolor	`1 1 1`		Light color
lightcolor_spd	`""`		Spectral power distribution for light color
from	`0 0 0`		Source position of light
falloff	`2`	`0, 1, 2`	Exponent for falloff in light intensity with distance
sceneunitsize	`1`		Size of one unit in world space
illuminanceunity	`500`		Divisor for emitted light intensity
lumensperwatt	`17`		Lumens per watt for watt intensity unit
iesname	`""`		file name for IES light profile
iesspace	`"shader"`	spaces	coordinate space for IES profile environment map lookup
iesrotatex	`0`		ies rotation about X axis in degrees
iesrotatey	`0`		ies rotation about Y axis in degrees
iesrotatez	`0`		ies rotation about Z axis in degrees
shadowname	`""`		Name of shadow map or `raytrace` for traced shadows
shadowbias	`0.01`		Offset to prevent incorrect self-shadowing
shadowblur	`0.01`	`0.0-0.5`	Blur for shadows
shadowsamples	`1`	`1-256`	Number of rays for traced shadows
shadowcolor	`0 0 0`		Color of shadows
shadowgroups	`""`		Optional groups for ray-traced shadows
shadowattribute	`"user:shadowgroups"`		Per-object shadowgroups override attribute name
__nondiffuse	`0`	`0, 1`	If 1 light is excluded from diffuse light
__nonspecular	`0`	`0, 1`	If 1 light will not produce highlights
__foglight	`1`	`0, 1`	If 1 light illuminates volumes
__channel	`-1`	`0-9`	Output channel

Description

The photometric_pointlight shader is a point light source with support for physically-based illumination values. A description of the physically-based parameters can be found in the section Photometric Lights and IES Profiles. The shadow controls in the shader behave exactly like those in the standard pointlight shader.

9.7.14 pointlight, pointlight_baked

point light source shader

Parameter	Default Value	Range	Description
intensity	`1.0`		Light intensity
lightcolor	`1 1 1`		Light color
lightcolor_spd	`""`		Light color spectrum file
maxemission	`0`		If greater than 0, the maximum emitted intensity (including falloff)
from	`0 0 0`		Source position of light
falloff	`2`	`0, 1, 2`	Exponent for falloff in light intensity with distance
shadowname	`""`		Name of shadow map or `raytrace` for traced shadows
shadowbias	`0.01`		Offset to prevent incorrect self-shadowing
shadowblur	`0.01`	`0.0-0.5`	Blur for shadows
shadowsamples	`1`	`1-256`	Number of rays for traced shadows
shadowcolor	`0 0 0`		Color of shadows
shadowgroups	`""`		Optional groups for ray-traced shadows
shadowattribute	`"user:shadowgroups"`		Per-object shadowgroups override attribute name
__category	`"lensflare"`		Light category
__nondiffuse	`0`	`0, 1`	If 1 light is excluded from diffuse light
__nonspecular	`0`	`0, 1`	If 1 light will not produce highlights
__foglight	`1`	`0, 1`	If 1 light illuminates volumes
__channel	`-1`	`0-9`	Output channel

Description

The pointlight shader emits light in all directions from the source point `from`.

Light Intensity

The emitted light value is the product of the `intensity` and `lightcolor` parameters and an attenuation factor determined by the `falloff` parameter. The `falloff` parameter allows the light's intensity to diminish with distance from the light. A falloff value of 2 (which is physically accurate) reduces the incident light value by the square of the distance from the light:

```
  Cl = intensity * lightcolor / (Distance * Distance)
```

Although a falloff of 2 is physically correct, it can produce very large intensity values at locations close to the light source. The `maxemission` parameter can be used to set the maximum emitted intensity regardless of distance from the light source.

Alternatively, a falloff value of 0 produces no attenuation with distance, which may be easier to control.

Traced Shadows

If the shadowname parameter is set to the special value `"raytrace"`, shadows are generated by tracing shadow rays. The `shadowsamples` parameter specifies the number of shadow rays to trace. The `shadowblur` parameter gives an angle (in radians) to distribute the rays around the point being shaded. Adjust the `shadowbias` parameter to prevent incorrect self-shadowing.

The `shadowgroups` parameter can be used to restrict shadow-casting to the specified list of groups. The shadow group can be overridden on a per-object basis by providing a user attribute:

```
Attribute "user" "string shadowgroups" ["my custom groups"]
```

The `shadowattribute` parameter gives the name of the user attribute to check for a custom shadow casting-group. By default all lights look for a custom shadow set in the same user attribute. By providing a different user attribute for each light, one can precisely specify for each object and each light which objects are tested for shadows.

If the shadow group has the special value "null", no shadow rays will be traced and the object will not receive shadows.

Shadow-mapped Shadows

Assign the file name of a previously generated shadow map to the `shadowname` parameter to generate shadow-mapped shadows. The `shadowblur` parameter gives a blur value as a fraction of the shadow map size. AIR ignores the `shadowsamples` parameter for shadow-mapped shadows, but other renderers may use it to control the quality of mapped shadows. Adjust the `shadowbias` parameter to prevent incorrect self-shadowing.

Output only variables

`color __shadow`	Shadow value
`color __unshadowed_Cl`	Emitted illumination without shadows

Baked Shadows

AIR includes an extended version of the pointlight shader - pointlight_baked - with additional parameters for creating 3D shadow textures and using 2D or 3D baked shadows:

bakemap	""		Bake map file name
bakemode	""		Bake mode
bakemapblur	0		Blur for bake map query
bakemapfilterwidth	1		Filter width for bake map query
motionbound	0 0 0 0 0 0		bounding box for moving objects
motionspace	"world"	space	coordinate space of motionbound

See Baking Shadows and Illumination for more information.

9.7.15 portallight

shader for area lights representing windows or other openings

Parameter	Default Value	Range	Description
intensity	1.0		Light intensity
lightcolor	1 1 1		Light color
falloff	2		Exponent for attenuation with distance
environmentmap	""		Optional environment map for background
environmentspace	"world"	spaces	Coordinate space for environment map
environmentblur	0.5		Blur for environment map lookup
environmentsamples	16		Samples for environment map lookup
shadowname	"raytrace"		raytrace for traced shadows or "" for no shadows
shadowbias	0.01		Offset for traced rays
shadowcolor	0 0 0		Shadow color
shadowgroups	""		Shadow-casting groups
__category	"indirect"		Light categories
__nonspecular	0	0, 1	If 1 light will not produce highlights
__foglight	0	0, 1	If 1 light illuminates volumes
__nondiffuse	0	0, 1	If 1 light contributes diffuse illumination
__channel	-1	0-9	Output channel index

Description

Use the portallight shader for area lights representing a window or other opening admitting illumination from an external source captured in an environment map. For interior scenes using an area light with this shader can be more efficient than trying to capture the effects of an external environment with indirect diffuse illumination.

The quality of the illumination and shadows is controlled with the light's nsamples attribute:

```
  Attribute "light" "integer nsamples" [n]
```

where *n* ranges between 1 and 256. A large number of samples produces smoother results.

Output only variables

`color __shadow`	Shadow value
`color __unshadowed_Cl`	Emitted illumination without shadows

See Also

area lights
arealight shader

9.7.16 spherelight

light shader to make a point light behave like a spherical area light

Parameter	Default Value	Range	Description
intensity	`1.0`		Light intensity
lightcolor	`1 1 1`		Light color
lightcolor_spd	`""`		Light color spectrum file
lightradius	`1.0`		Radius of the virtual sphere
from	`0 0 0`		Source position of light
falloff	`2`	`0, 1, 2`	Exponent for falloff in light intensity with distance
shadowname	`""`		Name of shadow map or `raytrace` for traced shadows
shadowbias	`0.01`		Offset to prevent incorrect self-shadowing
shadowblur	`0.01`	`0.0-0.5`	Blur for shadows
shadowsamples	`4`	`1-256`	Number of rays for traced shadows
shadowcolor	`0 0 0`		Color of shadows
shadowgroups	`""`		Optional groups for ray-traced shadows
shadowattribute	`"user:shadowgroups"`		Per-object shadowgroups override attribute name
__category	`""`		Light category
__nondiffuse	`0`	`0, 1`	If 1 light is excluded from diffuse light
__nonspecular	`0`	`0, 1`	If 1 light will not produce highlights
__foglight	`1`	`0, 1`	If 1 light illuminates volumes
__channel	`-1`	`0-9`	Output channel

Description

A spherical light source can provide more realistic illumination than a traditional point light because it gives the light an actual size and shape. The usual way to render a sphere light is to use an arealight shader with sphere geometry. That approach works, but it has some drawbacks:

- Render time can be significantly slower because the area light shader must be evaluated at many different locations on the sphere to produce a smooth result.
- Some software (such as Massive) does not provide an interface for area lights.

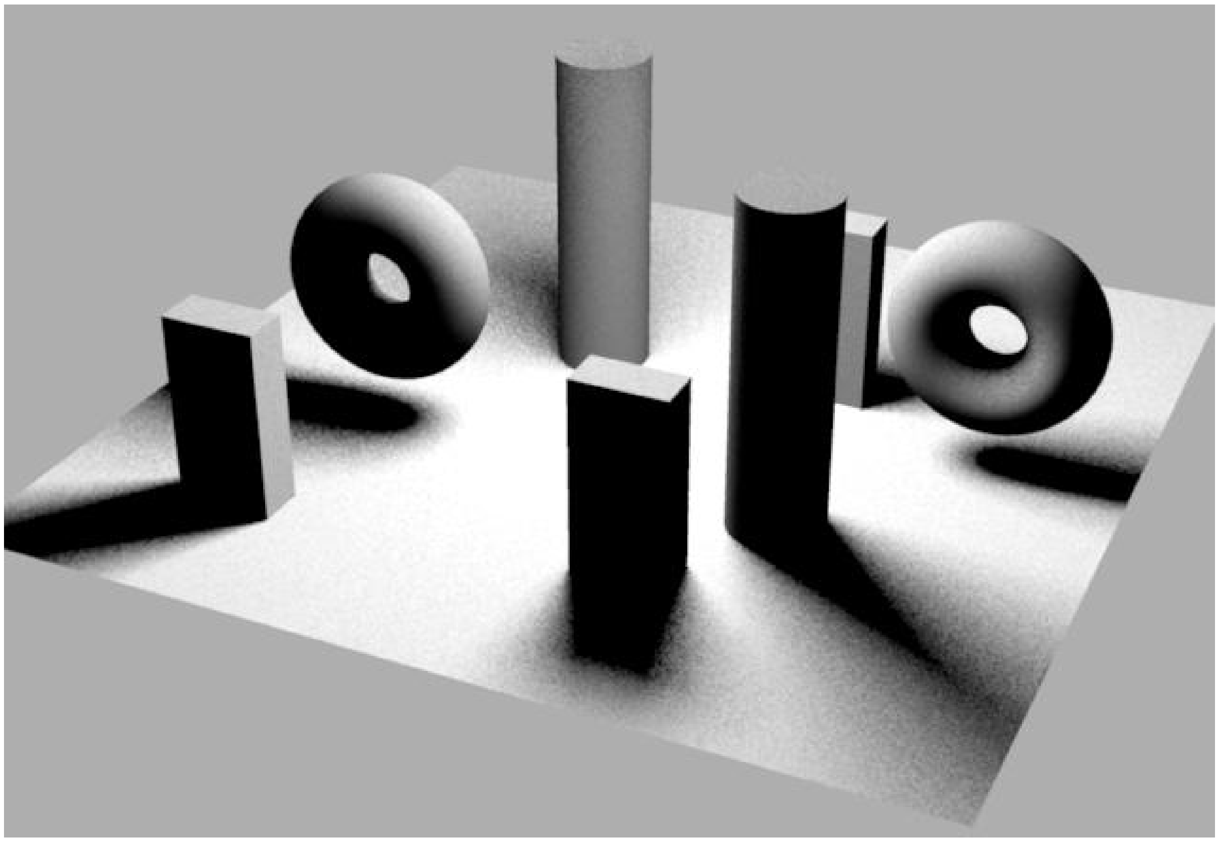
To address those drawbacks, we've put together a spherelight light shader that can be used to simulate the effect of a spherical light source using a standard point-type light. The shader is similar to the pointlight shader with the addition of a `lightradius` parameter for the radius of the (virtual) spherical light source. The shader uses the light radius in a couple computations that approximate the effects of a spherical source:

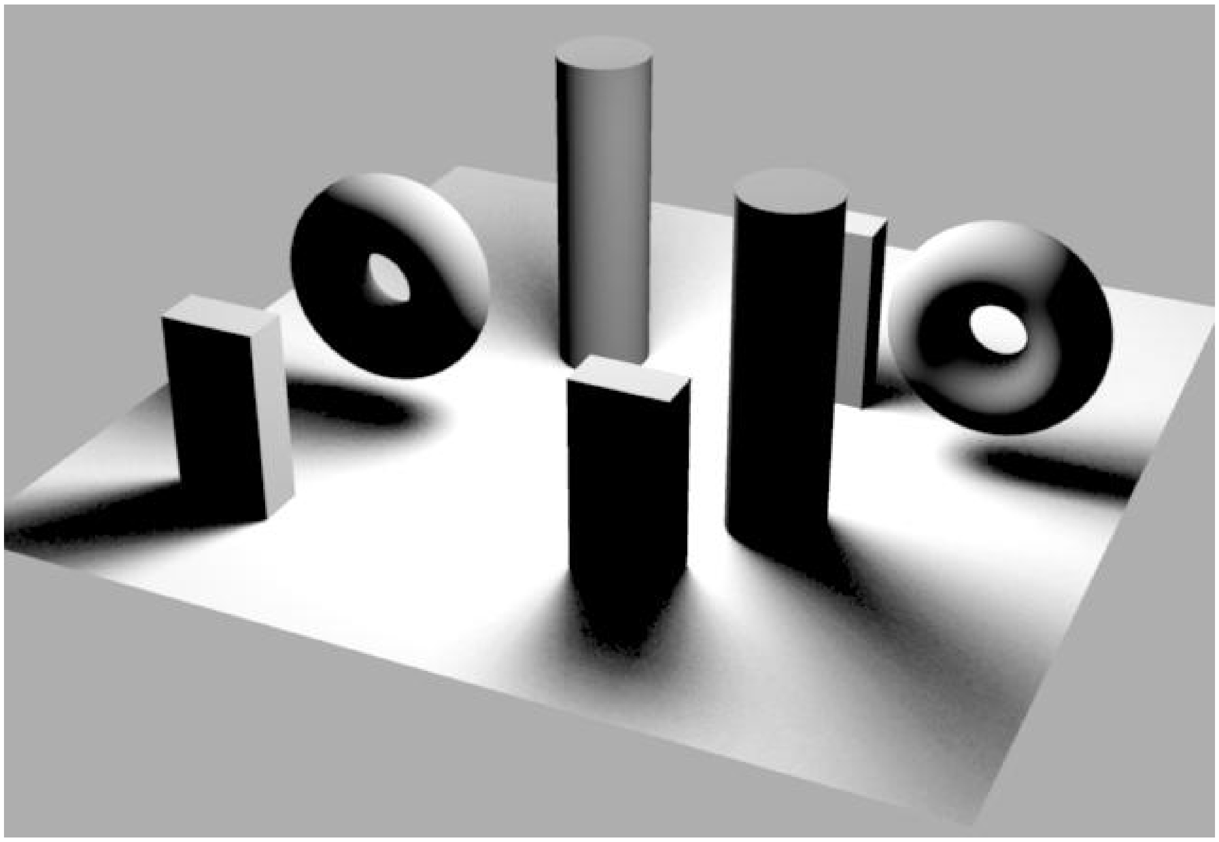
- Shadows are blurry close to the source and sharper farther away. The shader simulates this effect by computing a shadow blur angle based on the distance to the light source and the light radius:

  ```
  lenL=max(sqrt(L.L),radius);
  blur = abs(atan(radius,sqrt(lenL*lenL-radius*radius)));
  ```

- For locations close to the sphere surface, the light source no longer behaves as an idealized point light. The shader accounts for this by supersampling the sphere surface to account for the variation in incoming illumination from different parts of the sphere surface.

The images below compare the results for a true spherical area light (top) with 64 light samples and the spherelight shader (bottom) using 64 shadow rays:





Light Intensity

The emitted light value is the product of the `intensity` and `lightcolor` parameters and an attenuation factor determined by the `falloff` parameter. The `falloff` parameter allows the light's intensity to diminish with distance from the light. A falloff value of 2 (which is physically accurate) reduces the incident light value by the square of the distance from the light:

```
  Cl = intensity * lightcolor / (Distance * Distance)
```

Traced Shadows

If the shadowname parameter is set to the special value `"raytrace"`, shadows are generated by tracing shadow rays. The `shadowsamples` parameter specifies the number of shadow rays to trace. The `shadowblur` parameter gives an angle (in radians) to distribute the rays around the point being shaded. Adjust the `shadowbias` parameter to prevent incorrect self-shadowing.

The `shadowgroups` parameter can be used to restrict shadow-casting to the specified list of groups. The shadow group can be overridden on a per-object basis by providing a user attribute:

```
Attribute "user" "string shadowgroups" ["my custom groups"]
```

The `shadowattribute` parameter gives the name of the user attribute to check for a custom shadow casting-group. By default all lights look for a custom shadow set in the same user attribute. By providing a different user attribute for each light, one can precisely specify for each object and each light which objects are tested for shadows.

If the shadow group has the special value "null", no shadow rays will be traced and the object will not receive shadows.

Shadow-mapped Shadows

Assign the file name of a previously generated shadow map to the `shadowname` parameter to generate shadow-mapped shadows. The `shadowblur` parameter gives a blur value as a fraction of the shadow map size. AIR ignores the `shadowsamples` parameter for shadow-mapped shadows, but other renderers may use it to control the quality of mapped shadows. Adjust the `shadowbias` parameter to prevent incorrect self-shadowing.

Output only variables

`color __shadow`	Shadow value
`color __unshadowed_Cl`	Emitted illumination without shadows

9.7.17 spotlight, spotlight_baked

spotlight shader

Parameter	Default Value	Range	Description
intensity	`1.0`		Light intensity
lightcolor	`1 1 1`		Light color
lightcolor_spd	`""`		Light color spectrum file
maxemission	`0`		If greater than 0, the maximum emitted light intensity
from	`0 0 0`		Source position of light
to	`0 0 1`		Target of light
coneangle	`0.52`	`0.0-1.5`	Cone angle
conedeltaangle	`0.087`	`0.0-1.5`	Angle for cone edge blur
beamdistribution	`2`	`0-2`	Exponent for cross-beam falloff
falloff	`2`	`0, 1, 2`	Exponent for falloff in light intensity with distance
shadowname	`""`		Name of shadow map or `raytrace` for traced shadows
shadowbias	`0.01`		Offset to prevent incorrect self-shadowing
shadowblur	`0.01`	`0.0-0.5`	Blur for shadows
shadowsamples	`1`	`1-256`	Number of rays for traced shadows
shadowcolor	`0 0 0`		Color of shadows
shadowgroups	`""`		Optional groups for ray-traced shadows
shadowattribute	`"user:shadowgroups"`		Per-object shadowgroups override attribute name
__category	`"lensflare"`		Light categories

__nondiffuse	0	0, 1	If 1 light is excluded from diffuse light
__nonspecular	0	0, 1	If 1 light will not produce highlights
__foglight	1	0, 1	If 1 light illuminates volumes
__channel	-1	0-9	Output channel

Description

The spotlight shader emits a cone of light with apex at the source point `from` and direction given by the vector `to-from`. The `coneangle` parameter gives the half-angle of the cone in radians.

Light Intensity

The emitted light value is the product of the `intensity` and `lightcolor` parameters and an attenuation factor determined by the `falloff` parameter. The `falloff` parameter allows the light's intensity to diminish with distance from the light. A falloff value of 2 (which is physically accurate) reduces the incident light value by the square of the distance from the light:

```
  Cl = intensity * lightcolor / (Distance * Distance)
```

Although a falloff of 2 is physically correct, it can produce very large intensity values at locations close to the light source. The `maxemission` parameter can be used to set the maximum emitted intensity regardless of distance from the light source.

Alternatively, a falloff value of 0 produces no attenuation with distance, which may be easier to control.

The emitted light also varies based on the angle between the shading location and the cone axis. The `beamdistribution` parameter gives an exponent that is applied to the cosine of the angle. The light intensity smoothly fades to 0 as the angle ranges from `coneangle-conedeltaangle` to `coneangle`.

Traced Shadows

If the shadowname parameter is set to the special value `"raytrace"`, shadows are generated by tracing shadow rays. The `shadowsamples` parameter specifies the number of shadow rays to trace. The `shadowblur` parameter gives an angle (in radians) to distribute the rays around the point being shaded. Adjust the `shadowbias` parameter to prevent incorrect self-shadowing.

The `shadowgroups` parameter can be used to restrict shadow-casting to the specified list of groups. The shadow group can be overridden on a per-object basis by providing a user attribute:

```
Attribute "user" "string shadowgroups" ["my custom groups"]
```

The `shadowattribute` parameter gives the name of the user attribute to check for a custom shadow casting-group. By default all lights look for a custom shadow set in the same user attribute. By providing a different user attribute for each light, one can precisely specify for each object and each light which objects are tested for shadows.

If the shadow group has the special value "null", no shadow rays will be traced and the object will not receive shadows.

Shadow-mapped Shadows

Assign the file name of a previously generated shadow map to the `shadowname` parameter to generate shadow-mapped shadows. The `shadowblur` parameter gives a blur value as a fraction of the shadow map size. AIR ignores the `shadowsamples` parameter for shadow-mapped shadows, but other renderers may use it to control the quality of mapped shadows. Adjust the `shadowbias` parameter to prevent incorrect self-shadowing.

Output only variables

`color __shadow`	Shadow value
`color __unshadowed_Cl`	Emitted illumination without shadows

Baked Shadows

AIR includes an extended version of the spotlight shader - spotlight_baked - with additional parameters for creating 3D shadow textures and using 2D or 3D baked shadows:

bakemap	`""`		Bake map file name
bakemode	`""`		Bake mode
bakemapblur	`0`		Blur for bake map query
bakemapfilterwidth	`1`		Filter width for bake map query
motionbound	0 0 0 0 0 0		bounding box for moving objects
motionspace	`"world"`	space	coordinate space of motionbound

See Baking Shadows and Illumination for more information.

9.7.18 spot_projector

spotlight shader that projects a texture map like a slide projector

Parameter	Default Value	Range	Description
intensity	1.0		Light intensity
lightcolor	1 1 1		Light color
lightcolor_spd	""		Light color spectrum file
maxemission	0		If greater than 0, the maximum emitted light intensity
from	0 0 0		Source position of light
to	0 0 1		Target of light
coneangle	0.52	0.0-1.5	Cone angle
conedeltaangle	0	0.0-1.5	Angle for cone edge blur
beamdistribution	0	0-2	Exponent for cross-beam falloff
falloff	0	0, 1, 2	Exponent for falloff in light intensity with distance
shadowname	""		Name of shadow map or `raytrace` for traced shadows
shadowbias	0.01		Offset to prevent incorrect self-shadowing
shadowblur	0.01	0.0-0.5	Blur for shadows
shadowsamples	1	1-256	Number of rays for traced shadows
shadowcolor	0 0 0		Color of shadows
shadowgroups	""		Optional groups for ray-traced shadows
shadowattribute	"user:shadowgroups"		Per-object shadowgroups override attribute name
__category	"lensflare"		Light categories
texturename	""		File name of texture map to project
textureblur	0.0		Blur for texture map
texturescale	1.0		Scale factor for texture
texturerotate	0		Rotation angle in degrees about the cone axis
textureonce	1	0 or 1	When set to 1, only one copy of the texture is displayed
textureup	0 1 0		Vertical axis used to orient the map projection
__nondiffuse	0	0, 1	If 1 light is excluded from diffuse light
__nonspecular	0	0, 1	If 1 light will not produce highlights
__foglight	1	0, 1	If 1 light illuminates volumes
__channel	-1	0-9	Output channel

Description

The spot_projector shader allows a basic spotlight to be used to project a texture map as a light source like a slide projector. Like the spotlight shader, the spot_projector shader emits a cone of light with apex at the source point `from` and direction given by the vector `to-from`. The `coneangle` parameter gives the half-angle of the cone in radians.

The texture map given by texturename parameter is mapped to the inside of the spotlight cone. The textureup parameter defines a vertical axis used to orient the map. The texture scale parameter can be used to resize the map, and the texturerotate parameter to rotate the map.

By default only one copy of the map is displayed. You can change that behavior by setting the textureonce parameter to 0 and converting the texture map to an Air texture file using the Air texture conversion tools mktex or mktexui, selecting the desired wrap mode in the process.

Light Intensity

The emitted light value is the product of the `intensity` and `lightcolor` parameters and an attenuation factor determined by the `falloff` parameter. The `falloff` parameter allows the light's intensity to diminish with distance from the light. A falloff value of 2 (which is physically accurate) reduces the incident light value by the square of the distance from the light:

```
  Cl = intensity * lightcolor / (Distance * Distance)
```

Although a falloff of 2 is physically correct, it can produce very large intensity values at locations close to the light source. The `maxemission` parameter can be used to set the maximum emitted intensity regardless of distance from the light source.

Alternatively, a falloff value of 0 produces no attenuation with distance, which may be easier to control.

The emitted light also varies based on the angle between the shading location and the cone axis. The `beamdistribution` parameter gives an exponent that is applied to the cosine of the angle. The light intensity smoothly fades to 0 as the angle ranges from `coneangle-conedeltaangle` to `coneangle`.

Traced Shadows

If the shadowname parameter is set to the special value `"raytrace"`, shadows are generated by tracing shadow rays. The `shadowsamples` parameter specifies the number of shadow rays to trace. The `shadowblur` parameter gives an angle (in radians) to distribute the rays around the point being shaded. Adjust the `shadowbias` parameter to prevent incorrect self-shadowing.

The `shadowgroups` parameter can be used to restrict shadow-casting to the specified list of groups. The shadow group can be overridden on a per-object basis by providing a user attribute:

```
Attribute "user" "string shadowgroups" ["my custom groups"]
```

The `shadowattribute` parameter gives the name of the user attribute to check for a custom shadow casting-group. By default all lights look for a custom shadow set in the same user attribute. By providing a different user attribute for each light, one can precisely specify for each object and each light which objects are tested for shadows.

If the shadow group has the special value "null", no shadow rays will be traced and the object will not receive shadows.

Shadow-mapped Shadows

Assign the file name of a previously generated shadow map to the `shadowname` parameter to generate shadow-mapped shadows. The `shadowblur` parameter gives a blur value as a fraction of the shadow map size. AIR ignores the `shadowsamples` parameter for shadow-mapped shadows, but other renderers may use it to control the quality of mapped shadows. Adjust the `shadowbias` parameter to prevent incorrect self-shadowing.

Output only variables

`color __shadow`	Shadow value
`color __unshadowed_Cl`	Emitted illumination without shadows

9.7.19 sunlight

sun color and position based on observer location, date/time, and atmospheric conditions

Parameter	Default Value	Range	Description
intensity	1.0		Light intensity
lightcolor	1 1 1		Light color
use_sun_color	1	0 or 1	whether to use the computed sun color
turbidity	5	2-9	atmospheric turbidity (integer 2-9)
date_month	4	1-12	date month
date_day	15	1-31	day of month
date_hour	12	0-24	time of day (24-hour clock)
latitude_longitude	47.45 -122.3		observer location
time_zone	-8		difference between local time and GMT
animate_hour	0		When non-zero, the end time for animation
animate_time_scale	1		Multiplier for the global time value
override_zenith	-1		When positive, sets sun zenith position
override_azimuth	0		Override for sun azimuth position
coord_sys_y_is_up	1	0 or 1	when 1, Y is up; otherwise Z is up
coord_sys_north_angle	0		North direction as an angle in degrees relative to the X axis
coord_sys_right_handed	1	0 or 1	Set to 1 for a right-handed coordinate system
shadowname	""		Name of shadow map or `raytrace` for traced shadows
shadowbias	0.01		Offset to prevent incorrect self-shadowing
shadowblur	0.01	0.0-0.5	Blur for shadows
shadowsamples	1	1-256	Number of rays for traced shadows
shadowcolor	0 0 0		Color of shadows

shadowgroups	""		Optional groups for ray-traced shadows
shadowattribute	"user:shadowgroups"		Per-object shadowgroups override attribute name
__nondiffuse	0	0, 1	If 1 light is excluded from diffuse light
__nonspecular	0	0, 1	If 1 light will not produce highlights
__foglight	1	0, 1	If 1 light illuminates volumes
__channel	-1	0-9	Output channel

Description

The sunlight shader computes the position and color for a sun-like light source based on the observer's position, the local data & time, and the atmospheric turbidity.

The turbidity parameter sets the relative portion of atmospheric scattering due to haze instead of molecules. Larger values model atmospheres with more large (dust) particles.

Output only variables

`color __shadow`	Shadow value
`color __unshadowed_Cl`	Emitted illumination without shadows

See Also

envPhysicalSky environment shader

9.7.20 texturedarealight

area light shader with texture map control over emitted light

Parameter	Default Value	Range	Description
intensity	1.0		Light intensity
lightcolor	1 1 1		Light color
lightcolor_spd	""		Light color spectrum file
maxemission	0		If greater than 0, the maximum emitted intensity (including falloff)
coneangle	1.57	0.0-1.5	Angle for emitted light range
falloff	2	0, 1, 2	Exponent for falloff in light intensity with distance
texturename	""		Texture map file name
sizexy	1 1		Texture map size
originxy	0 0		Texture map origin
shadowname	"raytrace"		Name of shadow map or `raytrace` for traced shadows
shadowbias	0.01		Offset to prevent incorrect self-shadowing
shadowcolor	0 0 0		Color of shadows

shadowgroups	`""`		Optional groups for ray-traced shadows
shadowattribute	`"user:shadowgroups"`		Per-object shadowgroups override attribute name
__nondiffuse	`0`	`0, 1`	If 1 light is excluded from diffuse light
__nonspecular	`0`	`0, 1`	If 1 light will not produce highlights
__foglight	`1`	`0, 1`	If 1 light illuminates volumes
__channel	`-1`	`0-9`	Output channel

Description

The texturedarealight shader allows the emitted illumination from an area light to be controlled with a texture map. The quality of the illumination and shadows is controlled with the light's nsamples attribute:

```
  Attribute "light" "integer nsamples" [n]
```

A large number of samples produces smoother results. When tweaking the number of samples, it usually makes sense to increase or decrease the amount by a factor of 2.

Light Intensity

The emitted light value is the product of the `intensity` and `lightcolor` parameters and an attenuation factor determined by the `falloff` parameter. The `falloff` parameter allows the light's intensity to diminish with distance from the light. A falloff value of 2 (which is physically accurate) reduces the incident light value by the square of the distance from the light:

```
  Cl = intensity * lightcolor / (Distance * Distance)
```

Although a falloff of 2 is physically correct, it can produce very large intensity values at locations close to the light source. The `maxemission` parameter can be used to set the maximum emitted intensity regardless of distance from the light source.

Alternatively, a falloff value of 0 produces no attenuation with distance, which may be easier to control.

Shadows

The `shadowgroups` parameter can be used to restrict shadow-casting to the specified list of groups. The shadow group can be overridden on a per-object basis by providing a user attribute:

```
Attribute "user" "string shadowgroups" ["my custom groups"]
```

The `shadowattribute` parameter gives the name of the user attribute to check for a custom shadow casting-group. By default all lights look for a custom shadow set in the same user attribute. By providing a different user attribute for each light, one can precisely specify for each object and each light which objects are tested for shadows.

If the shadow group has the special value "null", no shadow rays will be traced and the object will not receive shadows.

Output only variables

`color __shadow`	Shadow value
`color __unshadowed_Cl`	Emitted illumination without shadows

See Also

area lights
arealight shader
arealight2 shader

9.7.21 uberlight

multipurpose light shader

Parameter	Default Value	Range	Description
lighttype	"spot"	"spot" "omni" "arealight"	Light type
intensity	1.0		Light intensity
lightcolor	1 1 1		Light color
cuton	0.01		Min depth value that is illuminated
cutoff	1000000		Max depth value illuminated
nearedge	0.0		Transition region at near end of depth range
faredge	0.0		Transition region at far end of depth range
falloff	0		Exponent for falloff with distance
falloffdistance	1.0		Distance at which falloff begins
maxintensity	1.0		Maximum intensity
parallelrays	0		When 0 rays emanate from a single point; when 1 rays are parallel
shearx	0.0		
sheary	0.0		Shear applied to light cone
width	1.0		Width of the light opening at z=1
height	1.0		Height of the light opening at z=1
wedge	0.1		Width edge fuzz
hedge	0.1		Height edge fuzz
roundness	1.0	0.0-1.0	Shape of the light cross-section: 0=rectangle, 1=perfect ellipse
beamdistribution	0.0		Controls falloff in intensity with angle for a spot light
slidename	""		Texture map for filtering the light
noiseamp	0.0		Amplitude for noise applied to the light
noisefreq	4		Frequency for noise
noiseoffset	0 0 0		Offset for noise (useful for animation)

shadowname	`""`		Shadow map name or `raytrace` for traced shadows
shadowblur	`0.01`	`0.0-1.0`	Blur for shadow map
shadowbias	`0.01`		Bias for shadow map
shadowsamples	`1`	`1-256`	Rays for traced shadows
shadowcolor	`0 0 0`		Color for shadows
blockercoords	`""`		Coordinate system for blocker
blockerwidth	`1.0`		
blockerheight	`1.0`		Width and height of blocker (in x and y of blockercoords)
blockerwedge	`0.1`		Blocker width edge fuzz
blockerhedge	`0.1`		Blocker height edge fuzz
blockerround	`1.0`	`0.0-1.0`	Roundedness of blocker
__nondiffuse	`0`	`0, 1`	If 1 light is excluded from diffuse light
nonspecular	`0`	`0, 1`	Set to 1 to exclude from specular calculations
__foglight	`1`	`0, 1`	If 1 illuminates volumes
__channel	`-1`	`0-9`	Output channel

Description

The uberlight shader is a multipurpose light shader with many parameters. The uberlight shader included with AIR is a slightly modified version of the shader discussed by Ronen Barzel in chapter 14 of Advanced RenderMan : Creating CGI for Motion Pictures by Apodaca and Gritz.

Light Type

The lighttype parameter determines how the shader casts light:

`spot`	casts light towards the positive z axis
`omni`	casts light in all directions
`arealight`	casts light within a hemisphere on the front side of the light surface

Distance Shaping and Falloff

The `cuton` and `cutoff` parameters define the range of depths that are illuminated. Emitted intensity will be zero outside that range. `nearedge` and `faredge` give transition zones at either end of the range in which intensity smoothly diminishes to 0.

`falloff` is the exponent for decay with distance. A value of 0 produces no falloff; a value of 2 produces the square-law falloff. The `falloffdistance` parameter gives the distance at which falloff-with-distance begins. No distance-based attenuation occurs at locations closer than `falloffdistance`.

`maxintensity` sets the maximum intensity, preventing the emitted intensity from growing too large.

When the `parallelrays` parameter is 0, light rays diverge; when the parameter is 1 light rays remain parallel.

Cross-section Shaping

Uberlight casts light with a cross-sectional shape that can vary from a rectangle to a perfect ellipse.

`width` and `height` give the width and height of the light opening. `roundness` determines whether the shape is a rectangle (0) or ellipse (1) or a super-ellipse somewhere in between. `wedge` and `hedge` define a fuzzy region along the edges of the light.

`shearx` and `sheary` can be used to shear the light cone.

`beamdistribution` controls the falloff in intensity with angle from the central axis. A value of 0 results in constant intensity across the light. A value of 1 is physically correct for a spotlight. A value of 2 matches the default behavior the spotlight shader.

Cookie or Slide

The light can optionally project a texture map like a slide if a `slidename` is provided.

Fake Blocker Shadows

The `blocker` parameters can be used to define a superellipse that casts fake shadows. `blockercoords` gives a coordinate space for the blocker (which should be defined in the scene file using `CoordinateSystem`.) `Blockerwidth` and `blockerheight` give the dimensions of the blocker which is defined to lie on the x-y plane of the blocker coordinate system. `Blockerwedge`, `blockerhedge`, and `blockerround` define the shape of the superellipse in the same manner as the corresponding parameters for the light shape.

Traced Shadows

If the `shadowname` parameter is set to the special value `"raytrace"`, shadows are generated by tracing shadow rays. The `shadowsamples` parameter specifies the number of shadow rays to trace. The `shadowblur` parameter gives an angle (in radians) to distribute the rays around the point being shaded. Adjust the `shadowbias` parameter to prevent incorrect self-shadowing.

Shadow-mapped Shadows

Assign the file name of a previously generated shadow map to the `shadowname` parameter to generate shadow-mapped shadows. The `shadowblur` parameter gives a blur value as a fraction of the shadow map size. AIR ignores the `shadowsamples` parameter for shadow-mapped shadows, but other renderers may use it to control the quality of mapped shadows. Adjust the `shadowbias` parameter to prevent incorrect self-shadowing.

Output only variables

`color __shadow`	Shadow value
`color __unshadowed_Cl`	Emitted illumination without shadows

Credits

Original shader coded by Larry Gritz based on Ronen Barzel's paper "Lighting Controls for Computer Cinematography" (in Journal of Graphics Tools, vol. 2, no. 1: 1-20).

9.8 Procedures

AIR 8 introduces a new procedure shader type for generating new primitives on-demand at render time.

An instance of a procedure shader is created with the new RIB `Procedure` call:

`Procedure "`*`shadername`*`" [`*`minx maxx miny maxy minz maxz`*`]` *`parameter list`*

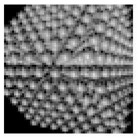
The array of 6 floats after the shader name gives a bounding box in object space encompassing all objects that will be created by the procedure shader. If and when AIR encounters the bounding box during rendering, the procedure shader will be executed, and the resulting objects inserted into the scene.

Writing Procedure Shaders

A procedure shader creates new geometry by emitting RIB commands using the new `ribprintf()` shading language command.

Source code for the procedure shaders included with AIR can be found in `$AIRHOME/vshaders/Procedures`

9.8.1 V3DArray

 creates a 3D rectilinear array of archive instances

Parameter	Default Value	Range	Description
Archive	""		RIB archive file name
Scale	1		Uniform scale applied to archive instances
CountXYZ	4 4 4		Array size in each dimension
OffsetX	1 0 0		Translation per increment in X index
OffsetY	0 1 0		Translation per increment in Y index
OffsetZ	0 0 1		Translation per increment in Z index
Origin	0 0 0		Offset for entire array

9.8.2 VTree

procedural tree generator

Introduction

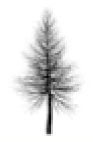
The VTree procedure shader "grows" a tree at render time. Many variations of the same basic tree "look" can be generated simply by changing the `Seed` value used to for generating random numbers during construction of the tree.

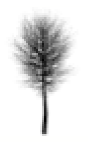
The tree model has four components: a trunk, major branches emanating from the trunk, twigs growing out of the branches, and leaves attached to the twigs.

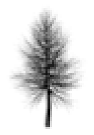
Tree Shape and Size

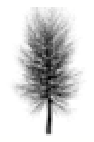
Parameter	Default Value	Range	Description
TreeShape	0	0-7	Index into tree shape table (see below)
TreeUpAttraction	0.5		Tendency of twigs and leaves to point up or down
TreeScale	1		Uniform scale applied to tree
TreeRotate	0	0-360	Rotation in degrees about vertical axis
TreeOffset	0 0 0		Center of trunk base
TreeUp	0 1 0		Up direction for tree
Seed	123	integer	Seed value for random variation

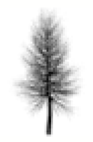
The overall shape of the tree is determined by the `TreeShape` parameter, which selects from among the following 8 basic tree shapes:

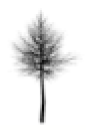

0 cone

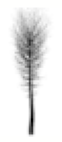

1 sphere

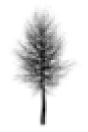

2 hemisphere


3 cylinder


4 tapered cylinder


5 flame


6 inverse cone


7 tend flame

There overall size of the tree is determined by the trunk height, given as an average `TrunkHeight` and an allowed variation `TrunkHeightVary`. The actual height is chosen within that range based on the random `Seed` value. Other size parameters for the trunk, branches, and twigs are given relative to the trunk height. The only other absolute size parameter is the `LeafSize` value.

Trunk

Parameter	Default Value	Range	Description
TrunkHeight	4		Average trunk height
TrunkHeightVary	0.5		Allowed variation in trunk height
TrunkWidthRatio	0.04		Ratio of trunk width at base to trunk height
TrunkWobble	0.1		Random variation in trunk orientation
TrunkBend	0 0		angle range for bent trunk (in degrees)
TrunkSurface	"VRubber"		surface shader for trunk
TrunkColor	.5 .5 .5		base surface color for trunk
TrunkColorMap	""		optional color map for trunk
TrunkBumpMap	""		optional bump map for trunk
TrunkBump	0.1		bump amplitude
TrunkTextureSize	1 1		texture map scale
TrunkCurveType	"tube"		trunk curve type

The trunk width at the base of the tree is computed by multiplying the trunk height by `TrunkWidthRatio`. Trunk width at the top is the produce of the base width and `TrunkTaper`.

`TrunkWobble` adds random variation along the tree trunk. `TrunkBend` gives a range of angles that cause the tree to bend as it grows.

If the `TrunkSurface` parameter is not null, it defines the surface shader for the trunk. The other shading-related parameters for the trunk are passed to the surface shader as follows:

Tree parameter	Surface parameter
`TrunkColorMap`	`ColorMapName`
`TrunkBumpMap`	`BumpMapName`
`TrunkBump`	`BumpMax`
`TrunkTextureSize`	`TextureSizeXY`

If the `TrunkSurface` parameter is empty, the trunk inherits the surface shader assigned to the VTree procedure primitive.

The trunk geometry is defined using a RIB Curves primitive. The curve definition includes values for the second standard texture coordinate (t) giving the length along the trunk. This information allows textures to be applied to trunks of different lengths without unnatural stretching. `TrunkCurveType` gives the curve type to use for rendering, defaulting to a tube. For trees that are far away, the polyline or ribbon curve types may be used. If the `TrunkCurveType` is set to the empty string, the trunk will not be rendered.

Branches

Parameter	Default Value	Range	Description
Branches	`50`		Total number of branches
BranchLengthRatio	`0.3 0.3`		Branch length relative to trunk height
BranchTilt	`-10 -110`		Branch angle relative to trunk
BranchBend	`10 -90`	`0-360`	Bend angle along branch
BranchBendBack	`0 0`		Optional bend angle for second half of branch
BranchTwist	`135 145`		Angle of rotation about trunk applied to successive branches
BranchStart	`0.4`		Relative position of first branch along trunk
BranchCurveType	`"tube"`		Branch curve type
BranchSurface	`""`		Optional branch surface shader

`Branches` gives the maximum number of branches for a tree of height `TrunkHeight`. For trees shorter or taller than the average height, the `Branches` value is scaled proportionally. Nominal branch length is given as a range of ratios relative to the tree height. The actual branch length at any position along the trunk is also strongly influenced by the tree shape. `BranchStart` gives the position of the first branch relative to the base of the trunk.

`BranchTilt` gives the angle (as a range) in degrees to tilt the branch away from the trunk. E.g., a `BranchTilt` of 90 would result in a branch perpendicular to the trunk. If the `BranchTilt` values are negative, a special mode is enabled that smoothly interpolates the tilt angle from tree base to tree tip instead of choosing the tilt angle for each branch randomly.

`BranchBend` defines an angle range used to bend each branch along its length. If the `BranchBendBack` range is defined, the `BranchBend` range is used for the first half of the branch, and the `BranchBendBack` range is used for the second half, allowing simple S-shaped branches to be generated.

`BranchTwist` is a range of angles used to rotate the starting position of each successive branch about the trunk axis.

The branch geometry is defined using a RIB Curves primitive. The curve definition includes values for the second standard texture coordinate (t) giving the length along the branch. This information allows textures to be applied to branches of different lengths without unnatural stretching. `BranchCurveType` gives the curve type to use for rendering, defaulting to a tube. For trees that are far away, the polyline or ribbon curve types may be used. If the `BranchCurveType` is set to the empty string, branches will not be rendered.

Twigs

Parameter	Default Value	Range	Description
Twigs	30		Max number of twigs per branch
TwigLengthRatio	0.6 0.6		Twig length relative to branch length
TwigTilt	35 55		Twig angle relative to branch
TwigBend	10 -90	0-360	Bend angle along twig
TwigTwist	135 145		Angle of rotation about branch applied to successive twigs
TwigStart	0		Relative position of first twig along branch
TwigCurveType	"ribbon"		Twig curve type
TwigSurface	""		Optional twig surface shader

`Twigs` sets the maximum number of twigs per branch. `TwigStart` is the relative position of the first twig along a branch.

Other parameters behave similarly to the equivalent branch parameters.

Leaves

Parameter	Default Value	Range	Description
Leaves	25		Max leaves per twig
LeafSize	0.06 0.08		Leaf size range (absolute values)
LeafAspectRatio	0.9		Leaf width relative to leaf length
LeafTilt	35 55		Leaf angle relative to twig axis
LeafTwist	70 80		Angle of rotation about twig for successive leaves
LeafColor	.18 .45 .04		Base leaf color
LeafColorVaryHSL	.2 .2 .2		Per leaf variation in hue, saturation, lightness
LeafSurface	"VClay"		Leaf surface shader
LeafMask	""		Optional opacity mask for leaf

`Leaves` sets the maximum leaves per twig. If `Leaves` is 0, no leaves will be rendered.

The leaf length varies randomly within the range given by `LeafSize`. By default leaves are represented as a polygonal leaf-like pointed oval. Use `LeafAspectRatio` to set the leaf width as a fraction of the leaf length. `LeafTilt` and `LeafTwist` have the same meaning as the equivalent twig and branch parameters.

`LeafColor` gives the average leaf color. The color for an individual leaf is randomly chosen based on the average color and the maximum variation in hue, saturation, and lightness specified in

LeafColorVaryHSL.

LeafSurface defines the surface shader to use for the leaves. LeafMask allows an opacity map to be used to define more complicated leaf shapes. If a leaf mask is provided, the leaves are rendered as square patches (using a Points primitive with point type patch). Note that using a mask requires that the leaf surface shader be evaluated during shadow ray evaluation, which can substantially slow down rendering of traced shadows. You may wish to use shadow maps or automap shadows with a leaf mask.

Level of Detail

The VTree shader incorporates a simple level of detail feature for leaves. Leaves closest to the camera are represented as a six-sided polygonal shape. Leaves around 1 pixel in size are rendered with as 4-sided polygons. Leaves smaller than a fraction of a pixel are represented as oriented point primitives. Each coarser level of detail uses approximately half the memory of the previous level. The TreeQuality parameter can be used to shift the transition points for the different levels of detail. A large TreeQuality value will force the more detailed representation to be used; a smaller value causes coarser levels of detail to be used at larger leaf sizes on screen.

Creating a Tree Archive

The VTree shader was created in the Vshade shader creation tool included with AIR. Vshade can also be used as a simple interface for creating a tree type as a rib archive. Here's how:

- Start Vshade.
- Open the VTree shader, located in the vshaders/Procedures directory of your AIR installation.
- Save a second copy of the VTree shader to a different directory so the original shader is not overwritten (keep the VTree name, just save to a different directory).
- You can now experiment with changing parameters under the Parms tab in the Vshade window.
- By previewing in AIR Show you can easily compare the effects of different parameter changes. To render previews to AIR Show instead of the small window in Vshade, check the Use Air Show item in the Run menu.
- AIR includes a few sample trees in the archives directory. To load a tree archive (which is just a shader definition with a set of parameter values), choose Import Parameter Values from the File menu and pick one of the Tree* rib files in the archives directory of the AIR installation.
- To save a tree for future use, select Export Parameter Values in the File menu to create a RIB archive with a complete shader definition. Be sure to set the Procedure bound appropriately on the View page before exporting.

Using a Tree Archive

A tree archive created with Vshade can be included in a scene using the standard ReadArchive command. E.g.,

```
Translate 1 0 1  # translate the tree into position
ReadArchive "Tree_Quaking_Aspen.rib"
```

The following user attributes can be used to customize the tree's appearance without modifying the tree archive:

```
Attribute "user" "float treeseed" [123]
```

Sets the tree Seed parameter used for generating random numbers.

```
Attribute "user" "float treeheight" [6]
```

Sets the tree height to a specific value. The height should be in the same units as the tree archive.

```
Attribute "user" "float treequality" [1]
```

Sets the `TreeQuality` parameter.

```
Attribute "user" "color leafcolor" [0.2 0.3 0.1]
```

Sets the average leaf color.

AIR also includes a VInstanceTree instancer shader that creates instances of tree archives at point locations in a particle system.

Optimization

Here are some tips for reducing the memory requirements and complexity of the trees generated by the VTree shader:

- For trees that are far from the camera, omit rendering the twigs or branches by setting `TwigCurveType` or `BranchCurveType` to the empty string. For nearer trees that are still somewhat far away, set the `BranchCurveType` to ribbon or polyline instead of tube.
- Setting the `LeafColorVaryHSL` values to all 0 allows all leaves to share the same color, saving the storage used by a per-leaf color value.
- Setting the `LeafSize` min and max to the same value saves a per-leaf width value when leaves are rendered as points primitives (the lowest level of detail representation).
- For distant trees using fewer leaves that are larger may produce a similar appearance at lower memory cost han many smaller leaves.

References

The VTree shader was inspired by "Creation and Rendering of Realistic Trees" by Jason Weber and Joseph Penn. The shader lacks many features mentioned in the article including:

- Trunk splits
- Branch splitting
- More than 3 levels of branches/leaves
- Automatic detail reduction

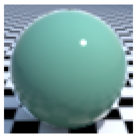
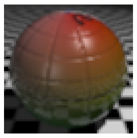
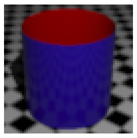
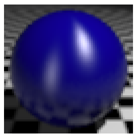
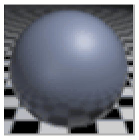
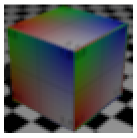
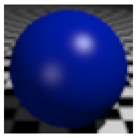
See Also:

VInstanceTree instancer shader

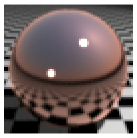
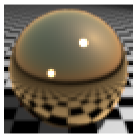
9.9 Surfaces

Surface shaders compute the opacity and color of a surface.

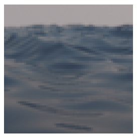
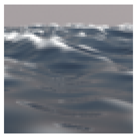
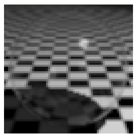
Plastics

	SimplePlastic	physically plaustible plastic material
	VPlastic	general plastic with reflections and texture maps for color, specular, opacity, and bump
	V2SidedPlastic	reflective, textured plastic with separate settings for the front and back side of a surface
	VAnimatedMap	plastic shader with animated texture map
	VBrushedPlastic	brushed or anisotropic plastic
	VBlinn	plastic with Blinn specular model
	VDecal2D	reflective, textured plastic with decals positioned using standard texture coordinates
	VDecal3D	reflective, textured plastic with projected decals
	paintedplastic	simple plastic with texture map (no reflections)
	plastic	basic plastic without reflections or texture maps

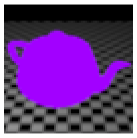
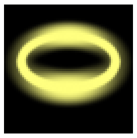
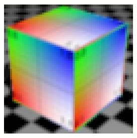
Metals

	SimpleMetal	simple physically plausible metal shader
	VMetal	metal with reflections and optional color map
	VBrushedMetal	brushed or anisotropic metal
	VPhysicalMetal	physically plausible metal with color map
	VRustyMetal	metal with a rust pattern

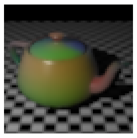
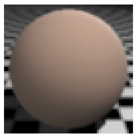
Glass and Water

	OceanSurface	ocean water surface (no waves)
	OceanSurfaceWithFoam	ocean water with foam
	VGlass	glass shader for thick, solid glass with refraction effects
	VThinGlass	shader for thin glass without refraction

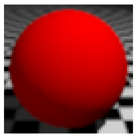
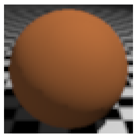
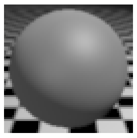
Constant

	constant	simple constant-colored shader
	Emitter	makes a surface behave like an area light when used with indirect diffuse illumination
	Glow	constant color with opacity falloff based on facing ratio
	VTexturedConstant	constant-colored shader with texture map
	VShadowedConstant	constant-colored shader with shadows
	VShadowedTransparent	transparent surface with shadow used for opacity

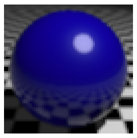
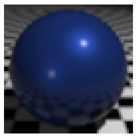
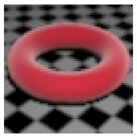
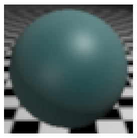
Subsurface Scattering

	VTranslucent	basic subsurface scattering shader
	VSkin	skin shader using surface scattering
	VTranslucentMarble	marble with subsurface scattering

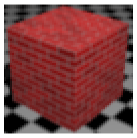
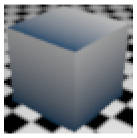
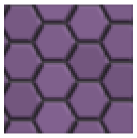
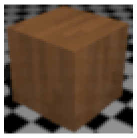
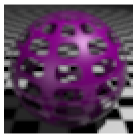
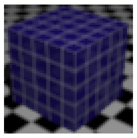
Rough Materials

	matte	basic matte/flat-shaded material
	VClay	rough diffuse surface with optional color map
	VConcrete	concrete with grooves
	VRubber	rubber

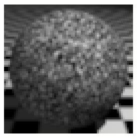
Shiny Materials

	VCeramic	ceramic with reflections and optional texture maps
	VCarPaint	metallic car paint with clear finish
	Velvet	velvet
	VFabric	shader for fabric or cloth

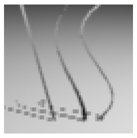
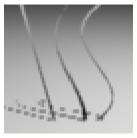
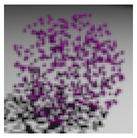
2D Patterns

	VBrick2D	bricks
	VGradient	color gradient
	VGrid2D	grid of 2D lines
	VHexTile	hexagonal ceramic tile
	VPlanks	wooden planks
	VScreen	metallic or plastic screen with round or square holes
	VShinyTile2D	ceramic tiles
	VWeave	basket-weave pattern

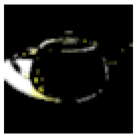
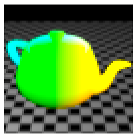
3D Patterns

	VMarble	simple veined marble
	VGranite	granite
	VWood	solid wood

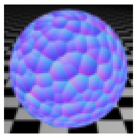
Curve and Point Shaders

	VHair	shader to make curves look like hair
	VFur	shader to make curves look like fur (for Shave and a Haircut)
	VDashes	dashed curves
	VShadeCarpet	surface shader for carpet created with the instCarpet instancer shader
	particle	shader for particles and volumetric primitives

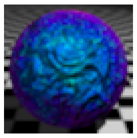
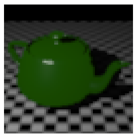
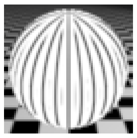
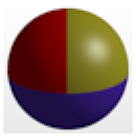
Passes

	CausticPass	emits caustics only at the current location
	ClownPass	assigns a unique color based on toon id
	depthpass	z depth pass
	IndirectPass	emits indirect diffuse illumination only
	massive_occlusion pass	occlusion pass shader for use with Massive
	MotionPass	surface shader for recording a motion vector
	occlusionpass	computes an ambient occlusion pass
	ReelSmartMotion	computes a motion blur vector for use with the Reel Smart Motion Blur plug-ins
	shadowpass	emits shadow values
	ShowPosition	emits the current shading location X,Y,Z values
	UseBackground	emits reflection and shadow values for compositing over a background image

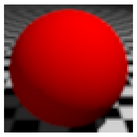
Baking

	Bake3d	baking illumination to a 3D texture
	BakedSurface	surface using baked illumination maps
	BakeTangentNormalMap	bake displacement to a tangent space normal map

Additional Shaders

	DarkTreeSurface	shader for using Dark Tree shaders
	FakeCarpet	turns a rectangle into a carpet
	VSketchOutline	uneven outlines for illustration
	VToon	shader for cartoon-style rendering
	VLines	shader for line-drawing illustration
	VLayeredMaterials	shader with multiple materials per surface

9.9.1 Bake3d

 shader for baking diffuse illumination to a 3D texture

Illumination Model: diffuse

Parameter	Default Value	Range	Description
Bake	`1`	`0 or 1`	Switch to enable baking (1) or reading (0)
BakeMap	`""`		Name of 3D texture to read or write
BakeChannels	`"diffuse_shadowed,diffuse_unshadowed"`	`Cs,diffuse_shadowed,diffuse_unshadowed`	Channels to bake
BakeRadius	`0.0`		Max distance between shading samples if baking with collect()
QueryBlur	`0.0`		Blur distance to add to query region
QueryFilterWidth	`1.0`		Multipler for sample region size
UseNormal	`1`	`0 or 1`	Whether to use the surface normal when baking or reading the map
FaceForward	`1`	`0 or 1`	If 1, ensure that the shading or query normal is forward-facing

Description

The Bake3d surface shader shows how to bake diffuse illumination to a 3D texture and reuse baked data for accelerated rendering.

Baking

The Bake3d shader creates a 3D texture map named `BakeMap` when the `Bake` parameter is set to `1`. The point map format is based on the file name extension. In most cases the Air Point Map format (extension `.apm`) is the most efficient format for storing and accessing 3D data.

Bake3d supports two methods of sampling a surface during baking:

Image-based sampling

When the `BakeRadius` parameter is `0` (the default), the shader simply records every shading sample to the bake map as the image is rendered. Only those parts of the object visible in the image will have shading samples recorded. Surfaces that are nearly parallel to the camera direction will be more sparsely sampled than those that are nearly perpendicular to the camera.

Multiple threads can write to the same point cloud in this mode. The baked point cloud cannot be queried during the same render pass.

This mode works well if the 3D point map will be used from approximately the same camera position. For greater coverage of a scene and better sample distribution, 3D point clouds can be generated from multiple points of view and combined with the AIR Point Tool (airpt).

Sampling using collect()

If a positive `BakeRadius` value is provided, the Bake3d shader uses the collect() function to sample the surface uniformly using `BakeRadius` as the maximum distance between samples. The call to collect samples all objects in the same point set as defined by

```
Attribute "pointset" "string handle" "name"
```

This attribute must be set when using this bake mode.

Only 1 thread will be used when sampling the surface. The baked map can be queried during the same render pass in which it is created.

The `BakeChannels` parameter determines which of the supported bake channels are actually written to 3D point cloud (any or all may be baked):

Cs	diffuse illumination multipled by object color
diffuse_shadowed	diffuse illumination with shadows
diffuse_unshadowed	diffuse illumination without shadows

Reading a baked map

When the `Bake` parameter is 0, the shader uses the baked map to provide illumination values during rendering instead of calling the diffuse illumination function.

Sampling of the map is controlled by the `QueryBlur` and `QueryFilterWidth` parameters. By default the sample region is equal to the area of the current shading sample. The `QueryBlur` parameter can be used to specifier a larger sample region for a blurrier result.

See Also:

3D Textures
collect()

9.9.2 BakedSurface

surface shader using baked maps for diffuse illumination

Illumination model: plastic with Blinn specular

Parameter	Default Value	Range	Description
MapAttributeName	`"user:bakedmap"`		user attribute holding the baked map name
MapFileName	`""`		Direct file name for baked map to use instead of querying the above user attribute
ChannelWithUnshadowed	`-1`		First channel for unshadowed diffuse values
ChannelWithIndirect	`-1`		First channel with indirect diffuse values
Diffuse	`1`	`0.0-1.0`	Diffuse reflectivity
Specular	`0`	`0.0-1.0`	Specular reflectivity
SpecularEccentricty	`0.3`		Specular size (Blinn eccentricity)
SpecularRolloff	`0.7`		Blinn rolloff parameter
SpecularColor	`1 1 1`		Specular color
SpecularApplyShadow	`0`		Whether to apply the diffuse shadow value to the specular lighting
Reflection	`0.0`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup

Description

The BakedSurface surface shader provides a standard plastic material appearance with diffuse values taken from baked texture maps.

By default the shader queries the user attribute specified in `MapAttributeName` for the baked map file name. If a `MapFileName` value is provided, that will be used as the bake map name.

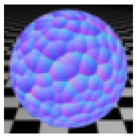
At a minimum each baked map should have 3 channels with the total diffuse lighting component (illumination and surface color). If the baked map also has 3 channels with the unshadowed diffuse value, the shader will use that information to compute proper `__diffuse_unshadowed` and `__shadow` output values. The shader needs to be told where the unshadowed values are stored in the baked maps by setting the `ChannelWithUnshadowed` value to the index of the first channel of the baked values in the map.

Similarly, if the baked map has indirect illum values (again light plus surface color), set the `ChannelWithIndirect` value to the index of the first channel.

Specular highlights using a Blinn BRDF can be added to the basic diffuse illumination. If the SpecularApplyShadow value is set to 1, the diffuse shadow value will also be applied to the specular component, which may allow final rendering to be done w/o shadow computations in the light. (Assuming a pretty simple lighting setup).

Reflections can also be added by setting Reflection to a positive value.

9.9.3 BakeTangentNormalMap


surface shader for baking displacement to a tangent space normal map

Parameter	Default Value	Range	Description
RightHand	`1`	`0, 1`	whether the current coordinate system is right-handed

Description

Use this surface shader with BakeAIR to produce a tangent-space normal map from a displacement shader. This shader works best when the displacement shader provides the undisplaced normal vector in a __undisplaced_N output variable. If the displacement shader does not provide such an output variable, the BakeTangentNormalMap shader computes the undisplaced normal using the tangent vectors and accounting for the handedness of the coordinate system.

9.9.4 CausticPass

surface shader returning photon-based caustics at the current shading location.

Parameter	Default Value	Range	Description
Intensity	`1.0`		Multiplier for recorded value

Description

The CausticPass shader returns the caustics visible at the current shading location if any.

This shader may be useful for baking caustics for use with the caustic_baked light shader.

See Also

caustics
caustic_baked light shader

9.9.5 ClownPass

surface shader that assigns a random color to each object based on its toon id attribute

Parameter	Default Value	Range	Description
Seed	`0`		Seed for random number generator

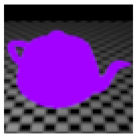
Description

The ClownPass surface shader assigns a unique color to each object or facet based on the toon id attribute used for outline rendering. The clown pass image can be used to as a source for selection masks in a paint program.

See Also

Outlines for Toon Rendering and Illustration

9.9.6 constant


constant-colored surface

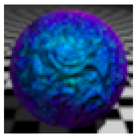
Description

The constant surface shader simply shades an object a constant color by applying the object's color attribute modulated by the object's opacity.

See Also

VTexturedConstant shader
VShadowedConstant shader

9.9.7 DarkTreeSurface


interface for Dark Tree shaders

Illumination model: custom

Features: reflections

Parameter	Default Value	Range	Description
DarkTreeShader	`""`		DarkTree shader file
Diffuse	`1.0`	`0.0-1.0`	Diffuse multiplier
Specular	`1.0`	`0.0-1.0`	Specular multiplier
Reflection	`1.0`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
Incandescence	`1`		Self-illumination multiplier

Use2DCoordinates	0	0 or 1	Set to 1 for a 2D pattern using an object's standard texture coordinates
Texture2DSize	1.0 1.0		Texture width and height for 2D pattern
Texture2DOrigin	0.0 0.0		Location of top, left corner of texture
Texture2DAngle	0.0		Rotation angle in degrees for 2D pattern
Pattern3DSize	1.0		Size of 3D pattern
Pattern3DOrigin	0 0 0		Offset for 3D pattern
Pattern3DSpace	"shader"	space	Coordinate space for pattern
BumpScale	1		Multiplier for bump effect
TimeScale	1.0		Multiplier for global time value
ColorTweak1Name	""		Tweak name
ColorTweak1	1.0 1.0 1.0		Tweak value
ColorTweak2Name	""		Tweak name
ColorTweak2	1.0 1.0 1.0		Tweak value
ColorTweak3Name	""		Tweak name
ColorTweak3	1.0 1.0 1.0		Tweak value
FloatTweak1Name	""		Tweak name
FloatTweak1	0.5		Tweak value
FloatTweak2Name	""		Tweak name
FloatTweak2	0.5		Tweak value
FloatTweak3Name	""		Tweak name
FloatTweak3	0.5		Tweak value
FloatTweak4Name	""		Tweak name
FloatTweak4	0.5		Tweak value

Description

This shader allows any DarkTree shader to be used as a surface shader for AIR.

To use DarkTree shaders, you must have the RMSimbiont from Darkling Simulations installed. The simbiont is available at `www.darksim.com`. The RMSimbiont file should be placed in a directory that is in your shader search path. The `shaders` directory of the AIR home directory is a good place.

To use a DarkTree shader, enter the name of the DarkTree shader in the DarkTreeShader parameter. The DarkTree simbiont will look for the DarkTree shader in the search pathes defined with the `SHADERS` and `SIMBIONT_RM_SHADERS` environment variables.

The DarkTreeSurface shader provides a number of tweak parameters that can be used to alter the tweaks defined for a particular shader. Some user interfaces, such as MatEd, will automatically fill in the list of tweaks for a particular shader. For those that do not, the tweak names and values can be entered by hand.

2D or 3D Shading

The DarkTree shading pattern can be calculated in 2D or 3D. If the Use2DCoordinates parameter is set to 1, the base shading location will be (s,t,0), where s and t are the standard 2D texture coordinates for a primitive.

Optimization

If you know that a particular parameter is not used by a given DarkTree shader, setting that parameter to 0 will speed up rendering. For example, if a shader does not perform bump mapping, setting the BumpScale parameter to 0 will avoid an unnecessary call to the RMSimbiont.

Set any of the following parameters that are not used to 0 to accelerate rendering: `Reflection`, `Incandescence`, `BumpScale`.

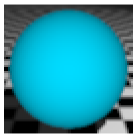
Animation

DarkTreeSurface uses the shading language time value (set with the `Shutter` RIB call) multiplied by the `TimeScale` parameter as the frame number input for DarkTree shaders. By varying the time value in the `Shutter` call, animated DarkTree shaders can be queried at different times.

See Also

DarkTree overview
DarkTreeDisplacement shader

9.9.8 defaultsurface


default surface shader

Description

The defaultsurface shader is used whenever no surface shader is assigned to an object.

The default implementation colors the surface as though it were lit by a distant light shining from the view direction. This shading allows objects to be visualized even in scenes without lights.

9.9.9 depthpass

simple surface shader for rendering a depth pass

Parameter	Default Value	Range	Description
mindistance	`0.0`		Start distance for interpolation
maxdistance	`1.0`		End distance for interpolation
invert	`0`	`0 or 1`	Invert output color

Description

This shader emits a grey-scale color whose intensity corresponds to the distance of the shading location to the camera. Points closer than `mindistance` to the camera are colored black. Points farther away than `maxdistance` are colored white. Points that lie between `mindistance` and `maxdistance` are assigned a grey color by linearly interpolating based on the position between `mindistance` and `maxdistance`.

If the `invert` parameter is set to 1, the output color is inverted (i.e., white becomes black, and black becomes white).

9.9.10 Emitter

surface shader for treating an object as a light emitter

Parameter	Default Value	Range	Description
IndirectIntensity	`1`		intensity for indirect rays
CameraIntensity	`1`		intensity for camera rays
ReflectionIntensity	`1`		intensity for reflection rays
Width	`1`		nominal width of surface
Length	`1`		nominal height of surface
TextureName	`""`		Color map name

Description

The Emitter surface shader emulates an area light by adjusting the intensity of the emitted incandescent color based on the size of the surface. The Width and Length parameters should be set so their product is the total surface area of the associated primitive.

See Also

cardlight light shader
arealight light shader

9.9.11 FakeCarpet

surface shader for turning a rectangle into a simple carpet

Parameter	Default	Range	Description
SizeX	10		Width of box in the X direction
SizeY	10		Width of box in the Y direction
StrandCount	20		Strand count per unit length
StrandLength	0.2		Strand length
StrandWidthFraction	0.1		Strand width as a fraction of strand length
StrandTiltAngle	90		Average angle in degrees between a strand tip and the vertical axis
StrandTiltVary	100		Variation in tilt angle
StrandTiltFrequency	40		Frequency of variation relative to the patch as a whole
StrandPolarAngle	0		Base rotation about the vertical axis
StrandPolarVary	360		Variation in polar angle
StrandPolarFrequency	30		Frequency of polar variation relative to the carpet rectangle
StrandStiffness	0.2		Stiffness affects the curve shape of each strand
StrandSelfShadowMax	0.76		Self-shadowing at the base of each strand
StrandColorVary	0.2		Variation in color of each strand
StrandBaseVary	0.2		Variation in the base position of each strand
FrontScatter	0.9		Forward scattering light multiplier
BackScatter	0.3		Backward scattering multiplier
ScatterPower	0.5		Exponent applied to the variation in scattering due to angle
ColorMapName	""		Optional color map
BaseColor	.76 .76 .65		Color for the rectangle patch under the strands
BaseSelfShadow	0.85		Shadowing the rectangle base due to the carpet strands
SampleGridSize	2		Grid size for internal ray tracing
IndirectDiffuse	0.5		Multiplier for the diffuse result returned when the shader is queried by an indirect ray

TestForIntersections	`1`	`0 or 1`	Whether to trace rays to test for objects intersecting the carpet
TraceBias	`0.02`		Offset used when tracing for intersections
TraceMargin	`0.05`		Border region with no strands
TraceTip	`1`	`0 or 1`	Whether to trace a sphere at the tip of each strand
TraceJoints	`1`	`0 or 1`	Whether to trace a sphere at the joint between each curve segment
ShadingSpace	`"world"`	spaces	Coordinate space for shading

Air 13 introduces a new FakeCarpet surface shader that turns a simple rectangle into a patch of fuzzy carpet. The carpet effect is produced by ray tracing a virtual field of carpet fibers inside the surface shader; no additional geometry is created in the scene.

Getting Started

Here's how to add carpet to a room model:

- Create a four-sided polygon and position it slightly above the floor. The polygon should represent the top of the area to be covered by the carpet. Make the polygon invisible to shadow rays.
- Assign the FakeCarpet surface shader. You will probably want to disable the material preview window in your application when tuning this shader because the preview render can take a long time.
- Set the SizeX and SizeY parameters to the width and length of the carpet rectangle. SizeX gives the size along the first texture coordinate axis, SizeY the size along the second texture coordinate axis.
- Set StrandLength to the length of the carpet fibers, which should be less than or equal to the height of the polygon above the floor.

That completes the the required steps to use the FakeCarpet shader. You should then be able to render a test image of your scene and see some carpet.

Carpet Shape

The carpet is modeled as a grid of carpet fibers. The geometric model of the fibers can be tuned using the following parameters:

StrandCount, StrandWidthFraction

The overall density of the fibers is set with StrandCount, which gives the number of fibers per unit length. The diameter of a fiber is given by the StrandWidthFraction parameter as a fraction of the fiber length.

StrandTiltAngle, StrandTiltVary, StrandTiltFrequency, StrandStiffness
StrandPolarAngle, StrandPolarVary, StrandPolarFrequency

Individual fibers are represented as curved tubes. StrandTiltAngle gives the average angle (in degrees) between the vertical direction and the tip of a fiber. The tilt angle can be varied for each fiber using StrandTiltVary and StrandTiltFrequency. StrandTiltFrequency is relative to the entire patch of carpet. StrandStiffness controls how each fiber bends. Higher values push the curve more toward the tip of each tube.

The direction of tilt can be changed using the StrandPolar parameters, which control a rotation in degrees about the vertical axis.

StrandSegments, TraceTip, TraceJoint

Each fiber is represented as a chain of tubular line segments. StrandSegments gives the number of segments per fiber. Each segment can be capped with a sphere. TraceTip controls whether a sphere is traced at the tip of each fiber. TraceJoints controls whether spheres are traced at the joints between segments in a fiber.

Carpet Shading

The carpet is shaded using an empirical model that assumes some light is reflected from the front of each fiber and some is transmitted.

FrontScatter, BackScatter, ScatterPower

FrontScatter gives the fraction of light reflected from the front of a fiber. BackScatter gives the amount transmitted from behind. Both are modulated based on the angle between the shading normal and the view direction such that more light is transmitted along the fiber edges and less in the middle. The ScatterPower parameter is an exponent applied to the angle cosine used for this edge-based variation.

SelfShadowMax, ColorMapName, BaseColor, IndirectDiffuse

SelfShadowMax gives the maximum self-shadowing of the carpet (the shadowing effect of neighboring fibers). Self-shadowing is greatest at the base of each fiber and smoothly attenuates to zero at the tip.

ColorMapName allows a color map to be applied to the fibers.

BaseColor gives the color of the "fabric" under the fibers, which is visible where fibers are missing.

IndirectDiffuse is a simple multiplier for the diffuse result returned for indirect illumination. The shader does not trace fibers for indirect rays.

Carpet Trace Controls

The FakeCarpet shader performs ray tracing to determine which fibers are visible at each shading location. The following parameters control this internal ray tracing process:

SampleGridSize

Because the carpet fibers will typically be much smaller than a pixel, the shader provides an option to trace multiple rays to produce a smoother, more accurate result. SampleGridSize defines the size of an NxN grid of rays to trace at the current shading location. The grid size is automatically reduced to 1 for reflection rays and IPR (interactive preview) rendering.

TraceMargin

The FakeCarpet shader treats the rectangular polygon as a sort of window into a virtual grid of 3D fibers. There can be artifacts along the edges if this "window" clips the carpet. To help prevent those artifacts, the TraceMargin parameter strips away a fraction of the fibers along the carpet border. Larger values remove more rows of fibers.

TestForIntersections, TraceBias

TestForIntersections determines whether the shader checks for objects intersecting the carpet (such as a chair leg). If there are no objects on the carpet, set this parameter to 0 to save some time and potentially avoid some artifacts. The TraceBias parameter gives an offset used to avoid incorrect intersections in this test.

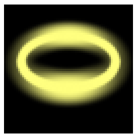
Final Remarks

The FakeCarpet surface is a fairly complicated shader, and you will likely need to experiment with the parameters to reproduce the appearance of a particular type of carpet. An IPR session (utilizing TweakAir) can be used to interactively view the effects of parameter changes.

See Also

instCarpet instancer shader

9.9.12 Glow


surface shader with constant color and opacity falloff based on facing ratio (angle between viewer and surface normal)

Parameter	Default Value	Range	Description
Intensity	`1.0`		Overall emitted intensity multiplier
AlphaFalloff	`4.0`		Exponent applied to facing ratio for alpha/opacity interpolation
ColorFalloff	`1.0`		Exponent applied to facing ratio for color interpolation
EdgeAlpha	`0`		Alpha/opacity value at edges
EdgeColor	`1 1 1`		Color at edge if EdgeColorEnable is 1
EdgeColorEnable	0	0 or 1	When 1, interpolate color using ColorFalloff and EdgeColor

Description

The Glow surface shader can be used to simulate illumination emitted from a glowing object. The Glow shader is assigned to an object modeled to represent the extent of glow emitted by the glowing object. Typically the Glow object will encompass the glowing object. E.g., in the thumbnail image above, the Glow shader is assigned to a second torus with a larger minor radius than the glowing torus.

The Glow shader interpolates opacity based on the facing ratio, which is the cosine of the angle between the viewing direction and the surface normal. The facing ratio ranges from 0 when looking at a surface on-edge to 1 when looking directly at a surface. The `AlphaFalloff` parameter controls how quickly the opacity value changes with angle.

When `EdgeColorEnable` is set 1, color is also interpolated based on the facing ratio.

9.9.13 IndirectPass

surface shader recording indirect diffuse illumination

Parameter	Default Value	Range	Description
Intensity	`1.0`		Multiplier for recorded value
Samples	`-1`	`0.0-1.0`	If greater than 0, the number of rays to trace for each indirect sample

Description

This surface shader stores the indirect diffuse illumination result at the current shading location as returned by the indirectdiffuse() function.

Note that this shader REQUIRES use of an indirect shade mode that does not evaluate the surface shader (mode `matte` or mode `constant`).

If the `Samples` value is less than 1, the number of traced rays is taken from the indirect nsamples attribute value for the object.

This shader may be useful when "baking" indirect illumination for use with AIR's indirect_baked light shader.

See Also

Indirect lighting
indirect_baked light shader
`indirectdiffuse()` shading language function

9.9.14 massive_occlusionpass

surface shader for rendering an ambient occlusion pass with Massive

Parameter	Default Value	Range	Description
coneangle	`1.57`		Half-angle for cone of directions to sample
invert	`0`		When set to 1, the output value is inverted
samples	`256`		rays to trace to estimate occlusion
maxhitdist	`500`		max distance to search for occluding objects
shadowbias	`0.1`		offset to prevent incorrect self-shadowing

Description

The massive_occlusionpass surface shader sets the output color to the ambient occlusion value at the surface location. The average unoccluded direction is emitted in the `Nunoccl` output variable.

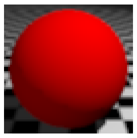
Output only variables

`normal Nunoccl`	Average unoccluded direction

See Also

Ambient occlusion
envlight light shader
Ambient occlusion in Massive

9.9.15 matte


standard matte surface shader

Illumination Model: matte

Parameter	Default Value	Range	Description
Ka	`1.0`	`0.0-1.0`	Ambient reflectivity
Kd	`1.0`	`0.0-1.0`	Diffuse reflectivity

Description

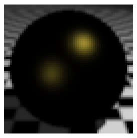
The shader simulates a flat or matte shaded surface. The surface color is taken from the object's color attribute.

For a matte surface with a texture map, use the VClay shader with the `DiffuseRoughness` parameter set to 0.

See Also

VClay surface shader

9.9.16 metal


simple metal shader with no reflections

Illumination Model: metal

Parameter	Default Value	Range	Description
Ka	`1.0`	`0.0-1.0`	Ambient reflectivity
Kd	`0.0`	`0.0-1.0`	Diffuse reflectivity
Ks	`1.0`	`0.0-1.0`	Specular reflectivity
roughness	`0.1`	`0.01-1.0`	Specular roughness

Description

The very simple metal shader with no reflections is included for compabitility with the RenderMan® Interface Specification. For metal with reflections, use the VMetal shader instead.

See Also

VMetal shader

9.9.17 MotionPass

surface shader for recording a motion vector

Parameter	Default Value	Range	Description
MotionSpace	`"camera"`	space	coordinate space for output vector
OpacityMap	`""`		optional opacity map
BinaryTest	`0`	`0, 1`	whether to convert the opacity map result to binary (0 or 1) based on the BinaryThreshold
BinaryThreshold	`0.5`		threshold for converting opacity map result to 0 or 1

Description

This shader emits the motion vector at the current shading location in the specified coordinate space.

9.9.18 occlusionpass

surface shader for rendering an ambient occlusion pass

Parameter	Default Value	Range	Description
coneangle	`1.57`		Half-angle for cone of directions to sample
invert	`0`		When set to 1, the output value is inverted
mapname	`"raytrace"`		`raytrace` or name of occlusion map
mapblur	`0.01`	`0.0-1.0`	Blur for occlusion map
mapbias	`0.01`		Bias for occlusion map
maxsolidangle	`0.05`		Max angle in radians for grouping of points in point-based occlusion

Description

The occlusionpass surface shader sets the output color to the ambient occlusion value at the surface location. The average unoccluded direction is emitted in the `Nunoccl` output variable.

When the `mapname` parameter is set to "raytrace", ray tracing is used to compute the ambient occlusion at each point. Otherwise, `mapname` is treated as the file name for an occlusion map with occlusion information.

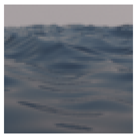
Output only variables

`normal Nunoccl`	Average unoccluded direction

See Also

Ambient occlusion
envlight light shader
massive_occlusionpass surface shader

9.9.19 OceanSurface

 surface shader for ocean water (no wave simulation is included in this shader)

Illumination Model: custom

Parameter	Default Value	Range	Description
Diffuse	0	0.0-1.0	Diffuse reflectivity
DiffuseColor	.44 .47 .6		Diffuse color
DeepSeaIntensity	.1		Deep sea color multiplier
DeepSeaColor	.21 .29 .47		Deep sea color
Specular	0.5		Specular intensity
SpecularRoughness	0.1		Specular roughness
Reflection	0.9		Reflection multiplier
ReflectionSamples	1		Number of reflection rays to trace
ReflectionBlur	0		Blur angle for reflection (in radians)
Transmission	0		Transmission multiplier
TransmissionMaxHitDist	1000		Max distance to trace transmission rays
TransmissionSamples	1		Rays to trace for refraction
TransmissionFalloff	0		Exponent for transmission decay with distance
TransmissionBlur	0		Blur angle for transmission rays
TransmissionNearColor	.3 .5 .4		Color for sea close to observer

Description

The OceanSurface shader provides a basic illumination model for a deep ocean surface. No waves are provided (use a displacement shader for those).

The deep sea color result is emitted as an incandescence value and simply added to the other shading components.

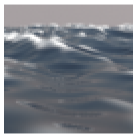
When Transmission is not 0, refraction rays are traced to capture objects beneath the water. The transmission result can be faded out based on distance if the TransmissionFalloff value is non-zero.

See Also

OceanSurfaceWithFoam surface shader

OceanWaves displacement shader

9.9.20 OceanSurfaceWithFoam

 surface shader for ocean water with foam (companion to the OceanWaves displacement shader)

Illumination Model: custom

Parameter	Default Value	Range	Description
Diffuse	0	0.0-1.0	Diffuse reflectivity
DiffuseColor	.44 .47 .6		Diffuse color
DeepSeaIntensity	.1		Deep sea color multiplier
DeepSeaColor	.21 .29 .47		Deep sea color
Specular	0.5		Specular intensity
SpecularRoughness	0.1		Specular roughness
Reflection	0.9		Reflection multiplier
ReflectionSamples	1		Number of reflection rays to trace
ReflectionBlur	0		Blur angle for reflection (in radians)
Transmission	0		Transmission multiplier
TransmissionMaxHitDist	1000		Max distance to trace transmission rays
TransmissionSamples	1		Rays to trace for refraction
TransmissionFalloff	0		Exponent for transmission decay with distance
TransmissionBlur	0		Blur angle for transmission rays
TransmissionNearColor	.3 .5 .4		Color for sea close to observer
Foam	1		Foam multiplier
FoamColor	1 1 1		Foam color
FoamOpacity	1		Extent to which foam obscures water
FoamIllumination	0.5		Multiplier for foam lighting
FoamIncandescence	0.5		Constant added to foam lighting
FoamRawOffset	0.95		Raw offset added to the base foam value
FoamRawMultiplier	10		Multiplier for raw foam value
FoamPower	2		Exponent for foam value

GridCountX	128		Grid size in X direction
GridCountY	128		Grid size in Y direction
SizeX	200		Nominal size of X dimension
SizeY	200		Nominal size of Y dimension
WindSpeed	5		Wind speed
WindAlign	8		Wave alignment with wind direction
WindDir	0		Wind direction as an angle in degrees
SmallWave	0.1		Small waves will be ignored
Dampen	0.75		Amount to dampen waves not aligned with the wind
Chop	1		Amount of chop to add to waves
FramesPerSecond	24		Value used to compute current time for animation
FoamEnabled	1	0 or 1	Whether foam needs to be simulated for the surface shader
RandomSeed	123		Base for random number generator

Description

The OceanSurfaceWithFoam shader provides a basic illumination model for a deep ocean surface with a foam option for use with the OceanWaves displacement shader.

The deep sea color result is emitted as an incandescence value and simply added to the other shading components.

When Transmission is not 0, refraction rays are traced to capture objects beneath the water. The transmission result can be faded out based on distance if the TransmissionFalloff value is non-zero.

Waves

The companion OceanWaves displacement shader computes waves by evaluating a simulation. In order for foam to appear in the proper location, the common parameters between the OceanWaves and OceanSurfaceWithFoam shaders must be set to the same values.

The wave simulation is evaluated on a discrete grid, which is mapped to the unit interval in texture space. X corresponds to the first texture coordinate, Y to the second coordinates. The GridCountX and GridCountY parameters give the grid size. A larger grid produces more detail but takes longer to compute.

SizeX and SizeY give the nominal size of the ocean area.

The waves can be tiled in X and Y.

WindSpeed gives the average wind speed. Higher values produce larger waves. WindDir can be used to change the wind direction by providing an angle in degrees about the vertical axis.

WindAlign controls how much the wind direction influences the wave directions by applying an exponent to the dot product of the wind direction and the direction of each wave. Larger values reduce the amplitude of waves not in the direction of the wind base on the Dampen factor.

Chop

The appearance of waves in windy conditions can be improved using the Chop parameter, which sharpens wave peaks and stretches troughs by shifting the surface laterally. Because the chop displacement shifts the surface in arbitrary directions, you'll need to tell Air that the displacement is not strictly in the direction of the surface normal by applying the following attribute:

```
  Attribute "render" "normaldisplacement" [0]
```

Note that too much chop can tear the surface apart.

Foam

The Chop parameter must be greater than 0 in order for foam to be generated. You will need to experiment with the FoamRawOffset and FoamRawMultiplier values to obtain a suitable base level of foaminess.

Animation

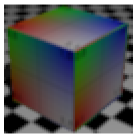
The waves will be animated based on the current frame number. The current time value is the current frame number divided by the FramesPerSecond value.

For speed the simulation is computed using the seawave dynamic shadeop (DSO) included with Air.

See Also

OceanWaves displacement shader

9.9.21 paintedplastic


plastic surface with a single texture map for color

Illumination Model: plastic

Features: texture map

Parameter	Default Value	Range	Description
Ka	1.0	0.0-1.0	Ambient reflectivity
Kd	0.5	0.0-1.0	Diffuse reflectivity
Ks	0.5	0.0-1.0	Specular reflectivity
roughness	0.1	0.01-1.0	Specular roughness
specularcolor	1 1 1		Specular highlight color
texturename	""		Texture file name
repeatx	1		Copies in the X direction
repeaty	1		Copies in the X direction
originx	0.0		Location of left edge of texture map
originy	0.0		Location of top edge of texture map

Description

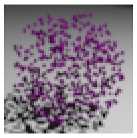
The shader simulates a colored plastic surface with a (usually white) shiny specular highlight. The base surface color is taken from the primitive's color attribute or from the optional texture map. The texture map is placed using the object's standard texture coordinates.

The roughness parameter controls the size of the specular highlight: larger roughness values produce larger highlights.

See Also

VPlastic shader

9.9.22 particle


surface shader for volume and point primitives

Illumination Model: custom

Parameter	Default Value	Range	Description
Diffuse	`1.0`		Diffuse reflectivity
Incandescence	`0.0`		Self-illumination value
Ambient	`0.0`	`0.0-1.0`	Ambient reflectivity
OpacityMultiplier	`1.0`		Opacity multiplier
PrimaryScatter	`0.0`	`-1 to 1`	Primary scattering direction
PrimaryScatterWeight	`1.0`	`0.0-1.0`	Relative weight of primary scattering
PrimaryScatterColor	`1.0 1.0 1.0`	color	Per-channel scattering multiplier
SecondScatter	`0.0`	`-1 to 1`	second scattering direction
SecondScatterColor	`1.0 1.0 1.0`	color	Per-channel scattering multiplier
AngleFalloff	`0.0`		Exponent for angle-based opacity falloff (for spherical particles)

Description

The particle shader implements a single-scattering illumination model for lighting volume and point primitives.

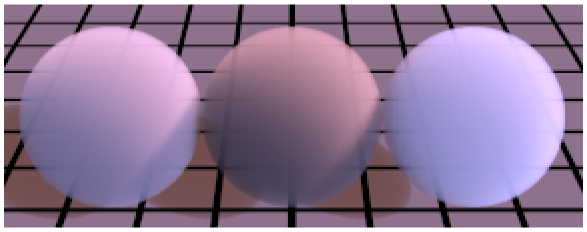
Isotropic Scattering

When the `PrimaryScatter` parameter is 0 and the `PrimaryWeight` parameter is 1, this shader produces isotropic scattering, in which incoming light is scattered equally in all directions. The result is a simple sum of the incoming light. (This shader omits the 1/4PI term from the usual isotropic scattering equation; you can adjust for this factor by setting the `Diffuse` parameter to ~0.08.)

Anisotropic Scattering

When the `PrimaryWeight` parameter is 1, a non-zero `PrimaryScatter` value will produce preferential scattering in one direction. If the scatter value is greater than 0, more light is reflected forward, along the direction of the incoming light. If the scatter value is negative, more light is reflected back towards the light source. The `PrimaryScatterColor` parameter can be used to adjust the scatter direction independently for each color channel.

The following image illustrates isotropic scattering, forward scattering (0.5), and backward scattering (-0.5) with blue light striking the front of the spheres and red light striking the back.



Scattering in Two Directions

When the `PrimaryWeight` parameter is less than `1`, the shader uses a weighted sum of scattering in two directions, given by the primary and secondary parameter settings. The second scattering contribution is weighted by `1-PrimaryScatterWeight`.

Reference Values

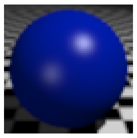
In Principles of Digital Image Synthesis Glassner presents values for several common types of scattering, summarized in the table below:

Type	Examples	PrimaryWeight	PrimaryScatter	SecondScatter
Rayleigh	dust, cigarette smoke	0.5	-0.46	0.46
Hazy Mie	sparse water droplets (fog)	0.12	-0.50	0.70
Murky Mie	dense droplets (thick fog)	0.19	-0.65	0.91

See Also

Volume Primitives

9.9.23 plastic


standard plastic surface shader

Illumination Model: plastic

Parameter	Default Value	Range	Description
Ka	1.0	0.0-1.0	Ambient reflectivity
Kd	0.5	0.0-1.0	Diffuse reflectivity
Ks	0.5	0.0-1.0	Specular reflectivity
roughness	0.1	0.01-1.0	Specular roughness
specularcolor	1 1 1		Specular highlight color

Description

The shader simulates a colored plastic surface with a (usually white) shiny specular highlight. The base surface color is taken from the primitive's color attribute.

The roughness parameter controls the size of the specular highlight: larger roughness values produce larger highlights.

See Also

VPlastic shader
V2SidedPlastic shader
VBrushedPlastic shader

9.9.24 ReelSmartMotion

surface shader that computes a motion vector for the ReelSmart Motion Blur plugin from RE:Vision Effects, Inc. (www.revisionfx.com)

Parameter	Default Value	Range	Description
MaxDisplace	`32`		Max pixel distance that can be stored
TargetSpace	`raster`	spaces	Coordinate space for output vector

Description

This surface shader computes a motion vector compatible with the ReelSmart Motion Blur plugin. To obtain useful output, the scene must have motion blur defined and enabled.

Set the MaxDisplace parameter to the same value used in the plugin. The vector image should be rendered with the following settings:

- `PixelSamples 1 1`
- A box filter with a filter width of 1.
- Gamma correction disabled (gamma set to 1).
- For an 8-bit image, quantization should be set as

  ```
  Quantize "rgba" 254 0 255 0
  ```

  For a 16-bit image, quantization should be

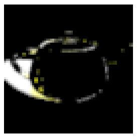
  ```
  Quantize "rgba" 65534 0 65535 0
  ```

Reference

This shader is based on the information at:

http://www.revisionfx.com/generalfaqsMVFormat.htm#alpha

9.9.25 shadowpass


surface shader that computes a shadow pass

Parameter	Default Value	Range	Description
backshadow	0	0 or 1	Set to 1 for shadows on surfaces facing away from a light
fuzz	0.05	0-1	Transition softness along silhouette edges when backshadow=1

Description

This shader sets the output color to a shadow value for the current shading location. This shader only works properly with lights that export a color `__unshadowed_Cl` output variable. All AIR light shaders as well as the MayaMan and RhinoMan light shaders should work properly.

9.9.26 shinymetal

metal shader with reflections

Illumination model: metal

Features: reflections

Parameter	Default Value	Range	Description
Ka	1.0	0.0-1.0	Ambient reflectivity
Kd	0.0	0.0-1.0	Diffuse reflectivity
Ks	1.0	0.0-1.0	Specular reflectivity
roughness	0.1	0.01-1.0	Specular roughness
Kr	1.0	0.0-1.0	Reflection strength
texturename	"raytrace"		Name of reflection map or "raytrace" for ray tracing

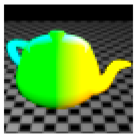
Description

The shinymetal surface shader is included for compabitility with the RenderMan® Interface Specification. For a more flexible metal shader with reflections use the VMetal shader instead.

See Also

VMetal shader

9.9.27 ShowPosition


surface shader for displaying X,Y,Z coordinates at each location

Parameter	Default Value	Range	Description
UnitSize	`1`		Divisor for emitted position values
Origin	`0 0 0`		Center point for position values
Clamp	`0`	0 or 1	When set to 1, emitted values are clamped to the unit interval
Cyclic	`0`	0 or 1	When set to 1, emitted values will repeat over the unit interval
ShadingSpace	`world`	spaces	coordinate space for emitted values

Description

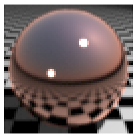
The ShowPosition surface shader sets the output color to the X,Y,Z coordinates of the current shading position P, with optional offset and scaling:

Output Color = (Position-Origin)/ UnitSize

When the Clamp parameter is set to 1, the output color is clamped to the unit interval.

When the Cyclic parameter is set to 1, the output color is restricted to the fractional part of the offset and scaled position.

9.9.28 SimpleMetal


physically plausible metal shader with simple parameters

Illumination Model: physical metal

Parameter	Default Value	Range	Description
Reflectance	`0.9`	`0.0-1.0`	Reflectivity
Roughness	`0.03`	`0.0-1.0`	Surface roughness
Samples	`8`	`1-1024`	Number of rays traced for reflections

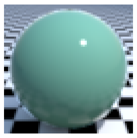
Description

This surface shader provides a physically plausible metallic appearance. The metal's color is taken from the standard color attribute. The Reflectance parameter scales the result for both specular highlights and reflections. Roughness controls the blurriness of reflections and the size of specular highlights - larger values produce blurrier reflections and larger highlights. Samples gives the number of rays to trace for reflections. For larger Roughness values, more samples may be needed to produce a smooth result.

See Also

VMetal surface shader
VPhysicalMetal surface shader

9.9.29 SimplePlastic

 physically plausible plastic shader with simple parameters

Illumination Model: physical plastic

Parameter	Default Value	Range	Description
Diffuse	1.0	0.0-1.0	Multiplier for diffuse
Specular	1.0	0.0-1.0	Multiplier for specular and reflections
Roughness	0.03	0.0-1.0	Surface roughness
Samples	4	1-1024	Number of rays traced for reflections

Description

This surface shader provides a physically plausible plastic or dielectric material. The base color is taken from the standard color attribute. The Specular parameter scales the result for both specular highlights and reflections. Roughness controls the blurriness of reflections and the size of specular highlights - larger values produce blurrier reflections and larger highlights. Samples gives the number of rays to trace for reflections. For larger Roughness values, more samples may be needed to produce a smooth result.

See Also

VPhysicalPlastic surface shader
VPlastic surface hader

9.9.30 UseBackground

surface shader that computes shadows and/or reflections for compositing over a background image

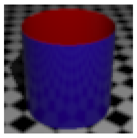
Parameter	Default Value	Range	Description
ShadowMask	1.0	0.0-1.0	Multiplier for shadow value
Reflection	1.0	0.0-1.0	Multiplier for reflections
ReflectionSamples	1		Number of rays to trace for reflections
ReflectionBlur	0		Blur angle in radians for reflections
ReflectionAlpha	1		Multiplier for the reflection alpha

Description

The UseBackground surface shader produces output useful for compositing over a background image. The output color is the reflection result, multiplied by the Reflection parameter. A reflection alpha is value is provided in an extra output variable named __reflection_alpha. If no reflections are desire, set the Reflection parameter to 0, and no reflections will be traced.

The emitted opacity value is the maximum of the reflection alpha and the computed shadow value. If no shadows are desired, set the ShadowMask value to 0, and no shadows will be computed.

9.9.31 V2SidedPlastic


reflective plastic with separate controls for the front and back sides of a surface

Illumination model: plastic

Features: texture maps, reflections

Parameter	Default Value	Range	Description
DiffuseFront	0.5	0.0-1.0	Diffuse reflectivity of front side
DiffuseBack	0.5	0.0-1.0	Diffuse reflectivity of back side
SpecularFront	0.5	0.0-1.0	Specular reflectivity of front side
SpecularBack	0.5	0.0-1.0	Specular reflectivity of back side
RoughnessFront	0.1	0.0-1.0	Specular roughness of front side
RoughnessBack	0.1	0.0-1.0	Specular roughness of back side
ReflectionFront	0.5	0.0-1.0	Reflection strength of front side
ReflectionBack	0.5	0.0-1.0	Reflection strength of back side
ReflectionSamples	1	1-256	Number of rays when ray tracing
ReflectionBlur	0.0	0.0-1.0	Blur for reflections
ReflectionName	"raytrace"		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	"current"	spaces	Coordinate space for reflection lookup
BackColor	1 0 0		Base color for back side
FrontTextureName	""		Texture name for front side
FrontTextureBlur	0.0	0.0-1.0	Texture blur for front side
FrontTextureSizeXY	1 1		Texture size in X and Y directions
FrontTextureOriginXY	0 0		Location of left, top texture edge
FrontTextureProjection	"st"	projections	Projection to use for 2D coordinates
FrontTextureSpace	"shader"	spaces	Coordinate space for projection
FrontTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection

BackTextureName	""		Texture name for back side
BackTextureBlur	0.0	0.0-1.0	Texture blur for back side
BackTextureSizeXY	1 1		Texture size in X and Y directions
BackTextureOriginXY	0 0		Location of left, top texture edge
BackTextureProjection	"st"	projections	Projection to use for 2D coordinates
BackTextureSpace	"shader"	spaces	Coordinate space for projection
BackTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection

Description

The V2SidedPlastic surface shader provides a basic reflective, textured plastic shader with separate controls for the front and back sides of a surface.

The base color of the front side is taken from the object's color attribute. The base color for the back side is set with the `BackColor` shader parameter.

Front and back sides may each have a texture map with completely separate mapping controls.

9.9.32 VAnimatedMap

surface shader with animated texture map

Illumination model: plastic

Features: reflections, animated texture map

Parameter	Default Value	Range	Description
Diffuse	0.5	0.0-1.0	Diffuse reflectivity
Specular	0.5	0.0-1.0	Specular reflectivity
SpecularRoughness	0.2	0.0-1.0	Specular roughness
Reflection	0.0	0.0-1.0	Reflection strength
ReflectionSamples	1	1-256	Number of rays when ray tracing
ReflectionBlur	0.0	0.0-1.0	Blur for reflections
ReflectionName	"raytrace"		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	"current"	spaces	Coordinate space for reflection lookup

MapBase	`""`	Map base name minus frame number
MapExtension	`".tif"`	File name extension for maps
MapCount	`1`	Number of maps
MapDuration	`1`	Number of frames to display each map
FrameStart	`0`	Frame for map number 0
FrameEnd	`9999`	Frame at which to stop animating
TextureSizeXY	`1 1`	Texture size in X and Y direction
TextureOriginXY	`0 0`	Location of left, top texture edge
TextureAngle	`0`	Texture rotation angle in degrees

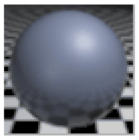
Description

The VAnimatedMap surface shader allows a sequence of numbered maps to be displayed during an animation. The maps should have a common base name, followed by a 4-digit index (zero-padded), followed by the file name extension. The first map should be numbered 0.

Animation begins at the frame given in FrameStart with map number 0 (which is also displayed for any earlier frames). Every MapDuration frames, the current map number is incremented. When the current map number reaches MapCount, the current map number is reset to 0.

If the texture map has an alpha channel, the alpha channel result is applied as the surface opacity.

9.9.33 VBlinn

 plastic shader with Blinn specular model, reflections, and texture maps

Illumination model: plastic with Blinn specular

Features: reflections, texture map

Parameter	Default Value	Range	Description
Diffuse	`0.5`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.5`	`0.0-1.0`	Specular reflectivity
SpecularSize	`0.3`	`0.0-1.0`	Specular size (Blinn eccentricity)
SpecularRolloff	`0.7`		Blinn rolloff parameter
Reflection	`0.0`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup

ColorMapName	`""`		Texture map for base color
ColorMapBlur	`0.0`	`0.0-1.0`	Texture blur
SpecularMapName	`""`		Texture map for specular color
SpecularMapBlur	`0.0`	`0.0-1.0`	Texture blur
OpacityMapName	`""`		Texture map for opacity
OpacityMapBlur	`0.0`	`0.0-1.0`	Texture blur
BumpMax	`0.0`		Maximum bump height
BumpMapName	`""`		Texture map for bump mapping
BumpMapBlur	`0.0`	`0.0-1.0`	Texture blur
TextureSizeXY	`1 1`		Texture size in X and Y direction
TextureOriginXY	`0 0`		Location of left, top texture edge
TextureAngle	`0`		Texture rotation angle in degrees
Projection	`"st"`	type	Projection to use for 2D coordinates
ProjectionSpace	`"shader"`	spaces	Coordinate space for projection

ProjectionTransform	`1 0 0 0`
	`0 1 0 0`
	`0 0 1 0`
	`0 0 0 1`
			Transformation applied prior to projection

Description

The VBlinn surface shader provides a plastic appearance using the Blinn specular model.

The optional color map, if present, is composited over the underlying base color (given by the color attribute) based on the texture alpha. For a texture map without an alpha channel, the map result replaces the base surface color.

The specular texture map modulates both the specular and reflection components.

Per-light Output Variables

In addition the standard output variables provided by most Air shaders, the VBlinn shader provides output variables with per-light results for diffuse, unshadowed diffuse, shadow, and specular components of the shading model:

`__lights[`*i*`]`	surface as illuminated by light channel i
`__lights_diffuse[`*i*`]`	diffuse component for light channel i
`__lights_unshadowed[`*i*`]`	unshadowed diffuse for light channel i
`__lights_shadow[`*i*`]`	shadow for light channel i
`__lights_specular[`*i*`]`	specular for light channel i

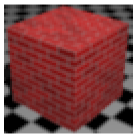
Each per-light array holds up to 10 channels of data, numbered 0 through 9. Assign a light to a particular channel by setting the light shader's `__channel` variable.

Sample RIB usage:

```
Declare "__lights" "varying color[10]"
Declare "__lights_diffuse" "varying color[10]"
Declare "__lights_unshadowed" "varying color[10]"
Declare "__lights_shadow" "varying color[10]"
Declare "__lights_specular" "varying color[10]"

Display "+light0.tif" "framebuffer"
"__lights[0],__lights_diffuse[0],__lights_unshadowed[0],__lights_shadow[0],
__lights_specular[0]"
  "quantize" [0 255 0 255]

Display "+light1.tif" "framebuffer"
"__lights[1],__lights_diffuse[1],__lights_unshadowed[1],__lights_shadow[1],
__lights_specular[1]"
  "quantize" [0 255 0 255]

LightSource "spotlight" 22 "float __channel" [0]

LightSource "pointlight" 23 "float __channel" [1]
```

See Also:
Light Channels

9.9.34 VBrick2D


basic brick shader with grooves

Illumination Model: matte

Diffuse	1.0	0.0-1.0	Diffuse reflectivity
BrickWidth	0.25		Brick width
BrickHeight	0.08		Brick height
BrickColorVary	0.4		Per-brick color variation
MortarWidth	0.01		Mortar width
MortarColor	0.6 0.6 0.6		Mortar color
Stagger	0.5	0.0-1.0	Fraction to offset every other row
EdgeVary	0.01		Unevenness of brick edges
EdgeVaryFrequency	4.0		Frequency of brick edge unevenness
GrooveDepth	0.01		Depth of mortar groove
PockDepth	0.01		Depth of indentations
PockWidth	0.01		Average size of pock marks

TextureSizeXY	`1 1`		Texture size in X and Y direction
TextureOriginXY	`0 0`		Location of left, top texture edge
TextureAngle	`0`		Texture rotation angle in degrees
Projection	`"st"`	type	Projection to use for 2D coordinates
ProjectionSpace	`"shader"`	spaces	Coordinate space for projection
ProjectionTransform	`1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1`		Transformation applied prior to projection

Description

This shader produces a simple brick surface with grooves for the mortar.

The base brick color is taken from the object's color attribute. The BrickColorVary parameter allows the intensity of the brick color to be varied for each brick.

9.9.35 VBrushedMetal

brushed metal surface with reflections

Illumination model: reflective anisotropic metal

Features: reflections

Parameter	Default Value	Range	Description
Ambient	`0.1`	`0.0-1.0`	Ambient reflectivity
Diffuse	`0.1`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.2`	`0.0-1.0`	Specular reflectivity
SpecularRoughnessX	`0.1`	`0.01-1.0`	Specular roughness in X direction
SpecularRoughnessY	`0.3`	`0.01-1.0`	Specular roughness in Y direction
Reflection	`0.95`	`0.0-1.0`	Reflection strength
ReflectionSamples	`4`	`1-256`	Number of rays when ray tracing
ReflectionBlurX	`0.1`	`0.01-1.0`	Reflection blur in X direction
ReflectionBlurY	`0.3`	`0.01-1.0`	Reflection blur in Y direction
ReflectionName	`"raytrace"`		Name for reflection map or `"raytrace"` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
ReflectionGroups	`""`		Groups to test for traced reflections

Description

This surface shader simulates brushed or rolled metal.

The shader uses an anisotropic reflection model, in which specular reflectivity depends on the orientation of the surface with respect to the incoming light. The surface orientation is determined by the primitive's tangent vectors. The X direction corresponds to the tangent vector in the u direction; the Y direction corresponds to the tangent vector in the v direction. Only parametric surfaces such as NURBs, bicubic patches, and bilinear patches have well-defined tangent vectors. This shader may not produce reasonable results when applied to subdivision surfaces or polygon meshes.

The SpecularRoughnessX and SpecularRoughnessY parameters control the size of the specular highlight in the X and Y directions respectively. Using a different value for X and Y roughness generates oblong highlights.

Similarly, the ReflectionBlurX and ReflectionBlurY parameters control the shape of blurry reflections; using a different value for each parameter produces anisotropic reflections.

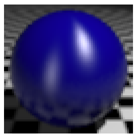
Normally the ReflectionBlurX and ReflectionBlurY values should be proportional to the RoughnessX and Roughness Y parameters, respectively, to produce a consistent appearance. However, the RoughnessX value does not have to be exactly the same as the ReflectionBlurX parameter, and the RoughnessY value need not match the ReflectionBlurY value.

The illumination model is based on "Measuring and Modeling Anisotropic Reflection" by Greg Ward published in SIGGRAPH '92.

See Also

VBrushedPlastic shader

9.9.36 VBrushedPlastic


brushed plastic surface with reflections

Illumination model: anisotropic plastic

Features: reflections

Parameter	Default Value	Range	Description
Diffuse	0.45	0.0-1.0	Diffuse reflectivity
Specular	0.1	0.0-1.0	Specular reflectivity
SpecularRoughnessX	0.1	0.01-1.0	Specular roughness in X direction
SpecularRoughnessY	0.25	0.01-1.0	Specular roughness in Y direction
Reflection	0.95	0.0-1.0	Reflection strength
ReflectionSamples	4	1-256	Number of rays when ray tracing
ReflectionBlurX	0.1	0.01-1.0	Reflection blur in X direction
ReflectionBlurY	0.25	0.01-1.0	Reflection blur in Y direction
ReflectionName	"raytrace"		Name for reflection map or "raytrace" for ray tracing
ReflectionSpace	"current"	spaces	Coordinate space for reflection lookup
ColorMapName	""		Texture map for base color
ColorMapBlur	0.0	0.0-1.0	Texture blur
SpecularMapName	""		Texture map for specular color
SpecularMapBlur	0.0	0.0-1.0	Texture blur
OpacityMapName	""		Texture map for opacity
OpacityMapBlur	0.0	0.0-1.0	Texture blur
BumpMax	0.0		Maximum bump height
BumpMapName	""		Texture map for bump mapping
BumpMapBlur	0.0	0.0-1.0	Texture blur
TextureSizeXY	1 1		Texture size in X and Y direction
TextureOriginXY	0 0		Location of left, top texture edge
TextureAngle	0		Texture rotation angle in degrees
Projection	"st"	type	Projection to use for 2D coordinates
ProjectionSpace	"shader"	spaces	Coordinate space for projection
ProjectionTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection

Description

This surface shader simulates the appearance of anisotropic surfaces with a plastic finish such as painted wood. The default parameters are based on physical measurements for a semi-gloss latex paint applied to wood.

The shader uses an anisotropic reflection model, in which specular reflectivity depends on the orientation of the surface with respect to the incoming light. The surface orientation is determined by the primitive's tangent vectors. The X direction corresponds to the tangent vector in the u direction; the Y direction corresponds to the tangent vector in the v direction. Only parametric surfaces such as NURBs, bicubic patches, and bilinear patches have well-defined tangent vectors. This shader may not produce reasonable results when applied to subdivision surfaces or polygon meshes.

The SpecularRoughnessX and SpecularRoughnessY parameters control the size of the specular highlight in the X and Y directions respectively. Using a different value for X and Y roughness generates oblong highlights.

Similarly, the ReflectionBlurX and ReflectionBlurY parameters control the shape of blurry reflections; using a different value for each parameter produces anisotropic reflections.

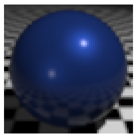
Normally the ReflectionBlurX and ReflectionBlurY values should be proportional to the RoughnessX and Roughness Y parameters, respectively, to produce a consistent appearance. However, the RoughnessX value does not have to be exactly the same as the ReflectionBlurX parameter, and the RoughnessY value need not match the ReflectionBlurY value.

The illumination model is based on "Measuring and Modeling Anisotropic Reflection" by Greg Ward published in SIGGRAPH '92.

See Also

VBrushedMetal shader

9.9.37 VCarPaint


car paint simulated as a base metal layer with clear overcoat

Illumination Model: metal and ceramic

Features: reflections

Parameter	Default Value	Range	Description
Diffuse	`0.3`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.7`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.1`	`0.01-1.0`	Specular roughness
FinishSpecular	`0.7`	`0.0-1.0`	specular reflectivity for clear finish layer
FinishReflection	`0.5`	`0.0-1.0`	reflection multiplier for clear finish layer
FinishSpecularRoughness	`0.015`	`0.01-1.0`	surface roughness
FinishSpecularSharpness	`0.4`	`0.0-1.0`	sharpness of highlight edge

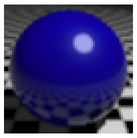
Flake	`0`		Flake intensity
FlakeFrequency	`1000`		Flake frequency
FlakeDensity	`0.5`	`0.0-1.0`	Fraction of area covered by flake
FlakeRoughness	`0.4`		Specular roughness for flakes
FlakeColor	`1 1 1`		Flake color
Reflection	`0.0`	`0.0-1.0`	Reflection strength for base
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup

Description

The VCarPaint surface shader simulates metallic car paint with a clear finish. The base metal color is taken from the object's color attribute. The clear finish adds glossy highlights and reflections (unmodulated by the base color) to the underlying metal.

Metallic flakes embedded in the paint can be added using the Flake controls. The flakes are distributed in texture space, using standard texture coordinates. They are generated using an antialiased noise pattern, so small flakes will fade out with distance.

9.9.38 VCeramic


glossy, reflective surface with optional texture maps for color and shininess

Illumination model: ceramic

Features: reflections, texture maps

Parameter	Default Value	Range	Description
Diffuse	`0.5`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.5`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.2`	`0.0-1.0`	Specular roughness
SpecularSharpness	`0.8`	`0.0-1.0`	Sharpness of highlight edge
Reflection	`1.0`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup

ColorMapName	""		Texture map for base color
ColorMapBlur	0.0	0.0-1.0	Texture blur
SpecularMapName	""		Texture map for specular color
SpecularMapBlur	0.0	0.0-1.0	Texture blur
OpacityMapName	""		Texture map for opacity
OpacityMapBlur	0.0	0.0-1.0	Texture blur
BumpMax	0.0		Maximum bump height
BumpMapName	""		Texture map for bump mapping
BumpMapBlur	0.0	0.0-1.0	Texture blur
TextureSizeXY	1 1		Texture size in X and Y direction
TextureOriginXY	0 0		Location of left, top texture edge
TextureAngle	0		Texture rotation angle in degrees
Projection	"st"	type	Projection to use for 2D coordinates
ProjectionSpace	"shader"	spaces	Coordinate space for projection
ProjectionTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection

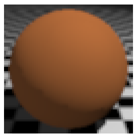
Description

The VCeramic surface shader simulates a glossy surface with reflections and texture maps for color, specular, opacity, and bump. All texture maps use the same set of texture coordinates.

See Also

VShinyTile2D shader

9.9.39 VClay

 surface shader for rough, dusty surfaces such as clay with no specular highlights

Illumination Model: clay

Features: texture map

Parameter	Default Value	Range	Description
Diffuse	0.7	0.0-1.0	Diffuse reflectivity
DiffuseRoughness	0.5	0.0-1.0	Diffuse roughness

ColorMapName	""		Texture map for base color
ColorMapBlur	0.0	0.0-1.0	Texture blur
OpacityMapName	""		Texture map for opacity
OpacityMapBlur	0.0	0.0-1.0	Texture blur
BumpMax	0.0		Maximum bump height
BumpMapName	""		Texture map for bump mapping
BumpMapBlur	0.0	0.0-1.0	Texture blur
TextureSizeXY	1 1		Copies of texture in X and Y direction
TextureOriginXY	0 0		Location of left, top texture edge
TextureAngle	0		Texture rotation angle in degrees
Projection	"st"	type	Projection to use for 2D coordinates
ProjectionSpace	"shader"	spaces	Coordinate space for projection
ProjectionTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection

Description

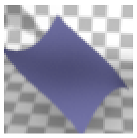
The VClay surface shader produces the appearance of rough surfaces such as clay.

The base output color is taken from the object's color attribute, which is multiplied by the color map value if a color map is provided.

See Also

VConcrete shader

9.9.40 VCloth

 surface shader for cloth based on a physical model of woven fabric

Illumination Model: microcylinder

Features: texture map

Parameter	Default Value	Range	Description
Diffuse	1.0	0.0-1.0	Volume scattering multiplier
Specular	1.0	0.0-1.0	Reflection multiplier
ColorMapName	""		Texture map for base color
ColorMapBlur	0.0	0.0-1.0	Texture blur

WeavePattern	`1 1 1`		Weave pattern as warp run, warp skip, weft run
WeaveWarpWidth	`1`		Warp thread width
WeaveWarpGap	`1`		Gap between warp threads
WeaveWeftWidth	`1`		Weft thread width
WeaveWeftGap	`1`		Gap between weft threads
WarpColor	`1 1 1`		Color for warp threads
WarpMapName	`""`		Texture map for warp color
WarpDiffuseSpread	`24`		Volume scattering angle in degrees
WarpDiffuseIsotropy	`0.3`	`0.0-1.0`	Isotropic scattering tendency
WarpSpecularSpread	`12`		Specular reflectance angle in degrees
WeftColor	`1 1 1`		Color for weft threads
WeftMapName	`""`		Texture map for weft color
WeftDiffuseSpread	`24`		Volume scattering angle in degrees
WeftDiffuseIsotropy	`0.3`	`0.0-1.0`	Isotropic scattering tendency
WeftSpecularSpread	`12`		Specular reflectance angle in degrees
ThreadAngle	`0`		Rotation for thread directions in degrees
ThreadAngleMap	`""`		Texture map for rotation angle
ThreadAngleMapMultiply	`90`		Multiplier for texture result
TextureSizeXY	`1 1`		Copies of texture in X and Y direction
TextureOriginXY	`0 0`		Location of left, top texture edge
TextureAngle	`0`		Texture rotation angle in degrees

Description:

The VCloth surface shader provides a physically based shading model for woven cloth based on the paper "A Practical Microcylinder Appearance Model for Cloth Rendering" by Sadeghi et al.

Thread Directions

Woven cloth is made by interleaving two perpendicular sets of threads or yarns, known as *warp* and *weft*. VCloth uses the tangent vectors for a surface (which are usually based on the texture coordinates) as the basis for the thread directions. By default warp threads follow the tangent in the direction of the second texture coordinate (Y); weft threads follow the X tangent. The thread directions can be rotated using the `ThreadAngle` parameters. By setting the `ThreadAngle` to 90, you can effectively swap the warp and weft directions. Rotating the thread directions does not affect the coordinates used for any texture maps.

Cloth appearance is affected by the type of weave pattern that is used and by the physical properties of the threads. This shader allows both sets of properties to be controlled.

Weave Pattern

The weave pattern is taken from the smallest rectangular piece of the cloth that could be tiled to reproduce the entire cloth. The shader assumes that this smallest area is much smaller than a pixel, and the shader returns the average reflectance over that patch. Individual threads will not be visible.

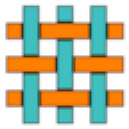
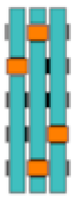
The WeavePattern parameter gives the pattern as 3 values:

- The length of the visible warp thread, given as the number of weft threads crossed over.
- The number of weft threads to skip before the next warp thread is visible
- The length of the visible weft thread, given as the number of warp threads crossed over.

The rest of the weave definition gives the thread width and the gap between threads in each direction. Because the exact pattern is not visible, this is enough information to compute the reflectance for the simple weave patterns supported by this shader: simple or plain weave, twill, and satin.

The weave pattern affects the final appearance in a two ways. First, it determines how much of each thread type is visible. Second, it determines the shape of the threads as they are interwoven, which affects how light responds to each thread. The shader automatically estimates the thread profiles based on the weave pattern.

Here are two sample patterns with the corresponding parameter values (the warp threads are vertical and blue-green; weft threads are horizontal and orange):

WeavePattern	1 1 1	3 1 1
WeaveWarpWidth	1	1
WeaveWarpGap	1	0.1
WeaveWeftWidth	1	1
WeaveWeftGap	1	1

Thread Properties

The way thread scatters light is computed as the sum of two functions: a "diffuse" component computed as the effect of volume scattering in the thread, and "specular" component for light that is reflected at the thread surface.

The specular (reflection scattering) behavior of the thread is determined by the thread shape and the specular spread value (`WarpSpecularSpread` or `WeftSpecularSpread`) which gives an angle in degrees over which reflected light is scattered. Larger values produce a broader but dimmer highlight.

Similarly, the diffuse spread parameter (`WarpDiffuseSpread` or `WeftDiffuseSpread`) determines how light scattered through the thread is emitted. The volume scattering is also affected by the isotropy parameter (`WarpDiffuseIsotropy` or `WeftDiffuseIsotropy`), which specifies the threads tendency to scatter light in all directions (versus scattering preferentially along the incoming light direction).

Threads are constructed from cloth fibers in two different ways, and the method of construction affects the optical properties of the thread. Staple threads (such as cotton) are made out of twisted fibers, and they are held together by friction. Filament threads are made of long straight fibers (such as silk) with little twist.

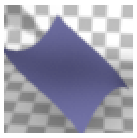
Here are some measured values for different thread types from the above cited paper:

Thread Type	Diffuse Spread	Diffuse Isotropy	Specular Spread
Linen (flax)	24	0.3	12
Flat silk	10	0.2	5
Twisted silk	32	0.3	18
Flat polyester	5	0.1	2.5
Twisted polyester	60	0.7	30

See Also:

VClothAdvanced surface shader
VFabric surface shader

9.9.41 VClothAdvanced

 surface shader for cloth based on a physical model of woven fabric

Illumination Model: microcylinder

Features: texture map

Parameter	Default Value	Range	Description
Diffuse	1.0	0.0-1.0	Volume scattering multiplier
Specular	1.0	0.0-1.0	Reflection multiplier
ColorMapName	""		Texture map for base color
ColorMapBlur	0.0	0.0-1.0	Texture blur
WarpColor	1 1 1		Color for warp threads
WarpMapName	""		Texture map for warp color
WarpDiffuseSpread	24		Volume scattering angle in degrees
WarpDiffuseIsotropy	0.3	0.0-1.0	Isotropic scattering tendency
WarpSpecularSpread	12		Specular reflectance angle in degrees

WeftColor	`1 1 1`		Color for weft threads
WeftMapName	`""`		Texture map for weft color
WeftDiffuseSpread	`24`		Volume scattering angle in degrees
WeftDiffuseIsotropy	`0.3`	`0.0-1.0`	Isotropic scattering tendency
WeftSpecularSpread	`12`		Specular reflectance angle in degrees
ThreadAngle	`0`		Rotation for thread directions in degrees
ThreadAngleMap	`""`		Texture map for rotation angle
ThreadAngleMapMultiply	`90`		Multiplier for texture result
TextureSizeXY	`1 1`		Copies of texture in X and Y direction
TextureOriginXY	`0 0`		Location of left, top texture edge
TextureAngle	`0`		Texture rotation angle in degrees

Description:

The VCloth surface shader provides a physically based shading model for woven cloth based on the paper "A Practical Microcylinder Appearance Model for Cloth Rendering" by Sadeghi et al.

Thread Directions

Woven cloth is made by interleaving two perpendicular sets of threads or yarns, known as *warp* and *weft*. VCloth uses the tangent vectors for a surface (which are usually based on the texture coordinates) as the basis for the thread directions. By default warp threads follow the tangent in the direction of the second texture coordinate (Y); weft threads follow the X tangent. The thread directions can be rotated using the `ThreadAngle` parameters. By setting the `ThreadAngle` to 90, you can effectively swap the warp and weft directions. Rotating the thread directions does not affect the coordinates used for any texture maps.

Cloth appearance is affected by the type of weave pattern that is used and by the physical properties of the threads. This shader allows both sets of properties to be controlled.

Weave Pattern

The VClothAdvanced shader provides low-level controls over the thread curves in each direction. The thread profile is defined as one or more curve segments with the `WarpCurve*` and `WeftCurve*` parameters. Each curve segment is defined as three values: start angle, end angle, and length. The angles are relative to the tangent direction, with negative angles pointing up and positive angles pointing down with respect to the surface normal. The shading computations return the length-weighted average response for the set of curves defined for each thread.

For an alternate method of specifying weave curves, see the VCloth shader.

Thread Properties

The way thread scatters light is computed as the sum of two functions: a "diffuse" component computed as the effect of volume scattering in the thread, and "specular" component for light that is reflected at the thread surface.

The specular (reflection scattering) behavior of the thread is determined by the thread shape and the specular spread value (`WarpSpecularSpread` or `WeftSpecularSpread`) which gives an angle in degrees over which reflected light is scattered. Larger values produce a broader but dimmer highlight.

Similarly, the diffuse spread parameter (`WarpDiffuseSpread` or `WeftDiffuseSpread`) determines how light scattered through the thread is emitted. The volume scattering is also affected by the isotropy parameter (`WarpDiffuseIsotropy` or `WeftDiffuseIsotropy`), which specifies the threads tendency to scatter light in all directions (versus scattering preferentially along the incoming light direction).

Threads are constructed from cloth fibers in two different ways, and the method of construction affects the optical properties of the thread. Staple threads (such as cotton) are made out of twisted fibers, and they are held together by friction. Filament threads are made of long straight fibers (such as silk) with little twist.

Here are some measured values for different thread types from the above cited paper:

Thread Type	IOR	Diffuse Spread	Diffuse Isotropy	Specular Spread
Linen (flax)	1.46	24	0.3	12
Flat silk	1.35	10	0.2	5
Twisted silk	1.35	32	0.3	18
Flat polyester	1.54	5	0.1	2.5
Twisted polyester	1.54	60	0.7	30

See Also:

VCloth surface shader
VFabric surface shader

9.9.42 VCumulusCloud

cloud surface shader for volumetric primitives

Illumination Model: particle

Parameter	Default Value	Range	Description
Diffuse	`1.0`		Diffuse reflectivity
Incandescence	`0.0`		Self-illumination value
Ambient	`0.0`	`0.0-1.0`	Ambient reflectivity
OpacityMultiplier	`1.0`		Opacity multiplier
PrimaryScatter	`-0.1`	`-1 to 1`	Primary scattering direction
PrimaryScatterWeight	`1.0`	`0.0-1.0`	Relative weight of primary scattering
PrimaryScatterColor	`1.0 1.0 1.0`	color	Per-channel scattering multiplier
SecondScatter	`0.0`	`-1 to 1`	second scattering direction
SecondScatterColor	`1.0 1.0 1.0`	color	Per-channel scattering multiplier
PatternSize	`0.5`		size of cloud pattern
SphereRadius	`0.8`		radius of soft sphere shape
Turbulence	`0.1`		turbulence strength
TurbulenceExponent	`0.5`		exponent for turbulence sharpness
TurbulenceLevels	`4`	`1-9`	octaves of fractal noise
TurbulenceRoughness	`0.5`		relative strength of successive noise levels
Distortion	`0.5`		distortion of basic shape
DistortionVector	`1 -1 1`		distortion direction
ShadingSpace	`"shader"`	space	space for shading computations

Description

This is a basic cloud shader for cumulus clouds. The illumination model is the same as that of the particle shader.

The cloud pattern is based on David Ebert's cumulus cloud model in Chapter 8 of Texturing and Modeling. The `SphereRadius` parameter gives the radius for a soft sphere shape within the volume primitive. If `SphereRadius` is 0, the soft sphere shape is not used, and the entire volume primitive is filled with the turbulent cloud pattern.

The basic sphere shape can be modified to look less like a sphere by distorting it along the `DistortionVector` using the turbulence result multiplied by the `Distortion` parameter.

The pattern within the cloud is generated using a turbulence function controlled by the various Turbulence parameters. The overall size of the pattern is determined by the `PatternSize` parameter. Both the `PatternSize` and `OpacityMultiplier` parameters should be adjusted based on the scale of volume primitive.

See Also

particle shader
VSmokeSurface shader

9.9.43 VConcrete

concrete with grooves and color variation

Illumination Model: clay

Parameter	Default Value	Range	Description
Diffuse	`0.8`	`0.0-1.0`	Diffuse reflectivity
DiffuseRoughness	`0.8`	`0.0-1.0`	Diffuse roughness
GrooveDepth	`0.01`		depth of grooves
GrooveWidth	`0.05`		Width of grooves in the shading space
GrooveAxis	`1 0 0`		axis perpendicular to grooves
Splotchiness	`1.0`		how much color variation to add
SplotchSize	`0.2`	`0.0-1.0`	size of color-varying splotches
SplotchColor	`0.89 0.89 0.86`		color of splotches
ShadingSpace	`"shader"`	spaces	coordinate space for shading calculations

Description

The VConcrete surface shader produces the appearance of weathered concrete with grooves.

The base surface color is taken from the object's color attribute, to which splotches are applied according to the Splotch parameters.

The grooves are oriented perpendicular to the axis defined by `GrooveAxis` in the `ShadingSpace` coordinate system. For concrete without grooves, set the `GrooveDepth` parameter to 0.

See Also

VClay shader

9.9.44 VDashes

dashes for curves

Parameter	Default Value	Range	Description
Frequency	10.0		Dashes per unit length in texture space
FirstDash	0.4	0-1	Length of first dash
FirstGap	0.2	0-1	Length of first gap
SecondDash	0.0	0-1	Length of second dash
SecondGap	0.1	0-1	Length of second gap
ThirdDash	0.0	0-1	Length of third dash

Description

The VDashes surface shader makes a curve primitive into a dashed line using the curve's texture coordinates. The color of the dashes is taken from the object's color attribute.

`Frequency` sets the number of dashes per texture coordinate unit. Other parameters are given as a fraction of the unit interval.

The pattern can have up to 3 dashes.

9.9.45 VDecal2D

reflective plastic shader with base texture map and up to 3 decals

Illumination model: plastic

Features: reflections, texture map

Parameter	Default Value	Range	Description
Diffuse	0.5	0.0-1.0	Diffuse reflectivity
Specular	0.5	0.0-1.0	Specular reflectivity
SpecularRoughness	0.2	0.0-1.0	Specular roughness
SpecularSharpness	0.8	0.0-1.0	Sharpness of highlight edge
Reflection	1.0	0.0-1.0	Reflection strength
ReflectionSamples	1	1-256	Number of rays when ray tracing
ReflectionBlur	0.0	0.0-1.0	Blur for reflections
ReflectionName	"raytrace"		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	"current"	spaces	Coordinate space for reflection lookup
ColorMapName	""		Texture map for base color
ColorMapBlur	0.0	0.0-1.0	Texture blur
SpecularMapName	""		Texture map for specular color
SpecularMapBlur	0.0	0.0-1.0	Texture blur

OpacityMapName	""		Texture map for opacity
OpacityMapBlur	0.0	0.0-1.0	Texture blur
BumpMax	0.0		Maximum bump height
BumpMapName	""		Texture map for bump mapping
BumpMapBlur	0.0	0.0-1.0	Texture blur
TextureSizeXY	1 1		Texture size in X and Y direction
TextureOriginXY	0 0		Location of left, top texture edge
TextureAngle	0		Texture rotation angle in degrees
Projection	"st"	type	Projection to use for 2D coordinates
ProjectionSpace	"shader"	spaces	Coordinate space for projection
ProjectionTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection
TopDecalName	""		Decal file name
TopDecalOrigin[2]	0 0		Location of top, left decal corner
TopDecalSize[2]	1 1		Width and height of decal
TopDecalColor	1 1 1		Decal color
SecondDecalName	""		Decal file name
SecondDecalOrigin[2]	0 0		Location of top, left decal corner
SecondDecalSize[2]	1 1		Width and height of decal
SecondDecalColor	1 1 1		Decal color
ThirdDecalName	""		Decal file name
ThirdDecalOrigin[2]	0 0		Location of top, left decal corner
ThirdDecalSize[2]	1 1		Width and height of decal
ThirdDecalColor	1 1 1		Decal color

Description

This plastic shader provides a base texture map with up to 3 decals on top. The decals and texture map are positioned using the object's standard texture coordinates.

9.9.46 VDecal3D

reflective plastic shader with base texture map and up to 3 decals

Illumination model: plastic

Features: reflections, texture map, projected decals

Parameter	Default Value	Range	Description
Diffuse	`0.5`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.5`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.2`	`0.0-1.0`	Specular roughness
SpecularSharpness	`0.8`	`0.0-1.0`	Sharpness of highlight edge
Reflection	`1.0`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
ColorMapName	`""`		Texture map for base color
ColorMapBlur	`0.0`	`0.0-1.0`	Texture blur
SpecularMapName	`""`		Texture map for specular color
SpecularMapBlur	`0.0`	`0.0-1.0`	Texture blur
OpacityMapName	`""`		Texture map for opacity
OpacityMapBlur	`0.0`	`0.0-1.0`	Texture blur
BumpMax	`0.0`		Maximum bump height
BumpMapName	`""`		Texture map for bump mapping
BumpMapBlur	`0.0`	`0.0-1.0`	Texture blur
TextureSizeXY	`1 1`		Texture size in X and Y direction
TextureOriginXY	`0 0`		Location of left, top texture corner
TextureAngle	`0`		Texture rotation angle in degrees
Projection	`"st"`	type	Projection to use for 2D coordinates
ProjectionSpace	`"shader"`	spaces	Coordinate space for projection

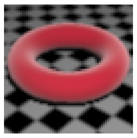
ProjectionTransform	`1 0 0 0`
	`0 1 0 0`
	`0 0 1 0`
	`0 0 0 1`
			Transformation applied prior to projection

TopDecalName	""		Decal file name
TopDecalSpace	"world"	spaces	Coordinate space for projection
TopDecalTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Projective transformation
TopDecalFrontOnly	0	-1, 0, 1	1=front only, -1=back only, 0=both sides
SecondDecalName	""		Decal file name
SecondDecalSpace	"world"	spaces	Coordinate space for projection
SecondDecalTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Projective transformation
SecondDecalFrontOnly	0	-1, 0, 1	1=front only, -1=back only, 0=both sides
ThirdDecalName	""		Decal file name
ThirdDecalSpace	"world"	spaces	Coordinate space for projection
ThirdDecalTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Projective transformation
ThirdDecalFrontOnly	0	-1, 0, 1	1=front only, -1=back only, 0=both sides

Description

This plastic shader provides a base texture map with up to 3 projected decals on top.

9.9.47 Velvet


velvet surface shader

Illumination Model: custom

Features: texture map

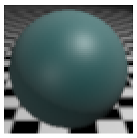
Parameter	Default Value	Range	Description
Diffuse	1.0	0.0-1.0	Diffuse reflectivity
BackScatter	0.5	0.0-1.0	Strength of reflection toward the viewer
Edginess	10.0		Strength of scattering at the horizon
Roughness	0.1	0.01-1.0	Specular roughness
SheenColor	1 1 1		Shiny color
ColorMapName	""		Texture map for base color
ColorMapBlur	0.0	0.0-1.0	Texture blur

SpecularMapName	""		Texture map for specular color
SpecularMapBlur	0.0	0.0-1.0	Texture blur
OpacityMapName	""		Texture map for opacity
OpacityMapBlur	0.0	0.0-1.0	Texture blur
BumpMax	0.0		Maximum bump height
BumpMapName	""		Texture map for bump mapping
BumpMapBlur	0.0	0.0-1.0	Texture blur
TextureSizeXY	1 1		Texture size in X and Y direction
TextureOriginXY	0 0		Location of left, top texture edge
TextureAngle	0		Texture rotation angle in degrees
Projection	"st"	type	Projection to use for 2D coordinates
ProjectionSpace	"shader"	spaces	Coordinate space for projection
ProjectionTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection

Description

The Velvet surface shader provides a basic velvet appearance with optional texture map for color.

9.9.48 VFabric

 surface shader for cloth with blinn specular and velvet sheen controls

Illumination Model: custom

Features: texture map

Parameter	Default Value	Range	Description
Diffuse	0.6	0.0-1.0	Diffuse reflectivity
DiffuseRoughness	0.8	0.0-1.0	Diffuse roughness
Specular	0.5	0.0-1.0	Specular reflectivity
SpecularSize	0.3	0.0-1.0	Specular highlight size
SpecularRolloff	0.7		Falloff of highlight edge
SpecularColor	1 1 1		Specular color

Sheen	0.2	0.0-1.0	Sheen intensity
SheenBackscatter	0.2		Tendency to scatter light towards viewer
SheenEdginess	3		Exponent applied to sheen strength
SheenRoughness	0.4		Sheen roughness
SheenMapName	""		Texture map for sheen color/strength
SheenMapBlur	0.0		Texture blur
ColorMapName	""		Texture map for base color
ColorMapBlur	0.0	0.0-1.0	Texture blur
SpecularMapName	""		Texture map for specular color
SpecularMapBlur	0.0	0.0-1.0	Texture blur
OpacityMapName	""		Texture map for opacity
OpacityMapBlur	0.0	0.0-1.0	Texture blur
BumpMax	0.0		Maximum bump height
BumpMapName	""		Texture map for bump mapping
BumpMapBlur	0.0	0.0-1.0	Texture blur
TextureSizeXY	1 1		Texture size in X and Y direction
TextureOriginXY	0 0		Location of left, top texture edge
TextureAngle	0		Texture rotation angle in degrees

Description:

The VFabric surface shader provides a custom illumination model for shading cloth: a combination of the VClay diffuse model, the VBlinn specular model, and the Velvet shaders' sheen component.

9.9.49 VFur

surface shader for fur modeled as curves primitives

Illumination Model: custom

Parameter	Default Value	Range	Description
Ambient	1.0	0.0-1.0	Ambient reflectivity
Diffuse	0.5	0.0-1.0	Diffuse reflectivity
Specular	0.5	0.0-1.0	Specular reflectivity
SpecularRoughness	0.1	0.01-1.0	Specular roughness

SpecularColor	1 1 1		Highlight color
SpecularStart	0.1	0.0-1.0	Distance from root to start specular
SpecularEnd	1.0	0.0-1.0	Distance from root to end specular
SpecularFadeRegion	0.1		Transition region at specular start and end
RootColor	0.5 0.5 0.5		Color at base of curve
TipColor	1.0 1.0 1.0		Color at tip of curve
ColorBias	0.5	0.0-1.0	Bias for color interpolation along curve
N_Srf	0 0 0		Surface normal at base of curve (provided per-curve in `Curves` primitive definition)

Description:

This shader is designed to shade fur generated with Joe Alter's Shave and a Haircut.

The diffuse color is based on the `RootColor` and `TipColor` which are linearly interpolated along the curve's length. The result of that interpolation is then multiplied by the curve's color attribute.

See Also

VHair shader

9.9.50 VGlass

glass shader for thick, solid glass with refraction effects

Illumination model: custom

Features: reflections

Parameter	Default Value	Range	Description
Specular	1.0	0.0-1.0	Specular reflectivity
SpecularRoughness	0.1	0.0-1.0	Specular roughness
SpecularSharpness	0.8	0.0-1.0	Sharpness of highlight edge
SpecularColor	1.0 1.0 1.0		Color of specular highlight
Reflection	1.0	0.0-1.0	Reflection strength
ReflectionSamples	1	1-256	Number of rays when ray tracing
ReflectionBlur	0.0	0.0-1.0	Blur for reflections
ReflectionMaxLevel	1	1-20	Max level of interreflection
ReflectionName	"raytrace"		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	"current"	spaces	Coordinate space for reflection lookup
ReflectionGroups	""		Groups to test for traced reflections
IndexOfRefraction	1.5	0.1-10.0	Index of refraction
Transmission	1.0	0.0-1.0	Multiplier for transmitted/refracted light
TransmissionColor	1.0 1.0 1.0		Color for transmitted rays
TransmissionSamples	1	1-256	Number of rays for refraction
TransmissionBlur	0.0	0.0-1.0	Blur for transmission
TransmissionMaxLevel	6	1-20	Max level for refracted rays

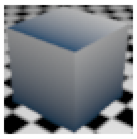
Description

The VGlass surface shader simulates thick transparent glass including the effects of refraction.

By altering the `IndexOfRefraction`, this shader can also be used simulate other transparent refractive materials such as water or diamond.

For thin glass objects modeled as a single surface (such as a window modeled as a single polygon), use the VThinGlass shader instead.

9.9.51 VGradient

 surface shader with color gradient

Illumination model: general

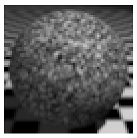
Features: reflections

Parameter	Default Value	Range	Description
Diffuse	`0.5`	`0.0-1.0`	Diffuse reflectivity
DiffuseRoughness	`0.0`	`0.0-1.0`	Diffuse roughness
Specular	`0.5`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.2`	`0.0-1.0`	Specular roughness
SpecularSharpness	`0`	`0-1`	Specular sharpness
Metallic	`0`	`0 or 1`	When 1, use metallic illumination model
Reflection	`0.0`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
Incandescence	`0`		constant emitted intensity
StartColor	`1 1 1`		Color at start location
StartColorBlend	`0.5`	`0-1`	Mix ratio with color attribute
StartAlpha	`1`	`0-1`	Alpha at start location
StartXY	`0 0`		Texture coordinates for start location
EndColor	`0 0 0`		Color at end location
EndColorBlend	`0.5`	`0-1`	Mix ratio with color attribute
EndAlpha	`1`	`0-1`	Alpha at end location
EndXY	`0 1`		Texture coordinates of end location
Projection	`"st"`	type	Projection to use for 2D coordinates
ProjectionSpace	`"shader"`	spaces	Coordinate space for projection
ProjectionTransform	`1 0 0 0` `0 1 0 0` `0 0 1 0` `0 0 0 1`		Transformation applied prior to projection

Description

The VGradient surface shader computes a gradient color pattern using 2D coordinates.

9.9.52 VGranite

 granite

Illumination model: plastic

Features: reflections

Parameter	Default Value	Range	Description
Diffuse	`0.8`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.2`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.1`	`0.0-1.0`	Specular roughness
Reflection	`0.0`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
PatternSize	`0.05`		Size or scale of pattern
PatternOrigin	`0 0 0`		Pattern origin
PatternSpace	`"shader"`	spaces	Coordinate space for shading calculations
VeinFraction	`0.45`	`0.0-1.0`	Fraction that is vein color
VeinFuzz	`0.3`	`0.0-1.0`	Blur for transition region
VeinColor	`0 0 0`		Vein color
VeinLevelsOfDetail	`4`	`1-7`	Levels of detail in pattern
VeinRoughness	`0.5`	`0.0-1.0`	How much each level adds to the pattern
VeinFilterWidth	`0.25`	`0.0-4.0`	Filter width multiplier

Description

The VGranite shader simulates a granite-like material with optional reflections.

9.9.53 VGrid2D

2D grid of lines on plastic

Illumination Model: plastic

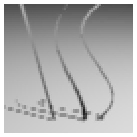
Parameter	Default Value	Range	Description
Ambient	`1.0`	`0.0-1.0`	Ambient reflectivity
Diffuse	`0.5`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.5`	`0.0-1.0`	Specular reflectivity
Roughness	`0.1`	`0.01-1.0`	Specular roughness

LineSpacing	0.1		Distance between line centers
LineWidth	0.01		Line width
LineColor	0 0 0		Line color
ProjectionType	"st"	projections	Projection to use for 2D coordinates
ProjectionSpace	"shader"	spaces	Coordinate space for projection
Transformation	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection

Description:

The VGrid2D surface shader generates an anti-aliased 2-dimensional line grid on a plastic surface.

9.9.54 VHair


surface shader to make curves look like hair

Illumination Model: custom

Parameter	Default Value	Range	Description
Ambient	1.0	0.0-1.0	Ambient reflectivity
Diffuse	0.5	0.0-1.0	Diffuse reflectivity
Specular	0.5	0.0-1.0	Specular reflectivity
Roughness	0.1	0.01-1.0	Specular roughness
SpecularColor	1 1 1		Highlight color
RootColor	0.5 0.5 0.5		Color at base of curve
TipColor	1.0 1.0 1.0		Color at tip of curve
ColorBias	0.5	0.0-1.0	Bias for color interpolation along curve

Description:

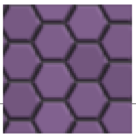
This shader makes a flat curve look like hair by illuminating it as though it were a cylinder.

The diffuse color is based on the `RootColor` and `TipColor` which are linearly interpolated along the curve's length. The result of that interpolation is then multiplied by the curve's color attribute.

See Also

VFur shader

9.9.55 VHexTile


hexagonal ceramic tile with reflections

Illumination model: ceramic

Features: reflections

Parameter	Default Value	Range	Description
Diffuse	`0.5`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.5`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.1`	`0.0-1.0`	Specular roughness
SpecularSharpness	`0.8`	`0.0-1.0`	Sharpness of highlight edge
Reflection	`0.5`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
TileWidth	`0.1`		Width of tile
TileBlur	`0.04`		Blur for antialiasing tile edges
TileColorVary	`0.3`	`0.0-1.0`	Color variation tile-to-tile
MortarWidthFraction	`0.1`		Mortar width as a fraction of tile size
MortarColor	`.1 .1 .1`		Mortar color
MortarDepth	`.1`		Groove depth
TextureSizeXY	`1 1`		Texture size in X and Y direction
TextureOriginXY	`0 0`		Location of left, top texture edge
TextureAngle	`0`		Texture rotation angle in degrees
Projection	`"st"`	type	Projection to use for 2D coordinates
ProjectionSpace	`"shader"`	spaces	Coordinate space for projection

ProjectionTransform	`1 0 0 0`
	`0 1 0 0`
	`0 0 1 0`
	`0 0 0 1`
			Transformation applied prior to projection

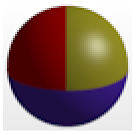
Description

The VHexTile shader produces a 2D pattern of ceramic hexagonal tiles.

See Also

VCeramic surface shader
VBrick2D surface shader
VShinyTile2D surface shader

9.9.56 VLayeredMaterials


surface shader with base material and 3 layers of different materials

Illumination model: plastic or metal

Features: reflections, textures, bump mapping

Parameter	Default Value	Range	Description
Diffuse	`0.7`	`0.0-1.0`	Diffuse reflectivity
DiffuseMap	`""`		Diffuse color map
DiffuseSizeXY	`1 1`		Size of texture map in X and Y
DiffuseOriginXY	`0 0`		Map position of left, top corner
Specular	`0.5`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.2`	`0.0-1.0`	Specular roughness/size
SpecularSharpness	`0.0`		Sharpness of highlight edge
SpecularMetallic	`0`	`0, 1`	Whether surface is metallic
SpecularMap	`""`		Specular color map
Reflection	`1.0`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
BumpMax	`0.0`		Maximum bump height
BumpMap	`""`		Texture map for bump mapping
Projection	`"st"`	type	Projection to use for 2D coordinates
ProjectionSpace	`"shader"`	spaces	Coordinate space for projection

ProjectionTransform	`1 0 0 0`
	`0 1 0 0`
	`0 0 1 0`
	`0 0 0 1`
			Transformation applied prior to projection

aMask	`""`		Texture map for layer mask
aMaskSizeXY	`1 1`		Size of mask in X and Y
aMaskOriginXY	`0 0`		Position of top, left corner of mask

aDiffuse	0.7	0.0-1.0	Diffuse reflectivity
aDiffuseColor	1 1 1		Diffuse color
aDiffuseColorMap	""		Diffuse color map
aDiffuseSizeXY	1 1		Size of texture map in X and Y
aDiffuseOriginXY	0 0		Map position of left, top corner
aSpecular	0.5	0.0-1.0	Specular reflectivity
aSpecularRoughness	0.2	0.0-1.0	Specular roughness/size
aSpecularSharpness	0.0		Sharpness of highlight edge
aSpecularMetallic	0	0, 1	Whether surface is metallic
aSpecularMap	""		Specular color map
aReflection	1.0	0.0-1.0	Reflection strength
aReflectionBlur	0.0	0.0-1.0	Blur for reflections
aBumpMax	0.0		Maximum bump height
aBumpMap	""		Texture map for bump mapping

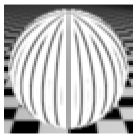
Two more layers - b and c - with parameters identical to layer a parameters.

Description

The VLayeredMaterials shader allows multiple materials to be applied to the same surface. The shader has a base material and up to 3 layer materials whose coverage can be controlled with a texture mask.

All textures and masks share the same underlying 2D coordinate system specified with the `Projection`, `ProjectionSpace`, and `ProjectionTransform` parameters.

9.9.57 VLines


line-drawing surface shader

Parameter	Default value	Range	Description
BlackThreshold	0.0		Diffuse level below which surfaces are shaded with full line coverage
WhiteThreshold	1.0		Diffuse level above which lines are invisible
Background	1 1 1		Explicit background color
BackgroundBlend	0	0.0-1.0	Blend between explict background color and object color

FirstFrequency	400		Stripe frequency
FirstFraction	0.3		Fraction of stripe covered by line
FirstAlpha	1	0.0-1.0	Line coverage (0=no lines)
FirstAngle	0	0-360	Rotation angle in degrees for lines
FirstColor	0 0 0		Explicit line color
FirstColorBlend	1.0	0.0-1.0	Blend between explicit line color and object color
SecondFrequency	400		Stripe frequency
SecondFraction	0.3		Fraction of stripe covered by line
SecondAlpha	1	0.0-1.0	Line coverage (0=no lines)
SecondAngle	0	0-360	Rotation angle in degrees for lines
SecondColor	0 0 0		Explicit line color
SecondColorBlend	1.0	0.0-1.0	Blend between explicit line color and object color

Description

Use the VLines shader to shade a surface by filling it with parallel lines. Up to two non-parallel sets of lines are supported.

Colors for the background and lines can be explicitly specified or based on the object's color attribute.

9.9.58 VMarble

layered marble with reflections

Illumination model: plastic

Features: reflections

Parameter	Default Value	Range	Description
Diffuse	0.7	0.0-1.0	Diffuse reflectivity
Specular	0.3	0.0-1.0	Specular reflectivity
SpecularRoughness	0.1	0.0-1.0	Specular roughness
Reflection	0.5	0.0-1.0	Reflection strength
ReflectionSamples	1	1-256	Number of rays when ray tracing
ReflectionBlur	0.0	0.0-1.0	Blur for reflections
ReflectionName	"raytrace"		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	"current"	spaces	Coordinate space for reflection lookup

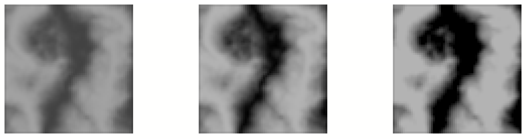
VeinColor	0.0 0.0 0.2		Color of marble veins
VeinSharpness	0.5	0.0-1.0	Contrast adjustment
VeinFraction	0.5	0.0-1.0	Fraction of layer that is vein color
VeinTurbulence	1.0	0.0-3.0	Amount of turbulence to apply
VeinFIlterWidth	1		Filter width multiplier for antialiasing
PatternSize	0.4		Width of one layer of marble
PatternOrigin	0 0 0		Pattern position
PatternAxis	1 0 0		Axis perpendicular to marble layers
PatternSpace	"shader"	spaces	Coordinate space for shading calculations

Description

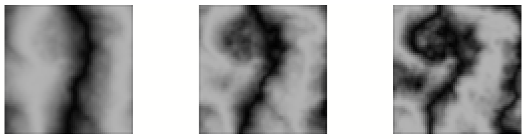
The VMarble shader simulates a marble-pattern material with reflections.

The size or scale of the marble pattern is set with the `PatternSize` parameter. The marble is modeled as alternating layers of a base substrate, whose color is taken from the primitive's color attribute, and a vein layer whose color is set by the `VeinColor` parameter. `VeinFraction` gives the fraction of each pair of layers that is the vein color. The layers are oriented perpendicular to the `PatternAxis`.

`VeinSharpness` determines the sharpness of the edge between the colored vein and the base material:


0.4 0.5 0.6

`VeinTurbulence` controls the how much the marble is distorted by a simulated turbulent flow:


0.5 1.0 1.5

This shader performs anti-aliasing by fading out the marble pattern as the region being shaded grows large. Lower `VeinFilterWidth` values allow more detail to appear, but the additional detail may cause aliasing in animation.

9.9.59 VMetal

metal shader with reflections and color texture map

Illumination model: metal

Features: texture map, reflections

Parameter	Default Value	Range	Description
Diffuse	`0.05`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.95`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.1`	`0.0-1.0`	Specular roughness
Reflection	`0.95`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
ReflectionGroups	`""`		Groups to test for traced reflections
ColorMapName	`""`		Texture map for base color
ColorMapBlur	`0.0`	`0.0-1.0`	Texture blur
OpacityMapName	`""`		Texture map for opacity
OpacityMapBlur	`0.0`	`0.0-1.0`	Texture blur
TextureSizeXY	`1 1`		Texture size in X and Y direction
TextureOriginXY	`0 0`		Location of left, top texture edge
TextureAngle	`0`		Texture rotation angle in degrees
Projection	`"st"`	type	Projection to use for 2D coordinates
ProjectionSpace	`"shader"`	spaces	Coordinate space for projection
ProjectionTransform	`1 0 0 0` `0 1 0 0` `0 0 1 0` `0 0 0 1`		Transformation applied prior to projection

Description

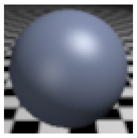
The VMetal shader provides a basic metallic appearance with reflections and optional texture map controlling the base metal color.

The base metal color is taken from the object's color attribute, multiplied by the color texture map if present.

See Also

VRustyMetal shader

9.9.60 VPhong

 plastic surface using the Phong specular model

Illumination model: diffuse plus Phong highlight

Features: reflections, texture map

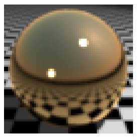
Parameter	Default Value	Range	Description
Diffuse	`0.5`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.5`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.1`	`0.0-1.0`	Specular roughness
SpecularSize	`1`		Size control for Phong highlight
Reflection	`0.0`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup

TextureSizeXY	1 1		Size of texture map in X and Y
TextureOriginXY	0 0		Position of left, top corner
TextureAngle	0.0		Rotation angle in degrees
ColorMapName	""		Color map name
ColorMapBlur	0.0	0.0-1.0	Color map blur
SpecularMapName	""		Specular map name
SpecularMapBlur	0.0	0.0-1.0	Specular map blur
OpacityMapName	""		Opacity map name
OpacityMapBlur	0.0	0.0-1.0	Opacity map blur
BumpMax	0.0		Bump amplitude multiplier
BumpSpace	"shader"		space in which to measure displacement
BumpMapName	""		Bump map name
BumpMapBlur	0.0	0.0-1.0	Bump map blur
Projection	"st"	projections	Projection to use for 2D coordinates
ProjectionSpace	"shader"	spaces	Coordinate space for projection
Transform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection

Description

This shader produces a reflective plastic surface with separate texture maps to control the surface color, opacity and specularity or shininess. An additional texture map is provided for bump mapping. All maps use the same texture coordinates.

9.9.61 VPhysicalMetal


physically plausible metal shader with texture maps

Illumination model: physical metal

Features: texture map, reflections

Parameter	Default Value	Range	Description
Reflectance	0.9	0.0-1.0	Specular reflectivity
Roughness	0.03	0.0-1.0	Specular roughness
Samples	8	1-256	Number of rays when ray tracing

ColorMapName	""		Texture map for base color
OpacityMapName	""		Texture map for opacity
TextureSizeXY	1 1		Texture size in X and Y direction
TextureOriginXY	0 0		Location of left, top texture edge
TextureAngle	0		Texture rotation angle in degrees
Projection	"st"	type	Projection to use for 2D coordinates
ProjectionSpace	"shader"	spaces	Coordinate space for projection
ProjectionTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection

Description

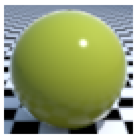
This surface shader provides a physically plausible metallic appearance with optional color and opacity maps.

The metal's color is taken from the standard color attribute. The Reflectance parameter scales the result for both specular highlights and reflections. Roughness controls the blurriness of reflections and the size of specular highlights - larger values produce blurrier reflections and larger highlights. Samples gives the number of rays to trace for reflections. For larger Roughness values, more samples may be needed to produce a smooth result.

See Also

SimpleMetal surface shader
VMetal surface shader

9.9.62 VPhysicalPlastic

 physically plausible plastic with texture maps

Illumination model: plastic

Features: reflections, texture map

Parameter	Default Value	Range	Description
Diffuse	1	0.0-1.0	Diffuse reflectivity
Specular	1.0	0.0-1.0	Specular and reflection multiplier
Roughness	0.03	0.0-1.0	Specular roughness
ReflectionSamples	4	1-256	Number of rays when ray tracing

TextureSizeXY	`1 1`		Size of texture map in X and Y
TextureOriginXY	`0 0`		Position of left, top corner
TextureAngle	`0.0`		Rotation angle in degrees
ColorMapName	`""`		Color map name
SpecularMapName	`""`		Specular map name
OpacityMapName	`""`		Opacity map name
BumpMax	`0.0`		Bump amplitude multiplier
BumpSpace	`"shader"`		space in which to measure displacement
BumpMapName	`""`		Bump map name
Projection	`"st"`	projections	Projection to use for 2D coordinates
ProjectionSpace	`"shader"`	spaces	Coordinate space for projection

Transform	`1 0 0 0`
	`0 1 0 0`
	`0 0 1 0`
	`0 0 0 1`
			Transformation applied prior to projection

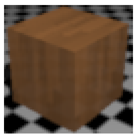
Description

This surface shader provides a physically plausible plastic or dielectric material. The base color is taken from the standard color attribute. The Specular parameter scales the result for both specular highlights and reflections. Roughness controls the blurriness of reflections and the size of specular highlights - larger values produce blurrier reflections and larger highlights. Samples gives the number of rays to trace for reflections. For larger Roughness values, more samples may be needed to produce a smooth result.

See Also

SimplePlastic surface shader
VPlastic surface shader

9.9.63 VPlanks


wooden planks

Illumination model: plastic

Features: reflections

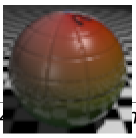
Parameter	Default Value	Range	Description
Diffuse	`0.8`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.2`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.05`	`0.0-1.0`	Specular roughness
Reflection	`0.4`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
PlankWidth	`0.05`		Plank width
PlankHeight	`0.8`		Plank height
PlankVaryColor	`0.3`		Per-plank color variation
WoodColor	`0.68 0.32 0.12`		Color of base wood
VeinColor	`0.08 0.05 0.01`		Color of dark wood vein
GrooveColor	`0.03 0.01 0.01`		Color of groove between planks
GrooveWidth	`0.001`		Width of grooves between planks
GrooveHeight	`0.001`		Depth of grooves between planks
RingScale	`22`		Number of rings per unit
GrainFrequency	`44`		Frequency of the fine grain
GrainFilterWidth	`0.25`		Filter width multiplier
TextureSizeXY	`1 1`		Texture size in X and Y direction
TextureOriginXY	`0 0`		Location of left, top texture edge
TextureAngle	`0`		Texture rotation angle in degrees
Projection	`"st"`	type	Projection to use for 2D coordinates
ProjectionSpace	`"shader"`	spaces	Coordinate space for projection

ProjectionTransform	`1 0 0 0`
	`0 1 0 0`
	`0 0 1 0`
	`0 0 0 1`
			Transformation applied prior to projection

Description:

The VPlanks surface shader produces a pattern of wooden planks useful for rendering wood flooring or paneling.

9.9.64 VPlastic



reflective plastic surface with texture maps for color, opacity, specular and bump

Illumination model: plastic

Features: reflections, texture map

Parameter	Default Value	Range	Description
Diffuse	`0.5`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.5`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.2`	`0.0-1.0`	Specular roughness
Reflection	`1.0`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
TextureSizeXY	`1 1`		Size of texture map in X and Y
TextureOriginXY	`0 0`		Position of left, top corner
TextureAngle	`0.0`		Rotation angle in degrees
ColorMapName	`""`		Color map name
ColorMapBlur	`0.0`	`0.0-1.0`	Color map blur
SpecularMapName	`""`		Specular map name
SpecularMapBlur	`0.0`	`0.0-1.0`	Specular map blur
OpacityMapName	`""`		Opacity map name
OpacityMapBlur	`0.0`	`0.0-1.0`	Opacity map blur
BumpMax	`0.0`		Bump amplitude multiplier
BumpSpace	`"shader"`		space in which to measure displacement
BumpMapName	`""`		Bump map name
BumpMapBlur	`0.0`	`0.0-1.0`	Bump map blur
Projection	`"st"`	projections	Projection to use for 2D coordinates
ProjectionSpace	`"shader"`	spaces	Coordinate space for projection
Transform	`1 0 0 0` `0 1 0 0` `0 0 1 0` `0 0 0 1`		Transformation applied prior to projection

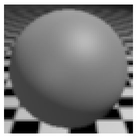
Description

This shader produces a reflective plastic surface with separate texture maps to control the surface color, opacity and specularity or shininess. An additional texture map is provided for bump mapping. All maps use the same texture coordinates.

See Also

V2SidedPlastic shader
VDecal2DPlastic shader
VDecal3DPlastic shader

9.9.65 VRubber

 rubber

Illumination model: rough matte with specular

Features: texture maps

Parameter	Default Value	Range	Description
Diffuse	0.6	0.0-1.0	Diffuse reflectivity
DiffuseRoughness	0.8	0.0-1.0	Diffuse roughness
Specular	0.5	0.0-1.0	Specular reflectivity
SpecularRoughness	0.2	0.0-1.0	Specular roughness
ColorMapName	""		Texture map for base color
ColorMapBlur	0.0	0.0-1.0	Texture blur
SpecularMapName	""		Texture map for specular color
SpecularMapBlur	0.0	0.0-1.0	Texture blur
OpacityMapName	""		Texture map for opacity
OpacityMapBlur	0.0	0.0-1.0	Texture blur
BumpMax	0.0		Maximum bump height
BumpMapName	""		Texture map for bump mapping
BumpMapBlur	0.0	0.0-1.0	Texture blur
TextureSizeXY	1 1		Texture size in X and Y direction
TextureOriginXY	0 0		Location of left, top texture edge
TextureAngle	0		Texture rotation angle in degrees
Projection	"st"	type	Projection to use for 2D coordinates
ProjectionSpace	"shader"	spaces	Coordinate space for projection
ProjectionTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection

Description:

Rubber appearance as a rough matte surface with soft specular highlights.

9.9.66 VRustyMetal

metal with rust spots

Illumination model: metal

Features: reflections

Parameter	Default Value	Range	Description
Diffuse	`0`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.98`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.2`	`0.0-1.0`	Specular roughness
Reflection	`1.0`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
Rustiness	`0.5`		How rusty the surface is
RustColor	`0.27 0.05 0.03`		Base rust color
RustColorVaryHSV	`0.15 0.1 0.1`		Variation in rust hue, saturation, and value
PatternSize	`0.2`		Overall pattern size
PatternOrigin	`0 0 0`		Pattern offset
PatternSpace	`"shader"`	spaces	Coordinate space for shading calculations

Description

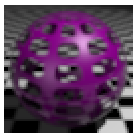
The VRustyMetal surface shader simulates a metallic surface partially covered with rust. Rust-covered regions do not have specular highlights or reflections.

`Rustiness` gives the approximate fraction of the surface covered with rust.

See Also

VMetal shader

9.9.67 VScreen


plastic or metallic surface with round or square holes

Illumination model: plastic or metal

Features: reflections

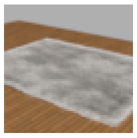
Parameter	Default Value	Range	Description
Diffuse	`0.5`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.5`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.2`	`0.0-1.0`	Specular roughness
Reflection	`1.0`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
Metallic	`0`	`0, 1`	0 for plastic, 1 for metal
HoleSizeXY	`0.7 0.7`		Hole size in X and Y
Round	`0.5`	`0.0-1.0`	0 to 1 value transitioning from a rectangle to an ellipse
TextureSizeXY	`1 1`		Texture size in X and Y direction
TextureOriginXY	`0 0`		Location of left, top texture edge
TextureAngle	`0`		Texture rotation angle in degrees
Projection	`"st"`	type	Projection to use for 2D coordinates
ProjectionSpace	`"shader"`	spaces	Coordinate space for projection

ProjectionTransform	`1 0 0 0`
	`0 1 0 0`
	`0 0 1 0`
	`0 0 0 1`
			Transformation applied prior to projection

Description

This surface shader produces a pattern of holes on a plastic or metallic surface. The hole pattern uses the primitive's standard texture coordinates.

9.9.68 VShadeCarpet


companion surface shader for the instCarpet instancer shader

Parameter	Default Value	Range	Description
Diffuse	1	0.0-1.0	Diffuse reflectivity
Specular	0	0.0-1.0	Specular multiplier
SpecularRoughness	0.2	0.0-1.0	Specular roughness
ShadeLikeCylinder	1	0 or 1	Whether to shade each curve as if it were a cylinder
FiberBounceLight	0.3		Indirect light bouncing from nearby carpet strands
ColorMapName	""		Color map name
ColorMapBlur	0		Color map blur
OpacityMapName	""		Opacity map name
OpacityMapBlur	0.0		Opacity map blur
TextureRepeatX	1		Texture copies in the X direction
TextureRepeatY	1		Texture copies in the Y direction
TextureOriginX	0		Texture origin in X
TextureOriginY	0		Texture origin in Y

See Also

instCarpet instancer shader

9.9.69 VShadowedConstant

constant-colored shader with shadows and optional texture map and reflections

Features: texture map, shadows, reflections

Parameter	Default Value	Range	Description
Intensity	1.0	0.0-1.0	color brightness multiplier
ShadowIntensity	1.0	0.0-1.0	Multiplier for shadow value
ShadowColor	0.0 0.0 0.0		Shadow color modifier

ColorMapName	""		Texture name
ColorMapBlur	0.0	0.0-1.0	Texture blur
TextureRepeatX	1		Copies of texture in X direction
TextureRepeatY	1		Copies of texture in Y direction
TextureOriginX	0.0		Location of left texture edge
TextureOriginY	0.0		Location of top texture edge
TextureProjection	"st"	projections	Projection to use for 2D coordinates
TextureSpace	"shader"	spaces	Coordinate space for projection
TextureTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection
Reflection	0	0.0-1.0	Reflection strength
ReflectionSamples	1	1-256	Number of rays when ray tracing
ReflectionBlur	0.0	0.0-1.0	Blur for reflections
ReflectionName	"raytrace"		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	"current"	spaces	Coordinate space for reflection lookup

Description

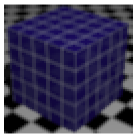
The VShadowedConstant shader produces a constant-colored surface with shadows. This shader only works properly with lights that have a `__unshadowed_Cl` output variable. All AIR light shaders, as well as the light shaders included with MayaMan and RhinoMan provide this variable.

9.9.70 VShadowedTransparent

renders a shadow value into the alpha channel for compositing over a background image

Parameter	Default Value	Range	Description
ShadowIntensity	1.0	0.0-1.0	Shadow multiplier

9.9.71 VShinyTile2D


ceramic tile with reflections

Illumination model: ceramic

Features: reflections

Parameter	Default Value	Range	Description
Diffuse	`0.5`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.5`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.1`	`0.0-1.0`	Specular roughness
SpecularSharpness	`0.8`	`0.0-1.0`	Sharpness of highlight edge
Reflection	`0.5`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
TileSize	`0.1 0.1`		Width of tile
TileStagger	`0.0`		Fraction to offset every other row of tiles
TileColorVary	`0.1`	`0.0-1.0`	Color variation tile-to-tile
GrooveWidth	`0.005`		Width of grooves between tiles
GrooveDepth	`0.01`		Depth of groove between tiles
GrooveColor	`0.91 0.91 0.91`		Groove color
BumpHeight	`0.0`		Maximum bump height
BumpWidth	`0.01`		Average size of noisy bumps
TextureSizeXY	`1 1`		Texture size in X and Y direction
TextureOriginXY	`0 0`		Location of left, top texture edge
TextureAngle	`0`		Texture rotation angle in degrees
Projection	`"st"`	type	Projection to use for 2D coordinates
ProjectionSpace	`"shader"`	spaces	Coordinate space for projection

ProjectionTransform	`1 0 0 0`
	`0 1 0 0`
	`0 0 1 0`
	`0 0 0 1`
			Transformation applied prior to projection

Description

The VShinyTile2D shader produces a 2D pattern of ceramic tiles.

See Also

VCeramic shader
VBrick2D shader

9.9.72 VSketchOutline

surface shader producing uneven, "sketch" outlines when used with AIR's outline capability

Parameter	Default Value	Range	Description
Diffuse	0.0	0.0-1.0	Diffuse reflectivity
DiffuseRoughness	0.0	0.0-1.0	Diffuse roughness
Specular	0.0	0.0-1.0	Specular reflectivity
SpecularRoughness	0.4	0.0-1.0	Specular roughness
Incandescence	1.0	0.0-1.0	Self-illumination
PatternSize	10.0		Size of random pattern
PatternSpace	"raster"	spaces	Coordinate space for pattern
LineWidthMax	2.0		Maximum line width in pixels
LineWidthMin	0.0		Minimum line width in pixels
LineWidthBias	0.5	0.0-1.0	Bias for line width change
LineWidthSharpness	0.5	0.0-1.0	Contrast control for line width change

Description

This shader generates randomly varying line widths for objects rendered with AIR's outline capability.

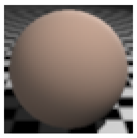
Outlining must be enabled with the maximum line width option set appropriately for this shader to function properly.

By default the shader produces a constant shaded surface with outlines. Set the `Incandescence` parameter to 0 and increase `Diffuse` and `Specular` to obtain a standard matte or plastic illumination model.

See Also

Outlines for illustration

9.9.73 VSkin


skin shader with subsurface scattering and optional texture map

Illumination model: custom

Features: subsurface scattering, texture map

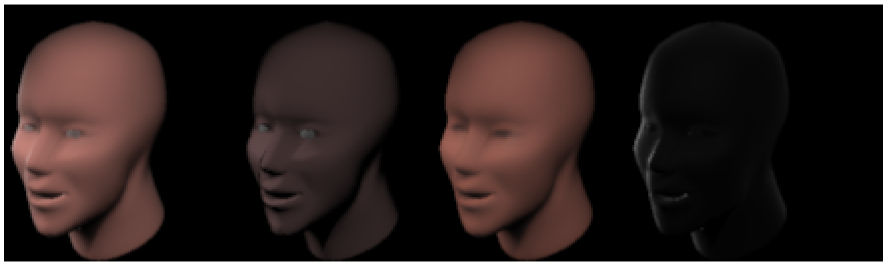
Parameter	Default Value	Range	Description
Ambient	1.0	0.0-1.0	Ambient reflectivity
SingleScatter	1.0		Strength of single scattering
MultipleScatter	1.0	0.0-1.0	Strength of subsurface scattering
Sheen	1.0	0.0-1.0	Sheen strength
SheenColor	1 1 1		Sheen color
ScatterDistanceScale	0.1		Multiplier for ScatterDistance
ScatterDistance	4.82 1.69 1.09		Average scatter distance for r g b
SubsurfaceReflectance	0.63 0.44 0.34		Diffuse reflectance
SampleDistanceOverride	-1		Maximum distance between subsurface samples (-1 for default)
ColorMapName	""		Texture map for base color
ColorMapBlur	0.0	0.0-1.0	Texture blur
TextureSizeXY	1 1		Texture size in X and Y direction
TextureOriginXY	0 0		Location of left, top texture edge
TextureAngle	0		Texture rotation angle in degrees
Projection	"st"	type	Projection to use for 2D coordinates
ProjectionSpace	"shader"	spaces	Coordinate space for projection
ProjectionTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection

Description

The VSkin shader models the apperance of human skin including subsurface scattering with an optional texture map for detail.

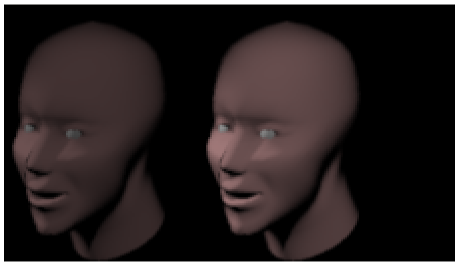
The skin illumination model has three distinct components:

- A *single scattering* component due to light that enters the skin and is then reflected.
- A *multiple scattering* component simuluating light that enters the skin, bounces around, and then exits.
- A *sheen* component simulating the shininess of the skin.


All Single scatter Multiple scatter Sheen

Single Scattering

The `SingleScatter` parameter sets the strength of the single scattering component. This component is also influenced by the `SkinThickness` parameter, which should be appropriate for the scale of the model. Larger values of `SkinThickness` produce a brighter single scattering component.



0.25 1.0

Multiple Scattering

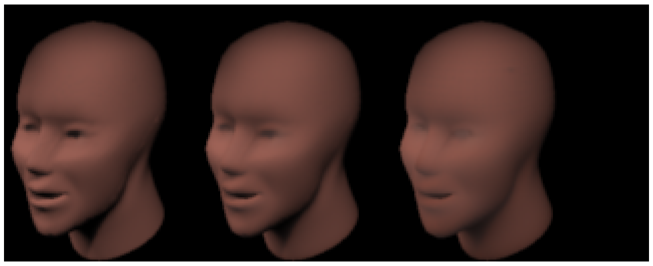
The multiple scattering component is computed using AIR's subsurface scattering capability. When a subsurface scattering object is first encountered, AIR calculates and stores a set of shaded points distributed over the object's surface. You may notice a delay as AIR performs these computations. After the point cache is generated, rendering should proceed relatively quickly.

To use subsurface scattering, assign the same point set handle to all primitives that are part of the current shading object with, for example:

```
Attribute "pointset"
  "string handle" "head.pts"
```

AIR shades points in the point cache from the "outside" of the object as determined by the current orientation. If your object looks unlit with subsurface scattering, try reversing the orientation of the object.

The `ScatterDistance` and `SubsurfaceReflectance` parameters are set to values appropriate for human skin. The `ScatterDistance` parameter is in units of millimeters; use the `ScatterDistanceScale` parameter to scale the scatter distance to an appropriate distance for the size of your model. Larger values of `ScatterDistanceScale` produce more blur and generally take less time to render:



0.4 1.0 3.0

Sheen

The `Sheen` parameter controls the specular component of the skin model.

Base Color

The base color is taken from the object's color attribute, multiplied by the texture map color if any.

Tip

When tweaking parameters for this shader it may be helpful to work on one component at a time by setting the strength of the other components to 0. For example, setting the `SingleScatter` and `Sheen` parameters to 0 will allow you to view the results of only the multiple scattering component.

The skin model is based on Matt Pharr's talk at SIGGRAPH 2001.

See Also

Subsurface scattering
VTranslucent shader

9.9.74 VSmokeSurface

smoke surface shader for volumetric primitives

Illumination Model: particle

Parameter	Default Value	Range	Description
Diffuse	`1.0`		Diffuse reflectivity
Incandescence	`0.0`		Self-illumination value
Ambient	`0.0`	`0.0-1.0`	Ambient reflectivity
OpacityMultiplier	`1.0`		Opacity multiplier
PrimaryScatter	`0.0`	`-1 to 1`	Primary scattering direction
PrimaryScatterWeight	`1.0`	`0.0-1.0`	Relative weight of primary scattering
PrimaryScatterColor	`1.0 1.0 1.0`	color	Per-channel scattering multiplier
SecondScatter	`0.0`	`-1 to 1`	second scattering direction
SecondScatterColor	`1.0 1.0 1.0`	color	Per-channel scattering multiplier
PatternSize	`0.5`		size of smoke pattern
SmokeStrength	`1.0`		smoke variation
SmokeLevels	`3`	`1-9`	octaves of fractal noise
SmokeRoughness	`0.5`		relative strength of successive noise levels
ShadingSpace	`"shader"`	space	space for shading computations

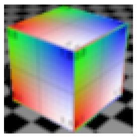
Description

This shader produces a smoke-like pattern using Brownian noise when applied to a volume primitive. The illumination model is the same as that of the particle shader. See that shader's documentation for details on the illumination parameters.

See also

particle surface shader
VCumulusCloud surface shader
volume primitives

9.9.75 VTexturedConstant

 constant-colored surface with texture map

Features: texture map

Parameter	Default value	Range	Description
Incandescence	1.0		Multiplier for constant color
ColorMapName	""		Texture map for base color
ColorMapBlur	0.0	0.0-1.0	Texture blur
OpacityMapName	""		Texture map for opacity
OpacityMapBlur	0.0	0.0-1.0	Texture blur
TextureSizeXY	1 1		Copies of texture in X and Y direction
TextureOriginXY	0 0		Location of left, top texture edge
TextureAngle	0		Texture rotation angle in degrees
Projection	"st"	type	Projection to use for 2D coordinates
ProjectionSpace	"shader"	spaces	Coordinate space for projection
ProjectionTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection

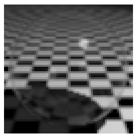
Description

The VTexturedConstant surface shader produces a constant-colored surface using a texture map.

See Also

VShadowedConstant shader
constant shader

9.9.76 VThinGlass

 shader for thin glass without refraction

Illumination model: custom

Features: reflections

Parameter	Default Value	Range	Description
Ambient	`0.01`	`0.0-1.0`	Ambient reflectivity
Diffuse	`0.01`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.5`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.1`	`0.0-1.0`	Specular roughness
SpecularSharpness	`0.8`	`0.0-1.0`	Sharpness of highlight edge
Reflection	`1.0`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
IndexOfRefraction	`1.5`		Index of refraction
Transmission	`1.0`	`0.0-1.0`	Fraction of light transmitted through the glass

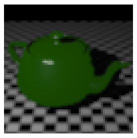
Description

The VThinGlass surface shader is intended for glass surfaces modeled as a single surface in which the effects of refraction are not noticable.

See Also

VGlass shader for thick glass with refraction effects

9.9.77 VToon


cartoon shader

Illumination model: custom

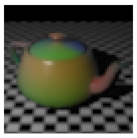
Features: reflections, texture map

Parameter	Default Value	Range	Description
Ambient	1.0	0.0-1.0	Ambient reflectivity
Diffuse	1.0	0.0-1.0	Diffuse reflectivity
DiffuseThreshold	0	-1.0 to 1.0	Cosine of angle for transition from lit to unlit
DiffuseSharpness	0.9	0.0-1.0	Sharpness of transition from lit to unlit
DiffuseLevels	2	1-9	Number of diffuse shading regions
Specular	0.5	0.0-1.0	Specular reflectivity
SpecularSharpness	0.9	0.0-1.0	Sharpness of highlight edge
SpecularRoughness	0.1	0.0-1.0	Specular roughness
Reflection	0.0	0.0-1.0	Reflection strength
ReflectionSamples	1	1-256	Number of rays when ray tracing
ReflectionBlur	0.0	0.0-1.0	Blur for reflections
ReflectionName	"raytrace"		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	"current"	spaces	Coordinate space for reflection lookup
ColorMapName	""		Texture map for base color
ColorMapBlur	0.0	0.0-1.0	Texture blur
OpacityMapName	""		Texture map for opacity
OpacityMapBlur	0.0	0.0-1.0	Texture blur
TextureSizeXY	1 1		Copies of texture in X and Y direction
TextureOriginXY	0 0		Location of left, top texture edge
TextureAngle	0		Texture rotation angle in degrees
Projection	"st"	type	Projection to use for 2D coordinates
ProjectionSpace	"shader"	spaces	Coordinate space for projection
ProjectionTransform	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1		Transformation applied prior to projection

Description

The VToon surface shader provides a basic cartoon shader with optional texture map.

9.9.78 VTranslucent


translucent shader using subsurface scattering with optional texture map

Illumination model: custom

Features: subsurface scattering, reflections, texture map

Parameter	Default Value	Range	Description
Diffuse	`0.5`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.5`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.2`	`0.0-1.0`	Specular roughness
Reflection	`1.0`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
ScatterDistanceScale	`0.3`		Multiplier for ScatterDistance
ScatterDistance	`1.0 1.0 1.0`		Average scatter distance for r g b
SubsurfReflectance	`0.9 0.9 0.9`		Diffuse reflectance
IndexOfRefraction	`1.3`		Index of refraction
SampleDistanceOverride	`-1`		Maximum distance between subsurface samples, or -1 for default
ColorMapName	`""`		Texture map for base color
ColorMapBlur	`0.0`	`0.0-1.0`	Texture blur
TextureSizeXY	`1 1`		Texture size in X and Y direction
TextureOriginXY	`0 0`		Location of left, top texture edge
TextureAngle	`0`		Texture rotation angle in degrees
Projection	`"st"`	type	Projection to use for 2D coordinates
ProjectionSpace	`"shader"`	spaces	Coordinate space for projection
ProjectionTransform	`1 0 0 0` `0 1 0 0` `0 0 1 0` `0 0 0 1`		Transformation applied prior to projection

Description

The VTranslucent surface shader simulates translucent surfaces using subsurface scattering with optional texture map.

When a subsurface scattering object is first encountered, AIR calculates and stores a set of shaded points distributed over the object's surface. You may notice a delay as AIR performs these computations. After the point cache is generated, rendering should proceed relatively quickly.

To use subsurface scattering, assign the same point set handle to all primitives that are part of the current shading object with, for example:

```
  Attribute "pointset"
    "string handle" "head.pts"
```

AIR shades points in the point cache from the "outside" of the object as determined by the current orientation. If your object looks unlit with subsurface scattering, try reversing the orientation of the object.

Use the `ScatterDistance` parameter to define the scatter properties for a particular type of material, e.g., skin. A table of real-world values can be found in Jensen et al "A Practical Model for Subsurface Light Transport" in the SIGGRAPH 2001 Proceedings.

The default scatter distances in the shaders provided with AIR are all in units of millimeters. Use the `ScatterDistanceScale` parameter to adjust the distance to the scale of your object. Larger distances produce more blur and generally take less time to render.

The `SubsurfReflectance` parameter sets the nominal reflectance for subsurface scattering. Real-world values can be found in Jensen et al. Although it is possible to control the color of an object by changing this parameter, the net effect in interaction with the scattering distance parameter can be difficult to predict. Instead, use the primitive's color attribute or a texture map to set the base color, with the `SubsurfReflectance` value set for a particular material type.

Base Color

The base color is taken from the object's color attribute, multiplied by the texture map color if any.

See Also

Subsurface scattering
VSkin shader

9.9.79 VTranslucentMarble

marble with subsurface scattering

Illumination model: custom

Features: subsurface scattering, reflections

Parameter	Default Value	Range	Description
Ambient	1.0	0.0-1.0	Ambient reflectivity
Diffuse	0.5	0.0-1.0	Diffuse reflectivity
Specular	0.5	0.0-1.0	Specular reflectivity
Roughness	0.2	0.0-1.0	Specular roughness
Reflection	1.0	0.0-1.0	Reflection strength
ReflectionSamples	1	1-256	Number of rays when ray tracing
ReflectionBlur	0.0	0.0-1.0	Blur for reflections
ReflectionName	"raytrace"		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	"current"	spaces	Coordinate space for reflection lookup
ScatterDistanceScale	0.3		Multiplier for ScatterDistance
ScatterDistance	1.0 1.0 1.0		Average scatter distance for r g b
SubsurfReflectance	0.9 0.9 0.9		Diffuse reflectance
IndexOfRefraction	1.3		Index of refraction
SampleDistanceOverride	-1		Maximum distance between subsurface samples, or -1 for default
MarblePatternSize	0.4		Size of one marble layer
MarbleVeinColor	0.0 0.0 0.2		Color of marble veins
MarbleSharpness	0.5	0.0-1.0	Sharpness of transition
MarbleVeinFraction	0.4	0.0-1.0	Vein fraction of each layer
MarbleTurbulence	0.5		Amount of turbulence to apply
MarbleAxis[3]	1 0 0		Axis perpendicular to marble layers
MarbleFilterWidth	1.0		Filter width multiplier
MarbleSpace	"shader"	spaces	Coordinate space for shading calculations

Description

The VTranslucentMarble shader simulates translucent marble using subsurface scattering with the same marble pattern as the VMarble shader.

See Also

Subsurface scattering
VTranslucent shader

9.9.80 VWeave

woven pattern

Illumination model: plastic or metal

Features: reflections

Parameter	Default Value	Range	Description
Diffuse	`0.5`	`0.0-1.0`	Diffuse reflectivity
Specular	`0.5`	`0.0-1.0`	Specular reflectivity
SpecularRoughness	`0.2`	`0.0-1.0`	Specular roughness
Reflection	`1.0`	`0.0-1.0`	Reflection strength
ReflectionSamples	`1`	`1-256`	Number of rays when ray tracing
ReflectionBlur	`0.0`	`0.0-1.0`	Blur for reflections
ReflectionName	`"raytrace"`		Name of reflection map or `raytrace` for ray tracing
ReflectionSpace	`"current"`	spaces	Coordinate space for reflection lookup
Metallic	`0`	`0, 1`	0 for plastic, 1 for metal
BumpStrength	`0.5`		Multiplier for displacement values
Columns	`15`		Copies in X direction
ColumnGap	`0.5`	`0.0-1.0`	Fractional gap between each column
ColumnWave	`0.15`		Amplitude for wavy pattern
ColumnRoundness	`0.0`		Cross-sectional rounding
ColumnColor	`.18 .08 .02`		Color of columns
Rows	`10`		Copies in Y direction
RowGap	`0.3`	`0.0-1.0`	Fractional gap between rows
RowWave	`0.2`		Amplitude for wavy pattern
RowRoundness	`0.1`		Cross-sectional rounding
RowColor	`1.00 1.00 0.41`		Color of rows
TextureSizeXY	`1 1`		Texture size in X and Y direction
TextureOriginXY	`0 0`		Location of left, top texture edge
TextureAngle	`0`		Texture rotation angle in degrees
Projection	`"st"`	type	Projection to use for 2D coordinates
ProjectionSpace	`"shader"`	spaces	Coordinate space for projection

ProjectionTransform	`1 0 0 0`
	`0 1 0 0`
	`0 0 1 0`
	`0 0 0 1`
			Transformation applied prior to projection

Description

This surface shader produces a woven pattern.

9.9.81 VWood

solid wood shader

Illumination model: plastic

Parameter	Default value	Range	Description
Diffuse	0.7	0.0-1.0	Diffuse coefficient
Specular	0.3	0.0-1.0	Specular coefficient
SpecularRoughness	0.1	0.0-1.0	Specular roughness
WoodColor	0.98 0.63 0.43		Wood color
WoodGrain	0.3		Graininess of base wood
VeinColor	0.23 0.10 0.05		Wood vein color
VeinGrain	1.0	0.0-1.0	Graininess of the vein
VeinWidth	0.8		Fraction of each ring in vein
VeinFuzz	0.8		Fraction of vein for transition
VeinBias	0.8		Relative sharpness of in and out transitions
GrainFrequency	200		Frequency of fine grain
GrainDepth	0.001		Depth of grain
RingFrequency	15		Frequency of wood rings
RingVary	1		Variation in ring width
TrunkVary	0.1		Deviation of trunk from a straight line
TrunkVaryFrequency	2.0		Frequency of trunk deviation
AngleVary	0.3		Variation in rings around the central axis
AngleVaryFrequency	2.0		Frequency for angular variation
PatternSize	1		Overall scale of pattern
PatternOrigin	0 0 0		Pattern offset
PatternAxis	0 0 1		Axis of trunk
PatternSpace	"shader"	spaces	Coordinate space for calculations

Description

The VWood shader simulates a solid wood texture with grain throughout.

`WoodGrain` and `VeinGrain` control the graininess of the base wood and the vein of darker wood respectively. `VeinWidth` gives the fraction of each ring that is part dark material.

`VeinFuzz` is the fraction of `VeinWidth` devoted to the transition between light and dark wood. `VeinBias` is the relative sharpness of the transition entering and leaving the dark material. A value of 0.5 results in equal transition regions on both sides.

See Also

VPlanks shader

9.10 Volumes

Volume shaders simulate volumetric effects such as smoke or fog.

fog	simple fog shader defined by the RI Spec
VFog	fast volumetric fog
VSmoke	volumetric smoke

9.10.1 fog

simple fog shader defined by the RenderMan Interface Specification

Parameter	Default Value	Range	Description
distance	`1.0`		distance beyond which fog completely obscures
background	`0 0 0`		fog color

Description

The fog volume shader implements a simple fog function fading to a background color with distance. This shader does not show the effects of light and shadow on the fog. The VFog shader provides such volumetric effects.

See Also

VFog shader

9.10.2 VFog

fast volumetric fog

Parameter	Default Value	Range	Description
StepSize	`0.1`		Step size for ray marching
LightDensity	`0.1`	`0.0-1.0`	Intensity with which fog reflects light
FogColor	`1 1 1`		Fog color
OpacityDensity	`0.0`	`0.0-1.0`	Tendency of fog to block light
AttenuateShadowRays	`0`	`0, 1`	If 1 fog will attenuate shadow rays
LightIndirectRays	`0`	`0, 1`	If 1 fog contributes to indirect rays

Description

The VFog shader produces a volumetric fog effect using AIR's builtin `foglight()` function.

9.10.3 VSmoke

volumetric smoke shader

Parameter	Default Value	Range	Description
UnitSize	5.0		Overall scale of shaded volume
OpacityPerUnit	0.5	0.0-1.0	Extent to which smoke obscures objects objects behind
ReflectancePerUnit	0.5	0.0-1.0	Extent to which smoke reflects light towards viewer
StepsPerUnit	0.1		Step size for ray marching
MaxSteps	200	1+	Maximum number of ray marching steps
SmokeVary	0		Strength of smoke variation
SmokeOctaves	3	1-7	Octaves of noise for smoke variation
SmokeFrequencyPerUnit	10.0		Smoke frequency per unit

Description

Simple volumetric smoke shader using ray marching.

The `UnitSize` parameter controls the overall scale of the shaded volume. The `UnitSize` setting should be roughly 10-100% of the overall dimension of the volume in the scene. By adjusting the `UnitSize` parameter, the same smoke parameters can used for objects and scenes of different scales.

`OpacityPerUnit` gives the extent to which the smoke obscures objects behind it.

`ReflectancePerUnit` gives the extent to which the smoke reflects light toward the viewer.

`StepsPerUnit` gives the number of ray marching steps to perform per UnitSize distance. More steps take longer to render but produce better results. `MaxSteps` gives the maximum number of steps to march for any one shader call.

If `SmokeVary` is greater than 0 the smoke will be modulated by some Brownian noise to produce an non-uniform effect.

This shader is based heavily on the smoke shader in section 12.6 of Advanced RenderMan by Apodaca and Gritz.

9.11 DarkTree Shaders

DarkTree is a visual shader creation tool from Darkling Simulations:

> "DarkTree 2.0 is an advanced procedural shader authoring tool. Its visual flow-based editor lets you interactively create photo-realistic procedural materials, surface shaders, and animated effects. DarkTree 2.0 includes 100 procedural components that can be combined to generate almost any texture or surface effect you need."

An evaluation version of DarkTree is available from the Darkling Simulations website.

Using DarkTree Shaders

Darkling Simulations has developed a DSO shadeop (a plugin for the shading language) that allows DarkTree shaders to be used with AIR (the shaders created by DarkTree are not directly compatible with with shading language used by AIR).

To use DarkTree shaders with AIR, you need to have the DarkTree DSO (called a simbiont by Darkling Simulations) installed. AIR version 4.1 and later includes the DarkTree DSO on Windows. If you are using an earlier version of AIR, you will need to install the DSO.

AIR comes with two special shaders for use with DarkTree shaders.

The DarkTreeSurface shader allows any DarkTree shader to be used as a surface shader for AIR.

The DarkTreeDisplacement shader allows any DarkTree shader with bump features to be used as a displacement shader for AIR.

The Air distribution includes several sample DarkTree shaders in the subdirectory:

```
viztools\darktree
```

Installing the DarkTree DSO

The installer for Air on Windows should automatically install the DarkTree simbiont. If that appears to have failed, follow the manual installation instructions below:

The simbiont files are part of the Air distribution. The only other step required is to define a `SIMBIONT_RM_COMPONENTS` environment variable to point to the location of the components used by the simbiont. Those components are located in the

```
shaders\simbiontrm
```

subdirectory of your Air installation.

To create an environment variable on Windows 2000/NT/XP:

- Right click the My Computer icon and choose Properties.
- Select the Advanced tab and click Environment Variables.
- Click New to create a new environment variable and enter the variable name and value.

The RMSimbiont archive includes many sample shaders as well as the plugin. The DarkTree plugin will search for DarkTree shaders in the path defined by the SHADERS and/or SIMBIONT_RM_SHADERS environment variables. Add the directory containing the DarkTree shaders to one of these variables. If you have the DarkTree application installed, add the DarkTree directory to the SIMBIONT_RM_SHADERS variable as well.

9.12 Layered Shaders and Shader Networks

Air 11 introduces the ability to use multiple shaders to compute the shading for surfaces, displacements, imagers, and environments.

This new capability provides a simple solution to several common shading tasks that would otherwise require complex custom shaders:

Decaling: apply an arbitrary number of texture maps to an object's color and feed the result into any surface shader.

Extra output variables: add an arbitrary number of additional shaders to an object to produce extra output values without affecting the normal shading of the object.

Assign different different shaders to different sections of an object: simple shader compositing allows existing surface shaders to be selectively applied to sections of a surface.

Replicate application shader networks: create arbitrary shader networks by connecting pre-compiled components.

See below for a description of how to use multiple shaders to address each of these tasks. Sample shaders and rib files can be found in

```
$AIRHOME/examples/layers
```

Simple Color Layers

Here's a simple RIB fragment that assigns multiple surface shaders to an object:

```
Surface "ColorMap" "string ColorMapName" "grid.tx"
Surface "+VMarble"
```

Prepending the shader name with "+" tells the renderer to append the shader to the list of shaders assigned to the object. The shaders are executed in the order in which they are assigned. Subsequent shaders inherit the current shading state (including the values of global variables) from shaders previously executed. Here's an implementation of the ColorMap shader that uses a texture map to set the object's color value:

```
generic ColorMap(
  string ColorMapName = "";
  float SetColor = 1;
  output varying color __Color = 0;
)
{
   if (ColorMapName!="") {
      __Color = color texture(ColorMapName,s,t);
      if (SetColor==1) Cs=__Color;
   }
}
```

In the RIB fragment given above, the VMarble shader will use the color map result as one of the input colors for the marble pattern. No modification of the VMarble shader is required.

The new generic shader type is compatible with any other shader type. Generic shaders can be used to construct re-usable components for layered and networked shaders.

Sample rib: `colormap.rib`

Decals

The `layers` example directory includes a `LayerDecal` shader that applies a masked color to a surface. Simple sequential shader assignment allows an arbitrary number of decals to be applied to a surface without constructing a complex custom surface shader:

```
Surface "LayerDecal"
  "ColorMapName" "sitex.tx"
  "OriginXY" [.3 .67]
```

```
    "SizeXY" [-.3 -.3]
    "DecalColor" [1 1 1]

  Surface "+LayerDecal"
    "ColorMapName" "sitex.tx"
    "OriginXY" [.25 .6]
    "SizeXY" [-.25 -0.25]
    "DecalColor" [1 0 0]

  Surface "+plastic"
```

Sample rib: `decals.rib`

Extra Passes

Another simple use for multiple shaders is adding extra output values to a rendering pass. Air 11 includes a new genOcclusion shader that can be added to any surface to produce an occlusion pass (stored in a `__occlusion` output variable). Sample usage:

```
  Display "extrapass.tif" "framebuffer" "rgba,color __occlusion"

  ...

  Surface "VMarble"
  Surface "+genOcclusion" "setshaderoutput" 0
```

Sample rib: `extrapass.rib`

Shader Compositing

Air's multi-shader support includes an option to composite one shader on top of another. Here's an example that adds a plastic decal to a metallic surface:

```
  Surface "VMetal"
  Surface "*VPlastic" "OpacityMapName" "sitex.tx" "ColorMapName"
  "square.tx"
```

The * before the shader name tells Air to blend the output color from the VPlastic shader with the output color of the VMetal shader using the output opacity from the VPlastic shader. Any output variables shared by the two shaders will also be composited using the output opacity.

Sample rib: `compshaders1.rib`

The above example shows how to composite a shader with controls for its output opacity. For surface shaders without opacity controls, an auxiliary shader can be used to control the opacity. Here is a simple generic shader that sets the input opacity value:

```
  generic OpacityMap(
    string OpacityMapName = "";
    float SetOpacity = 1;
    output varying color __Opacity = 0;
  )
  {
     if (OpacityMapName!="") {
        __Opacity = color texture(OpacityMapName,s,t,"fill",-1);
```

```
      if (comp(__Opacity,1)==-1) __Opacity=comp(__Opacity,0); // handle
single channel texture
      if (SetOpacity==1) Os=__Opacity;
   }
}
```

This shader allows any surface shader that modulates its output color by the input opacity to be composited over another shader. Most surface shaders modulate their output color in this way.

Here is how the OpacityMap shader might be used to add a marble decal to a 2D grid:

```
Surface "VGrid2D"
Surface "+OpacityMap" "OpacityMapName" "sitex.tx"
Surface "*VMarble"
```

To optimize the evaluation of composited shaders, Air will skip any shader layer whose input opacity value is 0.

Sample rib: `compshaders2.rib`

Shader Networks

Multiple shaders can be used to re-construct a general shader network by defining connections among the shaders assigned to an object. Connections are defined by including additional parameters in the shader declaration. First, each shader declaration is assigned a name using:

`"string layername" "`*`componentname`*`"`

Naturally `layername` does not need to be an actual parameter of the shader.

Each connection is defined with an additional string parameter:

`"string connect:`*`toparametername`*`" "`*`fromlayer`*`:`*`fromparametername`*`"`

The connection transfers the value of *fromparametername* in layer *fromlayer* to the target parameter in the current shader. The source parameter must be an output variable of the source shader or one of the global shader variables Ci and Oi. The target parameter is normally one of the shader's input parameters, but it can also be the global input color (Cs) or input opacity (Os). The source and target parameters must have the same type.

Sample usage:

```
Surface "ColorMap"
  "ColorMapName" "grid.tx"
  "layername" "gridlayer"
  "SetColor" 0

Surface "+VMarble"
  "connect:VeinColor" "gridlayer:__Color"
```

In this example, the marble vein color is taken from the ColorMap shader's `__Color` output variable.

The network shader capability is used by the Air Stream plug-in for Maya to translate Maya's shader networks for rendering with Air.

Sample rib: `net1.rib`

Limitations

User primitive data is currently not passed to objects with multiple surface shaders.

9.13 The Shading Language

The shading language used by AIR is a superset of the shading language described in the RenderMan® Interface Specification 3.2.

The shading language is similar to C, with the addition of special types and functions for shading and lighting calculations. You can find out more about the RenderMan® shading language in the following sources:

The RenderMan® Interface Specification 3.2

The RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics by Steve Upstill.

Advanced RenderMan: Creating CGI for Motion Pictures by Anthony A. Apodaca and Larry Gritz.

Derivatives in AIR

Several functions in the shading language rely on the ability to take derivatives of arbitrary expressions over a surface element. The built-in functions Du() and Dv() are supposed to give the change in an expression across a surface element in the u- and v-parametric directions, respectively. They are also supposed to give the change between shading samples.

Because AIR does not shade along the u- and v-axes, it interprets derivatives differently for efficiency. Shading is performed at samples which are distributed equally in screen space. `Du(e)` returns the change in e between adjacent samples in the horizontal direction, and `Dv(e)` returns the change between samples vertically. `du` and `dv` are always 1.

For the common use of derivatives to generate filter widths for anti-aliasing, this gives the desired result. E.g.,

```
  fwidth=abs(Du(e)*du)+abs(Dv(e)*dv);
```

is a good estimate of how much e changes between shading samples.

This implementation of derivatives obviously causes problems for shaders that expect `Du()` or `Dv()` to be aligned with their respective parametric axes. Shaders that rely on the length of `du` or `dv` also will not work properly.

An exception is made for the global variables `dPdu` and `dPdv`, which are always smooth estimates of the change in P along the u- and v-axes respectively.

Derivatives and Varying Conditionals

Like other renderers AIR interprets derivatives as differences between adjacent shading samples. When evaluating derivatives, AIR executes shading language instructions on a virtual SIMD (single-instruction, multiple data) machine: there is one path of execution, and each instruction is executed at multiple shading locations. This implementation ensures that derivative values are always available. However, it imposes two restrictions:

1. Derivatives must not be taken within varying conditionals.
2. Derivatives must not depend on values that are calculated using varying conditionals.

Functions that take derivatives either explicitly or implicitly are: Du(), Dv(), area(), calculatenormal() and Deriv().

Dynamic Shadeops (DSOs)

The shading language can be extended by writing custom functions in a programming language such as C which can be called from a shader. A DSO must be compiled to a DLL on Windows or a shared object under Linux, and the resulting file placed in the shader search path. The DSO location should also be included in the search path for header files given to the shading compiler.

AIR will look for a DSO function in a file with the same name or an abbreviated version of the function name.

A sample DSO with source code can be found in `$AIRHOME/examples/shadeop`.

9.14 Shading Language Extensions

This section documents additional shading language functionality supported by Air as well as Air-specific extensions to some of the standard shading language functions.

Variables

The following additional variables are available inside light shaders:

`Ns` and `angles` are read-only variables holding the normal and angle parameters of the calling `illuminance()` loop.

`ss` and `tt` are read-only variables containing the standard texture coordinate values (s,t) at the current shading location.

Structures

Air 10 and later support user-defined data structures as described here.

Types

Air 11 adds new standard data types integer and boolean:

`integer`

An integer variable stores whole signed or unsigned integers. For backwards compatibility, integer parameters are 100% compatible with floats. The compiled shader file includes an integer flag for parameters that an application can use to detect integer parameters and provide an appropriate user interface.

`boolean`

The boolean variable type stores a true or false value. Air 11 also adds reserved words `true` and `false` that are the only valid constants that may be assigned to a boolean variable. A boolean variable may be used anywhere a boolean value is expected.

Boolean parameter values in a RIB file should be either 0 (false) or 1 (true).

The compiled shader file includes a boolean flag for parameters which can be used by an application to detect boolean parameters and provide an appropriate user interface.

Functions

Illumination

areashadow
blinn
brushedspecular
caustic
cooktorrance
diffuse
foglight
indirectdiffuse
specular

Mapping

environment
shadow
texture

3D point sets

bake3d
collect
subsurfacescatter
texture3d

Ray tracing

fulltrace
gather
isindirectray
isphoton
isshadowray
occlusion
rayhittest
rayinfo
raylevel
trace
visibility

Patterns

voronoi

Procedure and Instancer shaders

meshinfo
ribprintf

DSO Shading Functions

dictionary
str_replace_last_number

Deprecated functions

irradiancecache

9.14.1 User Structures

AIR 10 introduces user-defined data structures for the shading language.

Custom record types can be created with the new structure keyword. Here's a simple example:

```
structure Layer {
   string Map;
   float Weight;
   color Tint;
}
```

For the most part user-defined types can be used anywhere a simple type is valid. Individual fields within a type are accessed using the following notation:

variablename.fieldname

For example:

```
Layer First;
First.Map = "grid.tx";
First.Weight = 1.0;
First.Tint = color(1,1,1);
```

A user type variable may also be initialized using a list of values. The above assignments could also be made using:

```
First = {"grid.tx", 1.0, color(1,1,1) };
```

User types may be passed as function parameters but may not be used as function return types.

Variables of the same user type are assignment compatible:

```
void CopyLayer(Layer from; Layer to)
{
  to = from;
}
```

A user type variable can be part of the shader parameter list, in which case each field of the user type is exported as a separate parameter. For example, given the following shader declaration:

```
surface Layers(Layer First = {"", 1.0, color(1,1,1)};)
{...}
```

the slbtell utility would return:

```
Surface "Layers"
  "string First.Map" ""
  "float First.Weight" [1]
  "color First.Tint" [1 1 1]
```

A complete sample shader with source code can be found in

```
$AIRHOME/examples/userstruct
```

This excerpt from the main shader code illustrates some of the advantages of user-defined structures:

```
surface DiffuseLayers(
  ...
```

```
    Layer First = LAYER_DEFAULT_VALUES;
    Layer Second = LAYER_DEFAULT_VALUES;
    Layer Third = LAYER_DEFAULT_VALUES;
  )
  {
    color C = 0;
    C = ApplyLayer(First, C);
    C = ApplyLayer(Second, C);
    C = ApplyLayer(Third, C);
    ...
```

1. The code is short and easy-to-understand: the higher-level functionality of adding layers stands out while the low-level details are encapsulated by the `Layer` type.

2. The code does not rely on complicated preprocessor definitions. It uses only one simple `#define` (for the default values).

3. The main source code is independent of the exact defintion of the `Layer` type and the operations of the `AddLayer` function. Adding a new field to the `Layer` type requires no modification to the main code.

Limitations

Arrays of user-defined type variables are not supported.

9.14.2 areashadow()

```
color areashadow(point P, normal N, float angle,
                 output varying point Psource,
                 "falloff", falloff,
                 "samples", samples,
                 "shadow, traceshadows)
```

The `areashadow()` function is valid only within a light source and is only useful in a light source attached to an area light. The function samples the geometry attached to an area light, using ray-tracing for shadows. The function returns a color giving the fraction of the sample location (defined by P, N, and angle) that is in shadow. `Psource` returns a carefully calculated virtual light origin usable in an illuminate statement. A typical usage would be:

```
light myarealight(float intensity=1; color lightcolor=color(1,1,1); float
falloff=2; float samples=64)
{
   point Psource;

   float inshadow = areashadow(Ps, Ns, angles, Psource,"falloff", falloff,
"samples", samples);

   illuminate(Psource) {
      Cl = (intensity / pow(length(L), falloff) ) * lightcolor * (1-
inshadow);
   }
}
```

In order to calculate an appropriate `Psource`, `areashadow()` requires information about the falloff in light intensity with distance, which can be set with the optional "falloff" parameter. The default value is 2. The optional `"samples"` parameter gives the number of rays used to sample the area light. If the area light geometry is two-sided, light will be cast from both sides of the surface; if the area light primitive is single-sided, light is cast from the front-facing side, as determined by the orientation.

`areashadow()` is a more efficient way to render area lights than methods that rely on supersampling a light because it avoids multiple passes through the light shader and the corresponding illuminance loops. The function is, however, only an approximation. It works best on convex light shapes: disks, rectangles and other convex polygons, spheres, etc. For complex shapes better results may be obtained by decomposing the shape into simpler sub-shapes which are each defined as a separate area light.

9.14.3 bake3d()

```
bake3d(string filename, string channels, point P, normal N, "float radius",
radius, "float radiusscale", radiusscale, token,value,...);
```

Bake3d generates a 3D point cache with an arbitrary list of per-point values. A cache file created with `bake3d()` can be queried with `texture3d()`. If a cache file is generated with the `collect()` statement, it can be read during the same rendering used to create it.

The first parameter gives the file name for the cache file. Each call to `bake3d()` adds a new point to the cache file specified by the *filename* parameter. An arbitrary list of user-supplied values is associated with each position and normal entry. If no applicable normal is available, the normal parameter should be set to 0.

The *channels* parameter can be used to select a subset of the user-supplied values to save to a file.

Each entry also has an associated radius that is automatically computed based on the location value P. An optional `"radius"` parameter can be used to set an explicit radius value. The optional `"radiusscale"` parameter acts as a multiplier for the associated radius.

Points can be added to an existing file by including an `"append"` parameter with a float value of 1.

Example:

```
  normal bent_normal;
  color occ=occlusion(P,Nf,PI/2,bent_normal);
  bake3d(cachename,P,Nf, "occlusion", occ, "bent_normal", bent_normal);
```

The `collect()` can be used to bake values for points distributed over an entire surface. By passing a maxsearchdistance of 0 to `collect()`, the normal automatic generation of a subsurface scattering cache can be suppressed.

See Also
- 3D textures
- `texture3d()`
- `collect()`
- Bake3d surface shader

9.14.4 blinn()

```
color blinn([string category,] normal Nf, vector V, float eccentricity,
float rolloff)
```

The `blinn()` function computes a specular highlight using the popular Blinn illumination model.

The eccentricity parameter (between 0 and 1) determines the overall size of the highlight.

The rolloff parameter controls how rapidly the highlight intensity diminishes. If rolloff is less than 0, its absolute value is treated as the index of refraction of the material (for compatibility with SoftImage).

The `blinn()` function also provides a `"channels"` output variable, an array of colors, holding per-light specular values indexed by a light shader's `__channel` output variable. A `"nchannels"` output variable gives the number of valid channels.

See Also

VBlinn surface shader

9.14.5 brushedspecular()

```
color brushedspecular([string category,] normal Nf, vector xdir, vector
ydir, vector V, float xroughness ,float yroughness, float gloss)
```

The `brushedspecular` function computes an anisotropic specular highlight. The `xroughness` and `yroughness` parameters control the highlight size along the `xdir` and `ydir` tangent vectors respectively. The `gloss` parameter provides an inverse exponent applied to the final result.

When gloss is set to 1, the `brushedspecular` illumination model differs from Greg Ward's model only in omitting division of the result by `(4*xroughness*yroughness)`.

9.14.6 caustic()

```
color caustic(point P, normal N)
```

Returns the result of a photon map lookup at point P.

9.14.7 collect()

```
collect(point P,
        float maxsearchdist,
        float maxsampledist,
        parameterlist) {
 statements;
}
```

AIR exposes the underlying point set query function used by `subsurfacescatter()` in the form of a new `collect()` shading language statement. `collect()` executes the body statements once for each point in the object's point set within `maxsearchdist` of P. Like the `subsurfacescatter()` function, `collect()` generates a point set the first time it is called (if a point set cache file is not used).

The parameter list can contain one or more of the following output parameters (similar to the usage for `gather`):

Type and Name	Description
`color surface:Ci`	sample irradiance
`point primitive:P`	sample position
`normal primitive:N`	sample normal
`float area`	area represented by the sample
`float attenuation`	attenuation function to prevent incorrect scattering for non-convex shapes
`float distance`	distance to sample point

9.14.8 cooktorrance()

```
color cooktorrance(normal N, vector V, float roughness, color ior, color
absorption)

If a positive absorption value is provided, the material is assumed to be
metallic and the corresponding fresnel equation is used.  For non-metals,
use an absorption value of 0.
```

9.14.9 dictionary()

```
float dictionary(string dictname, string keyname, float fvalue);
float dictionary(string dictname, string keyname, color cvalue);
float dictionary(string dictname, string keyname, string svalue);
```

The dictionary function alllows one to look up a set of values based on a name, like looking up a word's definition in a dictionary.

The dictionary function looks for a text file named *dictname* and tries to load a list of token value pairs, one per line. The token values should look like

```
LightBlue 0.6 0.6 1
```

The `dictionary()` call returns 0 if it succeeds. If an error occurs, the function returns one of the following error codes:

-1: unable to find or read dictionary file
-2: key name not found
-3: unable to convert value to the requested type

Examples and source code:

```
$AIRHOME/examples/shadeop/dictionary
```

9.14.10 diffuse()

```
color diffuse([string category,] normal Nf, "roughness", roughness, ...)
```

Diffuse roughness

The `diffuse()` function supports an optional `"roughness"` input parameter of type float giving the roughness used in the Oren and Nayer illumination model for rough diffuse surfaces such as clay. The roughness parameter defaults to 0, which is the value for the regular diffuse computation. The valid range for roughness is 0 to 1.

Diffuse output variables

The `diffuse()` function provides the following optional color output variables

`"unshadowed"`	`color`	diffuse lighting without shadows for all non-indirect lights
`"shadow"`	`color`	shadow value complementing the unshadowed output value
`"indirect"`	`color`	indirect lighting
`"channels"`	`color array`	per-light diffuse values (indexed by a light's `__channel` output variable)
`"unshadowedchannels"`	`color array`	per-light unshadowed diffuse values (indexed by a light's `__channel` output variable)
`"nchannels"`	`float`	number of valid channels

The unshadowed output value can only be computed correctly for lights that export a color `__unshadowed_Cl` output variable containing the light intensity without shadows. The shaders for AIR, MayaMan, and RhinoMan all provide this variable.

The indirect lighting output is computed by summing the diffuse contributions from lights that export a float `__indirectlight` variable set to 1. The indirect shader for indirect diffuse illumination and the envlight shader for ambient occlusion are both categorized as indirect lights.

The shadow output value is computed so that the following relation holds:

```
diffuse = unshadowed * (1-shadow) + indirect
```

Shaders can test a `DIFFUSEEXT` symbol to see if the shading compiler supports these extensions.

The extra output variables from diffuse() are typically used to provide extra output variables to a shader for multipass rendering. The expanded matte shader included with AIR illustrates typical shading language usage:

```
surface matte(
   float Ka = 1, Kd = 1;

   output varying color __ambient = color(0);
   output varying color __diffuse = color(0);
   output varying color __diffuse_unshadowed = color(0);
   output varying color __shadow = color(0);
   output varying color __indirect = color(0);
)
{
   normal Nf;

   Nf = faceforward(normalize(N), I);

   Oi = Os;
```

```
    __ambient = Oi * Cs * Ka * ambient();

    color diffuse_weight = Oi * Cs * Kd;

    __diffuse = diffuse_weight *
                diffuse(Nf, "indirect", __indirect,
                        "unshadowed", __diffuse_unshadowed,
                        "shadow", __shadow);

    __indirect *= diffuse_weight;
    __diffuse_unshadowed *= diffuse_weight;

     Ci = __ambient + __diffuse;
}
```

See Also

Multipass rendering
Light Channels

9.14.11 environment()

```
color environment(mapname, ...)
color environment("raytrace", ...)
color environment("environment", ...)
```

Optional Input Parameters:

Type and Name	Description
`float bias`	offset to prevent incorrect self-intersections
`float blur`	blur factor for fuzzy reflections
`string label`	user-assigned name for rays cast by this call
`float maxdist`	maximum distance to search for intersecting objects
`float samples`	number of rays to trace when ray tracing or samples to use for map access
`string subset`	restricts intersection tests to members of the specified groups
`vector majoraxis`	tangent vector for anisotropic reflections
`float majorblur`	blur in majoraxis direction for anisotropic reflections
`float minorblur`	blur in minor axis direction for anisotropic reflections
`color weight`	weight or importance of this environment call for importance-based ray tracing

Ray Tracing Reflections

If the file name passed to an environment call is `"raytrace"` or `"reflection"`, Air will use ray tracing to return an environment value. The standard "blur" and "bias" parameters to environment() have approximately the same meaning when ray tracing is used.

Sampling the Scene Environment

If the file name is the special value "environment", Air will sample the current environment shader. The blur parameter is interpreted as the half-angle (in radians) of a cone of directions over which to sample the environment, and the samples parameter can be used to super-sample the region.

Optional Output Parameters

The environment() function accepts an optional "alpha" output parameter that returns the next channel after the channels returned by the environment() call when an environment map is used. When ray tracing is used to compute the returned value, the alpha value returns the fraction of the sampled environment cone occluded by reflected objects. The alpha value returned by the ray tracer can be used to composite tracing results with a background environment map.

In AIR 9 and later environment() provides a "background" output parameter of type color that stores the contribution of the background environment map or color for reflection rays.

Beginning with AIR 9 environment() can return arbitrary output variables from the surface shaders of ray-traced primitives in the same manner as the gather() construct. For example:

```
color renv;
color Crefl = environment("raytrace",R,"surface:__environment",renv);
```

BRDF Sampling

Air 13 introduces a new option to importance sample the environment based on a BRDF. When sampling based on a BRDF, the environment() function should be passed the shading normal vector instead of the reflected vector. BRDF parameters are passed as extra parameters to the environment() call prefixed with "brdf:". Here's an example:

```
Cr = color environment("raytrace", Nf,
                        "samples", Samples,
                        "weight", weight,
                        "brdf", "cooktorrance",
                        "brdf:roughness", Roughness,
                        "brdf:ior", ior,
                        "brdf:absorption", absorption);
```

Currently only ray traced environment evaluation supports BRDF sampling. The following BRDFs are supported:

lambert
standard diffuse() BRDF, no extra parameters

air
BRDF for the builtin specular() function, one extra parameter "brdf:roughness"

blinn
BRDF for the builtin blinn() function, two extra parameters: "brdf:eccentricity", "brdf:rolloff"

ward
BRDF for the bultin brushedspecular() function, two parameters: "brdf:xroughness", "brdf:yroughness"

`phong`
BRDF for the builtin `phong()` function, one parameter "brdf:exponent"

`cooktorrance`
BRDF for the `cooktorrance()` function. Three parameters: "brdf:roughness", "brdf:ior", "brdf:absorption"

See Also

`trace()`
Environment shaders
BRDF Reflection Sampling

9.14.12 foglight()

```
color foglight(point from; point to; float stepsize,
               color lightdensity; color opacitydensity;
               output color opacity);
```

The foglight function integrates an illuminated uniform fog function between the `from` and `to` points. Only light shaders with a `__foglight` output parameter set to 1 or a `__category` parameter that includes "foglight" are considered by this function. The stepsize parameter gives the step size for marching along the ray; lower step sizes take more time but produce more accurate results.

The function returns the cumulative illumination, and the `opacity` variable returns the cumulative opacity along the ray segment.

9.14.13 fulltrace()

```
float fulltrace(point pos, vector dir,
                output color hitcolor,
                output float hitdist,
                output point Phit,
                output normal Nhit,
                output point Pmiss,
                output vector Rmiss);
```

Traces a ray from `pos` in direction `dir`. If the ray hits any object, `hitdist` is the distance to the closest object and `Phit` and `Nhit` contain the position and normal at that point. If no object is hit, `hitdist` will be large, and `hitcolor` will be (0,0,0).

Ray-tracing may spawn rays recursively (if a ray hits a reflective or refractive object). If any ray fails to hit an object, `Pmiss` and `Nmiss` will be set to the position and direction of the deepest ray that failed to hit any objects, and the function will return the depth of that ray. Otherwise the function returns 0.

9.14.14 gather()

```
gather(string category, point pos,
       vector axis, float angle,
       float samples, parameters) {
         statements
       } [else { statements }]
```

Ray Tracing

`gather` is a looping construct for sampling a cone of directions using ray tracing. Rays are fired from `pos` with the ray direction randomly distributed within `angle` radians of the `axis` direction. If a ray hits an object, the optional output variables are updated and the first set of statements is executed. If a ray misses all objects, the optional else block is executed if present.

Samples is the requested number of rays to trace. The actual number of rays traced may be less because AIR automatically reduces the number of rays cast at deeper levels of recursion. There may even be zero rays traced, and a shader should handle that case properly.

The category parameter is a hint to the renderer about the purpose of the gather query. The following categories are recognized:

`"illuminance"`	sample the incoming indirect light with indirect rays
`"environment"`	sample the surrounding environment with reflection rays
`"occlusion"`	sample with shadow rays

`gather` accepts a number of optional input and output parameters.

Optional Input Parameters

Type and Name	Description
`float bias`	offset to prevent incorrect self-intersections
`string distribution`	sample distribution: `"uniform"` or `"cosine"`
`string label`	user-assigned name for rays cast by this call; a shader can query the label with `rayinfo()`
`float maxdist`	maximum distance to search for traced objects
`string subset`	restricts intersection tests to members of the specified groups as determined by `Attribute "grouping" "membership"`
`float othreshold`	gives an opacity threshold. Tracing stops when the cumulative opacity along the ray exceeds the threshold. The default threshold is 0.

Optional Output Parameters

Type and Name	Description
`point ray:origin`	location from which the ray was traced
`vector ray:direction`	normalized ray direction
`float ray:length`	distance from ray origin to hit location, or 1.0E15 if the ray missed all objects
`primitive:`*`name`*	Returns the value of the named global variable of the object intersected by a ray prior to shader evaluation. Supported variable names are P, N, Cs, and Os.
`surface:`*`name`*	Returns the value of the corresponding variable after execution of the intersected object's surface shader. Supported variables are Ci, Oi, P, N, and any surface output variable.
`point:`*`name`*	Returns the value of the named variable associated with a point sample in a point cloud

Point Cloud Sampling

Air 12 extends the gather() construct to allow querying of point cloud data. Sample usage:

```
color csum = 0;
integer count = 0;
color tcolor;
gather("pointcloud:filename", P, N, maxsamples, "maxdist", maxdist,
"point:mycolor", tcolor) {
   csum += tcolor;
   count += 1;
}
color caverage = 0;
if (count>0) caverage=csum/count;
```

The normal vector is ignored. The query will return up to maxsamples points from the cloud within the search radius specified with the maxdist parameter. Values associated with a given point can be queried by including "point:*varname*" queries in the list of gather() parameters.

Limitations

`gather()` statements should not be nested.

9.14.15 indirectdiffuse()

```
color indirectdiffuse(point P, normal N,
                      float samples, parameterlist)
```

`indirectdiffuse` returns the indirect illumination at point P. If the `samples` parameter is less than

1, the number of samples is taken from the `indirect:nsamples` attribute.

`indirectdiffuse` accepts the following optional parameters:

Type and Name	Description
`float blur`	blur factor for optional environment map
`string environmentmap`	environment map used to provide color for rays that miss all objects
`string environmentspace`	coordinate space for environment map lookup
`string label`	user-assigned name for rays cast by this call; a shader can query the label with `rayinfo()`
`string subset`	restricts intersection tests to members of the specified groups as determined by `Attribute "grouping" "membership"`
`float maxhitdist`	maximum distance to search for intersecting objects
`float maxerror`	maximum error allowed when re-using samples from the occlusion cache. If 0, the occlusion cache is not used. If less than 0 or omitted from the occlusion call, the indirect:maxerror attribute value is used.
`float maxpixeldist`	maximum distance in pixels between samples in the occlusion cache. If omitted or less than 0, the indirect:maxpixeldist attribute value is used.
`float adaptive`	controls adaptive sampling: 1 enables, 0 disables, any other value defaults to the indirect:adaptivesampling attribute setting

9.14.16 irradiancecache()

This function has been deprecated in AIR 10 and later.

AIR exposes the irradiance caching functionality used by the `indirectdiffuse()` and `occlusion()` functions so that users can create and access custom caches for special purposes. The `irradiancecache()` function has two operating modes, insert and query:

Cache Insertion

```
irradiancecache("insert", string cachename,
                string cachemode,
                point P, vector N,
                "harmonicmeandist", float hmean,
                values);
```

Inserted values can be any of:

```
"irradiance", color irrad
"coverage", float alpha
"environmentdir", vector dir
```

The harmonic mean distance is the reciprocal of the average of the reciprocal hit distances of the rays that went into calculating the inserted values (presumably with a gather statement).

Cache Query

```
float irradiancecache("query", string cachename,
                      string cachemode,
                      point P, vector N,
                      "maxerror", float maxerror,
                      "maxpixeldist", float maxpixeldist,
                      values);
```

A cache query returns 1 if the cache holds values that satisfy the constraints of the maxerror and maxpixeldist parameters. If maxerror or maxpixeldist are omitted or are negative, the value of the corresponding indirect attribute for the primitive is used.

`cachemode` is used to control whether the cache is written to a file or loaded from a file. If `cachemode` contains "w", the cache will be written to file `cachename` when rendering is complete. If `cachemode` contains "r", the cache will be initialized with data from file `cachename` if it exists.

9.14.17 isbaking()

```
float isbaking()
```

returns 1 if the shader is being executed by BakeAIR, the Shading and Lighting Baker.

9.14.18 isindirectray()

```
float isindirectray()
```

Returns 1 if the shader is being executed to estimate indirect illumination; otherwise, returns 0.

9.14.19 isphoton()

```
float isphoton()
```

returns 1 if a light shader is being executed for a photon.

9.14.20 isshadowray()

```
float isshadowray()
```

Returns 1 if the shader is being executed to determine visibility for a shadow ray or shadow map; otherwise, returns 0.

9.14.21 meshinfo()

```
float meshinfo(token, variable)
```

AIR 8 introduces a new instancer shader type that can create new geometry at render time based on an existing primitive. The meshinfo() function provides access to information about the base primitive to which an instancer shader is attached. Meshinfo() is only valid inside an instancer shader.

Meshinfo() is typically used in a two-stage process: first, meshinfo() is called to establish a sampling method and location. Then meshinfo() is called to query primitive variable values at the current sample location.

Sample Methods

The following sampling methods are supported:

Sample by vertex

```
float vix=0;
if (meshinfo("sample:vertex",vix")==1)  // valid vertex index
```

This sample method allows vertex, varying, and facevarying data to be queried. For facevarying data Air picks one face's vertex value for the query result.

Sample at vertex on face

```
float facevx[2];
facevx[0] = faceix;
facevx[1] = vxindexonthisface;
if (meshinfo("sample:facevertex",facevx)==1) // valid vertex on this face
```

This sample method allows access to varying, vertex, and facevarying data.

Sample at standard texture coordinates

```
float xy[2];
xy[0]=scoord;
xy[1]=tcoord;
if (meshinfo("sample:st",xy)==1)  // valid (s,t) texture position on
surface
```

This method provides access to varying, vertex, and facevarying data. The base primitive must of course have a well-defined set of standard texture coordinates. If two locations on the surface share the same texture coordinates, the sample result is not well-defined.

Sample curve

```
float cv[2];
cv[0]=curveindex;
cv[1]=positionalongcurve;
if (meshinfo("sample:curve",cv")==1)  // valid position along specified
curve
```

This method samples data on the curve given by the first index and the position specifed by the second array parameter. The second parameter should lie between 0 (the start of the curve) and 1 (the curve end).

Sample time

```
float mytime=time;
meshinfo("sample:time",mytime);
```

This method sets the time at which to query a base mesh with vertex motion blur. This method is currently only supported for vertex queries, and only vertex position data P is blurred. In typical usage a shader will sample at the global 'time' value and at 'time'+'dtime'.

Querying Primitive Variables

Once the sample method and location have been set, subsequent calls to meshinfo() can query any standard or user-defined primitive variable at the current sample location. For example:

```
point pos;
if (meshinfo("P",pos)==1)  // got position data

normal nrm;
if (meshinfo("N",nrm)==1)  // got normal data
```

Point, normal, and vector values are returned in object space coordinates for the base primitive.

Standard texture coordinates are always stored as an array of 2 floats:

```
float st[2];
if (meshinfo("st",st)==1) //got standard texture coordinates
```

The underlying primitive representation will be a polygon mesh for all surface primitives, a list of line segments for curves, and a set of vertices for point primitives.

Information Functions

Meshinfo supports the following information functions for querying the base primitive:

Number of Vertices

```
  float nvx=0;
  meshinfo("count:vertices",nvx);
```

Number of Faces

```
  float nfaces=0;
  meshinfo("count:faces",nfaces);
```

9.14.22 occlusion()

```
color occlusion([string mapname,] point P,
                normal N, float angle,
                varying output normal Nunoccl,
                parameterlist)
```

Returns the extent to which the current location is blocked or occluded by surrounding objects. The direction in which the location is least obscured in `Nunoccl`.

The `angle` parameter gives the half-angle in radians of the cone of directions that is sampled for the occlusion result.

If the optional `mapname` parameter is not provided or it has the special value `"raytrace"`, AIR uses ray tracing to estimate the occlusion. Otherwise, `mapname` is the name of an occlusion map.

See Ambient Occlusion for more information.

Optional Input Parameters:

Type and Name	Description
`float bias`	offset to prevent incorrect self-intersections
`string label`	user-assigned name for rays cast by this call
`float maxhitdist`	maximum distance to search for intersecting objects
`float maxerror`	maximum error allowed when re-using samples from the occlusion cache. If 0, the occlusion cache is not used. If less than 0 or omitted from the occlusion call, the indirect:maxerror attribute value is used.
`float maxpixeldist`	maximum distance in pixels between samples in the occlusion cache. If omitted or less than 0, the indirect:maxpixeldist attribute value is used.
`float samples`	number of rays to trace when ray tracing
`string subset`	restricts intersection tests to members of the specified groups
`float adaptive`	controls adaptive sampling: 1 enables, 0 disables, any other value defaults to the indirect:adaptivesampling attribute setting
`float maxsolidangle`	for point-base occlusion, the maximum solid angle in radians for approximated point groups
`float optimizenormal`	for traced occlusion, set to 0 to use the normal value passed to occlusion as-is rather than the surface normal when computing occlusion. Useful when sampling occlusion at positions other than the current shading location.
`float sampleenvironment`	`1 enables sampling of the scene environment for unoccluded rays`

Optional Output Parameters

Type and Name	Description
`color environment`	result of sampling the scene environment when the sampleenvironment mode is enabled

See Also

Ambient Occlusion

9.14.23 rayhittest()

```
float rayhittest(point from, vector dir,
                 output point Phit,
                 output normal Nhit)
```

Tests for an object in direction `dir` from position `from`. If an object is encountered, `Phit` and `Nhit` contain the position and normal of the object and the function returns the distance to the object; otherwise the function returns a large number (1.0E15).

9.14.24 rayinfo()

`float rayinfo(string name, `*`variable`*`)`

Returns information about the ray that caused the shader to be run. `rayinfo()` returns 1 if the query succeeds, and 0 otherwise. Supported keywords are:

`"depth"` (float) - returns the current ray level, where level 0 is a camera ray.

`"type"` (string) - returns the type of ray: "camera", "trace", "transmission", "photon", or "collector"

`"label"` (string) - returns the label associated with the current ray by a "label" parameter to the calling ray tracing function

`"weight"` (color) - returns the weight or importance of the current ray. See Importance-Based Rendering for more information.

`"indirectdepth"` (float) - indirect trace depth for current ray

`"samplecount"` (float) - total sample count for rays spawned from `trace()` etc as set by the `"samples"` parameter

`"sampleindex"` (float) - index of current ray in the range 0..samplecount-1

9.14.25 raylevel()

```
float raylevel()
```

Returns the level of reflection/refraction at which the shader is being executed, with a value of 0 indicating the shader is being executed from the camera.

9.14.26 ribprintf()

`ribprintf(`*`patternstring`*`, `*`arglist`*`);`

AIR 8 introduces the ribprintf() statement to allow instancer and procedure shaders to send RIB commands to the renderer. Ribprintf works the same as printf() except that the output is sent to AIR's RIB parser instead of the console.

See Also

Instancer shaders
Procedure shaders

9.14.27 shadow()

```
color shadow("raytrace", ...)
```

If the file name passed to shadow is "raytrace" or "shadow", AIR will use ray tracing for shadows instead of a shadow map.

The standard `shadow()` function accepts several optional input parameters:

Type and Name	Description
`string subset`	restricts ray traced intersection tests to members of the specified groups
`string label`	user-assigned name for rays cast by this call
`color weight`	weight or importance of this trace call for importance-based ray tracing

9.14.28 specular()

```
color specular([string category,] normal Nf, vector V, float roughness,
sharpness)
```

The `specular()` function accepts an optional `"sharpness"` parameter with a value between 0 and 1. A value of 0 produces the normal specular result. Larger values produce a highlight with a sharper edge, giving a glossy appearance.

The `specular()` function also provides a `"channels"` output variable, an array of colors, holding per-light specular values indexed by a light shader's `__channel` output variable. A `"nchannels"` output variable gives the number of valid channels.

9.14.29 str_replace_last_number

```
string str_replace_last_number(string olds, float newnum)
```

This function searches backwards from the end of the string looking for an integer value. If one is found, it is replaced with the provided new value.

This function may be useful when dealing with a numbered sequence of file names.

9.14.30 subsurfacescatter()

```
color subsurfacescatter(
   point P, normal N,
   vector meandiffusepath,
   color reflectance,
   float ior, float maxsampledist)
```

The `subsurfacescatter()` function returns an estimate of the multiple scattering component of subsurface scattering at P.

`meandiffusepath` is the average distance light diffuses through the material for the red, green, and blue color components. The mean diffuse path can be calculated from the reduced scattering (sigma_s) and absorption coefficients (sigma_a):

```
 mdf = 1/sqrt(3*(sigma_s+sigma)*sigma_a)
```

`reflectance` is the average diffuse reflectance

`maxsampledist` is the maximum distance between samples in the point cloud. If `maxsampledist` is less than 0, AIR will automatically calculate the distance based on the meandiffusepath and reflectance values. This parameter is provided to allow users to override the default behavior.

Real world values for meandiffusepath and reflectance can be found in Jensen et al "A Practical Model for Subsurface Light Transport" in the SIGGRAPH 2001 Proceedings.

When `subsurfacescatter()` is first called, AIR generates a set of points covering all surfaces in the point set. Each point is shaded by querying its shaders with a ray type of `"collector"`; points are shaded on the "outside" of the surface as determined by the current orientation. A shader should test for evaluation by a `"collector"` ray, and return the irradiance to be stored in the point cloud. For example:

```
  string raytype="";
 rayinfo("type", raytype);

 if ((raytype=="collector") || (raytype=="photon"))

   Ci= Cs * (Ka * ambient() + Kd * diffuse(Nf));

 else

   Ci = Oi * (subsurfacescatter(P, Nf,
              SSDistance * SSDistanceScale,
              SSReflectance,
              IndexOfRefraction,
              SSSampleDistance)+
        Ks*specular(Nf,normalize(-I),roughness));
```

Sample shaders that use subsurfacescatter include VSkin, VTranslucent, and VTranslucentMarble.

Varying SSS Parameters

In Air 10 the `subsurfacescatter()` function handles varying parameter values properly. However using a varying mean diffuse path, reflectance, or ior will be much slower. When using varying parameters, the max sample distance should be explicitly set to produce consistent results. Use as large a max sample distance as feasible to save render time

See Also

`collect()`

9.14.31 texture()

```
color texture(..., "alpha", alpha)
```

The standard `texture()` accepts an optional `"alpha"` parameter that returns the next channel after the channels returned by the texture() call. For an RGBA file accessed with a color texture read starting at channel 0, the returned value will be the alpha channel of the file.

The extra channel does not significantly slow down rendering, so it is always better to request an alpha channel if it might be needed rather than making an extra texture call.

Air 13 and later provide a builtin antialiased 2D noise pattern which can be generated by providing the special value "noise" as the texture map name.

9.14.32 texture3d()

```
texture3d(string filename, point P, normal N, "float blur", blur, "float
width", filterwidth, "float fill", fillvalue, token, value,...);
```

Texture3d returns a filtered sample from a 3D texture file generated by bake3d. The filter region is automatically computed based on the input query position. The optional `"width"` and `"blur"` parameters can be used to scale or increment the sample region respectively. Texture3d will filter all points whose region of influence overlaps the query filter region. If a given token does not have an associated value in the point file, the corresponding variable value is filled in by the *fillvalue* (0 if the `"fill"` parameter is not specified).

Example:

```
surface bakematte3d(
  float Diffuse = 1;
  string BakeMapName="";
  float BakeMapBlur = 0;
  float BakeMapFilterWidth = 1;
  float BakeIt = 0;
  float BakeRadius = 0.1;)
{
  normal Nnrm = normalize(N);

  string raytype="";
  rayinfo("type",raytype);

  if ((BakeMapName!="") && (raytype!="collector")) {
    if (BakeIt==0) {
      texture3d(BakeMapName, P, Nnrm,
                "width", BakeMapFilterWidth,
                "blur",BakeMapBlur,
                "Cs",Ci);
    } else {
      collect(P,0,BakeRadius) {};
      Ci=1;
    }
  } else {
    normal Nf = faceforward(Nnrm,I);
    Ci = Diffuse * Cs * diffuse(Nf);
    if (BakeMapName!="") bake3d(BakeMapName,"Cs",P,Nnrm,"Cs",Ci);
  }
  Oi=Os; Ci*=Oi;
}
```

See Also

`bake3d()`
Bake3d surface shader

9.14.33 trace()

color trace(point pos, vector dir, *parameterlist*)

Optional Input Parameters

The standard trace() function accepts several optional input parameters:

Type and Name	Description
`float bias`	offset to prevent incorrect self-intersections
`float blur`	blur factor for fuzzy reflections
`string label`	user-assigned name for rays cast by this call
`float maxdist`	maximum distance to search for intersecting objects
`float samples`	number of rays to trace
`string subset`	restricts intersection tests to members of the specified groups
`vector majoraxis`	tangent vector for anisotropic reflections
`float majorblur`	blur factor in majoraxis direction for anisotropic reflections
`float minorblur`	off-axis blur for anisotropic reflections
`color weight`	weight or importance of this trace call for importance-based ray tracing

Optional Output Parameters

The `trace()` function accepts an optional `"alpha"` output parameter that returns the fraction of the sampled environment cone occluded by reflected objects. The alpha value returned by the ray tracer can be used to composite tracing results with a background environment map.

In AIR 9 and later `trace()` provides a `"background"` output parameter of type color that stores the contribution of the background environment map or color for reflection rays.

Beginning with AIR 9 `trace()` can return arbitrary output variables from the surface shaders of intersected primitives in the same manner as the `gather()` construct. For example:

```
  color renv;
  color Crefl = trace(P,R,"surface:__environment",renv);
```

See Also

`environment()`
Ray Traced Reflections

9.14.34 visibility()

```
color visibility(point p1, point p2)
```

Returns the visibility between two points. If there are no intervening objects, the function returns (1,1,1). If there is a fully opaque object in between, returns (0,0,0). Partial occlusion will return a partial transmission value.

9.14.35 voronoi()

```
void voronoi(string type, point pos, float jitter,
             output float dist1, output point center1);

void voronoi(string type, point pos, float jitter,
             output float dist1, output point center1,
             output float dist2, output point center2);
```

The voronoi() function computes a 2D or 3D cellular pattern. The `type` parameter should be either "grid2d" or "grid3d". The jitter parameter controls deviation from a regular grid and should lie between 0 and 1.

The `dist1` parameter returns the distance from the query point `pos` to the nearest cell center `center1`. Similarly, `dist2` returns the distance from `pos` to the second nearest cell center `center2`.

10 2D Texture Mapping

Preparing Texture Maps

For the most efficent rendering in terms of speed and memory use, all texture files should first be converted to AIR texture maps. AIR texture maps can be created with the MkTex command-line utility or the mktexui graphical user-interface (in addition to the RIB `MakeTexture` command). Your plugin may automatically perform this conversion for you.

Although AIR can directly use textures in a variety of graphics formats, creating an AIR texture file has several advantages:

- The wrapping behavior of the texture map can be specified.
- The new texture file is a mip-map - it stores multiple pre-filtered copies of the texture at different resolutions. When accessing the texture the renderer will use an appropriate resolution based on the region of the texture visible in the area being shaded, resulting in fewer aliasing artifacts.
- Texture files are tiled, allowing the renderer to selectively load only the portions of an image that are needed.
- Texture files are compressed to minimize disk space and bandwidth usage.
- AIR limits the memory used by tiled texture maps to a user-defined maximum, dynamically loading and unloading tiles as needed.

Texture Input Formats

AIR and its texture-conversion tools can read image files in a variety of popular graphics formats. AIR selects a texture reader based on the file name extension. If a file does not have an extension associated with a particular driver, AIR assumes the file is an AIR texture map.

The following is a list of currently supported raster image formats:

Format	Extension	Data Types	Channels
TIFF	`tif, tiff`	8-bit,16-bit, float	unlimited
Portable Network Graphics	`png`	8-bit and 16-bit	1-4
SGI File Format	`sgi, rgb`	8-bit and 16-bit	unlimited
JPEG Format	`jpg, jpeg`	8-bit	1 or 3
HDR Format	`hdr, rad`	float	3
Photoshop Format	`psd`	8-bit, 16-bit	3 or 4
Windows Bitmap	`bmp`	8-bit	3 or 4
Softimage PIC	`pic`	8-bit	3 or 4
IES light profile	`ies`	float	1
PTEX	ptex, ptx	8-bit, 16-bit, float,half	unlimited

High-Dynamic Range (HDR) Images

AIR can use High Dynamic Range (HDR) images as texture maps and environment maps. AIR can read images in the Radiance HDR image format that is popular for storing high-dynamic range images. As with other texture and environment maps, conversion to an AIR texture file using AIR's texture conversion utilities is recommended for best performance and results.

Texture Cache

AIR loads texture files, shadow maps, and environment maps as they are needed by the renderer. AIR uses a simple least-recently used cache to keep only the most recently accessed sections of texture in memory, thereby keeping the memory used by all types of textures to a minimum. The texture cache size is set with

```
Option "limits" "integer texturememory" [20000]
```

which gives the maximum cache size in K. The default texture cache of 20MB is usually sufficient unless a scene uses several large shadow maps or an occlusion map.

Textures and Gamma Correction

When a final image has gamma correction applied, you may find that your texture maps appear washed out or faded.

This problem occurs most often with commercial texture libraries that have been pre-gamma corrected to display properly on a CRT. To get such textures to render properly with AIR, apply an inverse gamma correction factor during the texture conversion process with the mktex or mktexui tools. Here are some common gamma factors and the corresponding inverses:

Gamma	Inverse
1.8	0.56
2.0	0.5
2.2	0.45

10.1 Projecting Textures ("Decaling")

Here's how to use the AIR Show display window to position textures for projective mapping:

A projected texture is mapped onto an object as though it were shown through a slide-projector. AIR Show allows you to use the camera-positioning controls in a modeling program to position a texture for projection. Shaders included with AIR that can project texture maps include VDecal3DPlastic.

- Pretend that the camera is a light that is going to shine the texture map onto an object.
- Choose or create a view window in your modeling program. If you have a choice, use an orthographic view (or a perspective view with a small field of view) to reduce distortion.
- Position the camera so the object you want to map is visible, and so that you could draw a rectangle on the object that would define the texture map position.
- Export the scene and render a quick, small draft image to AIR Show, the AIR framebuffer.
- In the display window, drag out a rectangle with the mouse to define the placement of the texture map. The texture map will exactly occupy the rectangle. You do not need to draw a rectangle if you want the texture to fill the entire window.
- Select **Copy Decal Projection** from the **Edit** menu in AIR Show. This copies the projection information as a text string of 16 numbers.
- Assign a shader that supports texture projection to the object. Many of the shaders included with AIR that generate 2D patterns provide texture projection.
- Assign the texture name to the texture map name parameter.
- Select the parameter for the transformation applied to the projected point and paste from the clipboard by pressing Ctrl-V or by using the right mouse button pop-up menu.
- If the shader takes a projection type, set it to `planar`.
- Set the projection space to `world`.
- Render the same view again to verify that the texture is correctly positioned.
- Render the scene from another view to see that the texture remains in place.

Because the texture is projected in "world" space, the texture will remain in place if the camera moves. The texture will not move with the object if the object moves.

You can use this technique to position any 2D pattern, not just those that rely on texture maps. In all cases the rectangle drawn in the display window defines the unit square in 2D texture coordinates.

10.2 Vector Textures

Beginning with release 3.0 AIR can use vector graphics files as texture maps with the aid of the Vtexture DSO shadeop written by Alex Segal. Vtexture can be downloaded from:

http://www.renderman.ru/vtexture/indexe.html

Copy the vtexture DLL or shared object file to the shaders directory of your AIR installation.

With Vtexture installed you can use a vector graphics file in Adobe Illustrator (.ai) or EPS format for any shader parameter that expects a texture map. When a file name with a `.ai` or `.eps` extension is passed to the standard shading language `texture()` or `environment()` functions, AIR automatically calls Vtexture with the appropriate parameters.

See the Vtexture documentation for information on supported file formats and customizing the operation of Vtexture.

10.3 PTEX: Per-Face Texture Mapping

Air 11 and later support per-face texture mapping using the open-source PTEX library.

Overview

The PTEX library was developed to address several common problems with traditional 2D texture mapping:

- Assigning "good" texture coordinates to complex models can be difficult and tedious
- Texture seams can be visible along discontinuities in the texture map
- Large numbers of texture files can create a significant IO bottleneck

The PTEX system addresses these concerns by eliminating UV assignment, providing seamless filtering, and allowing an arbitrary number of textures in a single file. A PTEX file stores a separate texture map for each face in a mesh, with adjacency information to allow proper filtering across face boundaries. The PTEX library provides a texture cache for managing large numbers of textures efficiently.

PTEX files can store 8-bit, 16-bit, float, and half precision data with an arbitrary number of channels.

Air PTEX Support

Air supports PTEX for the following applications:

- Surface and displacement textures for arbitrary convex polygon meshes
- Surface and displacement textures for subdivision meshes that are all triangles or all quads
- Cube-faced environment maps
- Texture baking for polygon meshes and subdivision surfaces with BakeAir

Using PTEX Textures

PTEX textures can be used with any shader that accesses texture maps using the standard texture coordinates (s,t). Just enable the following attribute to have Air automatically assign the necessary face indices and texture coordinates:

```
Attribute "render" "boolean ptex" [1]
```

The PTEX library maintains its own texture cache. The maximum memory used by the cache can be set with

```
Option "limits" "ptexmemory" [100]
```

giving the memory limit in MB.

The maximum number of open PTEX files can be set with:

```
Option "limits" "ptexfiles" [100]
```

By default the texture coordinates generated by Air for Ptex textures will be the standard texture coordinate pair, st. You can change the name for ptex coordinates with:

```
Attribute "render" "string ptexcoordinates" "st"
```

Limitations: Ptex support for subdivision meshes is limited to meshes that are all quads or all triangles

Using 3DCoat Textures

The PTEX files exported by the 3DCoat paint program appear to have the texture coordinates swapped. You can compensate for that in most Air surface shaders by setting the `TextureSizeXY` parameter to `1 -1` and the `TextureAngle` to `90`.

See Also

Baking PTEX Textures with BakeAir
PTEX web site: `http://ptex.us`

11 3D Textures & Point Sets

3D textures store 3D point-based information. Typically 3D textures are used to record time-consuming shading computations for accelerated re-rendering. Because 3D textures are accessed using a location in a 3D coordinate system, they do not require surfaces to have the well-defined set of texture coordinates required for 2D textures.

Many of AIR's advanced features allow expensive shading results to be saved to a file for later reuse, including:

- Indirect diffuse illumination
- Occlusion
- Caustics
- Subsurface scattering

AIR also provides a general mechanism for creating and accessing 3D textures with the bake3d() and texture3d() shading language functions.

3D Point File Formats

With version 8 AIR introduces a common set of file formats for all 3D point-based data. AIR uses the file name extension to determine the point file type.

Air Point Cloud	`.apc`	Unstructured point collection. The basic format for storing 3D point-based information. The entire point set will be loaded at once when accessed.
Air Point Map	`.apm`	Point set organized into spacially continguous clusters that can be loaded independently, allowing cached access to the point data. AIR can efficiently sample large point maps as 3D textures using only a small fixed-size cache.
Air Brick Map	`.abm`	3D data resampled into a hierarchy of 3D cells containing the original data filtered at different resolutions. Like point maps, sections of a brick map can be loaded on demand allowing cached access. Like 2D texture maps, the multiple levels of detail in a brick map allow the map to be sampled over large regions efficiently. Loss of data inherent in the brick map creation process may result in artifacts when used for fine details. Brick maps cannot be converted back into a point set.
RIB file	`.rib`	Stores the point data as a RIB Points primitive. If the file name contains the text `ascii`, an ASCII RIB file will be written; otherwise a more compact binary file is produced.
Text file	`.txt`	A simple text format enabling data exchange with other programs and formats.
Maya PDC	`.pdc`	(read only) Maya PDC file. Maya 'position' data is automatically converted to 'P' data for RIB. 'radiusPP' is converted to 'width' for RIB. If the PDC file has no radius info, a 'width' variable with value 1 for all points is added.
Memory cache	`.ram`	Creates a cache in-memory for use only during the current rendering process. Only useful for point sets created with the `collect()` function.

Point File Manipulation

Version 8 of AIR includes a new Air Point Tool (airpt) for converting between the different point-based formats and performing other manipulations of 3D point data.

Point Sets as Geometry

Air Point Clouds and Air Point Maps can be rendered as geometric primitives using the RIB Geometry call:

```
  Geometry "mycloud.apc"
```

A point set is represented converted into a RIB Points primitive, with file data attached as per-point primitive data. The normal point attributes such as point type are applicable. When rendering an Air Point Cloud file, AIR will load the entire point set the first time the set bounding box is encountered. When rendering an Air Point Map, subsets of the point map are loaded on-demand.

Examples

The Bake3d surface shader shows how to bake general data to a 3D texture. Source code for the shader is located in `$AIRHOME/shaders/src/surface`. Sample RIB files can be found in `$AIRHOME/examples/bake3d`.

See Also:

Bake3d surface shader

12 Lighting

Lighting in AIR is accomplished with the use of light shaders. Every light is produced with a light shader. Your plugin may include a special set of light shaders that mimic the behavior of the lights in your modeling or animation program. Most plugins also allow custom light shaders to be assigned to a light.

Standard Lights

AIR comes with extended versions of the standard pointlight, spotlight, and distantlight shaders. These shaders have been extended in the following ways:

- All support shadow-casting using ray tracing, shadow maps, or automaps
- Non-specular and non-diffuse behavior
- Extra output variables containing shadow values and unshadowed light colors for use in special applications such as computing a shadow pass
- Fine control of shadow-casting objects with shadow-casting groups (for ray-traced shadows)
- Per-light channel numbers facilitating per-light output channels in surface shaders
- Most lights have an extended version that allows baking shadows and illumination with 2D or 3D texture maps.

Special Lights

AIR includes additional light shaders for other purposes:

Area Lights:

`arealight` , `arealight2`, portallight, and `texturedarealight` shaders that can be attached to geometric primitives to produce area lights.

Indirect Diffuse Illumination:

`indirect` shader for simulating indirect diffuse illumination.

Dome Lighting and Image-Based Lighting:

`envlight` and occlusionpass shaders for ambient occlusion and image-based lighting.

Caustics:

The `caustic` shader is used with photon mapping to produce caustic effects.

Photometric lighting:

The photometric_pointlight shader allows illumination to be defined using physical units or an IES light profile

Sunlight

The sunlight shader computes the color and direction for a sun-like light source based on observer position, date & time, and atmospheric conditions.

Diffuse Only and Specular Only Lights

Most non-ambient light shaders can be configured to cast only diffuse light or only specular light by setting the following parameters.

To make a light diffuse only (i.e., produce no specular highlights), set the `__nonspecular` parameter of a light shader to 1.

To make a light cast only specular light but not diffuse light, set the `__nondiffuse` parameter of a light shader to 1.

Optimizing Distance-Based Lights

Often the illumination emitted by a light decreases with distance from the light source until it becomes a negligible part of the lighting at locations far from the light. You can help Air to avoid evaluating a light source at locations where it has little impact by setting the maxdist light attribute to the maximum distance at which the light should be evaluated. E.g.,

```
Attribute "light" "float maxdist" [1000]
```

12.1 Area Lights

Any primitive in a scene can be turned into a light source by attaching a suitable light shader to it. E.g.,

```
AreaLightSource "arealight" 33
```

(Trim curves are ignored by area lights.) In principle an area light casts light from the entire source surface onto the current shading location. Air provides two methods of simulating an area light:

Brute Force True Area Lights

One method of simulating an area light is to run the light source shader on a user-specified number of sample locations distributed over the surface area of the light. The number of samples is given by

```
Attribute "light" "integer nsamples" [n]
```

Shadows for an area light can be produced using ray tracing. The `arealight` shader included with Air can be used for brute force simulation of an area light.

Simulating Area Lights with `areashadow()`

Brute force simulation of area lights can be quite slow because it requires executing the light source shader multiple times. Air provides a shading language function called `areashadow()` that samples the area light in a single function call. The `arealight2` shader included with Air uses the areashadow function to provide faster area light simulation. When using the `arealight2` shader, the `light:nsamples` attribute is not used. Instead, the number of samples is set by a parameter in the shader.

The `areashadow()` function is an approximation, and it only works well for simple geometry such as a disk, square, or other convex polygon.

Area Lights and Indirect Illumination

If indirect illumination is employed, area light geometry should be made invisible to indirect rays with

```
  Attribute "visibility" "integer indirect" [0]
```

so that the area light is not included twice in the illumination calculations.

12.2 Shadows

Air provides three ways to produce shadows - ray tracing, shadow maps and automaps. Each method has advantages and disadvantages in terms of ease of use, speed, memory usage, and quality.

Ray-Traced Shadows

Ray tracing is one popular method for simulating shadows. A ray is traced from the shading location to a light source; the location is in shadow if the ray strikes any intervening objects. Ray traced shadows can easily accomodate colored shadows and shadows for semi-transparent objects.

Ray-traced shadows are easy to use - you only need to set the appropriate rendering options and attributes. However they have several drawbacks:

- Memory usage: Ray tracing requires that the entire scene be kept in memory for the duration of rendering, including parts of the scene that are off screen, which can use a lot more memory than normal scanline rendering.

- Speed: Ray-traced shadows can be slow for a large scene.

- Quality: You will notice that the shadow edges are jagged. Smoothing them requires casting extra shadow rays which increases rendering time proportionally.

Shadow-Mapped Shadows

Shadows can also be simulated using shadow maps. A shadow map is made by rendering an image of a scene from the point of view of a light source and recording the depth of the nearest object at each pixel location. The renderer can use the depth information to determine if a given point is in shadow by comparing its distance from the light source with the depth stored in the corresponding location in the shadow map.

Shadow maps are slightly more difficult to use than ray-tracing, but they have other significant advantages:

- Memory usage: Shadow map size is independent of the complexity of a scene.

- Speed: Shadow mapping is usually faster than ray tracing. Shadow map access has a fixed time cost, independent of scene complexity. Air can usually render shadow maps very quickly since shadow maps do not require as much shading computation as a color image.

- Reusability: Once rendered, shadow maps can be re-used as long as the relative positions of lights and objects remain the same.

- Quality: Shadow-mapped shadows have nice anti-aliased fuzzy edges.

Automaps

Automaps are a hybrid method of producing shadows, combining the ease-of-use of ray-traced shadows with the quality of shadow-mapped shadows. For automapped shadows Air dynamically creates tiles of a shadow map on-demand, and then uses those just as a normal shadow map to compute shadow values. Automaps require objects to be retained in memory just like ray tracing, so they use more memory than ordinary shadow maps. Automaps are also typically slower to generate than shadow maps created with a separate rendering pass.

Occlusion

Ambient occlusion can be used to simulate soft shadows. Air provides a special envlight light shader that uses occlusion shadows with illumination from a surrounding environment. Occlusion can also be used for shadows with any standard light by supplying the special shadow name "occlusion".

12.2.1 Ray Traced Shadows

AIR supports the use of ray tracing to produce shadows. One or more rays are traced from the light source to the point being shaded, and the incoming light is modulated by the opacity of any intervening objects.

To use ray traced shadows:

1. Make objects that will cast shadows visible to shadow rays with:

```
  Attribute "visibility" "integer shadow" [1]
```

By default all objects are invisible to shadow rays.

2. Enable shadow ray tracing for one or more lights. Your plugin may provide a special control to enable ray-traced shadows for a given light. For custom light sources, ray-traced shadows can be enabled by providing the special shadow name `"raytrace"`:

```
  LightSource "spotlight" 3 "string shadowname" ["raytrace"]
```

This special name tells the renderer to cast shadow rays instead of looking for a shadow map.

Shadow Quality

The quality of ray-traced shadows can be improved by casting more shadow rays. Most light shaders provide a `shadowsamples` parameter that sets the number of rays that are traced to compute the shadow value.

Shadow Acne and Shadow Bias

Occasionally some objects may exhibit incorrect self-shadowing, usually visible as small random dark dots or "shadow acne". A light shader typically provides a `shadowbias` parameter to help prevent incorrect shadows by providing a small offset for the depth information in the shadow map. Each primitive also has a bias attribute, used for all ray tracing operations, that can be set with

```
  Attribute "trace" "float bias" [0.01]
```

Soft Shadows

Most light shaders provide a `shadowblur` parameter that can be used to produce soft shadows by expanding the region sampled by the shadow rays. Shadow blur for ray-traced shadows is given as an angle in radians for the range of directions in which to cast rays.

Ray Traced Shadow Attributes

The following attributes can be set for each object:

```
Attribute "visibility" "integer transmission" [0]
```

When set to 1 an object will cast ray-traced shadows. Making objects that do not cast shadows invisible to shadow rays speeds up rendering. Good candidates are objects functioning as a ground plane or a surrounding volume.

`Attribute "shade" "string transmissionhitmode" "`*`type`*`"`

Tells the renderer how to treat an object when it is queried by shadow rays. Possible types are:

`primitive`	the primitive blocks shadow rays in proportion to its `Opacity` attribute setting
`shader`	the renderer should execute the surface shader attached to the primitive to estimate the opacity

For an object with a surface shader that computes a complicated opacity (e.g., a screen shader), you will need to set the shadow type to `"shader"` to force evaluation of the shader. Your plugin should provide a control for this setting.

```
Attribute "trace" "integer motionblur" [1]
```

Moving objects with this attribute set will cast motion-blurred shadows. Motion-blurred traced shadows require 2 or more shadow samples or rays. Use more rays for smoother blur.

Shadow Casting Groups

Many AIR light shaders provide extra parameters that allow shadow-casting to be restricted to a particular group of objects:

```
  "string shadowgroups" [""]
  "string shadowattribute" ["user:shadowgroups"]
```

The first parameter takes a list of groups to test for ray-traced shadows. This covers the common case in which one wants to restrict shadows cast by a specific light, and all objects receive shadows from the same groups. The shadow-casting groups can be overridden for a particular object by specifying a user attribute:

`Attribute "user" "string shadowgroups" ["`*`my custom groups`*`"]`

The second parameter above (`shadowattribute`) gives the name of the user attribute to check for a custom shadow casting group. By default all lights look for a custom shadow set in the same user attribute. By providing a different user attribute for each light, one can precisely specify for each object and each light which objects are tested for shadows.

If the shadow group has the special value "null", no shadow rays will be traced and the object will not receive shadows.

See Also

Ray Tracing -> Optimizing Memory for Ray Traced Objects

12.2.2 Shadow Maps

Shadow mapping is a fast and flexible method of producing shadows. A shadow map is generated by storing depth information for a scene when it is rendered from the point of view of a light source. When the scene is rendered normally, the light source can use the depth information to determine if a point is in shadow. Shadow-mapped shadows are usually faster than ray-traced shadows and have smoother soft edges.

Using Shadow Maps

Support for a shadow map must be coded in a light shader. If your plugin supports shadow maps, it will likely include an appropriate shader and use it automatically. Of the shaders included with AIR, the pointlight, spotlight, distantlight, and uberlight shaders support a shadow map.

Soft Shadows

Most lights that support shadow maps provide a `shadowblur` parameter that can be used to blur the shadow map lookup to produce softer shadow edges. Shadow blur for shadow maps is specified as a fraction of the total area of the shadow map.

Shadow Acne and Shadow Bias

Occasionally some objects may exhibit incorrect self-shadowing from a shadow-map, especially when using large shadow blur values. A light shader typically provides a `shadowbias` parameter to help prevent incorrect shadows by providing a small offset for the depth information in the shadow map.

Transparency and Motion Blur

AIR shadow maps work with transparent and motion-blurred objects. For higher quality shadows for transparent objects, consider using fragment shadow maps.

Creating Shadow Maps

Your plugin should have an option to automatically create shadow maps for a light.

A shadow map is created by rendering a scene from the point of view of the light source and recording depth information in an output file. The depth information can be saved in a floating-point TIFF file which AIR can use directly as a shadow map. However, it is better to first convert the depth map to an AIR shadow map with the `MakeShadow` RIB command or the `mktex` utility program. An AIR shadow map uses a tiled compressed format that allows AIR to efficiently cache shadow map requests and minimize memory use during rendering.

Instead of rendering a depth file and then converting that to a shadow map, AIR allows a shadow map to be rendered directly with the special `shadow` display driver. Using the `shadow` driver avoids creating a potentially large intermediate depth file.

Depth Map Values

AIR supports an optional parameter for the Hider command for choosing the type of depth values that are stored in a shadow map:

```
  Hider "hidden" "string depthfilter" ["midpoint"]
```

By default AIR writes a depth value that is the average of the two closest surfaces; this midpoint algorithm helps to minimize shadow artifacts caused by incorrect self-shadowing. If a depthfilter value of `min` is specified, the shadow map file will contain the nearest depth value at each pixel.

Shadow maps generated with the midpoint depth filter generally provide better shadows than maps generated with the min filter. However, midpoint shadow maps can cause artifacts where there is a discontinuity in the second surface visible from the light which results in a large difference between adjacent depth values. AIR 6 and later provide two options that can be used to limit how much the midpoint depth value differs from the min depth value. The first option sets the maximum absolute distance between the midpoint value and the nearest surface position:

```
Option "render" "float maxmiddistance" [1000000000]
```

The second option gives the relative position between the 2 closest surfaces of the value stored in the depth map:

```
Option "render" "float midposition" [0.5]
```

The default value of 0.5 stores a depth value half way between the 2 nearest surfaces. A value of 0.2 would produce a depth value closer to the nearest surface.

Cube-Faced Shadow Maps

Depth maps taken from multiple views can be combined into a single shadow map. This capability can be used to make a cube-faced shadow map for use with a point light. The mktex utility will combine up to 6 separate depth maps into a single shadow map. E.g.,

```
mktex -shadow map.px map.nx map.py map.ny map.pz map.nz shadow.shd
```

The RIB `MakeShadow` command has been extended to accept an array of strings as the first parameter, containing up to 6 depth maps to be joined into one shadow map.

AIR does not place restrictions on the views used for the depth maps that go into a multi-depth map shadow map. For a cube-faced map the FOVs of the depth maps should slightly overlap to prevent gaps.

Another way to create a cube-faced map is to use the shadow display driver and its `append` parameter to build a map. Render the first sub-map to the output file, and then append each additional submap. Sample files for creating and rendering with a cube-faced shadow map can be found in `$AIRHOME/examples/cfshadow`

12.2.3 Fragment Shadow Maps

AIR 5 introduces fragment shadow maps that store additional coverage information and multiple depth samples. The additional information allows fragment maps to produce a better shadow estimate for semi-transparent objects and objects that are thin or small such as hair or particles.

Fragment maps are enabled during the shadow map generation phase by giving the maximum number of depth samples to store in each pixel of the shadow map:

```
Option "limits" "integer shadowmapdepth" [n]
```

AIR creates a fragment map instead of a normal shadow map when the shadowmapdepth is greater than 1. If a given pixel in the shadow map is covered by more fragments than allowed in the depth map, AIR will remove and merge fragments until they fit in the map.

The minimum fragment map depth of 2 works well in many cases. For scenes with complex visibility -

many small or thin objects or many layers of transparency, a larger depth value of 4 to 8 may produce better results.

Another option provides a quality storage tradeoff:

```
Option "render" "integer shadowmapformat" [0]
```

A format value of 0 will generate smaller maps with less precision which may render faster. A format value of 1 stores more precise depth and coverage information.

You may wish to increase the texture cache limit when rendering with fragment maps because they tend to use more memory than conventional shadow maps.

Fragment Interpolation

AIR 8 introduces a new option to enable interpolation between fragments in a fragment shadow map:

```
Option "render" "integer lerpshadowmap" [1]
```

This option may reduce noise and jitter when render dense fur or hair.

12.2.4 Automaps

Automaps are shadow maps automatically created by AIR on-demand. Automaps are currently supported for spotlights, point lights, and directional/distant lights.

Using Automaps

To enable automapped shadows for a light, supply the special `"automap"` keyword as the shadow map name.

An object must be visible to shadow rays for it to cast automapped shadows. Your plugin should provide a control to enable visibility to shadow rays. Because automapped shadows use the "mid-point" filtering method for depth values, objects such as a ground plane that lie behind shadow-casting objects must also be visible to shadow rays.

Automap Size

The size of a light's automap is set with

```
Attribute "light" "integer automapsize" [1024]
```

Automap Coverage

For spotlights AIR looks for a light shader's `coneangle` parameter to determine the region covered by the shadow map.

For distant lights, the coverage of the automap must be set with

```
Attribute "light" "float[4] automapscreenwindow" [left right top bottom]
```

where the 4 parameters define the screen window that would be used to generate a shadow map for the light.

Automap Quality and Appearance

Automaps use the same high-quality filtering as shadow maps, producing smooth soft shadows. The shadow blur and shadow bias parameters are interpreted as they are for shadow maps.

Automap Cache

Automaps are stored in a special cache that limits the total amount of memory consumed by all automaps. The cache size can be set with:

```
Option "limits" "integer automapmemory" [10000}
```

which sets the maximum cache size in kilobytes.

12.3 Indirect Lighting

Topics:

Introduction

Indirect illumination is light that strikes a surface after being reflected, refracted or scattered by a surface, in contrast to direct illumination that strikes a surface directly from a light source.

For efficient rendering, AIR provides separate mechanisms for simulating indirect diffuse illumination - light which is scattered towards a surface by other surfaces - and caustics - light that is reflected or refracted in a focused manner by other surfaces. This section describes how to render indirect diffuse illumination effects with AIR.

AIR uses a variation of the irradiance method for indirect illumination calculations. AIR calculates the incoming indirect light at a point by sampling the hemisphere of directions above the current shading location using ray tracing. Each ray contributes illumination from the surface it strikes or a contribution from the indirect environment if no object is hit.

Adding Indirect Light

Many user interfaces for AIR provide special controls for enabling indirect illumination, often grouped under a section called GI or Global Illumination.

Indirect lighting can also be added to a scene manually as follows:

- Add an `indirect` light source to a scene.

- Make objects that contribute indirect illumination visible to "indirect" rays with

```
Attribute "visibility" "indirect indirect" [1]
```

Quality

The main control over the quality of the indirect illumination results is the number of rays cast, which can be set with the following attribute:

```
Attribute "indirect" "integer nsamples" [256]
```

More rays produce more accurate and smoother results at the expense of longer rendering time.

Sometimes it is only important to capture the indirect light from objects that are nearby the current shading location. The following attribute can be used to restrict the distance to search for objects that contribute indirect light:

```
Attribute "indirect" "float maxhitdist" [100000]
```

Shorter distances can reduce rendering time by reducing the number of objects tested for intersection with each indirect ray.

The following option limits the maximum number of bounces light can take from a light source to a point and appear in the global illumination calculations:

```
Option "indirect" "integer maxbounce" [1]
```

Lower values take less time and use less memory. Since most of the indirect illumination is usually the result of the first bounce, a value of 1 is usually sufficient to produce a noticable effect.

Adaptive Sampling

Adaptive sampling is a method of accelerating indirect illumination. When adaptive sampling is enabled, AIR casts fewer rays initially and then additional rays only in regions where the initial ray casts indicate high variation. Adaptive sampling can be enabled with

```
Attribute "indirect" "integer adaptivesampling" [1]
```

The maximum variance allowed before additional rays are cast can be set with

```
Option "indirect" "float maxvariance" [0.1]
```

Indirect Cache

Because calculating indirect light at every shading location can be prohibitively slow, AIR provides an efficient caching mechanism that stores previous computation results and allows them to be re-used at new locations. Two attributes affect the quality of indirect illumination results using the cache:

```
Attribute "indirect" "float maxerror" [0.0]
Attribute "indirect" "float maxpixeldist" [20]
```

The `maxerror` attribute sets the maximum error that is allowed in any one estimate of the local indirect illumination when reusing samples from the cache. Lowering the maximum error causes the renderer to take new samples more often. Setting maxerror to 0 disables the indirect cache, forcing AIR to compute a new indirect value at every shading location. A reasonable starting value for using the occlusion cache is 0.2.

The `maxpixeldist` attribute gives the maximum distance in pixels over which AIR will re-use a given sample. Lower values cause more samples to be taken more frequently.

The indirect illumination cache AIR calculates can be saved to a file and re-used when a scene is re-rendered. As long as no object or light moves, the same values will still be valid. If the camera position changes, AIR may need to calculate some new samples, but you can still save a lot of time by giving it an initial set of samples. To enable cache writing, use:

```
Option "indirect" "string savefile" [filename]
```

Load a cache file for re-use with:

```
Option "indirect" "string seedfile" [filename]
```

Although AIR uses ray tracing to generate new illumination samples, it does not need to trace rays to calculate indirect illumination using a seed file.

Using a Cache with Moving Objects

An indirect illumination seed file can be re-used in an animated sequence as follows:

1. Set the `maxhitdist` indirect attribute to a reasonably small value for all primitives.

2. Render the scene with all moving objects removed to create an indirect seed file.

3. Mark all animated objects as "moving" with

```
Attribute "indirect" "integer moving" [1]
```

4. Enable the following option:

```
Option "indirect" "moving" [1]
```

5. Render each frame using the seed file.

AIR will cull samples from the indirect illumination cache in the vicinity of moving objects based on the maxhitdist attribute of the objects. New samples will be generated in those areas.

Prepass

In some cases indirect illumination results computed using indirect caching may appear splotchy due to the abrupt way in which new samples are added to the cache. AIR supports an optional automatic pre-pass that records indirect cache values prior to rendering the final image, resulting in smoother indirect illumination. The prepass is enabled with

```
Option "indirect" "integer prepass" [1]
```

AIR can use different quality settings on the prepass than the second pass. Using higher quality parameters on the first pass results in a better final image. A scaling factor for the `maxerror` and `maxpixeldist` attributes is given by

```
Option "indirect" "float prepassfactor" [0.5]
```

If you want to use the same quality settings on both passes (the behavior of AIR prior to release 2.7), set this option value to 1. Increasing the prepassfactor will save rendering time.

Objects that are not illuminated by the indirect light source do not need to be evaluated during the prepass. Such objects should be tagged with the following attribute:

```
Attribute "indirect" "integer prepass" [0]
```

AIR's indirectchannels light shader can track light channels through indirect bounces. In order for the prepass to work properly with channel tracking, AIR must be told to record the per-channel indirect information with the following option:

```
Option "indirect" "string prepasschannels" "__lights"
```

Synchronization

Air 13 introduces a new option to force Air to wait until an indirect or occlusion prepass has finished before rendering the final pass:

```
Option "render" "syncprepass" [1]
```

This option provides a solution in cases where a bucket has not finished computing in the prepass when it is rendered in the final pass. Enabling this option will in general increase rendering time, since it forces all threads to wait until the last thread finishes the prepass. The option is enabled by default.

Prepass and Transparency

During the prepass the renderer only generates indirect illumination, saving time by not running the full set of shaders at each pixel. The renderer will calculate indirect light on the frontmost surface at each point, and for points behind that if the surface's opacity parameter (Os) indicates it is transparent. Some shaders modify the surface transparency in complex ways not tied to the opacity surface attribute. To force the renderer to generate samples behind those objects, set the surface opacity to a non-opaque value, even though it is isn't used by the surface shader.

Prepass and Reflections

Some surfaces have crisp reflections that may clearly reflect indirect illumination in other parts of the scene. To obtain the best results you may also need to precompute indirect values at those reflected locations by tagging a reflective object with the following attribute:

```
Attribute "indirect" "integer reflective" [1]
```

Prepass Groups

The objects that will contribute indirect illumination can be restricted to members of a list of groups by providing the following option:

```
Option "indirect" "string prepassgroup" "list of groups"
```

Shading for Indirect Illumination

AIR calculates the incoming indirect light at a point by sampling the hemisphere of directions around the surface normal using ray tracing. Each ray contributes illumination from the surface it strikes. AIR provides custom attributes for simplifying and thereby accelerating the shading performed to determine a primitive's contribution to indirect illumination:

```
Attribute "indirect" "string shading" [type]
Attribute "indirect " "float strength" [1]
Attribute "indirect" "color averagecolor" [-1 -1 -1]
Attribute "indirect" "string averagecolor_spd" ""
```

The `"indirect:shading"` attribute determines the type of shading performed when a surface is intersected by an indirect ray. Possible values are:

`shade`	Execute the surface's assigned shaders to determine the reflected color.
`matte`	Use a matte surface shader, with the color taken from `strength*averagecolor`. This is the default setting.
`constant`	Use a constant surface shader, with the color taken from `strength*averagecolor`.

If the `"indirect:averagecolor"` attribute is not set, the renderer will use the standard color attribute for the surface (`Cs`) as the average color.

Matte shading can save a considerable amount of time if a primitive has a complicated surface shader assigned. All that's required is an estimate of the average reflectance/color over the surface.

Constant shading saves even more time because the renderer doesn't have to execute the light shaders assigned to the surface to calculate its contribution to indirect illumination. For constant shading, the product of the `averagecolor` and `strength` attributes should be a rough estimate of the average color (including lighting) of the surface.

Environment

In Air 11 and later, indirect rays that miss all objects return a color based on the current environment shader(s) assigned to the scene, if any. Air 12 introduces a new option to automatically cache the environment for indirect queries with:

```
Option "indirect" "environmentcache" [1]
```

Using the cache typically produces smoother results with shorter rendering times.

If no environment shader is defined, the returned color is based on the following legacy options. The default returned color is set with

```
Option "indirect" "color background" [0 0 0]
```

Alternatively, the background can be defined using an environment map specified with the following options:

```
Option "indirect" "string envname" [""]
Option "indirect" "float envstrength" [1.0]
Option "indirect" "float envblur" [0.0]
Option "indirect" "integer envsamples" [4]
Option "indirect" "string envspace" ""
```

Radiosity Cache

Air 13 introduces a new radiosity cache that accelerates indirect lighting computations by storing and reusing the shading results for indiret ray hits. The cache can be enabled on a per object basis with the following atttribute:

```
Attribute "indirect" "radiositycache" [1]
```

When enabled, Air stores the result of shading each indirect ray hit with the associated polygon. If that polygon is hit by another indirect ray, the saved shading result is re-used (instead of evaluating the shaders at the new hit point).

By default one indirect shading sample is stored with each polygon. Large polygons can be shaded more accurately by setting a max size for each indirect shading sample with

```
Option "indirect" "radiositysize" n
```

where *n* gives the maximum radius of each shading point. Polygons larger than the specified size store multiple indirect samples for more accurate shading (at the expense of greater render time and memory use).

Caveats: the radiosity cache does not currently work when tracking light contributions though indirect diffuse calls. The cache also won't produce accurate results if both sides of a surface are visible.

See Also

indirect light shader
Caustics
Ray Tracing -> Optimizing Memory for Ray Traced Objects

12.4 Ambient Occlusion

Ambient occlusion refers to the extent to which a shading location is blocked or occluded by surrounding objects. It provides a good estimate of the extent to which an object is "shadowed" by surrounding objects with respect to an encompassing environment, such as the sky in an outdoor scene. Ambient occlusion is a faster alternative to computing indirect illumination for scenes where the soft lighting effects of hemispherical or "dome" lighting are desirable but color bleeding and bounce lighting effects aren't important.

AIR can compute ambient occlusion using ray tracing, depth maps, or point clouds.

Shaders

AIR includes four shaders that use ambient occlusion, which is encapsulated in the shading language function `occlusion()`.

The `envlight` and `massive_envlight` light shaders provide a "dome" or sky lighting effect with occlusion used for shadowing. The massive_envlight shader has parameters for most properties affecting ambient occlusion. This shader is a good choice when not using AIR's occlusion cache feature. With the envlight shader most occlusion properties are controlled with AIR attributes. This shader is recommended when using the occlusion cache feature.

The `occlusionpass` and `massive_occlusionpass` surface shaders set the surface color to the occlusion value, allowing a separate ambient occlusion pass to be rendered and used for compositing. The occlusionpass shader is recommended when using an occlusion cache. The massive_occlusionpass shader provides fairly complete controls for rendering occlusion without an occlusion cache.

Examples

Examples using occlusion can be found in

`$AIRHOME/examples/occlusion`.

12.4.1 Ray-Traced Occlusion

Ray traced occlusion estimates the occlusion at a location be casting shadow rays distributed over a cone of directions (usually a hemisphere) testing for shadowing or occluding objects.

Many applications and plugins provide a special interface for enabling and configuring ray-traced ambient occlusion, often grouped with other controls under a GI or Global Illumination dialog.

You can also add ambient occlusion to a scene manually by either:

1. Assigning an occlusionpass or massive_occlusionpass surface shader to all objects
2. Adding an envlight or massive_envlight light shader to the scene.

Because traced occlusion casts shadow rays to estimate occlusion, any objects that contribute to occlusion must be made visible to shadow rays. A plugin may provide a control for enabling traced shadow visibility, or visibility to shadow rays can enabled with

```
Attribute "visibility" "integer transmission" [1]
```

Occlusion Quality

The main control over occlusion quality is the number of rays cast to estimate occlusion. Some shaders (such as massive_envlight and massive_occlusionpass) provide a parameter for setting the number of rays to cast for occlusion. Shaders that do not provide such a parameter will inherit the number of rays from the corresponding indirect attribute:

```
Attribute "indirect" "integer nsamples" [256]
```

Using more rays will produce more accurate and smoother ambient occlusion results at the cost of longer rendering times. A useful guideline when making adjustments is to increase or decrease the number of rays by a factor of 2.

Often it is only important to capture the occlusion resulting from objects that are nearby the current shading location. The following attribute can be used to restrict the distance to search for occluding objects:

```
Attribute "indirect" "float maxhitdist" [100000]
```

Using as low a max hit distance value as possible can greatly accelerate occlusion estimation by limiting the number of potentially occluding objects. Using an appropriately low hit distance can also allow occlusion to be useful in interior scenes.

Adaptive Sampling

Adaptive sampling is another method of accelerating ambient occlusion. When adaptive sampling is enabled, Air casts fewer rays initially and then additional rays only in regions where the initial ray casts indicate high variation. Adaptive sampling can be enabled with

```
Attribute "indirect" "integer adaptivesampling" [1]
```

The maximum variance allowed before additional rays are cast can be set with

```
Option "occlusion" "float maxvariance" [0.1]
```

Occlusion Caching

Because tracing shadow rays to estimate occlusion is an expensive operation, Air allows occlusion values to be saved and re-used just as indirect illumination samples can be. Caching of occlusion values is controlled with the optional `maxerror` and maxpixeldist parameters passed to the occlusion function(), or if those are not provided, by the corresponding indirect attributes:

```
Attribute "indirect" "float maxerror" [0.2]
Attribute "indirect" "float maxpixeldist" [20]
```

If `maxerror` is 0, Air will not cache occlusion values, and occlusion will be recalculated at every shading location. Rendering without a cache is obviously slower, but it can produce superior results. Other values of `maxerror` give the maximum error allowed in an occlusion estimate before a new sample is generated.

The occlusion cache can be saved to a file with

```
Option "occlusion" "string savefile" filename
```

and later re-used with

```
Option "occlusion" "string seedfile" filename
```

Re-rendering with a seed file does not require ray tracing. The occlusion cache can be saved to any of the 3D point file formats supported by Air. For normal re-rendering with a saved cache, the Air Point Cloud format (.apc) is recommended.

Occlusion Prepass

Air supports an optional automatic pre-pass that records occlusion samples for re-use on a second pass, resulting in a smoother final image. The prepass is enabled with

```
Option "occlusion" "integer prepass" [1]
```

Air can use different quality settings on the prepass than the second pass. Using higher quality parameters on the first pass results in a better final image. A scaling factor for the `maxerror` and `maxpixeldist` attributes is given by

```
Option "occlusion" "float prepassfactor" [0.5]
```

Normally you will not need to change this option.

Air 13 introduces a new option to force Air to wait until an indirect or occlusion prepass has finished before rendering the final pass:

```
Option "render" "syncprepass" [1]
```

This option provides a solution in cases where a bucket has not finished computing in the prepass when it is rendered in the final pass. Enabling this option will in general increase rendering time, since it forces all threads to wait until the last thread finishes the prepass. The option is enabled by default.

Screen Space Cache

By default Air stores the cache as a set of 3D point samples. AIR 6 introduces a new screen-space cache mode that can be enabled by setting the maxpixeldist attribute to 0 and maxerror>0. In this mode Air will store one occlusion sample per pixel for possible re-use. This produces an appearance similar to rendering without a cache in less time. It also decouples the frequency of occlusion sampling from the shading rate.

Sampling the Environment

Air 11 introduces a new option to sample the current scene environment for unoccluded rays. The envlight shader provides a `sampleenvironment` parameter that enables this new mode. When the occlusion prepass is enabled, you should also enable environment sampling during the prepass with:

```
  Option "occlusion" "prepasssampleenvironment" [1]
```

When used with the occlusion cache, environment sampling can be noticably faster (and more accurate) than the old method of querying an environment map based on the average unoccluded direction vector.

See Also

Ray Tracing -> Optimizing Memory for Ray Traced Objects
Lighting -> Indirect Lighting

12.4.2 Depth-Mapped Occlusion

Occlusion maps offer an alternative to ray tracing for producing ambient occlusion estimates and dome lighting effects.

An occlusion map is basically just a collection of individual shadow maps. The key difference from an ordinary shadow map or a cube-face shadow map is that an occlusion map query returns the combined shadow results for all the shadow maps. Querying an occlusion map created with shadow maps from a range of directions around an object can be used to estimate ambient occlusion.

Creating an Occlusion Map

Method 1:

Combine multiple depth maps into an occlusion map with the RIB `MakeShadow` command:

```
MakeShadow [
  "shadow1.z"
  "shadow2.z"
  "shadow3.z"
  "shadow4.z"
      ...
  "shadowN.z"] "occlusion.sm"
```

Method 2:

An occlusion map can be built incrementally using the `"append"` parameter of the AIR `shadow` display driver. The first shadow map is rendered with a `Display` call such as:

```
  Display "occlusion.sm" "shadow" "z"
```

Subsequent shadow maps use

```
Display "occlusion.sm" "shadow" "z" "float append" [1]
```

to add to the occlusion map.

Using an Occlusion Map

The `occlusion()` function accepts an occlusion map as the first parameter. The optional bias, blur, and samples parameters function as they do for a shadow map. The `envlight` and `occlusionpass` shaders included with AIR both work with occlusion maps. For example, to add an environment light with an occlusion map, you might use:

```
LightSource "envlight" 9 "string mapname" "occlusion.sm"
```

When using an occlusion map, you may need to increase the texture cache size for good performance. E.g.,

```
Option "limits" "texturememory" [30000]
```

will increase the cache size from the default of 10MB to 30MB.

12.4.3 Point-based Occlusion

Air 10 introduces point-based ambient occlusion which computes occlusion using a 3D texture file or point cloud.

Step 1: Baking a 3D point file

The first step to utilizing point-based occlusion is to generate a suitable 3D point file.

The 3D point file should be saved as an Air Point Cloud (.apc) file. It is very important that the 3D point file include all occluding surfaces in the scene, not just those that are visible to the camera.

The simplest way to produce such a point set is to use the bake3d surface shader included with Air in "collect" mode, which uses the same surface-sampling capability employed by Air's SSS feature. To do that, set the `BakeRadius` parameter to the maximum radius or size for any point in the cloud. Smaller values produce a more accurate but larger point set.

Set the `BakeMap` parameter to the name of the 3D texture file, with a `.apc` extension.

The point cloud only needs basic position, normal, and width information for each point. To minimize the size of the point data, set the `BakeChannels` parameter to the empty string to avoid writing any additional channels.

To use the collect mode, all surfaces also need to be tagged members of the same point set with the following attribute:

```
Attribute "pointset" "handle" "occluders"
```

Here is a complete rib snippit that can be used to bake an exported rib file:

```
Surface "bake3d"
  "string BakeMap" "occlusion.apc"
  "string BakeChannels" ""
  "float BakeRadius" [0.1]

Attribute "pointset" "handle" "occluders"
```

```
Option "render" "commands" "-Surface"
```

The last line overrides all other surface shaders in the rib file. Save the above to a texture file, called say bakeocc.rib, then pass that file to Air prior to the main scene file:

```
air bakeocc.rib myscene.rib
```

Step 2: Rendering with point-based occlusion

The 3D point file representing occluding surfaces in the scene can be used to produce occlusion with the occlusionpass surface shader or the envlight light shader.

Set the `mapname` parameter to the file name of the 3D texture.

Adjust the `mapbias` parameter to prevent incorrect self-occlusion.

Control quality vs speed with the `maxsolidangle` parameter. To accelerate the occlusion estimate, Air groups points that are far away from the current shading location and treats them as a single occluder. The maxsolidangle parameter gives the maximum solid angle in radians that an approximate occluder may occupy before Air substitutes a more detailed representation. Larger values produce a coarser occlusion result with lower render times.

Here's a complete rib snippit for rendering an occlusion pass using a 3D point file:

```
Surface "occlusionpass"
  "string mapname" "occlusion.apc"
  "float mapbias" 0.1
  "float maxsolidangle" 0.05

Option "render" "commands" "-Surface"
```

Save the above as occpass.rib, then render with

```
air occpass.rib myscene.rib
```

Point-based occlusion may be faster than traced occlusion for very complex scenes. For simpler scenes, point-based occlusion may be slower than simple ray traced occlusion.

12.5 Image-Based Lighting

An environment map can be used to illuminate a scene with either indirect diffuse illumination or ambient occlusion. In either case, the source image should first be converted to an AIR environment map with AIR's texture conversion tools.

Converting a Light Probe Image

A light probe is a popular method of recording the lighting in an environment by photographing a reflective ball placed in the environment. By using the light information contained in a light probe image, one can make a computer-generated object look like it is being "lit" in that environment. For best results, the light probe image should be a high-dynamic range image, which captures the full range of the light in the environment.

Light probe images are often stored in an "angular" format or a mirrored ball format. Images in these formats can be converted for use with AIR by the mktex or mktexui utilities. For mktexui:

- Start mktexui

- Select the source image.
- Set the Map Type to Angular Map or Mirror Ball depending on how the source image was generated.

Image-based lighting usually only rquires a small environment map. You can set the output size to reduce a large source image.

- Set the Output Size to 256 128.

The texture converter also allows the orientation of the environment map relative to the scene to be altered using the Rotate 3D controls.

- Set the Output Directory and click the Make Map button.

IBL with Indirect Diffuse Illumination

If indirect diffuse illumination is being used in a scene, the IBL environment map can be added as the background map for the indirect diffuse lighting calculations with:

```
Option "indirect" "string envname" [""]
Option "indirect" "float envstrength" [1.0]
Option "indirect" "float envblur" [0.0]
```

Indirect rays that miss all objects in the scene will take their color from the environment map. The `envstrength` option can be used to scale the environment map intensity, while the `envblur` parameter adds blur to the environment map lookup.

IBL with Ambient Occlusion

Image-based lighting can also be accomplished using ambient occlusion with the `envlight` or `massive_envlight` light shaders. Add a light source with the `envlight` shader to your scene, and set the `envname` parameter to the name of the environment map.

Example: an example can be found in `$AIRHOME/examples/hdribl`

12.6 Photometric Lights and IES Profiles

AIR 10 and later can utilize IES light profiles to simulate the illumination properties of specific light fixtures. IES light simulation requires the new IES file reader and the new photometric_pointlight light shader included in AIR 10.

IES Profiles

IES light profile files represent the distribution of light from a fixture as a "photometric web" of measured intensity values over a range of directions. IES file support is implemented as an image reader that converts the directional intensity information into a lat-long environment map that can be queried using the standard shading language `environment()` function. The output of the IES reader is a single-channel file at float precision storing the interpolated light intensity in candela.

Viewing IES profiles

An IES file can be viewed in AIR Show just like images in any other image format supported by AIR. By default AIR Show assumes a single-channel file of float precision is a shadow map and applies some non-standard display processing to the image. You can turn off this special processing in AIR Show 4.03 or later by disabling the `Detect shadow map` option in the Options menu prior to loading an IES file.

Photometric lighting

The new photometric_pointlight shader in AIR 10 allows an IES profile to be used to illuminate a scene as follows:

1. Set the `iesname` parameter to the file name of the IES profile.

2. Optionally use the `iesrotatex, iesrotatey`, and `iesrotatez` parameters to rotate the orientation of the profile.

3. Set the `sceneunitsize` parameter to size of one unit in world space in meters. E.g., if your scene is modeled in centimeters, `sceneunitsize` should be 0.01

4. Set `illuminanceunity` to the illuminance (in lux) corresponding to a non-physical light source intensity of 1. This value is used as a divisor for the light's emitted intensity; larger values will cause the light to appear dimmer.

Notes

The `illuminanceunity` parameter tells the renderer how to map the physical light data into a rendering environment that lacks a physical camera model and may include non-physical lights. All lights in a scene (and in any scenes that will be compared) should have the same `illuminanceunity` value. To make it easier to specify a common value, the photometric_pointlight shader looks for a unity value in the following user option:

```
Option "user" "float illuminanceunity" [200]
```

When the user option is present in the scene rib file, the `illuminanceunity` parameter value is ignored.

Similarly, the photometric_pointlight shader looks for the scene unit size in

```
Option "user" "float sceneunitsize" [0.01]  # cm to m
```

and ignores the `sceneunitsize` parameter value when the option is detected.

Photometric illumination without an IES profile

When the photometric light shader is used without an IES profile, the light intensity can be specified in physical units by setting the `intensityunit` parameter to one of the following supported values:

`none`	intensity acts as a simple scalar with no conversion
`candela`	intensity is divided by illuminanceunity
`lumen`	intensity is converted to candela by dividing by 4 PI
`watt`	intensity is converted to lumen using `lumensperwatt` value

When the `watt` unit is selected, the `lumensperwatt` parameter should be set appropriately for the type of illuminant (e.g., 8-17 for an incandescent bulb, 60-72 for a compact flourescent).

12.7 Caustics

Caustics are illumination effects from light that has been reflected or refracted in a focused manner onto a surface. AIR simulates caustics using photon mapping.

Producing Caustics

Producing caustics requires several steps:

1. Add a caustic light source to the scene:

```
  LightSource "caustic" 99
```

Turn the light on for any surfaces that are to receive caustics, and off for all other surfaces.

2. Set the number of photons to trace for each light that is to produce caustics:

```
  Attribute "light" "integer nphotons" [100000]
```

3. Make primitives that are to receive or produce caustics visible to indirect/photon rays with

```
  Attribute "visibility" "integer indirect" [1]
```

4. For each primitive that is to produce caustics by reflecting or refracting light, set the reflective color, refractive color, and refraction index for caustics:

```
  Attribute "caustic" "color specularcolor" [0 0 0]
  Attribute "caustic" "color refractioncolor" [0 0 0]
  Attribute "caustic" "float refractionindex" [1]
```

For caustics to be visible, there must be at least one object that is lit by the caustic light and at least one object that reflects or refracts photons.

The `specularcolor` and `refractioncolor` attributes determine what happens to photons that intersect a surface. Photons will be reflected, refracted or absorbed in proportion to (respectively), the average intensity of `specularcolor`, the average intensity of `reflectioncolor`, and 1 minus the sum of the average intensities. The sum of `specularcolor` and `reflectioncolor` should be less than or equal to 1. If the sum is greater than or equal to 1, no photons will be stored on that surface and no caustics will be visible.

Example:

```
  Attribute "caustic" "color specularcolor" [.4 .4 .4]
  Attribute "caustic" "color refractioncolor" [.25 .25 .25]
```

In this case, 40% of incoming photons (light) would be reflected, 25% would be transmitted, and 35% would be stored and visible as part of the caustic illumination.

When simulating caustics AIR shoots photons for all lights to build the photon map prior to beginning the main rendering pass. No separate prepass is required for caustics.

Caustic Quality

The following attributes affect the quality of caustic results.

```
Attribute "caustic" "float maxpixeldist" [10]
```

Gives the maximum distance (in pixels) over which to gather photons for a caustic estimate. Larger values produce smoother results and require that fewer photons be used overall.

```
Attribute "caustic" "integer ngather" [75]
```

gives the minimum number of photons to gather for a caustic estimate. No caustics will appear at locations where insufficient photons are found within the distance given by the maxpixeldist attribute.

Recycling Photons

The photon map used for calculating caustics can be saved to a file with:

```
  Option "caustic" "string savefile" "filename"
```

and re-used with

```
  Option "caustic" "string seedfile" "filename"
```

Caustics can be saved and loaded from any 3D point format supported by AIR. The recommended point format for caustics is an AIR Point Cloud (extension `.apc`).

Loading a seed file does not preclude shooting additional photons. If you want to use only the photons in a seed file, be sure to set the number of emitted photons for all light sources to 0.

Note that while shooting photons requires ray tracing, rendering caustics with only a seed file does not. For example, a seed a file could be created using a small subset of objects from a larger scene and re-used for the entire scene.

Dispersion

AIR can simulate the effects of dispersion (e.g., prism-like effects) in the photon map.

```
Attribute "caustic" "float dispersion" [0]
```

gives the variation in the index of refraction over the visible spectrum. E.g., if the following caustic attributes were defined

```
  Attribute "caustic" "float refractionindex" [1.5]
  Attribute "caustic" "float dispersion" [0.3]
```

The index of refraction would range between 1.2 and 1.8.

An example can be found in

```
  $AIRHOME/examples/effects/prism.rib
```

12.8 Subsurface Scattering (SSS)

AIR provides fast simulation of the subsurface scattering effects common in soft or partially translucent materials such as skin and marble. Subsurface scattering is implemented as a new shading language function, `subsurfacescatter()`, and a capability to store and query a collection of points distributed over a group of surfaces.

Using Subsurface Scattering

Enable subsurface scattering for an object as follows:

1. Make all primitives that make up the object part of the same point set as defined with

```
Attribute "pointset"
  "string handle" "myobject.pts"
```

The handle name should be a valid file name if you wish to save the point cache to a file.

2. Assign a shader that uses the subsurfacescatter function, such as the VTranslucent, VSkin, or VTranslucentMarble shaders included with AIR.

3. Set the shader parameters appropriately for your object:

Scattering Distance

The most important parameter to set is the one that controls the scattering distance. Most shaders provide one parameter for setting the distance for the red, green, and blue color components, and another parameter to scale the distance.

Use the ScatterDistance parameter to define the scatter properties for a particular type of material, e.g., skin. A table of real-world values can be found in Jensen et al "A Practical Model for Subsurface Light Transport" in the SIGGRAPH 2001 Proceedings.The mean scattering distance can be calculated from the reduced scattering (sigma_s) and absorption coefficients (sigma_a):

```
msd = 1/sqrt(3*(sigma_s+sigma_a)*sigma_a)
```

The default scatter distances in the shaders provided with AIR are all in units of millimeters. Use the ScatterDistanceScale parameter to adjust the distance to the scale of your object. Larger distances produce more blur and generally take less time to render.

Subsurface Reflectance

This parameter sets the nominal reflectance for subsurface scattering. Real-world values can be found in Jensen et al. Although it is possible to control the color of an object by changing this parameter, the net effect in interaction with the scattering distance parameter can be difficult to predict. Instead, use the primitive's color attribute or other color options in the shader to tweak the overall color, with the SubsurfaceReflectance value set for a particular material type.

When a subsurface scattering object is first encountered, AIR calculates and stores a set of shaded points distributed over the object's surface. You may notice a delay as AIR performs these computations. After the point cache is generated, rendering should proceed relatively quickly.

AIR shades points in the point cache from the "outside" of the object as determined by the current orientation. If your object looks unlit with subsurface scattering, try reversing the orientation of the object.

Subsurface Quality

To save rendering time AIR aggregates points in the SSS cache that are far away from the current shading location. If this optimization is performed too aggressively, it can result in rendering artifacts in the form of spots of darker color in the subsurface scattering result. The max error tolerance for this optimization can be set with:

```
Option "render" "float subsurfacetolerance" [0.005]
```

Setting the tolerance to 0 disables the optimization entirely.

Saving and Re-using a Subsurface Cache

The point cache used to accelerate subsurface scattering can be saved to a file using the `filemode` attribute:

```
Attribute "pointset" "string filemode" "w"
```

The point set will be saved to the file name given by the point set handle.

To re-use a point set on a subsequent render, set the `filemode` to read with:

```
Attribute "pointset" "string filemode" "r"
```

Subsurface scattering point sets can be saved in any of the 3D point formats supported by Air.

Multithreading

In AIR 10 and later versions subsurface cache generation is multithreaded for better performance. Multithreading for SSS cache generation can be disabled with

```
Option "render" "integer sssmultithread" [0]
```

Automatic Point Set Handles

Air 11 introduces a new option to automatically generate a point set handles:

```
Option "render" "sssautocache" [1]
```

When this option is enabled, Air generates a unique point set handle for each primitive whose surface shader contains a call to the subsurfacescatter function. MayaMan users should use this option with caution: MayaMan shaders always include a subsurfacescattering() call, even when that option is not used, which will cause Air to retain objects for SSS cache generation when this option is enabled.

Custom Functions with collect()

If the builtin `subsurfacescatter()` function doesn't meet your needs, you can write your own subsurface scattering function using the `collect()` statement, which exposes the underlying point set cache and query capability used by subsurfacescatter. For an example of a custom subsurface scattering solution, see

```
$AIRHOME/examples/surbsurface/collectss
```

Naturally `collect()` is good for many other exotic effects as well.

See Also:

3D Textures & Point Sets
VTranslucent surface shader
VSkin surface shader

12.9 Baking Shadows and Illumination

Baked Shadows

The new distantlight_baked, pointlight_baked, spotlight_baked, and envlight_baked shaders extend the corresponding light shader with a few new parameters for baking. The two most important are:

`string bakemode` - bake mode
`string bakemap` - name of bake map

Bake Modes

The following bake modes are supported:

write: write shadow information to a 3D texture file. Recommended file format is .apm or .apc.

read: read shadow information from a 3D texture file. If the bake map does not contain relevant shadow information for the current shading location, the light shader executes the normal shadow evaluation code based on the shadowname setting.

append: adds samples to an existing 3D texture file. Useful for building a texture cache for an animated sequence

readNDC: read shadow values from a 2D texture map generated with a regular shadow pass. The texture map is indexed using Normalized Device Coordinates (NDC) which map the unit square to the current rendered image. The map is only used for primary (camera rays). Secondary rays generate shadows as usual.

readst: read shadow values from a 2D texture map indexed using standard texture coordinates. This mode is suitable for maps baked with BakeAIR. This mode only works in AIR 8.16 or later which allows lights access to the standard texture coordinates of an illuminated surface.

The bake map parameter can reference either a 2D or 3D texture file depending on the bake mode. Per-object bake maps can also be specified by setting the bakemap value to the name of a user attribute storing the map for each object. E.g.,

```
LightSource "distantlight_baked" 1
  "string bakemode" "readst"
  "string bakemap" "user:bakedshadow1"
...
Attribute "user" "string bakedshadow1" "shadow1object1.tx"
Sphere 1 -1 1 360
```

Per-object maps may be useful when using maps with baked shadows generated by BakeAIR.

Baked Shadows for Area Lights

The new arealight_baked and arealight2_baked shaders work similarly to the bake shaders mentioned above with the following differences:

The arealight_baked shader cannot create a 3D texture file; bake modes write and append are not supported. The arealight_baked shader can read a 3D shadow file generated by the arealight2_baked shader.

The arealight2_baked shader stores only shadow information in a 3D texture file. It does not store the virtual light location returned by the fast areashadow() function. As a consequence rendering using a 3D texture file will look somewhat different than a normal render with the arealight2 shader.

Baked Indirect Illumination and Caustics

The new indirect_baked and caustic_baked shaders allow baking of indirect diffuse and caustics respectively. These shaders support the same modes described above for standard shaders, with the natural difference that the light's emitted illumination is stored instead of a shadow value.

Note that for caustics rendering with AIR's builtin photon cache may be faster than using a baked 3D point file.

To facilitate the use of the readNDC mode for these shaders, the AIR 9 distribution includes new IndirectPass and CausticPass surface shaders for rendering an image with just the indirect diffuse or caustic illumination values.

Using Baked Maps with Moving Objects

Some baking shaders provide additional parameters that allow baked maps to be used in scenes with moving objects:

`float[6] motionbound` - 6 numbers defining a bounding box for moving objects in a scene as *minx maxx miny maxy minz maxz*
`string motionspace` - the coordinate space for the motion bound

A shader with these parameters will not query a baked map on points inside the motion bound or (for shadows) on points where a shadow ray might intersect the bound. Normal shadow computations are performed in those areas.

Food for Thought

It is perhaps worth mentioning that the 2D or 3D texture file used with one of the bake light shaders need not have been generated with that light. For example, it's possible to use an occlusion pass as the bakemap (with bakemode=readNDC) for a spotlight, or say paint a 2D light map for use with the indirect_baked shader.

13 Ray Tracing

Topics:

13.1 Importance-Based Rendering

Air optimizes ray tracing calls based on how important they are to the overall shading result. For example, if a shader indicates that a reflection call will contribute only 1% to the final pixel color, Air can use that information to produce a faster, lower-quality result than a reflection that contributes 50% to the final pixel color.

The following options apply to importance-based rendering:

```
  Option "trace" "float minweight" 0.01
```

gives a minimum weight below which ray tracing will stop. AIR fades trace results near the threshold to help avoid popping artifacts. This optimization can be very effective in scenes with many levels of interreflection.

```
  Option "trace" "float reducesamples" 0.1
```

gives a weight threshold below which Air will begin reducing the quality of shadow and reflection results.

Shader Writing and Importance

Specifying Importance

A shader is responsible for passing importance information to the renderer. The importance or "weight" of a ray is specified by providing a color "weight" parameter to a `trace()`, `environment()`, or `shadow()` function call.

In a surface shader that supports reflections, the weight of the `trace()` or `environment()` should be the total multiplier applied to the traced results. E.g., if the reflection component calculation were:

```
  color Crefl = Kr * BaseColor * trace(P,R);
```

an updated version with a weight parameter might look like

```
  color weight= Kr * BaseColor;
  color Crefl = weight * trace(P, R, "weight", weight);
```

In a light shader with a `shadow()` call, the shadow weight should be the unshadowed output color from the light. E.g.,

```
  Cl = intensity * lightcolor / (L.L);

  color inshadow=0;
  if (shadowname!="")
     inshadow = shadow(shadowname, Ps, "samples", shadowsamples,
                       "blur", shadowblur,
                       "bias", shadowbias,
                       "weight", Cl);
```

Querying Importance

The importance of the current shader evaluation can be queried with the `rayinfo()` function:

```
  color weight=1;
  rayinfo("weight", weight);
```

Importance information can be used to further optimize a shader for situations where lower quality results are acceptable. E.g., a shader that computes a detailed pattern might substitute a simpler pattern that is less costly to compute when the importance drops below a certain threshold.

13.2 Optimizing Memory for Ray Traced Objects

Air 11 introduces a new option for optimizing the memory used when storing objects that are ray-traced. Enable memory optimization with:

```
  Option "render" "optimizetracememory" [1]
```

The basic idea: some objects need to be ray traced but do not need to be shaded when they are hit by traced rays. Once such objects have been processed by the main scanline renderer, we can saved memory be freeing shading data such as surface normals and texture coordinates if that data is not required for ray tracing queries.

Specifically, Air can free at least some shading-related primitive data for traced objects that are:

- Invisible to reflection rays

- Invisible to shadow rays or visible to shadow rays with transmission hit mode `opaque` or `primitive`
- Invisible to indirect rays or visible to indirect rays with indirect shading mode `constant` or `matte`

For a typical Massive agent this optimization can reduce ray tracing memory by 1/3 to 1/2.

See Also

Lighting -> Indirect Lighting: Shading for Indirect Illumination
Lighting -> Shadows -> Ray Traced Shadows: Ray Traced Shadow Attributes

14 Reflections

Topics:

14.1 Ray Traced Reflections

AIR can simulate reflections using ray tracing or environment maps. Both methods may be used in the same scene, with some objects using environment maps and some using ray tracing.

Ray tracing reflections is accomplished with the aid of a suitable surface shader. Many of the shaders included with AIR support reflections, including the `VMetal` shader and the `VPlastic` shader.

Most shaders that support reflections offer the option of using either ray-traced reflections or some sort of reflection map.

- To enable ray-traced reflections, set the `envname` or `ReflectionName` parameter of a surface shader to the special name "raytrace".
- If the shader has a parameter such as `ReflectionSpace` or `envspace` for the coordinate space for the reflection vector, set that to "current". (Most shaders included with AIR will automatically detect the special raytrace keyword and use current space regardless.)

In addition to enabling ray tracing in a surface shader, objects that will appear in reflections must be made visible to reflection or trace rays with the following attribute:

```
Attribute "visibility" "integer trace" [1]
```

The plugin for your modeling package should provide a control for this property or translate the modeler's equivalent property.

Trace Bias

Occasionally you may observe spots or other artifacts in ray traced reflections due to incorrect self-intersection of a reflected ray with the reflecting object. These artifacts can often be eliminated by providing a small bias used to offset each traced ray from its original position:

```
Attribute "trace" "float bias" [0.01]
```

Note that the bias setting affects all ray tracing operations.

Blurry Reflections

Most shaders supporting reflections have a `ReflectionBlur` or `envblur` parameter for producing blurry reflections. Larger values produce blurrier reflections. To reduce the noise in blurry reflections, increase the number of reflection samples using the `ReflectionSamples` or `envsamples` parameter.

Maximum Ray Depth

The maximum depth to which AIR will recursively trace rays for reflection and refraction is set with

```
  Option "trace" "integer maxdepth" [6]
```

Reflected Background

AIR provides options to specify a color or environment map used to color reflection rays that miss all objects in a scene. A background color can be given with

```
  Option "reflection" "color background" [0 0 0]
```

A background environment map can be declared with

```
  Option "reflection" "string envname" [""]
  Option "reflection" "float envstrength" [1.0]
  Option "reflection" "integer envsamples" [8]
  Option "reflection" "float envblur" [0.0]
  Option "reflection" "string envspace" ""
```

Motion Blur in Reflections

AIR will blur motion for reflected objects with the following attribute set

```
  Attribute "trace" "integer motionblur" [1]
```

Valid parameter values are 0 (disabled) and 1 (traced blur enabled).

For motion blur to appear in reflections, the reflecting surface shader must use more than 1 ray sample or at least some reflection blur.

Traced motion blur will only display the first motion segment of multisegment motion blur.

#ReflectionGroups
### Reflection Groups

AIR can optionally restrict objects appearing in reflections to members of specific groups. AIR shaders that allow reflection groups will have a ReflectionGroups parameter accepting a space or comma-separated list of groups. Only members of the specified list of groups will appear in ray-traced reflections.

14.2 Reflections using Environment Maps

Environment mapping is a fast alternative to ray-traced reflections. Environment maps store a view of the environment surrounding an object. Reflections are simulated by simply looking up a color in the environment map, saving the time and memory used when reflections are ray-traced.

There are three main types of environment maps, distinguished by the method of projection: mirror reflection maps, spherical maps, and cube-faced maps.

Mirror Reflection Maps

Mirror or planar reflection maps are used to simulate reflections from flat surfaces, such as a tabletop, floor, or wall mirror. Reflection maps rely on the fact that the image seen in a planar mirror can be generated by reflecting the camera about the mirror plane. That relation implies that a mirror reflection map is only valid for a given camera position. A reflection map should have the same resolution and aspect ratio as the final image.

Your plugin may provide a facility for automatically generating mirror reflection maps.

A mirror reflection map can be used with any shader that provides an environment map parameter and means of setting the coordinate space used for indexing the environment map. The coordinate space should be set to "NDC" (Normalized Device Coordinates).

Sample shader declaration:

```
Surface "VMetal"
  "string ReflectionName" "floor.tx"
  "string ReflectionSpace" "NDC"
```

Spherical Environment Maps

Spherical or latitude-longitude environment maps store the surrounding environment using a spherical projection. Spherical maps are often generated by hand with a paint program.

A raw spherical texture map should be converted to a latitude-longitude environment map using the mktexui or mktex conversion utilties. Here's a sample command line for mktex:

```
mktex -envlatl mymap.tif mymap.tx
```

A spherical map can be used with any shader that supports environment maps. The coordinate space for the environment lookup should normally be set to `"world"`.

Sample shader declaration:

```
Surface "VMetal"
  "string ReflectionName" "blueskygreenground.tx"
  "string ReflectionSpace" "world"
```

Cube-faced Environment Maps

A cube-faced environment map is generated by rendering 6 orthogonal views of a scene from the point of view of the reflective object. The 6 maps are combined into a single reflection map using the mktex facility:

```
  mktex -envcube px nx py ny pz nz envfile.tx
```

A plugin may automatically generate cube-faced environment maps.

A cube-faced environment map can be used with any shader that supports environment maps. The coordinate space parameter (if present) should normally be set to `"world"`.

Sample shader declaration:

```
Surface "VMetal"
  "string ReflectionName" "teapotrefl.tx"
  "string ReflectionSpace" "world"
```

14.3 Anisotropic Reflections

AIR ships with two shaders for simulating anisotropic reflections such as those seen on brushed metal surfaces. The VBrushedMetal shader can be used for brushed metal surfaces, and the VBrushedPlastic shader for plastic surfaces with fine grooves.

Most shaders that compute anisotropic reflections use a tangent vector on the surface to orient the reflections. Valid tangent vectors are always available on parametric surfaces such as NURBs or bilinear or bicubic patches. For other primitive types such as polygon meshes and subdivision surfaces, AIR generates approximate tangent vectors based on the standard texture coordinates defined for the mesh.

14.4 BRDF Reflection Sampling

Air 13 and later can sample the environment using reflection rays based on a particular BRDF (bidirectional reflection distribution function). This importance sampling method allows the behavior of reflections and specular highlights to be controlled using the same parameters while producing consistent results. The new functionality enables a new set of shaders that implement physically plausible shading models while reducing the number shader parameters and unifying the behavior of specular highlights and reflections:

SimpleMetal
SimplePlastic
VPhysicalMetal
VPhysicalPlastic

For programmers, the documentation for the `environment()` function has a list of supported BRDFs and technical information on how to sample based on a BRDF.

15 Motion Blur, DOF, & LOD

Topics:

Motion blur
Depth of field
Level of detail

15.1 Motion Blur

AIR can blur the appearance of fast moving objects in an animated scene. The shutter command gives the open and close times for the camera shutter. In a scene file this command appears as:

```
Shutter opentime closetime
```

There should be an equivalent setting in a user interface that supports animation. An animation program will set the appropriate shutter times for each frame automatically.

If *opentime<closetime*, AIR will blur the appearance of moving objects described using motion blocks. To calculate the position of objects for motion blur, AIR records the position of objects at the open and close times, and linearly interpolates between those positions for intermediate times.

The smoothness of motion blur depends directly on the number of pixel samples used. AIR employs brute force supersampling for motion blur. The number of copies created of each object is currently set by the total number of pixel samples. The copies are scan-converted and averaged to produce a motion-blurred image. More samples produce smoother blur but also take longer to render.

The shutter for a typical motion picture camera is open about half the time between frames. You may wish to set the shutter times for animation to mimic that behavior.

Motion blur typically requires a large number of samples (8 x 8 or higher) to achieve good results.

Multi-segment Motion Blur

AIR supports multi-segment vertex and transformation motion blur with up to 32 time values. AIR ignores the time values, and uniformly interpolates among the supplied transformation or vertex values.

Ray Tracing Motion Blur

Ray tracing results will include motion blur for primitives that have the following attribute set:

```
  Attribute "trace" "integer motionblur" [1]
```

Motion blur is supported by caustics as well as the shading language functions `environment()`, `shadow()`, and `areashadow()`. Motion blur will only appear when 2 or more samples or rays are traced for query. Use more samples with these functions to increase the quality of the blur.

Ray tracing with motion blur can be very time-consuming, and users are advised to selectively enable it only where necessary. For curved surfaces, use of the following motion factor optimization can considerably improve performance in some cases.

Motion Factor Optimization

AIR provides an attribute to automatically reduce the number of polygons used to represent a curved primitive based on the how blurred the primitive appears. This feature can be enabled on a per-primitive basis with

```
  GeometricApproximation "motionfactor" [1]
```

Motion factor is a scalar value: large numbers will produce an even coarser tessellation; smaller numbers produce a finer one. A motion factor of 0 disables the optimization.

Time Shifting Motion Blocks in RIB Archives

The following attributes scale and offset the times passed in a motion block:

```
  Attribute "render" "float shutterscale" [1]
  Attribute "render" "float shutteroffset" [0]
```

Time shifting can be useful when re-using archives in an animation sequence.

Adaptive Motion Blur

AIR 6 introduces a new adaptive motion blur option. To enable adaptive motion blur, set the maximum

© 2001-2014 SiTex Graphics, Inc.

number of motion samples with

```
  Option "limits" "integer motionsamples" [n]
```

The minimum number of motion samples is taken from the pixel samples setting. AIR selects the number of motion samples for each object based on how far the object blurs on screen. The number of samples is calculated based on the maximum allowable distance between motion samples (in pixels), which can be set with:

```
  Option "render" "float motionthreshold" [0.5]
```

15.2 Depth of Field

AIR can simulate the limited focal range of a physical camera to produce depth-of-field effects. Physical cameras have a limited range over which objects appear in focus; objects that are too close or too far away appear blurred. Depth of field is set by 3 parameters

```
  DepthOfField f-stop focallength focaldistance
```

The parameters are taken from photography:

f-stop: the f-stop corresponding to the size of the aperture admitting light into the camera. Smaller numbers produce more blur. Typical values for a 35mm camera range from 2.8 (very blurry) to 16 (focused).

focallength: focal length of the lens. This should be in the same units as focaldistance

focaldistance: distance from the camera at which objects appear sharpest.

As with motion blur, the quality of an image rendered with depth of field depends directly on how many pixel samples are used. Blurrier images require more samples and take longer to render.

15.3 Level of Detail

AIR supports the RIB level of detail capability. Unlike some other renderers, the quality of the blending between levels of detail does not depend on the `PixelSamples` setting.

Alternate Detail Camera

AIR allows an alternate camera to be used for determining the level of detail in a scene. The following option

```
  Option "render" "integer detailcamera" [1]
```

marks the current camera definition (`Projection`, `ScreenWindow`, `Format`, and world-to-current transformation) as the camera to use to calculate level of detail. This option is followed by the normal camera definition for the scene. An alternate camera is useful for rendering a shadow map that uses the same detail models as the beauty pass.

Limitations

Blending between levels is not supported in the ray tracer; only the representation with the greatest blend weight is retained for ray tracing. Similarly, the shadow hider only tests the model with the largest blend weight in a transition region.

Automatic Level of Detail

AIR can automatically reduce the density of polygon meshes using an automatic LOD capability described here.

16 Outlines for Toon Rendering and Illustration

AIR can automatically draw outlines for cartoon rendering and illustration. Outline generation takes place at the sub-pixel level, producing smooth lines without the need to render large intermediate images or post-process the rendered image.

Outlining is enabled with the following option:

```
Option "toon" "float maxinkwidth" [n]
```

where *n* gives the maximum line width in pixels. AIR uses this option to expand the internal storage for each bucket to prevent lines from being clipped at bucket boundaries.

The smoothness of the outlines depends on the number of pixel samples used. Increase the number of pixel samples for smoother lines.

AIR provides several methods of generating outlines, which can be divided into two broad categories - image-based methods and object-based methods.

Image-based Outlines

Region Boundaries

Edge lines are drawn between different regions of an image. By default each primitive is assigned a distinct region id. A region id can be explicitly assigned to a primitive with:

```
Attribute "toon" "integer id" [n]
```

The image background is assigned a region id of 0.

The special region id of -2 causes every polygon to receive a distinct id, which can be used to outline a rendering mesh for illustration purposes.

The region id may also be generated in a surface shader by setting a `__toonid` output variable of type float. This capability allows shader-generated patterns to be outlined.

Depth Differences

Another method of identifying different regions or objects is to use depth-based detection, which can be enabled with:

```
Attribute "toon" "float zthreshold" [depth]
```

Where depth gives the minimum depth between distinct surfaces. The depth threshold can be set by a surface shader with a `__toonzthreshold` output variable of type float.

Silhouette Edges

AIR provides two methods of detecting silhouette edges. By default AIR detects silhouettes using an image-based technique that looks for differences in the surface normal. The threshold for image-based silhouette detection can be tuned with the following attribute:

```
Attribute "toon" "float silhouette" [0.5]
```

The valid range is 0 to 1. To disable this method of silhouette detection set the silhouette threshold to 0 or 1. The silhouette threshold may be set by a surface shader with a `__toonsilhouette` output variable of type float.

Object-based Outlines

Silhouette Edges

AIR provides a second object-based method of silhouette detection that looks for boundaries between front- and back-facing polygons in the rendering mesh. This method (disabled by default) can be enabled with:

```
Attribute "toon" "integer frontback" [1]
```

If you use the front-back method of silhouette detection, you may wish to disable to the image-based method.

Note that this method may not work well with the automatic meshing method for NURBs; using explicit meshing may help.

Face Border Edges

Edges that are shared by only one face can be included in vector-based outlines by enabling the following attribute:

```
Attribute "toon" "integer nakededges" [1]
```

Angle-based Edges

AIR can detect edges based on the angle between adjacent faces. The minimum angle in degrees for edge-detection is set with

```
Attribute "toon" "integer faceangle" [angle]
```

When angle is 0, angle-based edge detection is disabled.

Line Color

Line color can be set on a per-primitive basis with

```
Attribute "toon" "color ink" [0 0 0]
```

Ink color can also be set by a surface shader with a `__ink` output variable of type color.

Line Width

Line width is set with

```
Attribute "toon" "float inkwidth" [1]
```

which gives a distance in pixels. Line width can be set in a surface shader by exporting a `__inkwidth` output variable of type float.

The following option can be used to globally scale the width of all outlines:

```
Option "toon" "float inkwidthmultiplier" [1]
```

Line width can be automatically reduced with distance from the camera by setting a distance at which to begin fading with

```
Option "toon" "float fadethreshold" [n]
```

Set the fade threshold to 0 to disable line width attenuation with distance. When the fade-with-distance option is enabled, a minimum ink width can be set with

```
Option "toon" "float mininkwidth" [0.0]
```

to prevent outlines from completely disappearing at large distances.

Saving Outlines

The outline information may be saved to an image file by using the special `__toonline` output variable in a `Display` call. The `__toonline` output variable has a single value representing the fraction of each pixel that is covered by a line.

Exporting Outlines as Vectors

AIR 10 introduces the ability to export outlines as vectors. Specify the exported file name with:

```
Option "toon" "string vectorexportfile" "filename.svg"
```

Only outlines detected using one the vector-based methods described above will be included. The following file formats are currently supported: SVG, DXF, AI.

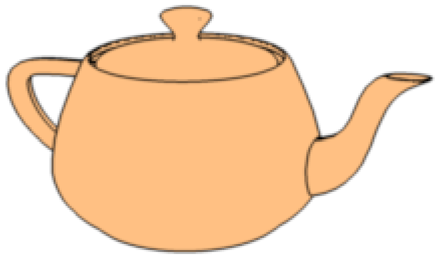
See Also

VToon surface shader

16.1 Illustration Shaders and Examples

Constant Shading

For simple illustration, you may wish to use the constant shader in conjunction with outlining.

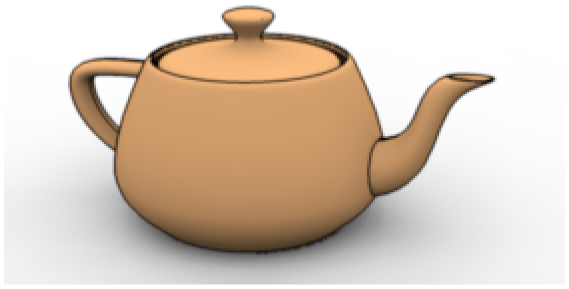


Cartoon Shading

For cartoon rendering, try the VToon shader included with AIR. Because toon-style shading often has sharpen transitions between dark and light regions, you may need to user a lower shading rate to produce smooth shading results.

Ambient Occlusion and Outlines for Illustration

For a softer look, try a matte surface shader with ambient occlusion and outlines:



More examples of toon rendering and illustration can be found in

`$AIRHOME/examples/toon`

17 Volume Rendering

AIR 4 introduces a new volume primitive type for efficient rendering of volumetric effects. Although AIR can still produce volumetric effects by ray marching in a volume shader, volume primitives are typically much more efficient.

Defining Volume Primitives

A volume primitive defines a volume as a basic shape within a bounding box. The shape can be a box, ellipsoid, cylinder, or cone.

Volume primitives can have arbitrary user data applied, either as a 3D grid of data values or as a point set with associated per-point data values.

Your RIB export plugin may automatically export fluid simulation or particle simulation data as a volume primitive. Users interested in creating their own volume primitives should see the section on the RIB `Volume` statement.

Shading Volume Primitives

Like other primitive types. the color and opacity for a volume primitive is computed by a surface shader - not a volume shader. The surface shader need only compute the shading and lighting at the current shading location, P. AIR takes care of storing and using the results of individual surface evaluations to compute the visibility and color for individual ray queries for a given volume. The following AIR shaders are designed to shade volume primitives:

particle
VCumulusCloud
VSmokeSurface

Shading Rate and Volume Quality

To accelerate rendering of volumetric effects, volume primitives use a dynamically created 3D shading cache, which decouples shader evaluation from visibility determination. The resolution of the shading cache, and hence the quality of the volume approximation - is based on the shading rate attribute. A larger shading rate will produce a faster, coarser image and consume less memory. Some common volume patterns do not require fine resolution to produce acceptable results. Many of the sample volume scenes included with AIR use a shading rate of 4, and that is recommended as a starting point for volume rendering.

Volume Visibility and Shadows

Volume primitives respect the normal attributes for visiblity to camera, reflection, and shadow rays. Volume primitives are not currently visible to indirect rays.

Volume primitives with complex transparency should have the shadow ray transmission type set to "shader":

```
Attribute "visibility" "string transmission" ["shader"]
```

By default all volume primitives will cast shadows upon themselves. Simple primitives that do not require self-shadowing can be accelerated by disabling self-shadowing with:

```
Attribute "visibility" "integer selfshadow" [0]
```

Volume primitives do not cast colored shadows by default. To enable colored shadows, set

```
Attribute "volume" "integer coloredopacity" [1]
```

Rendering with colored shadows is slower, but it does not use more memory that rendering with grey-scale shadows.

The look of volumetric primitives can often be improved by using separate opacity values for shadow rays and camera rays. AIR provides separate attributes to tint the opacity computed by the surface shader differently for shadow rays and camera rays:

```
Attribute "volume" "color shadowdensity" [1 1 1]
Attribute "volume" "color viewdensity" [1 1 1]
```

Motion Blur

Volume primitives support internal motion blur if additional motion data is provided in the primitive definition.

Transformation motion blur is currently applied to volume primitives as internal motion blur. Volume primitives boundaries will not blur.

Regular motion-blurred primitives may appear within and pass through volume primitives.

Tips

- The time required to render a volume primitive and the memory consumed is heavily influenced by the shading rate and the relative size of the primitive. For fast render times, use as high a shading rate as possible for the required image quality.

- When a volume primitive with a high shading rate is viewed face-on, - that is, from a direction that is nearly perpendicular to one of the faces of the bounding box - artifacts of the 3D shading cache may become visible. Such artifacts can be reduced by changing the point-of-view to a more oblique angle, lowering the shading rate, reducing the contrast of the shading results, or eliminating small pattern details.

17.1 Writing Shaders for Volume Primitives

Shading for a volume primitive is performed by its assigned surface shader. The surface shader need only calculate the color and opacity at P. AIR takes care of integrating shading information along query rays through the primitive. Surface shader variables are initialized so that `sqrt(area(P))` gives a good filter width estimate for anti-aliasing procedural patterns.

When applying the shader results to a ray through the volume medium, AIR treats the opacity result as the optical thickness of the material over the unit interval in object space. The actual attenuation applied for a distance D is

```
  1-exp(-Oi*D)
```

Output opacity values greater than 1 are permitted. A shader for a volume primitive should normally provide a parameter for scaling the output opacity as the proper value is dependent on the scale of the object. E.g., a sphere with radius 100 would need an opacity value .01 times smaller to match the appearance of a sphere of radius 1.

Unlike shader results for surface primitives, there is no general expectation that the output color will be proportional to the output opacity - with one exception: if the output opacity is 0 (the volume is transparent at this location), the output color should be set to 0.

Opacity Modulation for Different Ray Types

To accelerate rendering AIR stores surface shader results in a 3D cache, and only one set of values is stored at each shading location for all ray types. AIR provides two custom attributes for modifying the base opacity stored in the shading cache independently for shadow rays and camera (or reflection) rays:

```
Attribute "volume" "color shadowdensity" [1 1 1]
Attribute "volume" "color viewdensity" [1 1 1]
```

Shader Optimization

For volume primitives with complex opacity and self-shadowing, AIR often evaluates the surface shader twice at each shading location: once for a shadow ray, then a second time for a camera or reflection ray. AIR provides a method of optimizing the second evaluation by supplying the opacity result from the first evaluation in the Oi global opacity variable. If the shader is being executed for the first time at this location, Oi is set to -1. A surface shader can test Oi and avoid recomputing an expensive opacity function if it has already been evaluated:

```
   if (comp(Oi,0)<0) {
     /* compute expensive opacity function */
   }
```

Sample Shaders

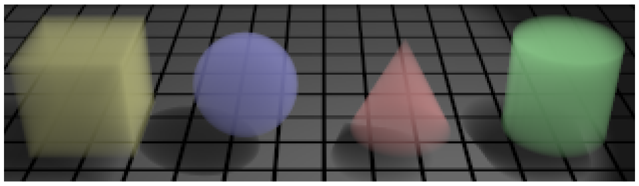
The particle, VCumulusCloud, and VSmokeSurface shaders included with AIR are designed to shade volume primitives.

17.2 RIB Volume Primitive

```
Volume "shape" [xmin xmax ymin ymax zmin zmax]
Volume "shape" [xmin xmax ymin ymax zmin zmax]
  [nx ny nz] parameter list
Volume "shape" [xmin xmax ymin ymax zmin zmax]
  [0 0 0] "P" [...] parameter list
```

A Volume primitive describes a region of space to be shaded as a 3D volume based on the assigned surface shader. The volumetric region is defined as a bounding box with one of the following shapes:

```
box ellipsoid cone cylinder
```



Cone and cylinder shapes are oriented around the z-axis in object space. The cone tip is at *zmin*.

Grid-based User Data

Volume primitives provide two ways of assigning varying user data: grid-based and particle-based. The first method is useful for rendering the results of voxel-based fluid simulations. Grid-based user data is assigned by defining a *nx* x *ny* x *nz* 3D grid. Each varying variable should have nx*ny*nz values. Typically one will want to assign varying opacity values. AIR provides an efficient way to specify a single channel opacity instead of the usual 3-channel color Os in the form of a `"varying float floatOs"` parameter.

```
Volume "box" [-2 2 -2 2 -2 2]
  [2 2 2] "varying float floatOs" [1 1 0 0  .5 .5  0 0]
```

Varying data is interpolated and supplied to the surface shader at each shading location. This allows a surface shader to modify the base output from a fluild simulation, for example, to add additional fine detail.

Particle-based User Data

Particle data can be rendered as a volume primitive by defining a null grid ([0 0 0]) and providing P position data with varying data associated with each point. The size of each particle can be set with a single `constantwidth` parameter or a varying `width` parameter. The particles are blended together like blobby primitives to obtain the base opacity value for each shading location, accounting for any varying `Os` or `floatOs` data supplied. Other user data is blended in the normal way.

Particle-based data is evaluated using a fast space-subdivision structure which makes it feasible to render large numbers of particles.

Motion Blur

Intravolume motion blur is support for grid-based user data by providing varying vector `dPdtime` data. A constant `__motionoffset` variable can be used to shift the motion vector relative to a given point.

The actual motion vector at point P extends from `P-motionoffset*dPdtime to P+(1-motionoffset)*dPdtime.` The default motion offset is 0.

Transformation motion blur is currently applied as intravolume blur. Boundaries of a volume primitive will not blur.

Procedural Primitives

Procedural primitives that generate volume primitives should be tagged with the following attribute:

```
   Attribute "visibility" "integer volume" [1]
```

18 Pipeline, Workflow, and Optimization

18.1 Conditional RIB

Syntax:

```
IfBegin "condition"
ElseIf "condition"
Else
IfEnd
```

Conditional RIB statements allow sections of RIB to be enabled or disabled based on an expression. The expression may including tests on user-defined options and attributes. For example:

```
Attribute "user" "string pass" "beauty"

IfBegin "$user:pass=='shadow'"
   Surface "constant"
ElseIf "$user:pass == 'beauty'"
   Surface "vmarble"
Else
   Surface "matte"
IfEnd
```

18.2 RIB Filters

AIR 6 and later provide a filter plugin capability for modifying the stream of RIB commands before they are processed by the renderer.

The current filter is defined using the following attribute:

```
Attribute "render" "string rifilter" "filtername arg1 arg2"
```

AIR will search the procedural search path for the filter program. Because the current filter is an attribute, a filter can easily be applied to only a subsection of a scene, say just the Massive agents. For example, to replace the surface shader assigned to a group of objects, one might use the following RIB snippit:

```
AttributeBegin
  Surface "occlusionpass"
  Attribute "render" "string rifilter" "cullrib Surface"
  ReadArchive "agents.rib"
AttributeEnd
```

No other scene elements will be affected by the filter.

To help with debugging filter plugins, AIR 8 includes a new `-echorif` command line option that echoes the output from a RIB filter to the standard output stream.

Sample Filters

The AIR distribution includes several sample filters along with their source code in `$AIRHOME/procedurals`.

echorib *list of RIB commands*

The echorib filter accepts a one or more RIB commands as arguments. Any matching commands found in the RIB stream are printed to the terminal on stderr and passed back to the renderer for normal processing.

Echorib is useful for seeing exactly what commands are included in a scene. AIR has a -rif command line switch for applying a filter to the whole rib stream. E.g., to see all attributes in a scene use:

```
air -rif "echorib Attribute" myscene.rib
```

cullrib *list of RIB commands*

The cullrib filter removes the specifed commands from the command stream before they are processed. For example, to disable all displacements in a scene:

```
air -rif "cullrib Displacement" myscene.rib
```

surftoprim *list of parameters*

The surftoprim filter takes one or more surface shader parameters as arguments. When the filter encounters a Surface call, it stores any parameters that match the argument list and adds those parameters and values to subsequent primitives as primitive variable data. The Surface call is removed from the RIB stream.

Sample Usage for Massive scenes:

The surftoprim filter allows the agent surface shader to be changed in another program such as Maya or Houdini while retaining the per-agent parameter values generated by Massive. For example, say agents have the VPlastic surface shader assigned in Massive and with a different color map for each agent. The RIB stream produced by the Massive procedural might look something like:

```
Surface "VPlastic" "string ColorMapName" "shirt1.tif"
PointsPolygons ....
Surface "VPlastic" "string ColorMapName" "shirt2.tif"
PointsPolygons ....
```

Applying the following filter:

```
Attribute "render" "rifilter" "surftoprim ColorMapName"
```

produces:

```
PointsPolygons .... "constant string ColorMapName" "shirt1.tif"
PointsPolygons .... "constant string ColorMapName" "shirt2.tif"
```

One can then assign the VPlastic shader to a RIB box referencing the filtered agents and manipulate other shader parameters while retaining the color map values generated by Massive. Naturally one can use a different surface shader as well, and any matching primitive variable data will be used. E.g.,

```
AttributeBegin
  Surface "VClay"
  Attribute "render" "rifilter" "surftoprim ColorMapName"
  ReadArchive "agents.rib"
AttributeEnd
```

Writing RI Filters

An RI filter is structured similarly to a procedural runprogram. The sample filters are written in C, but any programming language may be used.

When first invoked, the filter should emit a single line containing a list of RIB commands that the filter will process. Then the filter should enter a loop that waits on stdin. Each RIB command will be sent as a single line of potentially very long ASCII RIB. To pass RIB commands back to the renderer, send them to stdout. When the filter is done processing the current command, it should emit a byte value 255. The filter may also need to flush the stdout stream to force all data down the pipe to the renderer.

Source code for the sample filters included with AIR can be found in `$AIRHOME/procedurals`.

18.3 Geometry Export

Beginning with release 3.0 AIR allows the polygonal meshes used for rendering to be exported to a file, allowing the displaced geometry and other complex shapes such as a blobby system to be saved for re-use with AIR or for import into a modeling package. AIR can typically re-render a baked displacement mesh many times faster than the original object with true displacement.

To export the rendering mesh for an object, provide a filename with the following attribute:

```
  Attribute "render" "string meshfile" filename
```

The file format is determined by the file name extension:

Extension	Format
`.rib`	RenderMan RIB
`.dra`	RIB using DelayedReadArchive
`.obj`	Wavefront OBJ
`.ply`	PLY polygon format
`.stl`	STL format

RIB export normally uses the binary RIB format. If the mesh file name contains the string `ascii`, the RIB file will be in ASCII format instead of binary.

Additional Attributes for Geometry Export

```
Attribute "render" "integer makePref" [0]
```

Set this attribute to 1 to generate `__Pref` point data for the exported mesh containing the undisplaced vertex positions.

```
Attribute "render" "integer makeNref" [0]
```

Set this attribute to 1 to generate `__Nref` normal data for the exported mesh containing the vertex normals prior to displacement.

```
Attribute "render" "string meshspace" ["world"]
```

Specifes the coordinate space for exported point and normal data. The only support spaces are "world" and "object".

18.4 General Optimization Tips

Tips on optimizing rendering to reduce speed and/or memory used:

- Ray trace only when necessary to save time and memory.

Objects that do not need to be ray traced should be made invisible to all ray types - shadow rays, reflection rays, and indirect rays. AIR can discard such objects as soon as they have been rendered.

The AIR Show framebuffer can be used to see if any objects are retained for ray tracing in a scene. The second box from the right in the status bar displays one or more letters if any objects are visible to shadow rays (S), reflection rays (R), or indirect rays (I). If you are not using ray tracing in a scene, make sure that box is empty.

- For motion-blurred curved or displaced primitives enable the motion factor optimization.

- For complex scenes, using Procedural primitives to generate or load primitives on-demand can dramatically reduce peak memory use.

- In some cases of extreme motion blur the memory used by the shading cache can become significant. You can lower the cache size using an option.

- For scenes with many large shadow maps you may need to increase the texture cache size to avoid thrashing.

- Blobbies:
  - Using a higher tessellation factor can significantly reduce peak memory and render time
  - For true displacement enable the meshdisplacement (`Attribute "render" "meshdisplacement" [1]`)

19 TweakAir for Interactive Rendering

TweakAIR is an interactive rendering tool based on the AIR renderer. TweakAIR is designed to allow interactive tweaking of high-quality images with the full range of advanced rendering effects supported by AIR.

TweakAIR accepts regular RIB files as input. At the beginning of an interactive rendering session TweakAIR creates a "deep" framebuffer for the current view that stores information about the visible surfaces at each pixel. Once the deep framebuffer has been generated, shading and lighting settings for the scene can be changed interactively, and the rendered image will automatically update to reflect the changes.

TweakAIR employs various caching mechanisms and re-evaluation heuristics to accelerate the re-rendering of scene elements.

Users will typically interact with TweakAIR via their plugin or modeling software. TweakAIR is currently supported by AIR Space, the RhinoAIR plugin for Rhino, and the MayaMan plugin for Maya.

Features

TweakAIR provides the following capabilities:

- Select primitives by name or shading group
- Change any parameter for any assigned surface, displacement, or light shader
- Move lights
- Add new lights
- Assign new surface and displacement shaders

Limitiations

The following features are not supported by TweakAIR:

- Motion blur
- Depth of field
- Level of detail
- PixelFilter
- PixelSamples (use the corresponding ipr option instead)
- Shading rate

Developer's Kit

Documentation for the TweakAIR API can be found in this section of this user manual.

19.1 TweakAIR Options

The following options can be used to tune the performance of TweakAIR:

Deep Framebuffer Depth

```
Option "ipr" "integer mindepth" [1]
Option "ipr" "integer maxdepth" [4]
```

These options set bounds on the number of surface layers TweakAIR stores at each pixel.

Ray Trace Sampling

```
Option "trace" "integer maxshadowsamples" [4]
Option "trace" "integer maxtracesamples" [4]
```

These options set the maximum number of shadow and reflection rays to trace on any given ray tracing call. Use these options to optimize interactive rendering while allowing higher samples for final rendering with AIR.

Reflection Updating

```
Option "ipr" "integer retrace" [1]
```

When this option is enabled, TweakAIR will reshade an object whenever a reflected object's shading changes. Automatic reflection updates will adversely affect the overall IPR refresh rate.

Geometric Anti-aliasing

```
Option "ipr" "integer[2] pixelsamples" [1 1]
```

This option sets the number of pixel samples to use during IPR rendering. Currently the only supported values are 1 1 and 4 4. Using more pixel samples can have a significant impact on re-rendering time to the additional shading samples that are usually required.

19.2 TweakAIR Attributes

Use the following attributes to tune the performance of TweakAIR:

Shading

```
Attribute "ipr" "string shading" "shade"
```

This attribute allows shading to optimized during an IPR session without affecting shading during a final render. Supported types are

`matte`	shade object using a simple, fast matte shader
`relight`	optimize shading for re-lighting
`shade`	shade object using its assigned surface shader

The relight mode provides fast re-rendering for even complex surface shaders with the following properties:

The surface shader must use only builtin illumination statements: diffuse(), specular(), phong(), or blinn(). I.e., no custom illuminance loops.

The surface shader must provide __diffuse and __specular output variables.

TweakAIR is able to decouple lighting computations from surface shader evaluation for surface shaders that meet the above two conditions. After the inital surface evaluation. AIR can compute the effects of lighting changes on an object without having to re-execute the surface shader, which means that the refresh rate for relighting is independent of the complexity of the surface shader.

Although this mode is called "relight", it is still possible to tweak surface shader parameters - though modifying a parameter of the surface shader will require re-executing the shader.

Displacement

```
Attribute "ipr" "integer displacement" [1]
```

When set to 0 this attribute blocks execution of an object's displacement shader during an IPR session without affecting displacement during a final rendering.

Global Variable Storage

```
Attribute "ipr" "string globals" [""]
```

This attribute defines a list of global variables to store at a surface in addition to the position, normal, and texture coordinate variables that are always stored. Possible items: `dPdu`, `dPdv`.

19.3 TweakAIR API

Integration

Adding support for TweakAIR to an application that already supports AIR is straightforward. TweakAIR provides two mechanisms for managing an interactive session:

1. A C-based API for launching TweakAIR as an external process and sending update commands via a TCP/IP socket connection.
2. Simple text commands that can be sent to TweakAIR via the standard input stream - either from a command shell or via a connecting pipe.

TweakAIR C API

The C-based API header and source are included in the AIR distribution in `$AIRHOME/examples/tweakair`.

To begin an interactive rendering session, create a RIB file as usual and start a tweakair session with the `IrBegin` call. Then call the appopriate interface routine to update the rendering as the user changes the shading and lighting for the scene. The following interactive functionality is provided:

Primitive selection: Use `IrSelect` call to select one or more primitives to modify.

Shader parameter tweaking: Use `IrTweakSurface`, `IrTweakDisplacement`, and `IrTweakLight` to modify shader parameters for the selected primitives or light.

New shader assignment: Use `IrNewSurface` and `IrNewDisplacement` to assign new shaders to selected primitives.

New lights: Use `IrNewLight` to add new lights to the scene.

Moving a light: Use `IrLightTransform` to update the transformation applied to a light source.

Moving the camera: Use `IrTweakCamera` to change camera position or field of view.

Attribute tweaking: Use `IrColor` and `IrOpacity` to change the corresponding color and opacity attributes. Use `IrTweakAttribute` to change general attributes (be sure to consult the list of tweakable attributes).

The C API uses a TCP/IP socket connection to communicate with the TweakAIR process.

TweakAIR Text API

TweakAIR also accepts update commands from a command line or piped input stream.

For simple changes, you can start TweakAIR from a command shell, and then type update commands. E.g.,

```
tweakair myscene.rib
IrColor 1 1 0  # change all objects to yellow
IrNewSurface "VMetal" # change all surface shaders to metal
IrWriteImage "gold.tif"
IrEnd
```

You may need to press the Enter key twice after each command to send it to the rendering process.

A more flexible means of sending updates is to pipe a text file with commands to the standard input stream:

```
tweakair myscene.rib <myupdates.txt
```

An example of this method can be found in `$AIRHOME/examples/tweakair`.

Error Handling

Most calls return a non-zero error code if the call fails. Pretty much the only reason for failure is that the communication link with TweakAIR has been broken - in most cases because the TweakAIR instance has been shutdown by the user. In the event of an error, an application should call `IrEnd` to free the IPR structure and terminate the current IPR session.

19.3.1 IrBegin

```
IPR *IrBegin(char *cmdline);
```

This call starts an interactive rendering session. TweakAIR is started with the provided scene file. The call returns a pointer to an IR session that should be used as the first parameter for all IR functions. The IR library supports multiple simultaneous interactive sessions.

19.3.2 IrColor

`C:  int IrColor(IPR *ipr, float r, float g, float b);`

Text: `IrColor` *r g b*

The IrColor command changes the color attribute of the selected primitives.

19.3.3 IrDeleteLight

`C: int IrDeleteLight(IPR *ipr, char *handle);`

Text: `IrDelete` *handle*

The `IrDeleteLight` command deletes the light corresponding to the specified handle.

19.3.4 IrEnd

`C: void IrEnd(IPR *ipr);`

Text: `IrEnd`

This call terminates the interactive rendering session.

19.3.5 IrIlluminate

`C: int IrIlluminate(IPR *ipr, char *handle, int on);`

Text: `IrIlluminate` *handle* `0|1`

Use this command to turn the specified light on (1) or off (0) for selected objects.

19.3.6 IrLightTransform

`C: int IrLightTransform(IPR *ipr, char *handle, float *transform);`

Text: `IrLightTransform` *handle* [*matrix*]

This function sets the shader-to-world transformation matrix for the indicated light.

In the C call the `transform` parameter should point to an array of 16 floats.

19.3.7 IrNewDisplacement

```
C: int IrNewDisplacementV(IPR *ipr, char *shadername, int ntk, char **tk,
void **v);
```

Text: `IrNewDisplacement` *shadername parameter list*

This function assigns a new displacement shader to the selected primitives. Tokens in the parameter list must have full inline type definitions.

19.3.8 IrNewEnvironment

```
C: int IrNewEnvironmentV(IPR *ipr, char *shadername, int ntk, char **tk,
void **v);
```

`Text:  IrNewEnvironment` *`shadername parameter list`*

This function assigns a new environment shader to the scene. Tokens in the parameter list must have full inline type definitions.

19.3.9 IrNewImager

```
C: int IrNewImagerV(IPR *ipr, char *shadername, int ntk, char **tk, void
**v);
```

`Text: IrNewImager` *`shadername parameter list`*

This function replaces the current imager shader with a new one. Tokens in the parameter list must have full inline type definitions.

19.3.10 IrNewLight

```
C: int IrNewLightV(IPR *ipr, char *shadername, char *handle, float
*transform, int ntk, char **tk, void **v);
```

`Text: IrNewLight` *`shadername handle`* `[`*`matrix`*`]` *`parameter list`*

This call adds a new light to the scene. Tokens in the parameter list must have full inline type definitions. The `transform` parameter is a pointer to a matrix with the shader-to-world transformation.

Special Parameters:

`"float locallight" [1]`

By default a new light illuminates all primitives in the scene. If this parameter is present, the new light illuminates only the currently selected primitives.

19.3.11 IrNewSurface

```
C: int IrNewSurfaceV(IPR *ipr, char *shadername, int ntk, char **tk, void
**v);
```

`Text: IrNewSurface` *`shadername parameter list`*

This function assigns a new surface shader to the selected primitives. Tokens in the parameter list must have full inline type definitions.

19.3.12 IrOpacity

```
C: int IrOpacity(IPR *ipr, float r, float g, float b);
```

```
Text: IrOpacity r g b
```

This command changes the opacity attribute of the selected primitives.

19.3.13 IrRefreshWindow

```
C: int IrRefreshWindow(IPR *ipr, float left, float right, float top, float
bottom);
```

```
Text: IrRefreshWindow left right top bottom
```

Use this command to restrict refreshing of the IPR window to a subrectangle. The parameters have the same interpretation as the RI `CropWindow` call.

Examples:

Restricting refresh to the upper, right-hand quarter of the image:

```
  IrRefreshWindow(0.5, 1.0, 0.0, 0.5);
```

Restoring refresh for the entire image:

```
  IrRefreshWindow(0.0, 1.0, 0.0, 1.0);
```

19.3.14 IrSelect

```
C: int IrSelect(IPR *ipr, char *type, char *handle);
```

```
Text: IrSelect type handle
```

Use this call to select primitives for tweaking. Primitives can be selected by name or by shading group. A primitive's name should be set in the scene file with:

```
  Attribute "identifier" "string name" ["teapot"]
```

This object could be selected with:

```
  IrSelect(ipr,"name","teapot");
```

A shading group is meant to correspond to the concept of a material in a modeling program. The shading group for a primitive is set with:

```
  Attribute "identifier" "string shadinggroup" ["redbrick"]
```

All primitives in a given shading group can be selected with:

```
  IrSelect(ipr, "shadinggroup", "redbrick");
```

By default the new selection set replaces the old list of selected primitives. If the handle is preceded by a "+", new primitives will instead be added to the selection set. If the handle is preceded by a "-", matching primitives are removed from the selection set.

The special handle "*" can be used to select all primitives:

```
  IrSelect(ipr, "name", "*");
```

19.3.15 IrTweakAttribute

```
C: int IrTweakAttributeV(IPR *ipr, char *name, int ntk, char **tk, void
**v);
```

Text: `IrTweakAttribute` *name parameter list*

Use this function to modify attributes for the selected primitives. Tokens in the parameter list must have full inline type definitions.

The following attributes are tweakable:

```
Attribute "ipr" "string shading" type
Attribute "ipr" "integer displacement" [n]
Attribute "identifier" "string shadinggroup" name
```

19.3.16 IrTweakCamera

```
C: int IrTweakCamera(IPR *ipr, char *name, float *transform, float fov);
```

Text: `IrTweakCamera` *name* [*matrix*] *fov*

Use this command to update the camera position used for the interactive viewport. The `transform` parameter is a matrix giving the new world-to-camera space transform. For perspective views the `fov` parameter gives the new field of view in degrees.

The `name` parameter is currently ignored.

TweakAIR traces reflection rays to update the deep framebuffer when the camera changes. Objects must be visible to reflection rays in order to appear in the viewport after a camera move.

19.3.17 IrTweakCoordinateSystem

```
C: int IrTweakCoordinateSystem(IPR *ipr, char *name, float *transform);
```

Text: `IrTweakCoordinateSystem` *name* [*matrix*]

This function sets the space-to-world transformation matrix for the named coordinate system.

The `transform` parameter should point to an array of 16 floats.

19.3.18 IrTweakDisplacement

```
C: int IrTweakDisplacementV(IPR *ipr, int ntk, char **tk, void **v);
```

Text: `IrTweakDisplacement` *parameter list*

Use this function to modify displacement shader parameters for the selected primitives. Tokens in the parameter list must have full inline type definitions. The selected primitives need not have the same displacement shader assigned. This function simply looks for matching parameters in whatever displacement shader (if any) is currently assigned to a primitive.

19.3.19 IrTweakEnvironment

`C: int IrTweakEnvironmentV(IPR *ipr, int ntk, char **tk, void **v);`

`Text: IrTweakEnvironment` *`parameter list`*

Use this function to modify environment shader parameters. Tokens in the parameter list must have full inline type definitions.

19.3.20 IrTweakImager

`C: int IrTweakImagerV(IPR *ipr, int ntk, char **tk, void **v);`

`Text: IrTweakImager` *`parameter list`*

Use this function to modify parameters for the current imager shader. Tokens in the parameter list must have full inline type definitions.

19.3.21 IrTweakLight

`C: int IrTweakLightV(IPR *ipr, char *handle, int ntk, char **tk, void **v);`

`Text: IrTweakLight` *`handle parameter list`*

Use this function to modify the parameters of the light shader attached to a light. Tokens in the parameter list must have full inline type definitions.

Special Parameters:

```
"float refreshshadows" [1]
```

Including this parameter flushes the shadow cache and forces the light to be re-evaluated at all surfaces illuminated by the light.

19.3.22 IrTweakOption

`C: int IrTweakOptionV(IPR *ipr, char *name, int ntk, char **tk, void **v);`

`Text: IrTweakOption` *`name parameter list`*

Use this function to modify options for the current scene. Tokens in the parameter list must have full inline type definitions.

The following options are tweakable:

```
Option "indirect" "color background" [r g b]

Option "ipr" "integer retrace" [n]
Option "ipr" "integer[2] pixelsamples" [nx ny]

Option "reflection" "color background" [r g b]
Option "reflection" "float envblur" n
Option "reflection" "float envstrength" n
Option "reflection" "string envname" name
Option "trace" "integer maxtracesamples" [n]
Option "trace" "integer maxshadowsamples" [n]
```

19.3.23 IrTweakSurface

```
C: int IrTweakSurfaceV(IPR *ipr, int ntk, char **tk, void **v);
```

```
Text: IrTweakSurface parameter list
```

Use this function to modify surface shader parameters for the selected primitives. Tokens in the parameter list must have full inline type definitions. The selected primitives need not have the same surface shader assigned; this function looks for matching parameters in whatever surface shader is currently assigned to a primitive.

19.3.24 IrWriteImage

```
C: int IrWriteImage(IPR *ipr, char *filename);
```

```
Text: IrWriteImage filename
```

The command writes the image buffer to an image file. The file type is determined by the file name extension. TweakAIR waits for any pending updates to complete before writing the image.

20 BakeAir for Baking Textures and Meshes

BakeAir is a special-purpose 3D renderer for rendering ("baking") shading and lighting results to texture maps or meshes.

Baking to Textures

In texture baking mode, BakeAir produces one or more texture maps containing shading and lighting information for each primitive. BakeAir is based on Air and fully supports all applicable Air features, including indirect illumination, ambient occlusion, caustics, area lights, true displacement, and programmable shading.

Rendered data can include global color, opacity, position, and normal information, as well as indirect illumination and any user-defined output variable from a surface, displacement, or atmosphere shader. BakeAir allows considerable freedom in the mapping from primitives to texture maps:

- A single primitive can output different channels to different maps
- Multiple primitives can bake into a shared texture.
- The coordinates used for indexing textures can be standard (s,t) or (u,v) coordinates, or any user-defined 2D texture coordinates.

BakeAir 11 can also bake to PTEX per-face texture maps.

Baking to Meshes

Air 11 introduces a new mode for BakeAir that allows shading computations to be baked to mesh vertices and exported in various file formats.

See Also

PTEX: Per-Face Texture Mapping

20.1 Applications for Baked Textures

This section lists a few of the applications for BakeAIR organized into three broad categories: accelerating software rendering, baking maps for hardware rendering, and geometric manipulation. Examples refer to the `bakeexamples` directory of the BakeAIR distribution.

Accelerating Software Rendering

BakeAIR can be used to accelerate software re-rendering with a renderer such as AIR by baking information that does not change from frame-to-frame or shot-to-shot. Baked data can include

- diffuse illumination
- diffuse color pattern
- indirect lighting
- occlusion
- shadows

Examples: The `occlusion`, `output`, and `gourds` directories contain examples of using baked data to accelerate software re-rendering.

Baking Maps for Hardware Rendering

Most recent graphics hardware supports real-time rendering with texture maps. BakeAIR can be used to prepare maps for a real-time 3D engine for use in games or for visualization for product design, architecture, and production-planning. BakeAIR can produce maps containing final shading results, color patterns, bump maps, normal maps, light maps or maps to modulate any other shading parameter supported in a 3D engine. Per-vertex data can be obtained by rendering low-res maps and extracting vertex data based on the texture coordinates used for baking.

Examples: The `sphere` and `head` directories contain examples of baking information for use with VRML files.

Geometric Manipulation

In addition to color and lighting information BakeAIR can record geometric information like surface positions, normals, and tangents, opening up another range of applications. For example, position, normal, and tangent maps can be used to place detail objects on surfaces. Think of adding fur or horns to a monster, or placing grass or trees on rugged terrain. Because BakeAIR supports true displacement, objects can be positioned correctly on displaced geometry. As another example, an object whose texture coordinates map into the unit square can be stored in a texture (position) map, and reconstructed using displacement with a renderer such as AIR. Two objects stored in texture maps can be morphed by simply interpolating the texture lookup and using displacement.

Examples: The fuzz directory contains simple examples of positioning curves on surfaces used baked geometric information.

20.2 Baking Textures with BakeAir

This section describes the technical details of configuring a scene for baking textures with BakeAir. Your plug-in or application may provide a simpler interface for creating and utilizing baked textures.

To prepare a scene for baking:

- Define one or more maps to bake (file name, data values, map size, and image quality)
- Choose a point of view in light of its effect on tessellation

- Select the texture coordinates used for indexing the bake maps
- Assign one or more bake maps to each primitive

Displays as Texture Maps

The `Display` command defines a new texture map to be rendered with the current scene:

```
Display "name" "driver" "data"
   "float[2] exposure" [gain gamma]
   "float[4] quantize" [zero one min max]
   "string coordinatesytem" [name]
```

The new display inherits the current `Format`, `PixelSamples`, `PixelFilter`, `Exposure`, and `Quantize` settings. Note that pixel filter, pixel sampling, and shading rate settings all behave as one would expect when baking. For example:

```
PixelFilter "gaussian" 2 2
PixelSamples 2 2
ShadingRate 0.5
```

results in 4 samples per texel (to anti-alias facet edges in texture space) and 2 shading samples per texel (to anti-alias shading) processed with a gaussian filter with a width of 2 in x and y in texture space.

Any data value supported by Air can be baked to a texture map, including the standard rgb and rgba outputs, global variables, and any output variable from a surface or displacement shader.

When baking the global position `P` and and shading normal `N`, BakeAir recognizes an additional `"string coordinatesystem"` display parameter that specifies the coordinate system in which to save the baked data. The default is `"current"` space; `"world"` and `"object"` spaces are also supported.

Point of View and Tessellation

The camera defined for a conventional rendering of a scene is used by the baker to control adaptive tessellation of curved surfaces, just as in a conventional rendering.

Since the baker produces texture maps, not images of primitives, you can often get by with a much coarser tessellation when baking. A coarser tessellation will use less memory and render more quickly. The standard way to decrease tessellation density is by increasing the flatness attribute setting from the default value of 0.5:

```
GeometricApproximation "flatness" [1]
```

Texture Coordinates for Baking

The texture coordinates used for baking are specified with a custom attribute:

```
Attribute "bake" "string coordinates" ["st"]
```

Possible values are `"st"`, `"uv"`, or any user-defined varying, vertex, or facevarying variable of type `float[2]`.

Alternatively, texture coordinates may be supplied as two arrays of floats:

```
Attribute "bake" "string xcoordinate" ["u0"]
Attribute "bake" "string ycoordinate" ["v0"]
```

The texture coordinates used for baking should map primitives into the unit square in texture space. More than one primitive may bake into the same map, but no two points (on any surface) should share the same texture coordinates in a single map. In other words, a surface should not overlap itself or other surfaces in texture space.

Assigning Maps to Primitives

There are two methods for assigning bake maps to a primitive. The first method is to provide extra parameters in the primitive's variable list of the form:

```
  "constant string bakemap_data" "mapname"
```

where *data* and *mapname* match one of the `Display` calls previously defined.

The second method is to provide a list of bake maps in the following attribute:

```
  Attribute "bake" "string bakemaps" "list,of,maps"
```

where each map name corresponds to one of the `Display` calls in the scene.

Baking

Once the scene has been configured for baking, start BakeAir with the scene name:

```
  bakeair myscene.rib
```

BakeAir can use multiple processors to bake scenes with multiple maps by assigning a bake map to each processor.

Baked Texture Padding

In some cases a baked texture map may contain unrendered pixels. Unrendered pixels that border rendered pixels may need to be filled in to avoid rendering artifacts when using the maps. BakeAIR can automatically fill-in the border around rendered regions in the texture map by replicating pixels. The width of this border can be set with

```
  Option "bake" "float border" [1]
```

Setting the border width to 0 disables padding of the rendered texture.

In some cases it may be useful to have the border filling wrap around in texture space, which can be enabled with:

```
    Option "bake" "integer borderwrap" [1]
```

This option may be useful when baking Rhino polymeshes

See Also

Lighting -> Baking Shadows and Illumination

20.3 Baking PTEX Textures with BakeAir

BakeAir 11 and later can bake data to PTEX per-face texture maps for polygonal and subdivision meshes.

For PTEX textures, the texture size for an individual face is computed automatically based on the following option:

Option "bake" "float texelsize" [n]

which gives the size of a single texel in world-space coordinates. A smaller texelsize will produce a higher-resolution (and therefore more detailed) map for each face.

Baking to PTEX also requires the following attribute for the baked objects:

Attribute "render" "integer ptex" [1]

The PTEX display driver accepts two additional parameters in the Display parameter list:

`"float half" [1]`

enables saving data as 16-bit floating point numbers ('half' precision). The display output quantization should also be set for float precision when saving half data.

`"float makemipmaps" [1]`

enables mip-map generation for the created PTEX file. Each face map will be stored at multiple resolutions for faster, higher-quality rendering.

20.4 Writing Bake-Aware Shaders

With a little effort many shaders can be organized to work with the baker and baked maps. Consider the following shading language snippit:

```
surface bakingsample(
    string bakemap___diffuse = "";
    output varying color __diffuse = 0;

 /* other parameters... */
)
{

    if (bakemap___diffuse!="") && (isbaking()==0) {

      /* use baked values, assuming baking coordinates are (s,t) */

      __diffuse = color texture(bakemap___diffuse,s,t);

    } else {

      /* calculate value for use by the shader and possible baking */

      normal Nf;
      Nf = faceforward(normalize(N), I);
      __diffuse = diffuse(Nf));

   }

   /* rest of shader using __diffuse */
}
```

If a bake map is provided and the scene is being rendered conventionally (`isbaking()==0`), the shader will use the values stored in the map. Otherwise, the diffuse value is calculated normally and saved in an output variable, making it available for baking.

The `output` directory in the baker examples contains a scene illustrating use of a baker-aware shader.

20.5 Baking Meshes with BakeAir

BakeAir 11 introduces a new mode for baking shader output values to mesh vertices.

Enable mesh baking with the following option:

```
Option "bake" "bakemesh" [1]
```

or use the -bakemesh command-line switch:

```
bakeair -bakemesh myscene.rib
```

A baked mesh is exported to the file specified for geometry export:

```
Attribute "render" "string meshfile" "filename"
```

When baking meshes, you will typically want to export RIB files.

The values to be saved at each vertex are specified as a comma-separated list:

```
Attribute "render" "string meshoutput" "list,of,outputs"
```

Each vertex value is formatted as:

```
destination=source
```

where *destination* is the variable declaration that will appear in the output file, and *source* is a declaration defining the data source. Supported sources are any output variable from a surface or displacement shader and the global Ci color variable. The default output list is:

```
Attribute "render" "string meshoutput" "varying color Cs=varying color
Ci"
```

For meshes exported in PLY format, the Cs output variable is emitted as per-vertex color values (with red, green, & blue channel labels).

An vertex variable can also reference a component of a shader output parameter, e.g.,

```
Attribute "render" "meshoutput" "float prim_occ=Ci[0]"
```

By default a mesh will always include normal (N) and texture coordinate (st) data. You can exclude one or more primitive variables from the exported mesh with:

```
Attribute "render" "string meshexclude" "list,of,ids"
```

Tessellation

When baking a mesh, you may wish to use the new edge length attribute introduced in BakeAir 11 to help control the density of the exported mesh:

```
GeometricApproximation "edgelength" [n]
```

gives the max edge length in the tessellated mesh (in object space).

Shader Support for Baked Meshes

The following Air shaders can take advantage of baked mesh data:

- The occlusionpass, massive_occlusionpass, envlight, & massive_envlight shaders use occlusion values stored on a mesh in a `"float prim_occ"` variable
- The indirect light shader and IndirectPass surface shader to use baked mesh `"color prim_indirect"` data for indirect light if available
- The VTranslucent and VTranslucentMarble surface shaders to use a diffuse result baked to `"color prim_diffuse"` if available

Examples

Sample scene files can be found in:

```
$AIRHOME/examples/bakeexamples/bakemesh
```

21 Massive

Massive is the premier 3D animation system for generating crowd-related visual effects for film and television. This section describes tips and tools for rendering Massive scenes with Air.

More information about Massive can be found on the Massive Software web site:

```
www.massivesoftware.com
```

Tips for using Massive with Air

- Rendering with Air from Massive
- Command Line Rendering of Massive Scenes
- Display Customization and Multipass Rendering
- massrib

- Shading and Lighting in Air Space
- Ambient Occlusion in Massive
- Mattes, Layers, and Groups
- Eliminating Shadow Artifacts on Terrain

- Automatic Level of Detail
- Massive Agent Light Tool (malt)
- Optimization Tips for Massive Scenes
- Rendering a Depth Pass

- Adding a background image
- Rendering Massive Agents with Maya & MayaMan

21.1 Rendering with AIR from Massive

Rendering a Massive scene with AIR requires several steps:

- Create one or more render passes.

- Create one or more "renders". Each render is associated with a render pass.
- Assign shaders to agents, terrain, and lights for each pass.
- Export files used for rendering with the Sim dialog.
- Render using the resulting exported files.

Creating a Render Pass

- In the Options menu select Render to display the Render dialog.
- Choose the **renderpasses** tab and click **add** to create a new pass.
- Change the renderpass name to `beautyAir` by typing in the top text box.
- Change the renderer for the pass to `Air` using the selection list next to the render caption.

Creating a Render

The next step is to create a "render" with which to associate the render pass.

- Select the **render** tab and click **add** to create a new render.
- With the new render selected, click on the parameters page.
- Change the associated render pass for the new render to `beautyAir`.
- Use the **output ribs** entry to enter the path and file name specification for the exported rib file that will be used for rendering.

Important: the export directory must exist for the scene export to succeed. If the directory does not exist, you'll need to create it outside of Massive.

The remainder of the parameters dialog allows you to customize other aspects of the rendering, such as the image size.

Assigning Shaders

Massive allows custom shaders to be assigned to agents, terrain, and lights on a per-pass basis. If no custom shader is assigned, Massive will translate the basic shading and lighting information to a standard AIR shader.

Saving Simulation Data

In order to render a Massive scene with AIR, the simulation data must first be exported as a set of files.

- From the Run menu select Sim to display the Sim dialog.
- In the list of Outputs, select the sims button and choose an appropriate output path for the sims export. Note that the export directory must exist; Massive will not create it for you.
- Select the ribs button and provide an output path and file specification for the exported rib files.
- Next to the ribs export path, change the DynamicLoad option to RunProgram for Air.
- If the agents have cloth, select the cloth button and provide an export path for cloth files.
- In the list of renders at the bottom of the dialog, select the Air render.
- Choose a start and end frame at the top of the dialog.
- Click **go** to export the files required for rendering.

Rendering

The scene is now ready for rendering using the rib files specified in the output ribs section of the render dialog. Massive creates a small batch file in the output directory for rendering the frames. Alternatively, the frames can be sent to a render farm using appropriate software.

To render agents the ribs generated by Massive use a small procedural primitive (a program) to generate agents on-demand. That procedural and the agent data must be available to all machines used for rendering. If changes are made to an agent inside Massive, the agent must be re-saved for the changes to show up in a rendering. Some older sample agents included with Massive must be re-saved to render properly even if no changes have been made to the agents.

AIR 8 allows a frame sequence to be rendered with simple frame number substitution. For example,

```
  air -frames 8 10 scene#f.rib
```

would render `scene8.rib`, `scene9.rib`, and `scene10.rib`.   The frame number pattern can also be zero-padded:

```
  air -frames 8 10 scene#3f.rib
```

would render `scene008.rib`, `scene009.rib`, and `scene010.rib`.

21.2 Command Line Rendering of Massive Scenes

The scene files generated by Massive can be rendered by invoking AIR from a terminal window or command shell. An individual frame can be rendered by providing the name of the main rib file as an argument to air:

```
   air rib/frame0001.rib
```

AIR accepts multiple rib files on a single command line. Multiple files are processed in order from left to right. Additional small files with RIB commands can be used to customize a rendering without modifying the main scene file. For example, one could use the small file `quant16.rib` to render an output image with 16-bit values:

```
   air quant16.rib scene.rib
```

Command Line Switches

AIR provides many command line options for customizing the renderer's behavior without modifying the main rib file. A list of command line options can be printed with:

```
   air -
```

A list of options with brief descriptions can be found in the Reference section of this manual. This tutorial introduces a few of the more commonly used options.

Multithreaded Rendering

The number of threads can be set with a `-p` or `-threads` option:

```
   air -p 2 rib/frame0001.rib
```

The default number of rendering threads is equal to the number of detected processing cores.

Image Size and Output Destination

Use `-res` to override the image width and height set in the rib file:

```
   air -res 320 240 scene.rib
```

The output image file generated by the rendering can be set with `-file`:

```
air -file occlusion.tif scene.rib
```

The image type is determined by the file name extension. To force output to the AIR Show framebuffer instead of to a file, use the `-d` switch:

```
air -d scene.rib
```

Pixel Samples and Shading Rate

AIR has two main controls for image quality. The pixel samples option determines the number of times the scene geometry is sampled per pixel. E.g., a pixel samples setting of 4 will use a 4x4 grid of samples at every pixel. Use `-samples` to override the pixel samples setting on the command line:

```
air -samples 4 scene.rib
```

Increase the number of pixel samples to improve the quality of geometry sampling.

The shading rate determines the shading quality of an image. Shading rate is defined as the area in pixels covered by a single shading sample. A lower shading rate produces a higher quality image. For example, a shading rate of 0.25 will produce 4 shading samples per pixel. The `-sm` command line option is a multiplier for the shading rate setting in the rib file:

```
air -sm 0.25 scene.rib
```

Bucket Size and Bucket Order

By default AIR renders an image in 32x32 pixel tiles or buckets, one row at a time top to bottom. Using a smaller bucket size may reduce peak memory use and provide better load balancing with multiple threads. Set the bucket size with `-bs`:

```
air -bs 32 scene.rib
```

Rendering by column instead of by row can also help minimze memory use for crowd scenes. Use `-ltor` or `-columns` to render in columns, left to right:

```
air -ltor scene.rib
```

Surface Shader Override

The `-surf` switch overrides all Surface shaders declared in the rib file with the alternate shader provided. Alternative surface shaders can be used for rendering multiple output passes from a single scene file. E.g.,

```
air -surf occlusionpass -file occlusion.tif scene.rib
```

A related option, `-nosurf`, suppresses all Surface shaders in the scene file, rendering the scene with the defaultsurface shader which provides simple shading based on the surface normal. The nosurf option is useful for quick tests and for debugging. For example, if your scene is rendering black, rendering with -nosurf will indicate whether the agents are missing or just lacking illumination.

Statistics

Rendering statistics can be enabled with `-stats`:

```
air -stats scene.rib
```

Rendering Multiple Frames

AIR 8 allows a frame sequence to be rendered with simple frame number substitution. For example,

```
air -frames 8 10 scene#f.rib
```

would render `scene8.rib`, `scene9.rib`, and `scene10.rib`.   The frame number pattern can also be zero-padded:

```
air -frames 8 10 scene#3f.rib
```

would render `scene008.rib`, `scene009.rib`, and `scene010.rib`.

See Also

Command Line Switches

21.3 Display Customization and Multipass Rendering

The output image or images generated by a rendering are defined by one or more `Display` calls in a scene file. The Render dialog in Massive allows you to set up a single Display call that will generate an 8-bit image with 4-channel output (rgba). This standard output can be customized and enhanced with the use of additional rib commands in an auxilliary text file that is passed to the renderer prior to the main scene file.

Quantization for the Main Display

The output data type for the main display call can be changed from 8-bit to 16-bit or floating-point using the `Quantize` rib command. For 16-bit data (quant16.rib):

```
Quantize "rgb" 65535 0 65535 0.5
Quantize "rgba" 65535 0 65535 0.5
```

For floating-point data (quantfloat.rib):

```
Quantize "rgb" 0 0 0 0
Quantize "rgba" 0 0 0 0
```

A sample command to render 16-bit output might be:

```
air quant16.rib scene.rib
```

When rendering 16-bit and floating-point images be sure to use a file format that supports the corresponding data type. 16-bit images are supported by the PNG, PSD, SGI and TIFF formats. Floating-point data is supported in the TIFF, OpenEXR, and HDR formats. AIR Show can display images with 8-bit, 16-bit, or floating-point data.

Multipass Rendering with Multiple Displays

AIR provides two complementary features that enable multipass rendering:

1. Multiple images can be produced from a single rendering by providing additional `Display` calls with the file name prepended with a +.
2. Any output variable from a surface shader can be saved in an image.

Most AIR surface shaders provide a standard set of color output variables:

```
__ambient
```

```
  __diffuse
  __specular
  __reflect
  __refract
  __incandescence
  __constant
```

`__diffuse` is further broken down into:

```
  __diffuse_unshadowed
  __shadow
  __indirect
```

These extra output variables (also sometimes referred to as arbitrary output variables or AOVs) can be saved using an appropriate `Display` call. A sample rib file that saves diffuse and specular passes in addition to a beauty image would look like:

```
   Display "mp_beauty.tif" "file" "rgba" "quantize" [0 255 0 255]
   Display "+mp_diffuse.tif" "file" "varying color __diffuse" "quantize" [0
  255 0 255]
   Display "+mp_specular.tif" "file" "varying color __specular" "quantize"
  [0 255 0 255]
   Option "render" "commands" "-Display"
```

The last line is important: it tells AIR to ignore subsequent `Display` calls in the RIB stream. Save the above to a text file, say `displays.rib`, then render with

```
    air displays.rib massivescene.rib
```

You can save 16-bit or floating-point data instead of 8-bit data by changing the "quantize" parameters. For 16-bit output:

```
   "quantize" [0 65535 0 65535]
```

For floating-point output:

```
   "quantize" [0 0 0 0]
```

The frame number can be included in a `Display` call with `#nf` where n is the number of digits. E.g.,

```
   Display "beauty_#4f.tif" "file" "rgba"
```

would render to

```
   beauty_0021.tif
```

for frame 21.

Multiple Outputs in a Single Image

Instead of generating a separate image for every output variable, multiple output values can be saved in a single file to formats such as TIFF, OpenEXR, and PSD that support more than 4 channels per image. To save more than one value in a single image, provide a comma-separated list of output variables in the Display call. For example:

```
      Declare "__diffuse" "varying color"
      Declare "__specular" "varying color"
```

```
     Display "multi.psd" "file" "rgba,__diffuse,__specular"
```

Display Templates

The AIR distribution includes 3 template files with Display calls for rendering 8-bit, 16-bit, and floating-point output:

```
     displays8.rib
     displays16.rib
     displaysfloat.rib
```

These templates can be found in $AIRHOME/archives. They can be customized by adding or removing # symbols at the beginning of `Display` lines to enable or disable the corresponding output image.

As-is the templates render TIFF files. Change the extension of the image file name to select a different image format.

To render to AIR Show, change the output driver name from file to framebuffer.

See Also

Output

21.4 massrib

Massrib is a command-line tool for manipulating the rib files produced by Massive.

Note: Massrib is no longer needed to render Massive scenes with Air on Windows now that Massive and the Massive procedural primitive run under Windows.

Massive uses a procedural primitive to generate the members of a crowd at render-time, which is very efficient but not easy to customize. The main function of massrib is to convert the Massive procedural calls to a plain binary or text rib file. The resulting "flat" rib file can be rendered without access to the Massive scene files or Massive procedural. A flat rib is also amenable to further postprocessing, since it is just a plain rib file.

There are several disadvantages to using a flat rib. First, the flat rib files themselves can be rather large. Second, the procedural method of crowd-generation employed by Massive allows AIR to render huge scenes very efficiently, since each agent is generated only as (and if) it is needed. If an agent does not need to be ray-traced, AIR can also dispose of an agent as soon as it has been rasterized by the scanline renderer. When rendering a flat rib, AIR will load all agents which will consume more memory. Even with a flat RIB, AIR should be able to render many hundreds to thousands of agents before running out of memory.

Usage

```
  massrib render.0000.rib
```

Given a source rib name, massrib produces a "flat" rib with the same name in an output directory. By default the output directory is a `massrib` subdirectory of the current directory, which massrib will create if necessary. An alternate output directory can be specified using the `-outdir` option:

```
  massrib -outdir /usr/local/tmp/flat render.0000.rib
```

The simplest way to use massrib is to convert the main scene rib generated by Massive. The resulting flat rib will include the agents and terrain in a single file. You can also convert only the agent archive, producing an archive with just the agents for inclusion in a scene generated in another application. Multiple files can be processed using simple wildcards. E.g.,

```
massrib render.*.rib
```

Setting Massive Procedural Options

The Massive scene file includes an additional call to the procedural primitive that sets certain options such as motion blur and pass type for subsequent agent calls to the procedural primitive. In the current Massive version the call to set procedural options appears after the calls to create the agents in the rib stream, which means that massrib generates the agents before the options have been set. To work around this problem, massrib includes a `-massopt` command line option that can be used to provide option information which massrib will use to initialize the procedural primitive.

To find the Massive procedural options, look at the bottom of the main scene rib for a line that looks like:

```
Procedural "RunProgram" ["run_program.exe" "options motion_blur off
render_pass beauty air "]
[-2250.06 2249.94 -262.813 281.777 -2250.36 2249.64]
```

The second string enclosed in quotes contains the options. Pass that entire string (including the quotes) to massrib with `-massopt`:

```
massrib -massopt "options motion_blur off render_pass beauty air "
render.0000.rib
```

The options should be the same for every frame in a sequence. By default massrib will look for the procedural primitive in `/usr/local/massive/bin`. If Massive is installed in a different location, include a `-procpath` option prior to the `-massopt` option with the path to the Massive procedural.

Additional Command Options

`-ascii`

By default massrib produces a rib file in binary format for compactness. Use `-ascii` to generate a plain text rib file.

`-mapdir` *`olddir=newdir`*

This option allows path and file references in the source scene to be remapped. For example, with the following massrib invocation:

```
massrib -mapdir /usr/local/texture=i:/texture
```

a reference in the source rib file to

```
/usr/local/texture/shirt.tif
```

would become

```
i:/texture/shirt.tif
```

in the output rib file. Useful for changing texture references for rendering under Windows.

```
-optrib ribfilename
```

This option inserts additional rib commands from the specified file into the output stream just prior to the `WorldBegin` statement Useful for adding custom options and attributes to a scene.

Start massrib with no arguments to display a list of command line options.

21.5 Shading and Lighting in Air Space

A file Massive rib file that has been converted with massrib can be moved to Windows and edited with Air Space, the standalone shading and lighting tool included with the Windows distribution of AIR. Air Space provides an interactive shading and lighting environment utilizing TweakAIR, the interactive version of AIR. Air Space provides a simple means of lighting a scene and customizing the shading for individual agents. More information about Air Space can be found in the separate Air Space user manual.

Preparing a Massive Scene for Air Space

When preparing rib export in Massive, make sure the main rib file name is of the form:

```
basename.framenumber.rib
```

Air Space looks for a frame number between periods just before the .rib extension.

Creating a New Air Space Project

A new project is created using a single frame as reference for the IPR preview and opengl window. The shading and lighting in a project can be applied to an entire sequence. You can change the reference frame at any time using the Switch Model item in the File menu.

For a new project you will have the option of importing the existing shaders and lights from your rib file, allowing you to utilize the shaders assigned to agents in Massive. If your scene contains references to texture maps, make sure you use the -mapdir option in massrib to remap the texture file references if necessary.

Note: distant lights imported from Massive will appear quite small, near the scene origin.

Rendering a Frame Sequence

To render a sequence of frames, select File Sequence in the Animation section of the Render page and enter a frame range in the text box, such as

```
20-30
```

Click the Render Animation button in the toolbar to render the sequence, or choose Export Animation from the Render menu to export frames without rendering. Exported frames reside in a subdirectory of the current project directory. For a project named stadium with version number 1, the exported frames will be, e.g.,

```
stadium_v1/stadium_v1.0020.rib
stadium_v1/stadium_v1.0021.rib
stadium_v1/stadium_v1.0022.rib
...
```

While a sequence is being rendered, it is important not to overwrite the files in the export directory. You can continue working and rendering in Air Space by incrementing the version number, which will create a new export directory. Note that the export directory contains only shading, lighting, and basic rendering options. The scene geometry is stored in the mapped rib files created from the source ribs, and those are shared across render versions.

Making Changes to Animation

Air Space stores shading and lighting information in a manner that is independent of the underlying scene file. If you make changes to your animation in Massive, you can re-export rib files, convert them with massrib, overwrite the current ribs used by Air Space, and Air Space will automatically produce new mapped rib files and apply your current materials and lights to the new simulation. Use the Refresh Model button in the Air Space toolbar to reimport geometry from a changed source rib file.

Limitations:

Shadow maps cannot be generated per-frame. Use ray-traced shadows instead.

21.6 Ambient Occlusion in Massive

AIR provides two common ways of rendering ambient occlusion: a special envlight light shader that uses ambient occlusion to compute its shadows, and an occlusionpass surface shader.

The envlight and occlusionpass shaders included with AIR rely on attributes and options to control their behavior. Because Massive does not provide a convenient way of setting custom attributes and options, we have developed special versions of these shaders - massive_envlight and massive_occlusionpass - that do not rely on custom attributes or options.

These shaders should not be used with the occlusion prepass. If you wish to use the occlusion cache with an occlusion prepass, use the standard envlight and occlusionpass shaders with the process described below under occlusion caching.

For ambient occlusion to function properly, ray-traced shadows must be enabled in Massive's render dialog.

Shader Parameter Quick Reference

The massive_envlight and massive_occlusionpass shaders add the following parameters to those in the envlight and occlusionpass shaders:

The main quality control is the *samples* parameter, which gives the number of rays cast for each occlusion estimate. Use more rays to produce a smoother image with less noise at the expense of longer rendering time. It normally makes sense to increase or decrease the number of samples by a factor of 2.

The *maxhitdist* parameter sets the maximum distance to search for occluding objects. Lower values will reduce rendering time as well as noise.

The *coneangle* parameter gives the half angle in radians of the cone of directions within which occlusion rays are distributed. The default value of PI/2 (~1.57) distributes rays over a hemisphere. Use smaller values to concentrate the rays more in the normal direction, producing a more focused occlusion result.

The *shadowbias* parameter is used to prevent incorrect ray-object intersections by giving an offset to add to each ray's starting location.

AIR 6 introduced a new special screen-space caching mechanism that does not require an occlusion prepass. The screen space cache stores an occlusion sample at each pixel and reuses that sample if possible for nearby shading locations. You can enable this mode by setting the *maxerror* parameter to a positive number, usually 0.2 to 0.5. A higher value allows AIR to reuse samples more often. This new mode produces an appearance similar to rendering without a cache while reducing render time by 10-20% in many cases.

Occlusion Caching with Massive

The occlusion cache accelerates rendering by storing occlusion estimates and re-using them at nearby locations. Obtaining smooth results with the occlusion cache typically requires enabling the occlusion prepass. The easiest way to employ the occlusion cache with Massive is to use a small auxilliary text file with the RIB commands required for occlusion rendering. Here's a sample file for rendering an occlusion pass:

```
      # override all surface shaders
      Surface "occlusionpass"
      Option "render" "commands" "-Surface"

      #enable occlusion prepass
      Option "occlusion" "prepass" 1

      #set quality attributes
      Attribute "indirect" "maxerror" 0.1
      Attribute "indirect" "maxpixeldist" 10
      Attribute "indirect" "nsamples" 256

      #set max distance to search for occluding objects
      Attribute "indirect" "maxhitdist" 500
```

Save the above text in a file called `occpass.rib`. Then from a command shell render with:

```
  air occpass.rib myscene.rib
```

You can specify the output image by adding a `Display` call to the `occpass.rib` file:

```
  Display "occlusion.#4f.png" "file" "rgb"
  Option "render" "commands" "-Display"
```

When rendering multiple frames you'll need to include a frame block with the frame number in the air command line to generate a unique image name for each frame. For example:

```
  air occpass.rib {FrameBegin 22} render.0022.rib {FrameEnd}
```

will render to

```
  occlusion.0022.png
```

The shell script generated by Massive can be modified to include the additional command line parameters in the line that begins air. This method can be used to render an occlusion pass with the RIB files generated for a beauty pass in Massive; there is no need to generate a separate set of RIB files for the occlusion pass.

The same workflow can be used without the occlusion cache and prepass. To disable the occlusion cache, change the maxerror value to 0 in the auxilliary rib file. To disable the occlusion prepass set the prepass option value to 0. Here's a sample rib for rendering without an occlusion cache:

```
      # override all surface shaders
      Surface "occlusionpass"
      Option "render" "commands" "-Surface"

      #disable occlusion prepass
      Option "occlusion" "prepass" 0

      #disable occlusion cache
      Attribute "indirect" "maxerror" 0.0

      #set quality attributes
      Attribute "indirect" "nsamples" 256

      #set max distance to search for occluding objects
      Attribute "indirect" "maxhitdist" 500

      Display "occlusion.#4f.png" "file" "rgb"
      Option "render" "commands" "-Display"
```

Phil's Tips for Occlusion

Veteran AIR and Massive user Phil Hartmann offers the following tips on rendering occlusion:

1. In Massive set the shading rate to 0.25 or 0.5. Set Pixel Samples to 8x8. Set the trace bias in the range 1 to 10.

2. In the massive_envlight or massive_occlusionpass shaders, set samples to 32 or 64. Set maxhitdist to 200 to 500 cm.

3. If memory use is an issue, render the occlusion pass separately from other passes with the massive_occlusionpass shader.

4. To maximize processor usage on a multicore machine, run multiple processes with 1 thread each rather than a single process with multiple threads.

Selective Occlusion Testing with Groups

Problem: A user wishes to render an occlusion pass with agents and terrain in the same image. Agents should be rendered with normal occlusion from all geometry. The terrain should show occlusion from agents but not from the terrain itself.

Solution: Grouping can be used to restrict the objects tested for occlusion as follows:

1. The first step is to assign a group name to the agents by creating a small text file with the following:

```
  Attribute "grouping" "string membership" "+agents"
```

Assign that text file as a custom rib to the agents in Massive.

2. Assign the (new) massive_occlusionpass shader to the terrain. Set the new shadowgroups parameter to "agents" to restrict occlusion testing to just the agents.

3. Assign the massive_occlusionpass shader to the agents, leaving the shadowgroups parameter with its default empty value (so all objects occlude the agents).

See Also:

Ambient occlusion

21.7 Mattes, Layers, and Groups

This note describes two methods of rendering mattes for Massive scenes with AIR, as well as an extension of the second method for rendering by layers or groups.

A Simple RGB Matte Shader

The AIR distribution includes a very simple surface shader called rgbmatte for rendering an "rgb" matte pass in Massive. Since Massive does not allow an object's base color to be changed on a per-pass basis, this shader provides parameters for directly setting the output red, green, and blue channels.

This shader is about as simple as a surface shader can get:

```
surface rgbmatte(float r=0; float g=0; float b=0)
 {
   Oi=1;
   Ci=color(r,g,b);
 }
```

Rendering Mattes using Groups

AIR allows any object to be assigned to one or more groups. Include an object in a group with

```
Attribute" "grouping" "string membership" "+mygroupname"
```

The objects which appear in each output image can be restricted to members of particular groups by providing a "string subset" parameter with a list of groups in a `Display` call. Only objects in the listed groups will appear in the display image; other objects will be treated as matte objects for that image.

These capabilities can be utilized to produce an arbitrary set of mattes for different scene elements in a single render invocation.

Example: say one wanted to render a matte for agents and a matte for terrain.

Start by creating two small text files, `agents.rib`, with

```
Attribute" "grouping" "string membership" "+agents"
```

and `terrain.rib`, with

```
Attribute" "grouping" "string membership" "+terrain"
```

In Massive assign `agents.rib` as a custom rib archive to each agent. Assign `terrain.rib` as the custom rib archive for the terrain. Sim and generate the usual set of exported rib files.

The final piece is another small text file defining the matte images for agents and terrain, `mattes.rib`:

```
Display "+agent_matte#4f.tif" "file" "a" "string subset" "agents"
"quantize" [0 255 0 255]
Display "+terrain_matte#4f.tif" "file" "a" "string subset" "terrain"
"quantize" [0 255 0 255]
```

Include `mattes.rib` as the options rib in the massive render dialog, or pass it on the command line prior to the main scene file:

```
air mattes.rib render0001.rib
```

This method can be used to render a matte for any desired grouping of scene elements.

Rendering Separate Output Images by Group

The above technique easily generalizes to separating any output variable by group. For example, to render separate beauty images for agents and terrain, use:

```
Display "+agent_beauty#4f.tif" "file' "rgba" "string subset" "agents"
"quantize" [0 255 0 255]
Display "+terrain_beauty#4f.tif" "file' "rgba" "string subset" "terrain"
"quantize" [0 255 0 255]
```

See Also:

Rendering in Layers

21.8 Eliminating Shadow Artifacts on Terrain

Problem: When rendering with ray-traced shadows, terrain geometry may exhibit artifacts in the form of incorrect shadows along the boundaries between polygons.

Solution:

In most cases the shadow artifacts are caused by incorrect traced ray intersections. There are two solutions:

1. The best solution is to increase the shadow blur setting in the light to at least 0.005. In most cases that will completely eliminate the problem. (See below for why this works.)

2. Increase the shadow bias, either in the light shader or using the trace bias attribute:

   ```
   Attribute "trace" "bias" 2
   ```

   For Massive scenes start with a bias of around 2 and keep increasing until the lines disappear.

Why Increasing Blur Eliminates Artifacts

When shadow blur is less than 0.005, AIR antialiases shadows by distributing shadow sample positions over the area being shaded. Near polygon edges some of those positions can wind up "behind" neighboring polygons with respect to the light source. This is more likely to happen with a coarse polygon representation.

When shadow blur is 0.005 or greater, AIR antialiases shadows by varying the ray direction from the sample position over the cone angle defined by the blur value only. The sample position is fixed at the current shading point.

21.9 Rendering Massive Agents with Maya & MayaMan

This page describes how to render Massive agents as part of a Maya scene using Animal Logic's MayaMan plugin for Maya.

Background

Massive creates two RIB files when it exports a scene for rendering with AIR: a main RIB file containing the terrain, lights, and global scene options, and a second RIB file that defines the agents. The agents file can be rendered as part of a Maya scene with the aid of MayaMan.

Step 1: Modify the procedural search path.

Massive uses a procedural primitive named run_program.exe to generate agents at render time for AIR. We need to add the location of that program to the procedural search path so that AIR will be able to find it:

- From the MayaMan menu select MayaMan Globals.
- In the MayaMan Global Options dialog select RenderMan Search Paths.
- Select the Procedural tab.
- Type the path to the bin directory of your Massive installation (which contains the Massive run program).
- Click Add.

Step 2: Define options for the Massive procedural primitive.

The main RIB file generated by Massive contains a special command that initializes the procedural primitive with various options for the agents. This line must be duplicated for MayaMan. Look in the main rib file for a frame and find the line that begins Procedural. Copy that entire line. This line will be the same for every frame.

- In the MayaMan Globals dialog select the Advanced Options page.
- Scroll down to the section titled User Defined RIB Statements: World and expand it.
- Paste the Procedural declaration you copied above into the text box. Make sure the entire command line was pasted.
- Click Save User RIB at the bottom of the section.
- Close the MayaMan Globals dialog.

Step 3: Add agents.

- First, create a new simple primitive such as a box or sphere to act as a standin for the agents.
- With the new object selected, add a MayaMan Model Attributes node.
- Select the MayaManAttributes node and expand the section labeled ReadArchive.
- Check Substitute RIB and use the file dialog to choose an agent file generated by Massive. For an animated sequence, MayaMan will replace a #4f pattern in the archive name with the frame number.
- Make sure the DelayedReadArchive option is NOT checked.

Step 4: Enable motion blur (if applicable).

- In the MayaMan Globals dialog select the Motion Blur options page.
- Check Enable Motion Blur.
- Check Time in Zero to One Range
- Set the Motion Blur Amount to correspond to the motion blur setting in Massive.

You should now be able to render your Maya scene with the agents appearing at render time in place of the proxy object. The surface and displacement shaders used for the agents will be those configured for the selected render pass in Massive.

21.10 Automatic Level of Detail

AIR 7 and later can automatically reduce the number of polygons in a polygon mesh based on an object's on-screen size and other factors. Automatic LOD applies only to polygon mesh primitives.

Enable automatic level of detail for an object with the following custom attribute:

```
Attribute "render" "float autolod" [n]
```

where *n* gives an error tolerance when simplifying a model. Reasonable values to try are 0.1 to 0.3. A level of 0 disables automatic LOD.

Automatic LOD is also influenced by the "flatness" attribute:

```
GeometricApproximation "flatness" 0.5
```

which gives the maximum difference in pixels between the original surface and the simplified surface. The default flatness value is 0.5. A value of 1 or 2 will produce smaller final meshes.

Tips

A reasonable set of values to start with are:

```
Attribute "render" "float autolod" [0.2]
GeometricApproximation "flatness" 1
```

When applied to cloth, auto LOD may result in clothing that does not cover the underlying limbs. Using a smaller autolod tolerance may help prevent this condition. Another solution is to use an opacity map to hide parts of a limb that would be hidden by clothing anyway.

21.11 Massive Agent Light Tool (malt)

AIR 8 includes a new Massive agent light tool (malt) to enable rendering Massive agents with per-agent lights. Possible applications include vehicle headlights or crowds carrying torches or flashlights.

The basic idea is to use a piece of standin geometry to represent an agent light, which is converted to an actual RIB light source by the malt utility prior to final rendering. The workflow in detail:

1. Create and position standin geometry for each light associated with an agent. The shape should be a pyramid or triangle. By default malt uses the first vertex as the light position and the average of the other vertices as a target point for the light. If the -last command line option is passed to malt, it will use the last vertex as the light position.

2. In Massive assign a special surface shader to the standin geometry with the light shader properties. The AIR distribution includes two "proxy" surface shaders: proxy_spotlight for the standard spotlight shader, and proxy_pointlight for the standard pointlight shader. Most parameters of the proxy shaders mirror those of the corresponding light shader. There are a few special parameters:

   `LightShader` - the name of the light shader to use. Due to a bug in Massive 3.0 that initializes all string parameters to the empty string, the parameter must be set to "spotlight" for the proxy_spotlight shader or "pointlight" for the proxy_pointlight shader.

   `ShowProxyGeometry` - set the value to 1 to make the proxy geometry visible in the rendering.

   `LightMaxDistance` - sets the light maxdist attribute defining the range of influence of the light as a distance from the light source position. Setting this value appropriately can help reduce render times.

For initial tests you may also wish to set the `falloff` parameter to 0.

If the lights are to cast ray-traced shadows, set the shadowname parameter to "raytrace".

3. Make sure the proxy geometry is tagged not to cast shadows in Massive.

4. Sim to export rib files as usual.

5. Use the malt tool to create agent lights for each frame:

```
malt scene0001.rib
```

The malt tool parses through all the agent data looking for objects with a "light" surface shader. For each such object, malt creates a light in an agent light file named "scene0001_agentlights.rib". After processing the agents, malt re-writes the main scene file, adding a reference to the agent lights just after the `WorldBegin` statement.

6. Render the main scene file as usual.

21.12 Rendering Massive Agents in Rhino

This note describes how to include Massive agents when rendering a Rhino model with the RhinoAIR plug-in. This method uses the new MassiveAgents instancer shader to add agents at render time.

Preliminary: Massive uses a small program called runprogram.exe to create agents at render-time. AIR must be able to find the runprogram.exe file at render-time. The easiest way to ensure that is to copy the runprogram.exe file from your Massive installation to the procedurals directory of your AIR installation.

1. Select the terrain object in the scene (or another large object if there is no terrain). Display the AIR Material page for the object. Assign the MassiveAgents shader as the instancer shader.

2. Set the instancer Extent to a value such that the terrain object bounding box incremented by the Extent would encompass all the Massive agents in the scene.

3. Set the Agent archive prefix parameter to the file name of the agent archives exported from Massive minus the frame number and .rib extension.

4. Set the RenderPass parameter to the name of the render pass you set up for Air in Massive.

5. Set the Rotate x value to 90 to compensate for the different up directions in Rhino and Massive (the Y axis is up in Massive; the Z axis is up in Rhino).

6. Use the Frame offset value to pick the Massive agent frame that should correspond to the first Rhino frame in an animation.

7. Use the Scale parameter to uniformly scale the agents to match the scale of your model. By default agents in Massive are assumed to be modeled in units of centimeters.

8. To render an animated sequence, you must enable one of the animation modes in RhinoAir on the Air Animation page of the Rhino options dialog. If your scene does not employ any other animation, set the Animation mode to Turntable with a Max angle of 0.

9. Render to see your Rhino model with Massive agents generated at render-time.

See Also

MassiveAgents instancer shader

21.13 Optimization Tips for Massive Scenes

1. Make agents and terrain invisible to all ray types if the scene does not use raytracing. When rendering out of Massive, unchecking the Shadow control on the Render page should be sufficient.

There are two easy ways to see whether objects are being retained for ray tracing:

In the Air Show status bar, the 5th section from the left will contain S, R, or I if any objects are visible to shadow, reflection, or indirect rays respectively.

The statistics generated by Air report the memory used by objects retained for ray tracing. Statistics can be enabled with the `-stats` command line switch.

2. Render by column instead of by row. By default Air renders an image from top to bottom, one row of tiles at a time. For a typical wide crowd shot peak memory can often be reduced by instead rendering from left to right in columns. The bucket or tile order can be changed to left-to-right using the `-columns` command line switch or the following option:

```
Option "render" "string bucketorder" "columns"
```

3. Use a smaller bucket size to reduce memory use and improve load balancing when rendering with multiple threads. The bucketsize can be set using the `-bs` command line switch:

```
air -bs 16 scene.rib
```

or using the following option:

```
Option "limits" "bucketsize" [16 16]
```

The default bucket size is 32 x 32 pixels. A smaller bucket size will typically increase render time, so use as large a bucket size as feasible.

4. Convert all texture maps to Air texture files with mktex or mktexui to optimize memory use and texture quality.

5. Enable mesh data simplification with

```
Attribute "render" "integer simplifyfacedata" [1]
```

This will normally result in lower memory use for Massive agents.

6. Use multiple instances of the Massive procedural primitive (Air 10 and later)

Massive uses a RunProgram procedural primitive to generate agents at render time. That procedural can often become a bottleneck in multithreaded rendering. You can allow multiple instances of the RunProgram to run concurrently by increasing the maximum number of instances allowed with:

```
Option "limits" "runprogramthreads" [nmax]
```

Air will automatically send the scene options string to each instance of the Massive procedural primitive.

21.14 Rendering a Depth Pass

There are several ways to render a depth pass with Air. Here are two that work with an existing set of rib files (so you do not need to set up a separate render pass in Massive or export an additional set of files).

Method 1

Save the standard depth buffer z as the output image. First, create a small text file with the following commands:

```
Display "depth#4f.tif" "file" "z"
Option "render" "commands" "-Display"
Option "render" "zrange" [1000 10000]
```

The `zrange` option sets the minimum and maximum depth values which correspond to 0 and 1 respectively in the exported z value. If you wish to save raw depth values, omit the line that sets the `zrange` option.

Then render from a command shell including the extra file prior to main scene name:

```
air zdepth.rib scene0001.rib
```

For a sequence use:

```
air -frames 1 100 zdepth.rib scene#4f.rib
```

Method 2

Render a depth gradient using Air's depthpass surface shader. As in method 1, first create a small text file with a few RIB commands:

```
Display "depth#4f.tif" "file" "r"
Option "render" "commands" "-Display"

Surface "depthpass"
  "mindistance" 1000
  "maxdistance" 10000

Option "render" "commands" "-Surface"
```

Then include the extra file on the command line:

```
air depthpass.rib scene0001.rib
```

21.15 Adding a background image

It may sometimes be useful to render a preview image with Massive that includes a background plate. The Massive camera node includes an option to specify a background image, but that setting is not exported to rib for rendering with Air. Hopefully Massive Software will add this feature in a future release; until then, here are a couple alternatives:

Option 1: Use a small options file with an imager shader to add a background image:

- Create a small text file with the following imager declaration:

```
Imager "VBackdrop" "string TextureName" "filenameofbgimage"
```

- Enter the options rib file name as the "options rib include" entry in the 'render parameters' tab of the renders dialog.

Drawback: the imager shader will be included in the rib export for preview rendering and normal rendering. If you do not wish to include the background image in final rendering, you'll need to remove the options reference prior to exporting ribs for the final render.

Option 2: Use Air Show to view the rendered preview image with a background plate

Air Show has an option to display a rendered image over a background plate, but that option is only available for images with an alpha channel. Unfortunately, the Massive preview image is always "rgb" only, so again this is not as easy as one might hope.

A solution is to render the preview rib from a command shell and use Air's command line options to override the output declaration to include an alpha channel.

When rendering a preview image with Air, Massive exports the rib files and other data to $TEMP/massive. To render the preview rib from a command shell:

```
cd $TEMP
cd massive
air -d -mode rgba massive.rib
```

You can leave the command shell open and re-launch the last command for each preview. Note that this means each preview will be rendered twice. If the preview render takes more than a few seconds, you can use the red stop button in Air Show to abort the first preview render. To load the background image in Air Show, select Load Background from the View menu. Use the checkered toolbar button to enable or disabled display of the background for the current image.

22 Houdini

Side Effects Software's Houdini Escape and Houdini Master support Air as a 3rd-party rendering option.

Version Requirements

The information in this section covers Air 7.0 and later and Houdini 9.1 and later.

22.1 Configuring Houdini for Air

Set Air as the default RIB Renderer

Create a `HOUDINI_DEFAULT_RIB_RENDERER` environment variable with the value

```
air6.0
```

Add Air Shaders to Houdini

Air ships with a set of dialog scripts that make it easier to use Air shaders with Houdini. To add the dialog scripts to your Houdini installation, copy the contents of the directory:

```
$AIRHOME/houdini/shop
```

to the `shop` directory in your Houdini installation. If the `shop` directory already contains files named "SHOPsurface", "SHOPdisplace", "SHOPfog", or SHOPlight", append the Air versions of these files to the existing files.

When you re-start Houdini, the AIR shaders should appear in the list of generators. Each Air shader will have a prefix of `air_`.

Preview Air Shaders in Houdini

Air shaders to be previewed in Houdini by defining a `HOUDINI_VIEW_RMAN` environment variable with the value `air.`

Air 7.0 and later for Windows ship with a custom display driver used to send the preview image to Houdini. With earlier versions of Air or Linux, you will need to run the proto_install utility included with Houdini and install the renderman display driver. Remember the installation location you select. Then copy the display driver file from the installation location to the `displays` directory of your AIR installation.

Update Air Output Dialog Script

When a new RenderMan output operator is created in Houdini with Air as the default renderer, Houdini automatically creates a dialog for setting basic rendering parameters and custom options. That dialog has been updated with Air 7 release for Houdini 9.1. Compare the dialog script in the Air distribution:

```
$AIRHOME/houdini/air6.0.ds
```

with the one in your Houdini installation:

```
$HFS/houdini/soho/parameters/air6.0.ds
```

and keep whichever is more recent in your Houdini installation.

22.2 Rendering with Air from Houdini

To render with Air from within Houdini 9.1 or later:

- Display the Output network in a pane, and add a RenderMan generator, which should appear as an operator named `rib1`.
- Display the parameters for the `rib1` operator.
- On the Main page of the parameters dialog, make sure the Render Target is set to AIR 6.0.
- Set the Camera control to the desired camera to use for rendering.
- To render the scene:

  Click the **Render** button in the `rib1` properties dialog, or

  In the Houdini Render menu, choose **Start Render -> rib1**.

  Note that by default primitives in Houdini will not have any surface properties defined, so all surfaces will appear with a simple default surface shader that shows the geometry but no other shading details.

- By default Houdini starts the Air rendering process and sends the scene directly to Air via a pipe.

Alternatively, you can have Houdini create a RIB file that can then be edited and rendered outside Houdini. To enable RIB file generation, check the box next to **Disk File** on the **Main** page in the `rib1` properties dialog and provide a valid file name.

Applying Air Surface Shaders

To produce usable output you will need to assign an Air surface shader to each object. The surface can be a shader that has been written in the RenderMan shading language and compiled for Air, or a RenderMan shader that has been built in Houdini.

Air comes with dozens of pre-built surface shaders that are ready to use. In the configuration section you made those shaders available to Houdini by adding a set of dialog scripts. To apply an Air surface shader to an object in Houdini:

- Display the Shop network and bring up the Tool menu. Under RenderMan Surfaces you should find many shops with the prefix AIR. Choose one of these surface shaders to apply to your object. (If you do not see any Air shops, return to the Configuration section for instructions on installing the AIR dialog scripts.)

- Assign the new AIR shop to your object(s) using any of the standard methods of material assignment in Houdini.

- When an AIR shop is selected you will a list of shader parameters that can be used to modify the appearance of the shader. See the Shader Guide section of this manual for more information about the shaders included with AIR.

Rendering the scene with Air should now use the assigned surface shader.

Image Quality

Basic image quality controls can be found in the Quality sub-tab of the Properties tab of a RIB output op.

The **Pixel Samples** setting determines how often the scene geometry is samples per pixel, defining an NxM grid of pixel samples. The default setting of 4x4 should be adequate for images without fine detail or motion blur. When using motion blur or depth of field, pixel samples may need to be increased to 8x8 or higher.

Shading Quality sets the number of shading samples per pixel (the inverse of the RIB shading rate property). More shading samples produce smoother shading at the expense of longer render times.

Pixel Filter and **Filter Width** specify how the pixel samples are combined to produce final pixel values. The defautl gaussian filter with a filter width of 2 is somewhat blurry. For a sharper filter, try a mitchell filter with a filter width of 4.

Multithreading

Set the number of threads or processors used for rendering on the Render sub-tab under Properties for a RIB output driver.

Rendered Output

Select the desired output image name and display device on the **Display** sub-tab of the **Properties** tab of the RIB output op.

By default the rendered image will be sent to the standalone Air framebuffer Air Show. To render to the Houdini framebuffer, change the **Display Device** to `Houdini`. To render to a file, select the display device for the desired file format.

By default the rendered image will be saved at 8-bit precision. To render at 16-bit precision, change the Quantize values to

```
  65535 0 65535 0.5
```

To render a floating-point image, set the Quantize values to

```
  0 0 0 0
```

Air can produce an arbitrary number of additional output images from a single rendering. Additional images typically save *arbitrary output values* (AOVs) computed by a shader. Use the **AOV** tab to add additional output images.

Reflections

Many Air surface shaders can compute reflections using ray tracing. For objects to appear in reflections, they must be made visible to reflection or trace rays. To enable reflection visibility globally, go to the **Trace** sub-tab of a RIB operator's **Properties** tab, and check **Visibile in Reflections**. The **Max Trace Depth** control in the same tab gives the maximum number of "bounces" to follow reflection or refraction rays.

A background color or environment map can also be given that provides a color for traced rays that miss all objects in the scene.

Motion Blur and Depth of Field

Motion blur and depth of field can be enabled on the **Sampling** sub-tab under the **Properties** tab of the RIB output operator.

22.3 Houdini Lights and Air

Houdini automatically translates Houdini lights when exporting a scene to Air

Ray-traced Shadows

If a Houdini light's Shadow Type is set to Ray-Trace Shadows, Houdini will enable ray-traced shadows for the light when exporting for Air. One other attribute needs to be set to make objects visible to shadow rays.

- Display the Rib output operator in a panel and select the Properties tab.
- Look in the Attributes tab for a property named "Transmission Visibility". (If the attribute is not present, you can add it by editing the operator type and finding the attribute in the rendering section under AIR 6.0/Attributes.)
- Set the value to "Use Os for opacity" or "Opaque to shadows".

Note that as of Houdini 9.1.160, native Houdini lights do not provide much control over ray-traced shadows. In particular, there is no control over shadow blur or sampling. For more control, use a custom AIR light shader instead (see below for instructions).

Shadow-Mapped Shadows

- Set the light's Shadow Type to Shadow-Mapped Shadows
- Uncheck Transparent Shadows

- Make sure the Shadow Softness is set to at least 1.

Using Custom Light Shaders

AIR light shaders can be used in Houdini with the aid of a Light template. Make sure you have already installed the AIR shaders for Houdini as described here.

To use a custom light shader:

- In the SHOP network, open the Tool menu, and under the RenderMan Light category, choose an AIR light shader. For an initial test, try the AIR point light shader.
- In the parameter list for the point light shader, change the Falloff parameter to 0 to disable intensity decay with distance.
- In the OBJ network, create a Light Template.
- On the Shaders page of the light template, set the Light Shader value the AIR light shader added above.
- Move the light template position in a model view so the light is in a reasonable position to illuminate the scene.
- Render.

To enable ray-traced shadows for a custom light, set the Shadows parameter to the special value `raytrace`. The Shadow blur parameter determines the blurriness or softness of the shadows. The Shadow samples parameter sets the number of rays to trace when computing the shadow result.

22.4 Ambient Occlusion with Air and Houdini

Ambient occlusion can be used either as a surface shader or with a light shader.

Air's envlight shader provides occlusion-based shadows with optional image-based lighting.

To use the envlight light shader:

- In the SHOP network, open the **Tool** menu, and under the **RenderMan Light** category, choose the AIR envlight shader.
- In the OBJ network, create a Light Template from **Render** sub-menu under the **Tool** menu.
- On the **Render** tab of the light template, select the **Shaders** subtab, and set the **Light Shader** value to the AIR envlight shader added above.
- Make all objects in the scene visible to shadow rays. In the parameters window for the RIB output operator, look under the **Properties** tab for the **Trace** sub-tab. Set **Shadow Visibility** to **Use Os for opacity**.
- Render a test frame.

Occlusion Cache Acceleration

The default settings for ambient occlusion can be slow as Air computes a new occlusion sample at every shading location. Using an occlusion cache can greatly accelerate occlusion rendering by allowing Air to re-use occlusion samples. To enable occlusion caching:

- In the parameters window for the RIB output operator, look under the **Properties** tab for the **GI** sub-tab.
- Change the **Max Cache Error** value from the default of 0 to 0.25.
- For best results, also enable the **Occlusion prepass**.
- Render another test frame.

Occlusion Pass Surface Shader

As an alternative to the envlight, you can use Air's occlusionpass surface shader to render an occlusion pass.

See Also

Ambient occlusion

22.5 Global Illumination with Air and Houdini

Global illumination in the form of indirect diffuse illumination can be added to a Houdni scene by adding an AIR indirect light and enabling controls in the GI section of the rib output operator.

Step 1: Adding an indirect light:

- In the SHOP network open the **Tool** menu, and under the **RenderMan Light** category, choose the AIR Indirect light shader.
- In the OBJ network, create a Light Template from the **Render** sub-menu under the **Tool** menu.
- On the **Render** tab of the light template, select the **Shaders** subtab, and set the **Light Shader** value to the AIR indirect shader added above.

Step 2: Configure global GI options and attributes:

- Select the RIB output operator that will be used for rendering. Under the **Properties** tab, select the **GI** sub-tab.
- Check **Visible to indirect rays** to make all objects contribute to global illumination by default.
- Render a small test image.

The scene must include some direct illumination in order to see any GI results.

Global Illumination Cache Acceleration

With the default settings indirect diffuse illumination can be quite slow as AIR computes a new GI sample at every shading location. Gi can be accelerated by enabling the irradiance cache:

- In the **GI** subtab, change the **Max Cache Error** to 0.25 to enable the GI cache.
- For smooth results, check the **Indirect Prepass**.
- Another test render should finish much more quickly.

The quality of the GI cache is controlled by the Max Cache Error and Max Spacing settings. Max Cache Error takes a value between 0 and 1 giving the maximum error allowed when trying to re-use samples in the cache. A max error of 0 disables the GI cache.

The Max Spacing value gives the maximum distance in pixels between samples in the cache, ensuring that all areas of the image receive at least some GI samples.

Global Illumination Background

Use the controls at the bottom of the GI sub-tab to specify incoming illumination from the "environment" surrounding the scene. The GI background can be either a simple color or an environment map. For an environment map, the map intensity and map blur can also be set.

See Also

Indirect Diffuse Light

22.6 Creating Shaders for Air in Houdini

Houdini allows RenderMan-compatible shaders to be built in a manner very similar to building Mantra shaders.

RenderMan SHOPs are different from Mantra SHOPS. Mantra cannot render RenderMan SHOPs, and Air cannot render Mantra SHOPs.

An Air-compatible RenderMan SHOP can be started in two ways:

Method 1:

- Display the SHOP network.
- In the **Tool** menu, go to the **RenderMan Surface** item and select the **VOP RSL Surface Shader** sub item.

This action provides a bare-bones RenderMan surface shader. You'll need to edit the network and add more nodes to create a usable shader. For example:

- Edit the shader network.
- Go to the **Tools** menu, **Shading** section, and add a **Lighting Model** node.
- Connect the Lighting Model's clr output to the Ci input of the output1 node. (Ci is the emitted color result from an Air/RenderMan surface shader).
- Assign your new SHOP to an object and test render to see a basic no-frills surface shader.

Method 2:

- Display the Material Palette
- At the top of the lefthand pane, select RenderMan as the gallery filter to display.
- Look in the **General** section for a **Basic Surface** entry and add it to the SHOP list.
- The new SHOP provides a surface shader with some basic controls and hooks into Houdini.

Compiling

Houdini will automatically attempt to compile a RenderMan shader for Air. If you are using a version of Houdini prior to 9.1.169, the compile may fail because older version of the hrmanshader utility included with Houdini are not compatible with Air. You will need to update the hrmanshader binary (or hrmanshader.exe) in your Houdini installation with a more recent build in order to compile shaders for Air.

22.7 Adding New Shaders to Houdini

New shaders that you create can be added to Houdini with the aid of the MakeLIF utility.

23 Tools

The following programs are included in the `bin` directory of the Air distribution.

Tool	Command	Description
Air renderer	air	32-bit air binary under Windows 64-bit air binary under Linux
	air64	64-bit air binary under Windows
Air Control	aircontrol	User interface for starting and managing Air render jobs
Air Point Tool	airpt	Command-line tool for manipulating 3D point files
Air Show	airshow	Standalone framebuffer for displaying rendered images
Air Space	airspace	Standalone interface for adding materials and lighting to a model. See the separate Air Space User Manual for more information. Windows only.
BakeAir	bakeair	Air texture baking tool
Make LIF	makelif	Command-line tool for creating help files for shaders
Retexture	retexture	Interactive tool for editing materials as a post-process. See the separate Retexture User Manual for more information. Windows only.
Shading Compiler	shaded	Compiler used to compile shaders for use with Air
Shader Info	slbtell	Command-line tool for querying the parameters of a compiled shader
Texture Converter	mktex	Command-line texture conversion tool
Texture Converter UI	mktexui	User interface for mktex
TweakAir	tweakair	Interactive version of air
Vortex	vortex	Distributed rendering tool
Vshade	vshade	Visual interface for building shaders. See the separate Vshade User Manual for more information.

23.1 AIR Control

(Windows only)

Air Control is a simple user interface and render queue for rendering scene files with Air. Air Control also accepts rendering jobs submitted from the companion airq batch render command.

To render a single frame:

- Enter the RIB file name in the **Scene** control or click the ... button to the right to display a dialog for choosing a RIB file.
- Set the number of threads to use for rendering. Note that the Air demo is limited to a single thread.

- If **Low priority process** is checked, the rendering process will run at a lower Windows priority, minimizing the impact on the performance of foreground applications such as a modeling program.
- Click the **Render** button to start rendering.

The rendering job will appear in the list of tasks on the Jobs page. As the frame renders, you can check the console on the Jobs page for any error messages.

To rendering multiple frames:

- Enter the RIB file name for one frame in the sequence. The file name must include a frame number, left-padded with zeroes. E.g.,

  ```
  scene0001.rib
  ```

- Check the **Frame range** control, and enter the first and last frame numbers in the appropriate boxes.
- Click **Render**.

Submitting Batch Render Jobs

The companion airq program can be used to submit rendering jobs to the Air Control queue from a script or application. If Air Control is not already running, airq will automatically start Air Control.

For a single frame, just submit the frame name:

```
airq scene.rib
```

The number of rendering threads can be set with a `-threads` option:

```
airq -threads 4 scene.rib
```

For multiple frames, use one of the scene files as the rib name and specify the frame range with the `-frames` option:

```
airq -frames 0 19 -threads 4 scene0001.rib
```

23.2 AIR Point Tool (airpt)

Airpt is a command line tool for converting and manipulating 3D point sets. Airpt reads and writes the same point set file formats as AIR.

Airpt accepts one or more point set files as input.

```
airpt options pointfile1 ...
```

Airpt can print information about a point collection and/or create a new output file from a set of input files.

Options

`-o` *`filename`*

Specify the name of an output file to which the point set is saved after any modifications are applied.

The point set file format is based on the file name extension.

`-info`

Print information about the point set including the number of points and the channel names

`-rench` *`oldname newname`*

Rename a channel in the point set. The new name must not currently be in use by any other channel.

`-delch` *`name`*

Delete a channel from the point set. The position channel P and width channel cannot be deleted.

`-scalech` *`name scale`*

Multiplies values in the named channel by the specified scale value.

`-maxch` *`name max`*

Limits the values in the named channel to be less than or equal to the specified maximum.

`-minch` *`name min`*

Limits the values in the named channel to be greater than or equal to the specified minimum.

`-range`

Prints the min and max value of each channel

23.3 AIR Show

AIR Show is the standalone program that serves as a framebuffer for AIR. Images rendered to the "framebuffer" device in AIR appear in AIR Show. If AIR Show is not running, the framebuffer driver will automatically start AIR Show on the local machine.

AIR Show is much more than a simple framebuffer:

- As a separate program, AIR Show remains open after a render has completed. AIR Show can display any number of rendered images (limited only by available memory).
- Concurrent processes can write to different images in AIR Show
- Rendering processes can write to AIR Show running on any machine on a network.
- Multiple processes can write to different sections of the same image, permitting a single frame to be rendered over multiple machines
- A sequence of frames can be played as a simple flipbook animation.
- AIR Show can display 16-bit and floating-point image data, as well as the usual 8-bit data, with an unlimited number of channels per image.
- Gamma-correction for display without altering the raw image data that is saved to a file.

- Image navigation including pan and zoom, toggle between images, wipe between images, and view individual pixel values
- Special depth map display for previewing shadow maps
- Viewing of stereo images as color anaglyphs (version 5.0 and later)

23.3.1 Image Display

Gamma Correction

AIR Show can gamma correct an image for display purposes without altering the raw image data. This capability allows you to view an image properly on your monitor and still save it without gamma correction as a "linear" image suitable for post processing.

Gamma correction applied in AIR Show is independent of any correction applied by the renderer based on the `Exposure` settings in a scene file. Naturally, you should avoid gamma correcting an image in both the renderer and AIR Show. If an image is gamma-corrected in AIR, the framebuffer driver will automatically set the AIR Show gamma for that image to 1 (no correction).

The gamma correction applied to an image by AIR Show is determined as follows:

1. If an `AIRSHOWGAMMA` environment variable is defined, that value is the default gamma, otherwise the default gamma is 1.
2. If a gamma factor is provided using the `-g` command line option, that gamma becomes the default for all images.
3. If the display driver call passed to AIR contains a `"float gamma"` variable, that variable's value is used for that image only.

Applying gamma correction to an 8-bit image can result in banding artifacts. If you plan to gamma correct in AIR Show and not in the renderer prior to quantization, render 16-bit or floating-point data to avoid banding.

To repeat a point mentioned above, the gamma correction used by AIR Show for display purposes does not affect the image data that is saved.

sRGB

The latest versions of AIR Show have a new option to apply sRGB gamma correction to an image with the corresponding toolbar button. sRGB correction overrides any custom gamma defined for the image in AIR Show.

Image Background

Images with an alpha channel can be displayed over a background image or pattern by toggling the background button in the AIR Show toolbar. By default the background is a simple checked pattern. Use the Load Background item in the View menu to load an image to use as the background for the current image.

Multilayer Images

AIR can render a single image with multiple output values such as diffuse, specular, and reflection passes. AIR organizes such an image into multiple layers with layer names displayed in the lower right section of the AIR Show window. To view an individual layer, left click on its name.

Composite View

AIR Show 3 introduces a new simple composite view mode for multilayer images. Click the Comp button in the AIR Show toolbar to enable composite mode. The displayed image will then be the sum of all layers, with each layer's contribution scaled by a weight factor. In composite view mode, the list of layers displays the layer weight as a percentage next to each layer. To adjust the weight of a layer, left click in the layer name row. You may need to adjust the width and height of the layer list to facilitate manipulating the layer weights.

AIR recognizes `__shadow` and `__lights_shadow` output values as special layers. Shadow layers are assumed to apply to the immediately preceding layer (which should contain the corresponding unshadowed output).

Histogram

AIR Show displays a histogram for the currently displayed image layer or channel just above the list of image layers. For 3-channel layers the histogram shows the luminance distribution.

Status Bar

The status bar at the bottom of the AIR Show window displays the following information about the active image (from left to right):

- Elapsed time
- Image resolution and data format (8-bit, 16-bit, or floating point)
- Zoom level
- Memory usage: while a rendering is in progress, memory in use is the amount of memory used by the rendering process. Once a render has completed, memory in use reports the memory used to store the image.
- Ray tracing object retention indicator. This block contains letters indicating if any primitives have been stored for ray tracing - S for objects visible to shadow rays, R for objects visible to reflections, and I for objects visible to indirect rays. If you are using a particular ray tracing feature, be sure the corresponding letter is present, indicating that objects are available for interesection with the corresponding ray type. Conversely, if you are not using ray tracing features, make sure than no letters appear, ensuring that AIR will be able to dispose of primitives as soon as they are processed by the scanline renderer.

23.3.2 View Navigation

Zooming

Click an image with the left mouse button to zoom in and with the right mouse button to zoom out.

Panning

Scroll bars appear if the image is larger than the window. The image can be scrolled using the scroll bars, arrow keys, or by moving the mouse while holding down the middle mouse button.

Syncing Views

If the Sync Views option is checked in the Options menu, AIR Show will synchronize panning and zooming across images with the same width and height. With Sync Views enabled, you can pan and zoom while playing an animation.

Comparing Images

AIR Show provides a couple mechanisms for comparing images.

The jump function can be used to quickly toggle between two images. Click the Jump button in the toolbar or press J to jump from the current image to the previously selected image.

The current image and previously selected image can also be compared by wiping between the two with the mouse. Hold down the CTRL key and left click and drag from an edge of the view window to reveal a section of the previous image next to the current image.

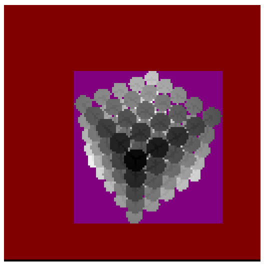
Viewing Pixel Values

Hold down the SHIFT key while mousing over an image to see the image channel values at the current pixel, listed in the upper right region of the AIR Show window.

23.3.3 Depth Map Display

AIR Show has a special display mode for depth maps designed to allow a shadow map to be previewed by rendering to AIR Show.

Here's a sample depth map display:



AIR Show performs the following image manipulations:

- The bounding rectangle of the non-empty pixels is computed, and the region outside the used rectangle is colored red.
- Empty pixels within the bounding rectangle are colored purple.
- Depth values for non-empty pixels are displayed using a grey-scale
- The status bar reports the percentage of the total map that is inside the bounding rectangle of the non-empty pixels. Low percentages indicate that large sections of the shadow map are unused, and that the user should consider refocusing the light.
- A warning is issued in the status bar if any pixels along the edges of the image are non-empty.

23.3.4 Rendering to a Remote Machine

(Not available in the demo version of AIR.)

AIR Show and the AIR `framebuffer` driver enable the output from a rendering process on one machine to appear in an AIR Show window on another machine. The destination machine for framebuffer data can be set with the `-dhost` AIR command line option

```
air -dhost 192.168.1.2 myfile.rib
```

or with a "host" parameter passed to the framebuffer driver in a RIB file:

```
  Display "remote.tif" "framebuffer" "rgb"
  "string host" ["192.168.1.2"]
```

A `host` parameter in a `Display` call overrides any `-dhost` command line option. The host parameter should be the IP address of the host machine in dot-3 notation or a name for a DNS lookup.

AIR Show must already be running on the remote machine.

23.3.5 Framebuffer Parameters

The AIR `framebuffer` driver accepts several optional parameters that are passed on to AIR Show:

`"float gamma" [`*`n`*`]`

Sets the gamma correction value for the displayed image. If provided this parameter overrides the default gamma in AIR Show. If omitted, the default gamma in AIR Show is used for the image.

`"string host" ["`*`ipaddress`*`"]`

Gives the IP address of the machine to which the image data is sent. If omitted, data is sent to the local machine. If data is sent to a remote machine, AIR Show must already be running on that machine. The IP address can be in dot-3 notation (e.g., 192.168.1.1) or a name for a DNS lookup.

`"float share" [0]`

When set to 1, AIR Show will attempt to merge output with that from other processes writing to the same image.

`"integer port" [47349]`

Gives the TCP port number to use when connecting to an external display program such as AIR Show. The default port number used by AIR Show is 47349.

23.3.6 Command Line Options

The following options may be included on the command line when starting AIR Show:

`-res` *`width <height>`*

Sets the default width and height of the client area of the AIR Show window. This is the maximum image size that can be displayed without scroll bars.

`-pos` *`left top`*

Sets the position of the upper left corner of the display window on the screen.

`-g` *`gamma`*

Sets the default gamma correction applied to images for display. This options overrides a gamma defined with the optional `AIRSHOWGAMMA` environment variable.

`-fps` *`framespersecond`*

Sets the default frames per second for animation playback. This option overrides the frames per second defined with the optional `AIRSHOWFPS` environment variable.

Loading Images at Startup

AIR Show also accepts a list of file names or a wildcard pattern for pictures to load when started from the command line.

23.3.7 Menus

23.3.7.1 File

Open... (Ctrl-O)

Open a TIFF file saved by ***AIR***. AIR Show can only read uncompressed TIFF files.

Open Sequence...

Open a numbered sequence of image files.

Save... (Ctrl-S)

Displays a file dialog for choosing a file name and saves the current image. The file name extension determines the file format.

Save All...

Allows all images to be saved at once. A dialog is displayed for choosing an output directory and display driver. All images are saved to the same directory with the same driver. Each image should have a different base name.

Save Sequence...

Saves all images as a numbered sequence.

Close (Ctrl-W)

Closes the current image window.

Close Previous

Closes all images prior to the current image in the image list.

Close All

Closes all image windows.

23.3.7.2 Edit

Copy Crop Coordinates

Copies the crop coordinates for the current selection rectangle as a string. The string is formated as xmin xmax ymin ymax.

Copy Decal Projection

Copies the transformation matrix that will place a decal within the selection rectangle from the view of the image as a text string of 16 numbers. This allows the viewport to be used to position decals on an object.

Annotate...

Add a descriptioin to an image, which is stored by some file formats such as TIFF.

Copy Image (Windows only)

Copies the current image (as displayed) to the Windows clipboard.

23.3.7.3 View

Refresh

Redraws the display window.

Zoom In

Zoom in on an image.

Zoom Out

Zoom out of an image.

Zoom Normal

Restore image view to the default in which one pixel on screen represents one pixel of the image.

Gain

Displays a dialog for setting the gain or intensity multiplier for an image.

Gamma

Displays a dialog for setting the gamma correction factor for the current image.

Pixel Zoom

Opens a separate window that displays the numeric values of the pixel under the mouse pointer

Fit Window to Image

Resizes the AIR Show window to display the current image without scroll bars or margins.

23.3.7.4 Animate

Frame Rate...

Displays a dialog for setting the playback rate for animation in frames per second.

Play Forward [3]

Play images as an animation once.

Play Backward [#]

Show images in reverse order as an animation once.

Cycle Forward [4]

Cycle images in forward order until stopped by the space bar or stop button.

CycleBackward [$]

Cycle images in backward order until stopped by the space bar or stop button.

Pong [5]

Show images in forward and reverse order alternately until stopped by the space bar or stop button.

Sort Frames

Puts images in alphabetical order based on image name.

Move Frame

Use the subitems to change the position of the current image in the list of images.

23.3.7.5 Sound (Windows only)

Select Sound Clip

Select a WAV file to play with an animation.

Use Sound Clip

Play sound clip when animation is played

23.3.7.6 Options

Always on Top (Windows only)

If checked the AIR Show window remains on top of all other windows on the screen.

Sync Views

Synchronize pan and zoom for same-sized images.

Show Toolbar

Display toolbar.

Auto Restore Size

When checked a minimized AIR Show window will restore to normal size when a new rendering starts. Uncheck this option to allow the window to stay minimized while rendering a sequence of frames.

Default Gamma...

Sets the default gamma correction applied to images.

23.3.8 Keyboard Shortcuts

Channels

r, R	red channel
g, G	green channel
b, B	blue channel
a, A	alpha channel
c, C	RGB channels
[	Decrement channel
]	Increment channel

Animation

1	Previous image
2	Next image
!	First image
@	Last image
3	Play animation once
#	Play animation in reverse once
4	Cycle animation
$	Cycle animation in reverse
5	Animate frames in pong mode
space bar	Stop animation
7	Swap with previous frame
8	Swap with next frame
&	Make first frame
*	Make last frame

Miscellaneous

Ctrl-F	fit window to image size
J	jump to last image viewed

23.4 MakeLIF

MakeLIF is a command-line utility that generates shader help files. MakeLIF can generate help files in one of two formats: .LIF shader information files used by the RhinoMan, PaxRendus, and Liquid plugins, and dialog scripts for use with Houdini.

Usage

```
makelif [options] source

Options:
   -o outputdir   output directory for created files
   -ds            create Houdini dialog scripts
   -spdl          create SPDL files for Softimage

   -surf          surface shaders only
```

```
    -disp          displacement shaders only
    -light         light shaders only
    -vol           volume shaders only
```

MakeLIF reads compiled AIR shader (`.slb`) files and VShade source (`.vsl`) files. The source parameter may be a wildcard pattern.

Creating Houdini Dialog Scripts

When a Houdini dialog script is created, makelif prints a line to the console for adding the dialog to Houdini. To make the dialog available in Houdini, add the line to the

```
  $AIRHOME/houdini/shop/SHOPsurface
```

file for surface shaders (or to the SHOPlight, SHOPdisplace, or SHOPfog files for light, displacement, and volume shaders respectively). After modifying the SHOP file you will need to re-start Houdini or reload the dialogs in Houdini with `dsreload` before the dialog will appear.

Technical note:

MakeLIF uses the `dictionary.lif` file in `$AIRHOME/viztools` as well as some builtin heuristics to produce a help file for a shader.

23.5 Shading Compiler (shaded)

Before shaders can be used, they must be compiled using the shading compiler `shaded`. If you are using a plugin, the plugin may automatically compile shaders for you. Otherwise, you can use `shaded` from a command prompt. The command syntax is

```
  shaded <options> filename
```

where *filename* is the name of the shader source file.

The shading compiler accepts the following options:

`-D`*`sym`*
  define preprocessor symbol sym

`-D`*`sym=val`*
  define preprocessor symbol sym as val

`-I`*`path`*
  specify a path to search for included files

`-line`
  print the source code line when an error occurs

`-o` *`filename`*
  specify an alternate output file name. The default name for the compiled shader is the name of the shader in the source code with the extension `.slb`

`-noopt`
  No compile time optimization.

`-norto`

No run-time optimization. Air normally optimizes shader code for each instance of a shader.

`-cpp`
Use an external pre-processor instead of the built-in one. The pre-processor must reside somewhere in the search path for executables.

`-fover`
Enable support for function overloading (multiple functions with the same name but different arguments and/or return type). Note that nested functions are not currently supported when function overloading is enabled.

Default Search Path for Include Files

A default search path for include files can be specified in a SHADECH environment variable.

Pre-processor Definitions

AIR pre-defines two identifiers - `AIR` and `RAYTRACE`. `RAYTRACE` is intended for conditionally compiling features that depend on ray tracing and/or the ray tracing extensions supported by AIR. E.g.,

```
#ifdef RAYTRACE

  fulltrace(P,R,Crefl,hitdist,Phit,Nhit,Pmiss,Rmiss);

#endif
```

Some shaders conditionally include ray-tracing features based on the symbol BMRT. To compile such shaders for AIR, use the `-Dsym` option. For example:

```
air -DBMRT glass.sl
```

23.6 Shader Info (slbtell)

Usage:

```
  slbtell [-obi] shader.slb
```

SLBTell prints a list of all parameters for a compiled shader. Using redirection, the output can be saved and included in a RIB file.

If the `-o` option is specified, the output is in a format similar to that used by the BMRT program `slctell`.

If the `-i` option is included, any parameters of type `integer` will be declared as such. By default, integer parameters are declared as `float` for backwards compatibility.

If the `-b` option is included, any parameters of type `boolean` will be declared as such. By default boolean parameters are declared as `float` for backwards compatibility.

23.7 Texture Converter (mktex)

Mktex is the command-line texture conversion utiltiy provided with AIR. The primary purpose of mktex is to convert source images to the special formats AIR uses for texture, environment, and shadow maps. mktex is also useful as a general purpose utility for converting between a variety of image and data formats.

mktex is a command-line utility that is normally invoked from a command shell. You can see a complete list of command-line options by entering mktex with no arguments at a command shell.

The mktexui program provides a user-friendly interface for mktex.

Topics:

23.7.1 Texture Maps

Create an AIR texture map from any image format recognized from AIR to an AIR texture map with:

```
mktex [options] infile(s) [outfile]
```

Although AIR can use image files directly as texture maps, creating a new texture file has several advantages:

- The wrapping behavior of the texture map can be specified.
- The new texture file is a mip-map, storing multiple pre-filtered copies of the texture at different resolutions. When accessing the texture AIR will use an appropriate resolution based on the region of the texture visible in the area being shaded, resulting in fewer aliasing artifacts.
- Texture files are tiled, allowing the renderer to selectively load only the portions of an image that are needed.
- Texture files are compressed to minimize disk space and bandwidth usage.
- AIR limits the runtime memory used by tiled texture maps to a user-defined maximum, dynamically loading and unloading tiles as needed.

If no output file is given, the name of the output file is derived from the input filename by changing the filename extension to `.tx`

Options

`-smode, -tmode, -mode (black|clamp|periodic)`

-smode, -tmode, and -mode set the wrapping mode for the s-axis, t-axis, and both axes, respectively.   Each is followed by one of the following possible modes:

`black`	return color (0,0,0) for texture access outside the range 0..1
`clamp`	clamp the coordinate value to the range 0..1
`periodic`	wrap the texture coordinate; e.g., use s=s-floor(s)

The default mode is black for both s and t coordinates.

`-size` *`width height`*

Resize the output image to width x height. When writing an AIR texture map, the output width and height should be a power of 2.

`-flipx`

Flip the image horizontally.

`-flipy`

Flip the image vertically

`-rot` *`angle`*

Rotate the image in 2 dimensions. Angle must be a multiple of 90 degrees.

`-8`

Output data as 8-bit unsigned integers

`-16`

Force output of data as 16-bit unsigned integers

`-float`

Output data as floating-point numbers.

`-tilesize n`

Use tiles of size n for mip-mapped images. The default size is 32 for texture maps and 64 for shadow maps. The tile size must be a power of 2 between 8 and 128 inclusive.

`-u`

Forces the creation of an uncompressed texture file.

`-d`

Send output to AIR Show. On Linux AIR Show must already be running.

`-lzw`

Enable LZW compression for TIFF images.

`-inc`

Convert the image incrementally, one row of pixels at a time. This option only works with image readers that support reading an image one row at a time, currently limited to TIFF, BMP, and PSD formats. This option can drastically reduce the amount of memory needed to convert large images.

`-fromsrgb, -tosrgb`

Convert the image from or to sRGB color space

Sample Use:

```
mktex -smode periodic -tmode clamp grid.tif grid.tx
```

Creates a texture file that is periodic in the first texture coordinate and clamps the second texture coordinate.

23.7.2 Shadow Maps

Make an AIR shadow map from one or more depth maps with

```
  mktex -shadow zfile1 (zfile2...) outfile
```

Up to 6 depth maps may be combined into a single shadow map. All depth maps should have the same dimensions. Multiple maps are useful for adding shadows to a source such as a point light that illuminates a wide range of directions. When accessing a shadow map with multiple depth maps, the renderer will use the first map whose viewing frustum contains the point being sampled. Maps whose regions adjoin should slightly overlap to prevent gaps.

Multiple depth maps can be used to make a cube-faced shadow map for a point light.

23.7.3 Latitude-Longitude Environment Maps

Create a lat-long environment map from an uncompressed TIFF file with

```
  mktex -envlatl TIFFfile [envfile]
```

If no output file is given, the name of the output file is derived from the input filename by changing the filename extension to `.env`

23.7.4 Cube-Face Environment Maps

Create a cube-face environment map from 6 image files with

```
  mktex -fov angle -envcube px nx py ny pz nz envfile
```

Each image contains the environment viewed from a point in one of the 6 orthogonal directions.

The optional -fov argument gives the angle in degrees of the perspective projection used to generate each face of the cube. Using an angle slightly larger than 90 degrees can help avoid artifacts along cube edges. If the -fov option is not present, the default angle is 90 degrees.

23.7.5 Mirror Balls and Angular Maps

Environment map images are often generated by photographing a reflective sphere. Such a "mirror ball" format image can be converted for rendering with

```
mktex -ball srcimage destimage
```

Another common format for probe images is the angular map, in which distance from the center is linearly related to angle (unlike a mirror ball image).

To convert an angular map to a lat-long map for rendering with Air, use

```
mktex -angular srcimage destimage
```

For image-based lighting a small image is usually sufficient. The output image size can be set using the -size option. E.g.,

```
mktex -angular -size 256 128 myprobe.tif myprobe.tx
```

mkex also supports 3D rotation for an angular or ball map, allowing you to orient the map properly for a given scene:

```
-rotx angle
-roty angle
-rotz angle
```

These options rotate the map in 3D dimensions around the X, Y, and Z axes respectively. The "up" axis is Z by default. For a modeling problem that uses Y as the up direction, rotate 90 degrees about the X axis. Rotation about the Z axis always rotates the scene about the vertical axis.

```
-blur amt
```

The blur option blurs the output image, which may reduce noise when a map is used for image-based lighting.

23.7.6 Compositing

New in AIR 9, mktex allows images to be created using simple but powerful command line compositing equations. The source images may differ in size, channel depth, or data type. mktex will automatically perform any necessary conversions.

Compositing Syntax and Examples

Compositing mode is invoked with the -comp command line option followed by an equation which can include source image file names, numeric constants, and operators. For example, a simple weighted sum of a diffuse and specular pass could be created with:

```
  mktex -comp result.tif = diffuse.tif mul 0.9 + specular.tif mul 1.1
```

Multichannel constants can be created as a comma-separated list enclosed in braces:

```
  mktex -comp greenish.png = {0.8.1,0.8} mul source.png
```

Equations can reference a subrange of channels in a multichannel image using brackets with a channel range after the image name:

```
  mktex -comp result.tif = m.tif[4-6] + 0.9 mul m.tif[7-9] + 1.1 *
  m.tif[10-12]
```

Braces can also be used to add channels to an image. E.g., to add an alpha channel to a composite of two 3-channel images:

```
  mktex -comp result.tif = {light0.tif + light1.tif, beauty.tif[3]}
```

or merge separate images into a single multichannel image:

```
  mktex -comp multi.tif = {diffuse.tif,specular.tif,reflect.tif}
```

A numbered sequence of images may be processed using the `-frames` option to specify a range of frames. mktex recognizes the same frame number expansion syntax as AIR - `#nf` where n is the minimum number of digits in the frame number, zero padded. E.g., the following command

```
  mktex -frames 9 11 result#4f.tif = diffuse#4f.tif + specular#4f.tif
```

would produce 3 images: `result0009.tif`, `result0010.tif`, and `result0011.tif`.

Compositing operators

A + B, A add B	addition
A - B, A sub B	subtraction
A * B, A mul B	multiplication
A / B, A div B	division
A min B	minimum of A and B
A max B	maximum of A and B
A dot B	dot product of A and B
byte A	convert A to unsigned 8-bit data
word A	convert A to unsigned 16-bit data
float A	convert A to float data
abs A	take absolute value of A
[*min*-*max*]	extract channels from preceding image
M ? A : B	blend between B (M=0) and A (M=1)
( )	group and order operations
{A,B}	append channels in B to channels in A

Note 1: Linux shells may expand * to a wildcard; mul should be used for multiplication.
Note 2: / used as a divide operator must be separated by white space

23.7.7 Stitching Images

Version 12 of mktex introduces a new stitching mode for joining a grid of small images into a single large image. The command syntax is:

```
mktex -stitch nx ny basename00.tif outimage.tx
```

where nx and ny give the number of subimages horizontally and vertically. All sub-images must have the same width, height, number of channels, and data type. Each sub-image should have a name like:

```
basenameXY.tif
```

where X (between 0 and nx-1) and Y (between 0 and ny-1) give the sub-image index. Image 0,0 is the top, left image. Stitching uses the new incremental conversion mode for handling large images. The input file format must be compatible with that mode.

23.8 Texture Converter User Interface (mktexui)

`mktexui` provides a graphical user interface for creating AIR texture maps from standard texture files. The single window contains the following controls:

Image Source

The source image file in any of the texture formats recognized by AIR. The source can be a single file, or a pattern for selecting several files. E.g.,

```
c:\temp\*.tif
```

would select all `.tif` files in the `c:\temp` directory.

Map Type

This control specifies the type of map to be created or the source format in the case of environment maps.

X Wrap Mode, Y Wrap Mode

For texture maps the wrap mode determines how texture queries outside the range 0..1 are handled.

`black`	return color (0,0,0) for texture access outside the range 0..1
`clamp`	clamp the coordinate value to the range 0..1
`periodic`	wrap the texture coordinate; e.g., use s=s-floor(s)

Transformations

The following transformations can be applied to an image during the translation process:

Flip X

Flip the image horizontally

Flip Y

Flip the image vertically

Rotate

Rotate the image in increments of 90 degrees.

Rotate 3D

For angular maps only, rotates the coordinate system for the environment map lookup.

Output Data Type

Specifies the data type of the output image: 8-bit, 16-bit, or float. By defaul the output image will have the same data format as the source image.

Output Size

Specfies the size of the output image.

Output Directory

The directory in which to place the created file.

View Source

Press this button to view the source image in AIR Show. On Linux AIR Show must already be running for this command to work.

Preview

Use the Preview button to view the converted image in AIR Show, with any transformations applied. On Linux AIR Show must already be running.

Make Maps

Press this button to create the texture map or maps. Standard texture maps and environment maps will have an extension of .tx. Shadow maps will have an extension of .shd.

23.9 Vortex

Vortex is a distributed rendering manager for AIR. Vortex allows multiple instances of AIR running on different machines to all work on the same image.

Vortex has a couple basic requirements and restrictions

- All files used in a scene should reside on a networked drive visible to all worker machines.
- The scene file should have only one pass, i.e., one `World` block, which may produce multiple output images.

Usage

Vortex is designed to be a transparent substitute for AIR: you can use Vortex to render just as you do AIR:

```
vortex myscene.rib
```

Vortex manages the rendering process by requesting pieces of the image from each worker task and merging the output from all workers to produces the final image(s).

Vortex requires one or more instances of AIR running as workers connect to the Vortex host. Vortex provides 3 mechanisms for launching worker tasks:

1. Automatic launch of workers on the local machine.
2. Broadcast message requesting workers from machines running VoluntAIR.
3. Execution of a user-defined command that, e.g., starts a batch job on a render farm or communicates with a centralized render manager.

These mechanisms may be used in any combination that is convenient.

Local Workers

To start an instance of AIR as a worker for Vortex manually use:

```
air -vhost ip:port
```

where ip is the IP address of the machine running vortex. The port is optional unless the `-port` option was used with Vortex.

By default Vortex will launch a single instance of AIR on the local machine. That behavior can be overridden with the -local command line option:

```
  vortex -local 2 myscene.rib
```

An argument of 0 will prevent any local workers from being started. The default number of workers can be set with a `VORTEX_NLOCAL` environment variable.

The exact command executed can be customized with a `VORTEX_LOCALCMD` environment variable. The default command is

```
  air -vhost %HOST%
```

Vortex replaces `%HOST%` in the command string with the IP address and port of the Vortex instance.

VoluntAIR (Windows only)

VoluntAIR is a small program that enables an idle machine to provide workers for Vortex instances on demand. To start VoluntAIR on a machine, simply enter voluntair at a command prompt. VoluntAIR waits for a message from a Vortex instance and starts an instance of AIR when it receives a request. The exact command executed can be customized with a `VOLUNTAIR_CMD` environment variable. The default command is

```
  air -vhost %HOST%
```

Batch Command

f a `VORTEX_BATCH` environment variable is defined, Vortex will execute the command given in the environment variable as a separate process. The command can be used to submit a batch job to a render farm or to request workers from a central render manager. Vortex performs the following substitutions on the command string:

`%HOST%`	ip address and port number of vortex instance
`%NWORKERS%`	number of requested workers

Number of Workers

The maximum number of workers used by a particular Vortex instance can be set with the -w command line option. The default maximum is 5 workers. Vortex will reject connection attempts by additional workers once the maximum has been reached.

Command Line Options

`-local` *n*

Sets the number of local ***AIR*** instances to launch.

`-novol`

Don't broadcast a request for VoluntAIR workers.

`-port` *n*

Set the port number on which to accept connection requests from workers.

`-w` *n*

Sets the maximum number of workers to use for a job.

`-bs` *`width (height)`*

Sets the size of the buckets used for rendering.

`-crop` *`left right top bottom`*

Restricts rendering to a subrectangle.

`-d`

This option forces output to a window. It overrides any Display call in the RIB stream.

`-reflect`

Shorthand for `Attribute "visibility" "reflection" [1]`

`-res` *`width height`*

Sets the resolution of the rendered image. This option overrides any subsequent `Format` call.

`-samples` *`xsamples`* `(`*`ysamples`*`)`

Sets the number of samples per pixel. This option overrides any subsequent `PixelSamples` call.

`-shadows`

Shorthand for `Attribute "visibility" "shadow" [1]`

24 Plugins and Companions

Plugins compatible with Air are available for many major modeling and animation programs. Other programs provide builtin support for rendering with Air.

Here is a partial list of compatible plugins for Air:

SketchAir *(new!)*

Our new plugin for SketchUp, currently in development.

Air Stream

Our plugin for Maya available as a separate download from the SiTex Graphics web site.

RhinoAir

Our own plugin for Rhino 4 and 5, available as a separate download from the SiTex Graphics web site.

CineMan

A new plugin for Cinema4D

Companion Products

Houdini

`www.sidefx.com`

SideFx's advanced 3D products for visual effects offers good support for rendering with Air. See the Houdini section of this manual for more information.

Massive

`www.massivesoftware.com`

Massive is the crowd simulation program from Massive Software used to generate crowds for the Lord of the Rings movies.

"Massive is the premier 3D animation system used for generating crowd-related visual effects and character animation, based on artificial life technology."

The Massive section of this guide has an extensive list of tips and tools for rendering Massive crowds with Air.

CityEngine

`www.procedural.com`

Procedural city construction from Procedural, Inc.

RealFlow Rendering Toolkit

`www.realflow.com`

Rendering toolkit for the RealFlow fluid and dynamics simulation software.

Temerity Pipeline

"Temerity Pipeline is the industry's first complete production control application designed for the film, television and game industries. Pipeline increases productivity and studio profitability by providing seamless control over distributed processing of jobs, revisions and storage management. Studio pipelines are managed through one environment allowing artists to spend more being creative instead of dealing with production issues.

Temerity believes that the development of in-house production solutions and pipeline methodologies can be more costly, difficult to maintain and less efficient than an integrated solution like Pipeline. The tight integration of Pipeline's asset management, revision control and distributed execution queue provides both efficiency and new functionality which cannot be achieved through gluing ad hoc combinations of existing production tools. Pipeline offers a solid yet highly flexible infrastructure suitable for any studio size or type of production."

Temerity Pipeline has full support for Air, BakeAir and their associated texture and shader utilities.

DarkTree

DarkTree is a visual shader creation tool from Darkling Simulations:

"DarkTree 2.0 is an advanced procedural shader authoring tool. Its visual flow-based editor lets you interactively create photo-realistic procedural materials, surface shaders, and animated effects. DarkTree 2.0 includes 100 procedural components that can be combined to generate almost any texture or surface effect you need."

An evaluation version of DarkTree is available from the Darkling Simulations web site. See also the section on using DarkTree shaders with AIR.

HDR Shop

"HDR Shop is an interactive graphical user interface image processing and manipulation system designed to view and manipulate High-Dynamic Range images."

HDR Shop is available from Paul Debevec's web site.

Vtexture

Alex Segal's DSO shadeop for using vector file formats as textures.

25 Resources

The RenderMan® Interface

AIR is compatible with the RenderMan® standard for the description of 3D scenes. To learn more about RenderMan®, consult the resources below:

Online documentation

The RenderMan® Interface Specification 3.2

Books

The RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics by Steve Upstill.

10 years old but still a good introduction to RenderMan®.

Advanced RenderMan: Creating CGI for Motion Pictures by Anthony A. Apodaca and Larry Gritz.

The latest reference for RenderMan®. Contains some introductory material as background to the advanced techniques discussed. Also has a good section on digital cinematography.

Essential RenderMan Fast by Ian Stephenson

This new book is specifically designed as an introductory text, covering the nuts and bolts of rendering and writing shaders. Highly recommended.

Web Site

The RenderMan Repository (www.renderman.org)

Many resources including shaders, RIBs, SIGGRAPH course notes, and links to RenderMan®-related programs.

Newsgroup

comp.graphics.rendering.renderman

Newsgroup FAQ

comp.graphics.rendering.renderman.FAQ

Creating Computer Graphics Imagery

Digital Lighting & Rendering by Jeremy Birn

Good general discussion of techniques for composing, lighting, and shading 3D scenes. Not application-specific.

26 License Agreement

AIR - Advanced Image Rendering Software and the Visual Shading and Lighting Toolkit (SOFTWARE PRODUCT) is owned by SiTex Graphics and is protected by United States copyright laws and international treaty provisions. Therefore, you must treat the SOFTWARE PRODUCT like any other copyrighted material (e.g. book or musical recording). You may make one copy of the SOFTWARE PRODUCT for archival purposes. You may not redistribute the SOFTWARE PRODUCT or any of its accompanying materials. You may not reverse engineer, decompile, or disassemble the SOFTWARE PRODUCT.

SiTex Graphics grants you the non-exclusive right to use one copy of the SOFTWARE PRODUCT per license as long as you comply with this agreement. Each license of the SOFTWARE PRODUCT may be used on only one computer at one time.

The SOFTWARE PRODUCT and accompanying materials are provided on an "as-is" basis. SiTex Graphics makes no warranty, either expressed or implied, including but not limited to warranties of merchantability or fitness for a particular purpose, with regard to the SOFTWARE PRODUCT and the accompanying materials.

In no event shall SiTex Graphics be liable for any damages whatsoever (including without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use of or inability to use the SOFTWARE PRODUCT. Because some states do not allow the exclusion or limitation of liability for consequential damages or incidental damages, the above limitation may not apply to you.

This agreement is governed by the laws of the State of Texas, USA.

By installing the supplied SOFTWARE PRODUCT, you acknowledge that you have read, understood, and agreed to these terms.

27 Copyrights, Trademarks, and Credits

Copyrights and Trademarks

AIR and the Visual Shading and Lighting Tool Kit software and documentation are Copyright 2000-2004 SiTex Graphics, Inc. All rights reserved.

The RenderMan® Interface Procedures and RIB Protocol are Copyright 1988, 1989, Pixar. All rights reserved. RenderMan® is a registered trademark of Pixar.

Rhinoceros (Rhino) is Copyright 1993-00 Robert T. McNeel & Associates. All rights reserved.

OpenEXR software

Copyright (c) 2002, Industrial Light & Magic, a division of Lucas Digital Ltd. LLC All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Industrial Light & Magic nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

LIBTIFF

Copyright (c) 1988-1997 Sam Leffler
Copyright (c) 1991-1997 Silicon Graphics, Inc.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that (i) the above copyright notices and this permission notice appear in all copies of the software and related documentation, and (ii) the names of Sam Leffler and Silicon Graphics may not be used in any advertising or publicity relating to the software without the specific, prior written permission of Sam Leffler and Silicon Graphics.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
OF THIS SOFTWARE.

PTEX software

Copyright 2009 Disney Enterprises, Inc. All rights reserved

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
* The names "Disney", "Walt Disney Pictures", "Walt Disney Animation Studios" or the names of its contributors may NOT be used to endorse or promote products derived from this software without specific prior written permission from Walt Disney Pictures.

Disclaimer: THIS SOFTWARE IS PROVIDED BY WALT DISNEY PICTURES AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT AND TITLE ARE DISCLAIMED. IN NO EVENT SHALL WALT DISNEY PICTURES, THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND BASED ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Credits

The JPEG display driver is based in part on the work of the Independent JPEG Group.

The PNG display driver uses libpng.

28 History

Change log.

AIR 12 (August 2012)
AIR 11 (July 2011)

AIR 10 (July 2010)
AIR 9 (September 2009)
AIR 8 (December 2008)
AIR 7 (March 2008)

AIR 6 (July 2007)
AIR 5 (December 2006)
AIR 4.1 (July 2006)
AIR 4 (November 2005)

AIR 3.1 (June 2005)
AIR 3 (February 2005)
AIR 2.9 (September 2004)
AIR 2.8 (June 2004)

28.1 AIR 12.0

What's new in Air 12. (For information on prior releases, see the History section.)

- First official release of the Air Stream plug-in for Maya (available as a free download from the SiTex Graphics web site).

- 64-bit build for Windows: the Windows distribution of Air now includes a 64-bit version of Air named `air64.exe`

- New ray tracing primary hider

- Occlusion
  - New `"axis"` parameter for the occlusion() function takes a vector value. If non-zero, the vector defines the "up" direction for hemisphere of incoming light. Any occlusion rays pointing down with respect to the up direction will be considered occluded by a virtual ground plane, and not traced at all.
  - Occlusion optimize normal option is now disabled by default when an occlusion cache is not used, which produces more detail for bump-mapped surfaces
  - Important fix for occlusion to prevent recursive calls for surfaces with shader-computed transmission.
  - New envlight parameter samplelikediffuse to enable sampling the environment using a cosine distribution. When the envlight envname parameter is set to "environment", the environment will be sampled using a lambert brdf, producing a result that is much closer to the result from indirect lighting and the result from using the envlight's "sampleenvironment" switch.
  - Added a shadowgroups parameter to the environment shader to restrict occlusion testing to the members of the specified groups.

- Indirect lighting
  - New option to automatically build an in-memory cache for environment queries from indirect rays:

    ```
    Option "indirect" "environmentcache" [1]
    ```

    Using the cache produces much smoother results and is usually at least 2X faster for environment queries than rendering without. Highly recommended.

  - Indirect diffuse environment sampling is now much cleaner
  - Indirect diffuse results are less noisy at higher bounce levels

- Ptex (per-face texture mapping)
  - New attribute to set the name of the Ptex texture coordinates generated by Air:

    ```
    Attribute "render" "string ptexcoordinates" "st"
    ```

- Shaders
  - New cardlight area light shader emulates the effect of using a constant-shaded card to add light with indirect diffuse illumination.
  - New OceanWaves displacement shader
  - New Emitter surface shader can be used to create the effect of an area light with indirect diffuse illumination
  - New surface shader ShowPosition
  - New OceanSurface and OceanSurfaceWithFoam shaders for oceans
  - New BakedSurface surface shader for rendering with baked illumination maps
  - New genQueryBakedMap generic shader for querying baked maps
  - New genUserAttributes generic shader for querying color or float user attributes
  - New genUV generic shader emits standard u, v coordinates (for use in Maya)
  - New genTextureUV generic shader queries a texture map using u,v coordinates
  - New instMultipleArchives instancer shader for instancing multiple archives
  - The portallight shader has been updated to sample the global environment by default
  - All indirect and envlight shaders have been updated to include indirect in the __category

  parameter.
  - Updated VInstanceArchive shader with parameters for rotation and for converting color or float prim vars to user attributes (which can be queried by a shader attached to an instanced object). A new ArchiveIsSimple parameter can be used to disable use of a temporary inline archive for instancing (which doesn't work with archives created by RhinoAir that contain Rhino blocks).
  - The VCarPaint shader has new controls for metallic flakes.
  - The sunlight and envPhysicalSky shaders have new parameters to explicitly set the sun position.
  - The VBackdrop imager shader now supports an image sequence.
  - The VArrayOnSurface instancer shader has new parameters to vary scale and rotation for each item.

- Shading Language
  - New method of transferring primitive variable data to shaders allows prim var data to be used with layered and networked shaders. The new method is also slightly faster.
  - Area lights can now query the total surface area of the area light geo with

    ```
    float surfarea;
    attribute("light:area",surfarea);
    ```

  - New function str_replace_last_number can be used to replace the last number in a string with a new value
  - The `gather()` function can now be used to query point clouds

- Instancers and instancing
  - New data structure greatly accelerates sampling a primitive using standard texture coordinates
  - Instancers can now be evaluated by multiple threads simultaneously
  - Instancers can reference the object space coordinate system of the base object
  - Instancers now handle scale for width and constantwidth prim var data
  - Instancers can reference user attributes attached to the base object
  - Objects created by an instancer now inherit any user attributes attached to the instancer's base object
  - Modified instancer bound computation to look for particle/curve width data
  - Multiple instancer shaders may now be assigned to an object

- Command line switches
  - The `-frames` command line option no longer uses the frame range to filter rib based on the frame number passed to FrameBegin. This change allows a numbered sequence of files to be rendered regardless of the frame numbers set in the rib. In particular, this feature now works with the shadow map ribs generated by Air Stream.
  - The -surf command line option has been extended to accept a comma-separated list of parameter values for the override surface shader. E.g.,

    ```
    air -surf massive_occlusionpass,samples=1 myscene.rib
    ```

    Float and string parameter types are supported.

- Shadows
  - Traced shadows with transmissionhitmode "primitive" now work for volume primitives with point-based data and no per-vertex opacity
  - Traced shadows with transmissionhitmode "primitive" now use prim var opacity (`Os`) data when available
  - Smooth interpolation for deep shadow maps can now be specified when the shadow map is created by including a "float lerp" parameter with value `1` in a `Display` call using the shadow display driver.

- Points
  - The patch point type accepts a "patchangle" parameter to set a rotation angle (in degrees) for the patch rectangle (for Maya sprite translation).

- Curves
  - Polyline and ribbon curves now compute a primitive-based dPdv value (to match TweakAir)
  - Tube curves now compute u,v when required
  - Curve primitives used as area lights may now use width or constantwidth to specify a radius value for treating the curve as a tube

- Outlines
  - Vector outline export in DXF and AI file formats

- Textures
  - Faster texture lookups when all texture derivatives are close to 0.
  - The Windows TIFF imager reader has been updated to libtiff 4.0.1 which includes support for BIGTIFF files (which can be larger than 4GB).

- TweakAir
  - Networked shaders can be tweaked by including a "layername" parameter in the IrTweak* command specifying which shader to tweak
  - Standard texture coordinates (s,t) are initialized for imager shaders.
  - The displayed image may be in 16-bit or float precision.
  - When the camera changes, TweakAir now casts rays of type "camera" to compute new visible surfaces, allowing objects that are invisible to reflection rays to appear in the IPR view.
  - Tweaking an environment shader now forces re-evaluation of any light shaders that sample the environment.
  - A new per-pixel occlusion cache speeds up re-rendering with tweaking the envlight shader.
  - Now displays a rough view as the initial deep buffer is filled.
  - dPdv is generated when needed for polyline and ribbon curves
  - u and v values are now cached when required.
  - Fix to correctly save user prim vars when re-tracing the camera view.
  - TweakAir detects if a tweaked camera position is really different from the current camera transform before re-tracing the view port (fixes an issue where Rhino now sends a camera change event even when the camera has not moved)

- Air Show
  - A new log window displays any error messages generated by Air during a rendering
  - Image data saved to OpenEXR files is automatically converted to float precision if necessary.
  - Histogram now ignores empty pixels in an rgba image

- mktex (texture conversion tool)
  - Ptex is supported as an output format for regular textures and all environment texture modes.
  - New incremental conversion mode (-inc) converts an image one row at a time to reduce memory used when converting large images.
  - New stitching option to create a single large image from a grid of sub-images.
  - New -lzw option to enable LZW compression for TIFF images

- Miscellaneous
  - Increased max threads allowed to 64
  - Under Windows the default shader search path now includes `$HOMEPATH/SiTex/shaders`

- Deprecated features
  - 32-bit Linux version of Air

  - The parallel shading attribute is no longer supported.

- Fixes:
  - Shading compiler fix to handle #undef preprocessor directive
  - Bug in ptex coordinate generation for polygon meshes that are not all triangles or all quads (mangled normals)
  - Sub-pixel edge mask handling of exactly horizontal edge
  - Bug that prevented light output variables from being correctly initialized when queried from a networked surface shader
  - Modified handling of traced reflections with only 1 sample and some blur to produce smoother (non-noisy) results for first reflections
  - Occlusion and indirect prepasses now work with fragment shading
  - Display driver open and close calls are now single-threaded (to work properly with libpng and libjpeg)
  - Fix for subtle bug in surface shader handling when rendering a shadow map: surface shader is no longer optimized away when creating an inline archive with transmissionhitmode other than shader
  - Fix for bug in prim var handling in ray tracer for constant and uniform data
  - Fixed bug in instancer handling of normals
  - Fixed bug in display subset mask computation for multilayer images with elements of arrays
  - Tube curve generation when both s,t and u,v prim vars requested.
  - Fix for shading language `spline("b-spline",...)` function

28.2 AIR 11.0

- Layered Shaders and Shader Networks

- New generic shader type

- Environment shaders for shading rays that miss all objects in the scene

- Physical sun and sky shaders

- Per-Face Texture Mapping: support for the open-source PTEX library

- Deep image hider

- Ray Tracing
  - New option to optimize memory for ray-traced objects

- Indirect diffuse lighting
  - Much more efficient handling of `indirectdiffuse()` evaluation with no caching in reflections (the number of indirect samples is now reduced based on the number of reflection samples)
  - `indirectdiffuse()` accepts "maxerror","maxhitdist","adaptive", and "maxpixeldist" parameters
  - New massive_indirect light shader
  - Indirect shading modes `constant` and `diffuse` now use prim var Cs data (of any class) if it is present. This allows a mesh with diffuse illumination baked as Cs to be used for fast indirect illumination with the `constant` indirect shade mode.

- Subsurface Scattering
  - New option to automatically generate point set handles

- Shading

- Micropolygon shading handles uniform and constant primitive data
- Micropolygon shading `texture()` evaluation is now smoother and more consistent
- Mesh displacement now works on meshes without vertex normals

- Lighting
  - Added `adaptivesampling` and rotation parameters to massive_envlight
  - Primitive variable data can now "write" to light shader parameters. The target light shader parameter must be class "varying", and it must not be an output variable.
  - Spectral profiles for light sources may now be specified using a color temperature for the spectral file name value. E.g.,

```
LightSource "distantlight" 1 "string lightcolor_spd" "2400" # sunset
```

- Occlusion
  - New option to automatically sample the environment for unoccluded rays
  - Modified the error calculation to work better with small max hit distances

- Textures
  - New algorithm greatly improves the efficiency of the texture cache in some conditions

- Geometric Representation
  - New attribute to control tessellation based on maximum edge length
  - Changed tessellation of `PointsPolygons` primitives with more than 4 sides (for compatibility with PTEX)

- Shading Language Extensions
  - New `boolean` and `integer` variable types
  - Additional `rayinfo()` queries
  - New mode added for `blinn()` rolloff: if rolloff<0, it's treated as the index of refraction of the material (for compatibility with Softimage)
  - New built-in `cooktorrance()` BRDF
  - New `dictionary()` DSO shadeop

- Imager shaders
  - Imager shaders may now have arbitrary output variables. Each imager AOV that is being saved will be initialized to the current pixel value for that AOV (the filtered result of the normal rendering process).
  - The standard Air imager shaders now provide a `__background` output variable with the imager shader contribution to the final image
  - Imager shaders now show up in images with multiple output channels
  - Imager shaders may be defined in inline archives
  - Imager shaders can query light sources with `illuminance()`. The imager shader is "illuminated" by all lights.
  - Imager shaders initialize (s,t) coordinates so the unit square covers the output image
  - New LensFlareImager shader

- Output
  - A `Display` declaration may now specify a different subset (group) for each output variable by including a list of groups separated by commas in the optional `"subset"` parameter (instead of a single group)
  - New display channel `__CPUtime` gives the time (in minutes) to render each bucket
  - User-defined attributes can be saved to output images:

```
Declare "objectid" "varying float"
Display "myuserdata.tif" "file" "user:objectid"
```

- An object's toon id can be included in an output image with:

  ```
  Display "tooninfo.tif" "file" "toon:id"
  ```

- Geometry Export
  - Export to PLY format now includes vertex color values when available
  - Geometry export now issues an INFO message for each file created

- Level of Detail
  - Improved automatic LOD for polygon meshes: custom user data is now handled properly, and edges and overall shape are better preserved

- Procedurals
  - A RunProgram procedural declaration may omit the bounding box, in which case the procedural is evaluated immediately.
  - The option to set the number of RunProgram threads has been converted to an attribute

- RIB processing
  - Air, BakeAir, and TweakAir now define a user option with the renderer name
  - New directory mapping capability for translating path names depending on the rendering machine's environment
  - Handles for `ObjectBegin` and `ObjectInstance` can be strings instead of numbers
  - RIB filters for `Procedural` now trap `DelayedReadArchive` procedurals
  - Added limited support for `ScopedCoordinateSystem` (converted to `CoordinateSystem`)
  - Archives can be found in the resource search path
  - Reorganized the processing of the RIB `Option` call

- Shaders
  - Fix for VPlastic to avoid tracing reflections twice
  - Updated the `bumpmap_polys` displacement shader (used by Massive) to scale the bump amplitude so it is relative to a local coordinate system (so scaling the agents also affects the bump amplitude)
  - Updated MassiveAgents instancer shader

- Messages
  - Missing shaders are only reported once
  - Missing textures are reported once (instead of once per thread)
  - Error message for a missing texture reports the shader containing the texture call
  - Errors in the RIB commands issued by an instancer shader now reference the instancer shader name

- The old cilver license system has been de-supported

- Linux: a single set of multithreaded binaries is now distributed for air and bakeair. Separate airmt, airst, and bakeairmt binaries are no longer included.

- Air Demo: the number of rendering threads is no longer restricted to 2

- Shading compiler (`shaded`):
  - New `-line` command line option will print the line of code where an error occurs
  - Support for prman-style struct field reference (eg, `a->x` instead of `a.x`)
  - Fix for dot product and color values
  - Missing quote error message reports the line with the unmatched quote
  - Fix for handling of undeclared identifier in `setxcomp()` etc.

  - The option to compile a shader to a DLL or shared object has been de-supported
  - Support for longer parameter names (up to 63 characters)
  - Fix for `ctransform()` code generation

- BakeAir
  - New mode for baking meshes (instead of texture maps)

- TweakAir
  - `IrNewEnvironment` and `IrTweakEnvironment` commands for updating environment shaders

- Air Show 6
  - New tone mapping display option
  - Fix for gain control applied to 16-bit image with sRGB
  - sRGB display mode is now enabled by default

- Vshade 4.0: see the separate Vshade user manual for what's new

- Vortex
  - New mode for distributed rendering of a sequence uses the same syntax as Air's sequence rendering:

    ```
    vortex -frames start end myscene#4f.rib
    ```

    Each frame is assigned to one worker. No output redirection is performed.

  - Number of local workers now defaults to 0

- Bug fixes:
  - Many fixes specific to the 64-bit linux build of Air
  - Several fixes for vector-based edge detection and vector outline export
  - Fix for a bug that could cause a crash with multiple threads in some rare conditions (specifically when the bounding box of a primitive is much larger than the contained mesh or objects)
  - Fix for xyY -> rgb color conversion with `ctransform()`
  - Fix for `environment()` read of texture with fewer than 4 channels
  - Increased buffer limits and added error checks for large numbers of output channels
  - Shader interpreter modified to clamp the third argument of float mix(a,b,c) to 0..1
  - Fix for map handling when map used as environment and texture
  - Fix for `rgbaz` output
  - Fix for stereo rendering with indirect or occlusion prepass
  - Fix to prevent matrix lock corruption
  - Fix to avoid using multiple instances of a DSO shadeop
  - Fix for runtime shading optimizer
  - Fixed `voronoi("grid2d")` to initialize the z-component of the cell centers
  - Updated OpenEXR driver for Windows fixes a bug with pixel aspect ratio

28.3 AIR 10.0

- 64-bit Air binary for Linux

- Support for new unlimited threading licensing option: new Air licenses allow an unlimited number of rendering threads on a single machine. Users with older 4-core licenses may upgrade their licenses to unlimited threading for a fee.

- Stereo and Multicamera Rendering

- Ambient Occlusion
  - New point-based ambient occlusion method
  - Additional optimization accelerates rendering when the max hit distance is small relative to the overall scene size (think local occlusion in a Massive crowd)

- Spectral color definition using spectral power distribution (SPD) files

- Spectral Prefiltering and Chromatic Adaptation

- Photometric lights and IES light profiles
  - New image reader for IES light profile files which converts IES data to a lat-long environment map
  - New photometric_pointlight shader for use with IES profiles

- Shading Language:
  - User structures
  - Support for `option("searchpath:texture",path)` and `option("searchpath:resource",path)`
  - `meshinfo()` vertex sampling now returns a value for facevarying data (it just picks one face's value for the query vertex)
  - New `sRGB` and `litRGB` color spaces
  - Shaders can query the toon id attribute with `attribute("toon:id",id)`
  - Deprecated `irradiancecache()` function

- Multithreading
  - The default number of rendering threads is now equal to the number of detected processing cores
  - Air no longer limits the number of rendering threads to the number of detected cores

- Procedural primitives
  - Air can now execute multiple instances of a RunProgram procedural primitive to accelerate multithreaded rendering. Massive users in particular can use this option to improve rendering performance.
  - Linux builds of Air now detect Python and Perl scripts and automatically invoke the corresponding helper application

- Texture maps:
  - Unconverted textures are now loaded once and shared by all threads (instead of being loaded separately by each thread). Converting all textures to Air's texture format is still recommended for optimal rendering.

- Subsurface scattering (SSS):
  - Subsurface cache generation is now multithreaded, which can greatly accelerate SSS rendering. Multithreading for cache generation is enabled by default. It can be disabled with:

    ```
    Option "render" "integer sssmultithread" [0]
    ```

  - The `subsurfacescatter()` function now handles varying parameter values properly. However using a varying mean diffuse path, reflectance, or ior will be much slower. When using varying parameters, the max sample distance should be explicitly set to produce consistent results. Use as large a max sample distance as feasible to save render time.
  - SSS memory usage has been reduced by 20%

- Polygon meshes
  - Much more efficient rendering of quad meshes with mesh displacement

- Mesh displacement on polygon meshes now applies the same normal correction code as the default displacement method
- The split mesh attribute now works with polygon meshes. There is a new option for the max number of faces in each sub mesh:

  ```
  Option "limits" "integer splitpolyfaces" [25]
  ```

- Subdivision meshes:
  - Simplification of facevarying data on subdivs is now supported when the split mesh attribute is enabled. Enable simplification with

    ```
    Attribute "render" "integer simplifyfacedata" [1]
    ```

    This optimization can save significant memory on objects with facevarying texture coordinates.

- Micropolygon shade mode:
  - Support for multisegment motion blur
  - Fix for polys without vertex normals
  - Fix for dof or motion blur with clipped objects

- Motion blur
  - new attribute to enable/disable adaptive motion blur on a per-primitive basis:

    ```
    Attribute "render" "integer adaptivemotionblur" [1]
    ```

- Instancer shaders: objects with vertex motion blur can be sampled at a given time with `meshinfo("sample:time"...)`

- Geometry export: meshes can now be exported in STL format

- New image readers for Softimage PIC, Windows Bitmap, and IES files

- New display driver: Softimage PIC

- Outlines
  - New option to export outlines in SVG vector format
  - New attribute to include border edges in vector-based outline rendering
  - New attribute to include edges in vector-based outlines based on angle between faces

- Shaders:
  - New surface shaders: ClownPass, Glow, VGradient
  - New light shader photometric_pointlight
  - New displacement shader VThreads
  - New imager shader sRGB
  - Updated VInstanceArchive instancer shader to use a temporary inline archive for instances, for much more efficient rendering of large point clouds
  - Most surface shaders with a color map parameter now convert the map color to "current" color space to work with spectral pre-filtering and chromatic adaptation

- Statistics:
  - Statistics now reports total memory used by unconverted textures
  - New option to report statistics on SSS cache reads:

    ```
    Option "runtime" "integer sssinfo" [1]
    ```

  - Stats include total memory used by all subsurface caches
  - Stats include memory used for occlusion and irradiance caches

- 3D Point clouds:
  - Point clouds with N values that are all 0 no longer save N to point cloud files
  - Support for reading Maya PDC files
  - Fixed bug in brick map loader that prevented Air from finding already loaded maps
  - Fixed `texture3d()` filtering of general point clouds (not applicable to Air Point Maps or Air Brick Maps)

- DSO shade ops:
  - Shade ops that return a string type are now supported
  - Fix to generate standard texture coordinates when only referenced by a shade op
  - Shade op calls now make string parameters available at auxilliary shading locations

- Reprise License Manager (RLM) version 8. See the separate RLM End User Manual in the `doc` directory of the Air installation for information on new and improved features.

- Command line switches:
  - Added `-rows` and `-columns` as shorthand for the corresponding bucket orders
  - Added switch for custom float or string options: `-opt name token value`
  - Updated the list of switches in this manual

- RIB
  - New `Echo` RIB command prints a message to stdout
  - New `Camera` RIB command for defining additional cameras for stereo and multicamera rendering
  - `MakeTexture`, `MakeShadow`, and `MakeCubeFaceEnvironment` now issue INFO messages with the name of the created map

- BakeAir:
  - Support for baking multiple outputs to a single map
  - Individual elements of array output variables can be saved
  - Map name display reports 16-bit and float precision
  - Fixed printing of display modes for rendered maps

- Air Show 5.0:
  - Stereo view mode
  - Interactive gain control
  - New option to disable detection of shadow maps, which allows single channel floating-point images to be displayed as normal images
  - Histogram and min/max computed only for valid region of a cropped image
  - Fixes for 3dl LUT loader

- Air Control
  - Bug fix for incorrect detection of task completion, resulting in "orphaned" air processes in the task manager

- mktex Texture Conversion Tool:
  - Added `byte`, `word`, `float` data conversion and `abs` composite ops

- airpt (Air Point Tool):
  - New `-range` option shows the min and max value of each stored channel
  - Support for reading Maya PDC files

- makelif
  - New option to create SPDL files for Softimage

- Bug fixes:
  - Important fix for motion blur combined with depth of field (bounding box for a facet)
  - Problem with SSS cache evaluation when triggered from a reflection ray
  - Increased thread stack size for Linux air to prevent overflow
  - Polygon auto LOD properly handles the case where an entire primitive is removed
  - Initialize description string to null in image reader
  - Inline archives fixed to prevent prim lock corruption
  - String-to-real conversion now properly handles excess zeroes after the decimal point
  - Real-to-string conversion handles small numbers better
  - Adjusted minimum flatness to allow lower flatness values that may be appropriate when flatness space is not raster space
  - Network cache copy for air on Linux
  - `environment()` "fill"
  - Custom clipping planes
  - Fix for `Display` mode to allow inline declaration of output arrays. E.g.,

    ```
    Display "channel1.tif" "file" "varying color[10] __lights[1]"
    ```

    should work now.

28.4 AIR 9.0

- New common product licensing: TweakAIR and BakeAIR can now use an AIR license. A separate TweakAIR or BakeAIR license is no longer required.

- New light shaders for baking shadows and illumination to 3D texture files

- Important fixes for deep/fragment shadow maps
  - Much better behavior on scenes with high depth complexity
  - Bug fix for the default shadow mode to correct a case where a nearly transparent fragment was recorded as fully opaque

- Light channel tracing
  - AIR surface shaders with reflections now allow light channels to include the effects of reflection and refraction. Light channel tracking through reflections is enabled with the following user option:

    ```
    Option "user" "float reflectlights" [1]
    ```

  - Light channels can be tracked throught indirect diffuse bounces (GI) with the new indirectchannels light shader. With this new shader, each light channel includes not only direct lighting but indirect illumination due to the corresponding light source.

- Indirect diffuse (GI)
  - Much faster rendering with bg environment map
  - Much more efficient rendering of multiple bounces without a cache
  - Fix for overblown results with non-normalized normals
  - new option to enable light channel recording during an indirect prepass:

    ```
    Option "indirect" "string prepasschannels" "__lights"
    ```

- Shading language:
  - environment() and trace() calls provide a "background" output value of type color that stores the contribution of the bg environment map or color
  - environment() and trace() can return arbitrary output variables from ray-traced primitives in a manner similar to the gather() construct.
  - texture3d() accepts a "fill" argument specifying the default value for channels if no points are in range or the channel is missing.
  - bake3d() accepts an "append" parameter with a float value. If the append value is 1, points are added to an existing file.
  - Shaders may query the current frame number with option("FrameNumber",frame)
  - Light shaders can access standard texture coordinates at the current shading location using the new global variables ss and ts.

- Improved environment maps:
  - The shading language environment() call now respects the samples parameter allowing users to fine tune the speed and quality of environment map queries
  - New options to set the number of environment samples for the indirect and reflected background environment maps:

```
Option "indirect" "integer envsamples" [4]
Option "reflection" "integer envsamples" [8]
```

- New micropolygon shading mode:

```
Attribute "shade" "integer mode" [1]
```

- New attribute to apply the flatness tessellation criterion as an absolute tolerance in object space

```
Attribute "render" "string flatnessspace" "object"
```

- Geometry export in PLY format

- Added Perspective RIB command

- Increased max rendering threads from 8 to 16.

- RIB filter additions: ScreenWindow, Projection

- Area lights:
  - Fix for area light normals
  - Better handling of geometry with facevarying data

- Shaders:
  - New surface shaders: BakeTangentNormalMap, CausticPass, IndirectPass, MotionPass, VAnimatedMap, VFabric
  - New light shader: portallight
  - New MassiveAgents instancer shader
  - Most surface shaders with ray-traced reflections now have a __environment output variable holding the contribution of the background environment map for reflection rays
  - All displacement shaders now provide __bump and __undisplaced_N output variables
  - Updated shadowpass shader provides per-channel shadow values

- TweakAIR:
  - Camera view can now be dynamically changed with new IrTweakCamera command
  - Faster IPR display refresh

  - New `IrTweakCoordinateSystem` command for updating coordinate spaces
  - New `IrWriteImage` command writes the image buffer to a file
  - TweakAIR can now be run in "batch" mode with no default display driver and commands piped from a text file
  - Faster storage to deep framebuffer
  - Support for standard Cs color data attached to a primitive
  - Documentation moved out of the Tools chapter to its own topic in the user manual
  - TweakAIR API is now included with the AIR distribution
  - Documentation for the TweakAIR API

- BakeAIR
  - Multithreading support
  - Updated legacy code to allow baking larger maps - up to 8K x 8K at PixelSamples 1 1.
  - New option to have border filling wrap around in texture space:

    ```
    Option "bake" "integer borderwrap" [1]
    ```

    This option may be useful when baking Rhino polymeshes

- AIR Show 4:
  - Simple render commands added to Tools menu
  - 3D LUT support
  - Drag-and-drop support from Windows Explorer (images and rib files)
  - Histogram displays min and max values
  - Image border color is now dark grey instead of black
  - AIR Show now attempts to organize a multichannel file loaded from disk into layers assuming 3 channels per layer with `rgb` or `rgba` as first layer

- AIR Space
  - Instancer shader support
  - support for IPR view updates
  - faster IPR display refresh
  - new option to import some RIB options
  - support for dropped files (.apj, .rib, .obj)
  - the Bake channels parameter now accepts a comma-separated list of channels for baking multiple maps. Each channel is baked to a separate map. Non-standard channel names should include inline type declarations.

- AIR Control:
  - accepts command line file name as default scene
  - new control for command line options passed to the renderer
  - drag-and-drop support from Windows Explorer

- Vortex
  - Allows more than 16 workers
  - Looks for max number of workers in a `VORTEX_MAXWORKERS` environment variable

- Vshade 3:
  - TweakAIR support
  - new improved parameter list
  - preview image size can be set to viewport
  - drag-and-drop support from Windows Explorer

- mktex texture converter:

  - new compositing operations
  - new -info switch to display basic image information
  - -ch option supports a range of channels to extract
  - improved cube-face environment map conversion: supersampling, output to any supported display driver, support for other output options, new -fov option
  - new buffering when writing an AIR texture file greatly improves performance when writing to a network drive from Windows

- Bug fixes
  - AIR quits if a vortex connection is refused (instead of waiting on stdin)
  - Fix for `sqrt()` range check
  - Fix to allow instancers in inline archives
  - AIR now clamps *umin*, *umax*, *vmin*, and *vmax* to the valid range for NURB patches
  - Fix for image scaling when `FrameAspectRatio` is set
  - Point maps are now closed after rendering completes
  - Modified test for degenerate polygons in ray tracer to be relative to coordinate size
  - updated shadeop.h file to include extern "C" declaration for shadeop table (for C++ compatibility)

28.5 AIR 8.0

- Procedure shaders: generate new primitives on-demand at render time using the shading language

- Instancer shaders: create new geometry at render time based on an existing primitive

- 3D Textures and Point Clouds
  - New common set of 3D point file formats for all 3D data
  - AIR Brick Maps
  - AIR Point Maps
  - AIR Point Tool (airpt) for converting and manipulating 3D point sets
  - Point Maps and Point Clouds can be rendered as geometry using the RIB Geometry call

- Points
  - Renamed `Attribute "render" "string pointtype"` to `Attribute "point" "string type"`
  - New point type "disk"
  - Patches and disks can be oriented perpendicular to a normal vector
  - Non-square patches can be specified with a `patchaspectratio` primitive variable
  - New attribute to automatically subdivide large point sets into smaller clusters for more efficient rendering
  - Point width is scaled by a scaling transform

- Curves
  - New attribute:

    `Attribute "curve" "string type" [`*`name`*`]`

    replaces `Attribute "render" "integer beveledcurve"`

  - New curve type "tube"
  - Ribbon curves compute a more accurate value for the global variable v along the curve length
  - Ribbon curves work with uniform normal data
  - Fix for ribbon curve orientation problem with right-hand coordinate system
  - Fix for linear curves and uniform data

  - Curve width is scaled by a scaling transform

- Volume primitives
  - New Dw(x) shading language function gives the derivative in the 3rd dimension for a voxel
  - Fix for calculatenormal()
  - Ray marching works with point-based data

- Procedurals
  - Under Windows AIR now recognizes RunProgram procedurals written in Python or Perl and automatically provides the necessary call to `python.exe` or `perl.exe` respectively
  - New documentation in this manual
  - New examples in `$AIRHOME/examples/primitives/procedurals`

- Subdivision mesh
  - New `interpolateboundary` tag option: when the single integer argument is 2, boundary edges are treated as sharp creases, but boundary vertices with 2 incident edges are not treated as sharp corners.

- RIB Filters
  - Added `Display`, `PointsGeneralPolygons`, `Shutter`, `MotionBegin`, `MotionEnd`, `Clipping`, `Volume`
  - New `-echorif` command line option echoes RI filter output to stdout

- Displays
  - Individual components of a color output variable can now be saved using array subscript notation. E.g.,

    ```
    Display "greylight.tif" "file"
    "__diffuse_unshadowed[0],__shadow[0],__specular[0]"
    ```

    would create a 3 channel file with the first (red) channel of each output variable.

  - AIR will automatically recompute a __shadow output value in a rendered image if the corresponding __diffuse and __diffuse_unshadowed outputs are also present, eliminating artifacts along edges and in semi-transparent areas.
  - Important fix to enable dither

- Geometry export
  - OBJ export now handles facevarying normals and texture coordinates
  - New attribute to select the coordinate space for exported positions and normals

- Frame number expansion for RIB file names on the command line. E.g.,

  ```
  air -frames 19 21 scene#4f.rib
  ```

  would render `scene0019.rib`, `scene0020.rib`, and `scene0021.rib`.

- New option to interpolate density in fragment shadow maps

- Shaders
  - Added `ReflectionGroups` parameter to surface shaders VMetal, VBrushedMetal, VGlass
  - Modified many surface shaders to only reference texture coordinates when needed, resulting in faster rendering and lower memory use when these shaders are used without textures
  - New shaders: texturedarealight, VPhong, VGreyscale
  - Shading language `option()` recognizes Clipping, Shutter, DepthOfField, CropWindow,

  FrameAspectRatio
  - Shading language `attribute()` recognizes ShadingRate, Sides, Matte

- RLM License Server
  - Update to RLM 5.0
  - AIR looks in SITEX_LICENSE and RLM_LICENSE environment variables for the location of the RLM license file or server if an RLM license file is not present in the bin directory of the AIR installation.

- Massive
  - New note on using groups to render mattes and layers
  - New tips for rendering occlusion and how to control occlusion with groups
  - New Massive agent light tool for rendering per-agent lights

- AIR Show 3.0
  - Histogram display
  - Integrated pixel zoom for viewing pixel values
  - Wipe between images for comparison
  - Simple compositing of multilayer images
  - Faster sRGB conversion for 16-bit and float images
  - Quick help for mouse actions and keyboard shortcuts

- AIR User Manual
  - Command line option list moved from Reference to the Getting Started section
  - Attributes list moved from Reference to the Attributes section
  - Options list moved from Reference to the Options section
  - Reorganized many topics including ambient occlusion and indirect lighting

- Vshade 2.0 (see the separate Vshade documentation for details)

- Many internal changes for the forthcoming 64-bit version of AIR

- Bug fixes:
  - Fix for mktex
  - Fix for shading interpreter runtime optimizer (recognize that attribute() may write to array elements)
  - FIx for extended `phong()` per-channel output
  - Prefilterclamptoalpha no longer clobbers outline data
  - Points as spheres now clipped correctly by near clipping plane
  - Fix for z channel output in a multichannel `Display` call
  - Outlines: fix for front/back edge detection
  - Fix for shading cache size without motion blur

28.6 AIR 7.0

- New RhinoAir plugin for Rhino 4 (available as a separate download from the SiTex Graphics web site)

- Automatic level of detail for polygon meshes

- Support for Houdini 9, including the ability to build and compile shaders within Houdini

- New prefilterclamptoalpha option for better filtering of rendered images with high-dynamic range.

- New alternate noise algorithm
- Most Air light shaders have been extended to allow precise specification of shadow-casting groups for raytraced shadows.
- New shading language extensions and optimizations to facilitate per-light output values for diffuse, specular, etc.
- Field rendering
- `ClippingPlane` RIB statement for arbitrary clipping planes (only affecting camera visibility)
- Indirect illumination and occlusion:
  - Occlusion prepass now respects attribute:indirect:reflective:1
  - Attribute:indirect:prepass:0 is now compatible with attribute:indirect:reflective:1
  - If an object has complex opacity (as indicated by attribute transmission hitmode) a prepass will also sample surfaces behind the object
  - New option to specify a subset/group to use for the indirect or occlusion prepass:

    ```
    Option "indirect" "string prepassgroup" ""
    ```
- Outlines:
  - Toon ids are now handled in a manner that eliminates most extraneous lines on transparent surfaces
  - New global multiplier for outline ink width:

    ```
    Option "toon" "float inkwidthmultiplier" 1
    ```
- Subsurface Scattering:
  - New option to set the max error tolerance for subsurface scattering:

    ```
    Option "render" "float subsurfacetolerance" [0.005]
    ```
  - New default subsurfacetolerance of 0.005 (vs 0.05 previously) to eliminate artifacts in some cases
- The names of AOV output channels have been modified to work better with Nuke, ProExr and other plugins and applications. The first four channels for each output variable are now appended with .R, .G, .B, and .A respectively.
- Curves and Points primitives can be used as area lights.
- Curves now compute an accurate t value along the curve length
- RIB filters support many more commands
- Imager shaders:
  - Imager shaders are now executed prior to gamma correction (contrary to the RI spec)
  - Using an imager shader no longer forces display gamma to 1.
  - `environment()` can safely be called from within an imager shader
- mktex and mktexui can convert mirror ball images to lat-long environment maps for rendering.
- New Air Control render queue and airq batch render job submission tool (Windows only)
- New shading compiler option `-fover` to enable function overloading

- New sample HDR environment maps from Aversis (www.aversis.be)

- TweakAir:
  - Relight shade mode now adds __refract ouput if present for better looking glass
  - Fix to prevent thread clash with communications thread
  - Fix for internal compositing with subpixel sampling

- Air Show:
  - New sRGB option to display an image with sRGB gamma correction
  - New View menu item to load a background image, which can be toggled with the checked bg button in the toolbar

- Vshade:
  - Better code generation for arrays for faster rendering
  - Vshade now adds `$AIRHOME/shaders` and `$AIRHOME/usershaders` to the include search path when compiling shaders
  - New DarkTree library category with components for accessing DarkTree shaders (Windows only)
  - Vshade now converts block names into valid identifiers when necessary.

- Grouping:
  - Group membership has been modified so a group of "" resets to no membership in any group rather than leaving the current group unchanged.
  - A null "subset" ("") value passed to `trace()` and other ray tracing calls now selects all objects, equivalent to no subset parameter.

- Shaders
  - New shaders: VShadowedTransparent, VFur, VLayeredMaterials, VLines, VBumpRegions, VBgEnvironment
  - VBlinn provides per-light output channels for diffuse, specular, etc.
  - VToon supports multiple diffuse illumination levels or bands
  - VDashes supports line patterns

- Bug fixes:
  - Air now checks the number and type of RIB shader parameter values against those in the shader and issues an appropriate error message when there is a mismatch.
  - Fix to prevent negative weights for importance sampling
  - Important fix to eliminate shadow map request for tile outside the map
  - Fix for true displacement with mulitsegment motion blur
  - Global time and dtime variables are initialized for auxilliary shading points
  - Fixed number of varying values expected for NuCurves
  - Corrected spherical projection in many Air surface shaders

- The old lightdome and shadowmap utilities have been removed from the distribution

28.7 AIR 6.0

- New user manual section with tips for rendering Massive scenes

- New massrib tool for manipulating Massive scenes

- RIB filters

- Adaptive motion blur

- New shading compiler optimizations can reduce initial code size by a factor of 3 or more for complex shaders generated by MayaMan. Runtime performance is also improved.

- New `bake3d()` and `texture3d()` shading functions for general point set caching and filtering

- New implicit surface primitive for rendering grid data as an isosurface

- The Reprise License Manager (RLM) is now supported in addition to AIR's cilver license manager

- Mesh displacement is now supported for polygon meshes and NURB surfaces

- New attribute to simplify facevarying data on polygon meshes

- New attribute to apply additional smoothing to displaced normals

- New options to cache texture maps locally for improved performance with textures stored on a network drive

- Overlapping volume primitives are now handled properly

- Occlusion improvements:
  - New improved sampling pattern when rendering without the occlusion cache
  - New intermediate screen-space cache when maxerror>0 and maxpixeldist=0.
  - Better performance with multiple reflection samples and no occlusion cache
  - Occlusion can be used with any light source with a shadow name parameter by setting the parameter value to "occlusion".

- Indirect illumination improvements and changes:
  - Much faster rendering of area lights with indirect illumination
  - New improved sampling pattern when rendering without an irradiance cache
  - The default indirect maxerror attribute value is now 0, which disables irradiance caching.

- Better shadow maps with new options to tune depth values stored with the default midpoint depth filter

- New method for a shader to query if an output variable needs to be computed for display:

  ```
  float saved=0;
  option("display:varname",saved);
  ```

  Sets `saved` to 1 if `varname` is being saved in an image.

- New option to enable progress reporting equivalent to the -Progress command line switch:

  ```
  Option "statistics" "integer progress" [1]
  ```

- Multisegment motion blur no long blurs when the shutter interval is 0, and an appropriate intermediate position is interpolated based on the shutter open time.

- Vertex motion blur with all motion times the same now uses the first time position without blurring

- Array arguments are now supported in dynamic shade ops

- Support for vertex transform blur for dynamic blob ops without an ImplicitMotion function

- Primitives with constant user data no longer generate a unique shader instance per primitive
- AIR is able to share shader instances across primitives in more cases
- Most AIR surface shaders have a new `__constant` output variable with the unlit surface color
- The shadowpass shader has a new backshadow switch to enable shadows on surfaces facing away from a light
- A sample dynamic shadeop (DSO) with source code is now included in the AIR distribution in `$AIRHOME/examples/shadeop`
- AIR Show has a new button to display a checker background under images with an alpha channel
- On Linux AIR Show and mktexui no longer reference libglade
- mktex and mktexui have a new option to apply gamma correction to an image
- AIR Space 0.3 (see the separate AIR Space user manual for more information):
  - Area light support
  - BakeAIR support, including OpenGL viewport display of baked maps and VRML export
  - Turntable and camera path animation
  - OpenGL display of texture maps
  - Material import from RIB files
  - Many bug fixes and user interface enhancements
- Bug fixes:
  - Motion vector output with Pixel Samples 1x1
  - Runtime shader optimization for `vtransform()`
  - Spiral bucket order with crop window and a prepass
  - Subsurface scattering and true displacement
  - Area computation for triangles during subsurface scattering collection phase
  - Shading compiler optimizer bugs affecting array references
  - Handling of separate alpha channel for matte objects
  - Compiler bug affecting aligned memory address computation in airmt on Linux

28.8 AIR 5.0

- Fragment shadow maps for "deep" mapped shadows
- Output enhancements:
  - Per-light output channels
  - Multilayer rendering in a single pass
- Ray Tracing
  - Much better sample distribution for shadows and reflections viewed in reflections
  - New options provide a coordinate space for background environment map lookups:

    ```
    Option "reflection" "string envspace" ""
    Option "indirect" "string envspace" ""
    ```
- Subdivision meshes:
  - New attribute to displace an entire subdivision mesh at once:

```
  Attribute "render" "integer meshdisplacement" [1]
  ```

  This method helps avoid cracks when the displacing function has discontinuities.

  - New attribute to split a mesh into subregions for tessellation:

    ```
    Attribute "render" "integer splitmesh" [1]
    ```

    The following option gives the maximum number of faces per subregion:

    ```
    Option "limits" "integer splitmeshfaces" [25]
    ```

  - Smoother seams between primitives stitched together by MayaMan

- Better true displacement for Blobby primitives with the new meshdisplacement method

- Points primitives
  - New mode to render points as spheres:

    ```
    Attribute "render" "string pointtype" "sphere"
    ```

  - Much better shading cache performance for motion blurred points
  - Bug fix for ray traced points

- Volume primitives:
  - Support for `mpoint` data
  - Support for `dPdtime` for point-based data
  - New attribute indicating when a procedural primitive contains a volume primitive:

    ```
    Attribute "visibility" "integer volume" [1]
    ```

- Much more efficient memory use for particle systems instanced with individual primitives with constant primitive shader variables

- Much more efficient shading for motion-blurred unbeveled curves

- TweakAIR:
  - Support for moving area lights
  - Fix for black dots in indirect and envlight light caches with multiple threads

- AIR Show
  - AIR Show can read AIR shadow maps (with a `.sm` or `.shd` extension)
  - Fix for refresh bug with small or rapidly rendered buckets
  - Fix for bug on Linux that prevented AIR Show from restarting immediately
  - When saving images AIR Show automatically converts data to the proper number of channels and data type for the selected image format.

- AIR Space
  - Cameras can be viewed and positioned in OpenGL views
  - Per-light output channels
  - New introductory tutorial in the AIR Space user manual

- New option to control shading cache memory

- New Optimization section in the AIR user manual

- Optimization to eliminate unnecessary primitive data for MayaMan shaders

- The Linux distribution is now built on Red Hat 9 instead of Red Hat 7.2

- On Windows AIR and other programs have the *largeaddressaware* flag set, allowing them to access up to 4 GB of memory on 64-bit versions of Windows.

- The maximum number of Pixel Samples has increased to 32 x 32.

- New ReelSmartMotion shader computes motion vectors for the Reel Smart Motion Blur plugin from RE:Vision Effects, Inc. (www.revisionfx.com)

- Motion blurred primitives with time intervals that are a subrange of the `Shutter` time interval are properly dimmed.

- AIR recognizes gzipped files without a `.gz` extension

- Vortex now works with display drivers that expect data in scanline order

- Configuration instructions for previewing AIR shaders in Houdini

- The BMP and TIFF display drivers now accept `"resolution"` and `"resolutionunit"` parameters

- New shaders: VSketchOutline, VNormalMap, VTangentNormalMap

- Bug fixes:
  - Fix for traced shadow bug causing horizontal line artifacts
  - Fix to prevent min z depth filtering from trashing AOVs
  - On Windows AIR no longer tries to open AIR Show locally when launched with the `-dhost` option
  - On Linux airmt supports the `-dhost` switch.

28.9 AIR 4.1

- AIR Space, a new interactive shading and lighting interface for AIR and TweakAIR. See the separate AIR Space User Manual for more information.

- New transmission attributes for compatibility with PRMan:

  ```
  Attribute "visibility" "integer transmission" [1]
  ```

  enables visibility to shadow rays.

  ```
  Attribute "shade" "string transmissionhitmode" ["primitive"]
  ```

  determines how an opacity value is generated for shadow queries: using the "primitive" opacity or evaluating the surface "shader".

- Support for conditional RIB

- AIR can now parse gzipped RIB files. Gzipped files must have a file extension of `.gz`.

- New interpolation option for facevarying subdivision mesh data

- New option to apply normal filtering to depth values:

```
Option "render" "string zfilter" "normal"
```

- Automap improvements:
  - Much better cache performance
  - Now work with point lights (in addition to spotlights and distant lights)

- TweakAIR improvements:
  - Better handling of displacement shader tweaking
  - New reflection cache improves refresh performance for objects with traced reflections
  - New option to automatically update an object when a reflected object changes:

    ```
    Option "ipr" "integer retrace" [1]
    ```

  - Many bugs fixed on Linux

- New, much improved environment map filtering

- AIR will automatically generate tangent vectors for polygon meshes based on the assigned texture coordinates

- Light shaders can query surface output parameters

- AIR Show can display AIR texture maps

- Support for texture-mapped area lights

- New `-surf` command line option to override the surface shaders in a RIB file

- Many surface and displacement shaders have been modified to employ new conventions for 2D and 3D patterns recognized by AIR Space

- New shaders: VRubber, VDecal2D (replacing VDecal2DPlastic), VDecal3D (replacing VDecal3DPlastic)

- Vshade now accepts varying input parameters

- Bug fixes:
  - Failure to free automaps at end of frame
  - Facevarying interpolation on subdivision meshes
  - `diffuse()` handling __indirectlight vars
  - Fix for subsurface scattering shaders included with AIR to use all 3 scatter distances
  - Random crash with outlining and multithreading
  - `rayhittest()` now performs a self-intersection check if the ray origin is the current shading location
  - Fix for builtin blinn() function
  - Per-curve s,t data is supported
  - Volume primitives accept user attributes
  - Fix for using a SSS cache with true displacement

28.10 AIR 4.0

- Volume primitives for fast volumetric rendering

- Automaps: automatic shadow maps

- TweakAIR improvements:
  - New fast re-lighting shading mode
  - TweakAIR now supports the indirect and occlusion prepasses
  - New option for anti-aliasing with multiple pixel samples:

    ```
    Option "ipr" "integer[2] pixelsamples" [1 1]
    ```

- Simple RIB command filtering

- Improved curve rendering:
  - More efficient rendering of catmull-rom and b-spline curves
  - Memory reduced up to 50% for general curve rendering

- Subdivision mesh improvements:
  - Optimized tessellation reduces memory usage and render time
  - Better interpolation of facevarying variables
  - Rendering of border faces with sharp edges w/o the interpolate boundary flag

- Points primitives may now be rendered as bilinear patches for sprites

- Shading language extensions:
  - new builtin `blinn()` function implements a Blinn specular illumination model
  - new builtin `brushedspecular()` function implements an anisotropic specular function similar to Ward's
  - `diffuse()` accepts an optional `"roughness"` parameter for Oren-Nayer shading
  - `specular()` accepts an optional fourth *sharpness* parameter for sharp-edged specular highlights
  - `phong()` accepts an optional fourth argument for the extended Phong model
  - New special spline types `"solvecatrom"` and `"solvelinear"` for `spline()`

- New and improved shaders:
  - Many shaders have been modified to take advantage of the new builtin illumination functions in AIR 4, yielding faster render times
  - The particle shader now supports a more general single-scattering illumination model
  - New shaders: VWeave, VBlinn, VSmokeSurface, VCumulusCloud

- Better filtering of transparent and semi-transparent objects for depth pass renders:
  - No depth value is stored for completely transparent shading locations
  - For semi-transparent surfaces, only the nearest depth is stored with no transparency

- New option to map a range of depth map values prior to quantization:

  ```
  Option "render" "float[2] zrange" [zmin zmax]
  ```

  If *zmin* is not equal to *zmax*, z depth values are scaled so that z is 0 at *zmin* and 1 at *zmax*, and clamped to the range 0..1.

- AIR Show enhancements:
  - New depth map display mode
  - New button to jump between the two most recently viewed frames

- New attributes to scale and offset the times passed in a motion block:

  ```
  Attribute "render" "float shutterscale" [1]
  Attribute "render" "float shutteroffset" [0]
  ```

These attributes are useful for re-using archives in an animated sequence.

- Display enhancements:
  - The `Display` call accepts inline variable declarations. E.g.,

    ```
    Display "Diffuse.tif" "file" "varying color __diffuse"
    ```

  - The real names of arbitrary output variables are passed to display drivers, so that drivers such as OpenEXR can save full channel names.

- New edge mask option when rendering shadow maps may improve shadows cast by small or thin objects:

  ```
  Option "render" "integer shadowedgemask" [1]
  ```

- `PointsGeneralPolygons` now detects and handles non-planar 4-sided polygons

- The `-Progress` command line switch now outputs current memory, peak memory, and elapsed time in addition to a percent complete value. E.g.,

  ```
  R90000  55%  1334K  2444K  19S
  ```

- New additions to the Plugins and Companions page:
  - XSIMan, a new plugin for XSI from Graphic Primitives
  - Temerity Pipeline, a production management application

- Bug fixes:
  - Uniform string primitive variables were not handled properly
  - Attribute comparison bug affecting bakemaps for BakeAIR
  - Subtle noise() bug for sample locations close to but less than an integer
  - Subtle bug in normal compression for normals near (0,1,0)
  - Binary rib bug in handling of 16-bit string ids
  - Long shader parameter names are compared properly with truncated names in `.slb` files
  - The shading compiler accepts a variable string argument to `ctransform()`

28.11 AIR 3.1

- TweakAIR, a new product for interactively tweaking shading and lighting in a scene

- New Photoshop display driver for producing layered Photoshop files

- New Photoshop image reader

- Shading language extensions:
  - New shading language voronoi() function that is 4 times faster than the equivalent shading language code
  - diffuse(), phong(), and specular() accept an optional initial light category
  - Extended diffuse() function with output variables to facilitate multipass rendering

- More efficient rendering of `PointsGeneralPolygons` and polygons with texture data

- Additional attributes for geometry export:
  - Export to ASCII RIB
  - Export to RIB using DelayedReadArchive

  - Automatic generation of `__Pref` data
  - Automatic generation of `__Nref` data

- The `gather()` function has a new `"othreshold"` parameter to control ray continuation

- Faster lighting calculations in MayaMan, especially global illumination with maxerror=0

- New toon option to set a minimum ink width when the fade-with-distance option is enabled:

  ```
  Option "toon" "float mininkwidth" [0.0]
  ```

- More accurate ray tracing with better detection of degenerate triangles

- New option to set the maximum grid size for displacement:

  ```
  Option "limits" "integer displacementgrid" [8]
  ```

- More efficient handling of uniform data

- Optimized shading structure storage for BakeAIR, especially for primitives that are not "baked"

- Inline archive support for Procedural RunPrograms

- Changes to the AIR shader library:
  - Most surface shaders have additional output variables for multipass rendering: `__diffuse_unshadowed`, `__shadow`, and `__indirect`
  - Parameter names for many older shaders have been made consistent with more recent shaders. (For example, the diffuse strength parameter is now named `Diffuse` instead of `Kd`.)
  - A new VMetal shader is the new standard metal shader (replacing the SIshinymetal shader)
  - New VGranite shader replaces VGranite2
  - Several shaders have been renamed: Dashes to VDashes, RustyMetal to VRustyMetal, Concrete to VConcrete
  - The ambient_indirect and ambient_envlight shaders are no longer included. Their functionality has been replaced by the addition of an `__indirect` output variable to the AIR shaders.
  - The following shaders are no longer included with AIR: wood2, VLaqueredInlaidWood, SIdents (use VDents), backgroundgradient (use VBgGradient), VShinyCeramic (use VCeramic)

- AIR Show enhancements:
  - On Windows a sequence of images can be saved as an AVI file
  - New Channels window displays the extra channels for images with multiple output variables

- New redesigned AIR Control tool

- Several old tools have been removed form the AIR distribution: Visual Browser, LightEd, MatMap, and the old AIR Control. The last versions of these tools are available for download:

  ```
  http://www.sitexgraphics.com/oldtools.zip
  ```

- Bug fixes:
  - Important fix for `illuminate()` bug that caused surfaces facing away from a light to be "lit" anyway
  - "shader" space for shader declarations in inline archives
  - Uniform primitive variables
  - Shadow maps with no blur and shadows at edges
  - Illuminance() cache category comparison

- Fix for outlining with an indirect or occlusion prepass
- Fix for runtime shading optimization that very rarely caused a silent crash
- Fixed failure to fully reset occlusion cache between frames

28.12 AIR 3.0

- Geometry export for displacement baking and modeling

- Improved toon rendering:
  - The viewport of a rendering with outlines now matches the view without outlines
  - New depth-based edge-detection method
  - New silhouette edge-detection method based on detecting boundaries between front- and back-facing polygons in the rendering mesh
  - Outlines may now be saved as a separate output image with the new __toonline output variable
  - Faster rendering with large maxinkwidth settings
  - New special region id of -2 can be used to outline every polygon in a rendering mesh

- Fast volumetric fog with a new foglight() function

- Vector texture support with Alex Segal's Vtexture

- New FreeObject RIB command frees an object created with ObjectBegin/ObjectEnd

- New FreeArchive RIB command frees an inline archive created with ArchiveBegin/ArchiveEnd

- Shading improvements:
  - New dead-code elimination and constant-folding capabilities in the shading compiler produce better code
  - Shader execution optimization yields better performance with many AIR shaders

- New depth (z) filtering for multiple pixel samples: with multiple pixel samples the output depth is now the minimum of the samples, making it easier to generate a depth map along with a beauty pass.

- Global illumination improvements:
  - Optimized texture access when shading indirect rays
  - Caustics are no longer evaluated during the indirect prepass

- Ray tracing improvements:
  - Optimized shadow tracing for many semi-transparent objects
  - Ray tracing storage reduced 2-3% for large scenes
  - Improved trace blur for interreflections

- Displacement improvements:
  - Automatic true displacement detection in AIR displacement shaders. Users no longer need to set a TrueDisplacement parameter for AIR shaders; this feature is now completely controlled by the true displacement attribute.
  - Primitives with custom user data no longer use excessive shading structures with true displacement

- Photon map queries are about 10% faster

- New reader and display driver for Radiance HDR image files

- New AIR texture display driver for rendering AIR texture maps directly with BakeAIR or AIR

- Shader additions and changes:
  - New VScreen surface shader replaces the old SIscreen and SIholes shaders (the old source code can still be found in `$AIRHOME/shaders/oldsrc` for those interested).
  - New DarkTreeDisplacement shader allows DarkTree shaders to be used for displacement.
  - New VFog volume shader for fast volumetric fog
  - New V2SidedPlastic shader allows separate shading for each side of surface
  - New VShadowedConstant shader provides a constant-colored surface with shadows
  - Most displacement shaders have been updated with more descriptive parameter names and automatic true displacement detection
  - The SIGlass surface has been deprecated in favor of the new VGlass shader

- AIR Show 2
  - Tabbed output windows have been replaced by a list of images with thumbnails along the right side of the main AIR Show window. Among other benefits this change circumvents the Windows limit on the number tabs in a single window.
  - The new AIR Show largely eliminates a problem on Linux where the display window would sometimes freeze during a rendering.
  - The toolbar may now be hidden and shown without restarting AIR Show.
  - On Windows the currently displayed image may be copied to the clipboard. The image will copied as displayed; i.e., the raw output data will not be copied.

- New and improved mktex and mktexui texture conversion utilities:
  - Angular probe images can be converted to AIR environment maps
  - Source images may be flipped, rotated and resized
  - Output data format may be specified as 8-bit, 16-bit, or float independent of source format
  - mktex can output to any image format recognized by AIR

- Vshade 0.4. See the separate Vshade manual for detailed information.

- Bug fixes
  - Important bug fixed in insertion & sorting code
  - Ray tracing hang bug
  - Spiral bucket order with a crop window failed sometimes
  - pow(x,y) now works properly with negative y
  - Rare divide by 0 error in occlusion
  - DelayedReadArchives failed to inherit the current true displacement attribute

28.13 AIR 2.9

- Multithreading on Linux (in addition to existing support for multithreading on Windows)

- Importance-based rendering for faster rendering

- Texture readers for additional image input formats

- Faster raw ray tracing performance

- Ambient occlusion improvements: faster rendering, better quality, less memory

- Scene storage optimization for memory reduction of 10-20% for typical scenes. Additional optimization for polygon meshes. Houdini users in particular should see a dramatic decrease in memory use for polygonal objects.

- Support for SideFx's Houdini
- New parameters for `environment()` and `trace()` to support anisotropic reflection
- Modified VBrushedMetal shader and a new VBrushedPlastic shader with anisotropic reflections
- Interior and Exterior volume shaders
- New VSmoke volume shader (see the example in `$AIRHOME/examples/effects`)
- New pixel filters: disk, mitchell, and lanczos
- Faster true displacement
- New attribute to indicate whether displacement takes place only in the direction of the surface normal:

  ```
  Attribute "render" "integer normaldisplacement" [1]
  ```

- New attribute to select triangles instead of quads for displacement meshes:

  ```
  Attribute "render" "integer triangledisplacement" [1]
  ```

  Using triangles may reduce artifacts at the cost of increased rendering times.

- Dispersion simulation for caustics
- AIR now reads from the standard input stream if a file name is not provided. To display help information, use a command line argument of - or ?
- A new TIFF display driver supports lzw compression on Windows (and on Linux systems whose libtiff library supports lzw compression).
- The `texture()` shading language function now supports a `"filter"` parameter, though its use is discouraged as it incurs a performance penalty and may produce inferior results if used improperly.
- `"cosine"` distribution for `gather()` now respects the angle parameter
- New mktexui graphical user interface for the AIR texture-making utility mktex
- RiDeformation has been de-supported
- BakeAIR improvements:
  - BakeAIR now implements the indirect and occlusion prepasses
  - A new option allows control of the border created around rendered regions in a baked map
- AIR Show improvements:
  - AIR Show reads image files in any format supported by AIR
  - AIR Show options are now saved between sessions
- The multithreaded version of the Cilver floating license manager is now included in the AIR distribution. All users are encouraged to upgrade to this version if they have not already done so.

- The default shading type for shadows is now "Os" (use the object's opacity attribute). For shaders with complicated opacity (e.g., a screen shader), you will need to set the shadow type to "shade".

- Bug fixes:
  - Better handling of wide filter-widths to prevent changes to the viewport based on filter width, which could cause a misalignment when compositing AIR renders with output from other renderers
  - Fix for bogus X shading location in motion blur
  - Fix for stall condition with multiple threads and GI
  - Fix for process priority on Windows
  - JPEG display driver fix
  - Imager shaders now work with outlining
  - Traced shadows with 1 sample blur properly
  - Shadow map filtering fixed near map edges

28.14 AIR 2.8

- Subsurface scattering

- New collection capability to generate and query a set of points distributed over a group of surfaces (the underlying capability on which the new subsurface scattering feature is built)

- Outline rendering for cartoon rendering and illustration

- Semisharp creases for subdivision meshes have been implemented using a new "linear" crease tag. (Conversion of the standard "crease" tag takes place automatically.)

- A new shadow map filtering algorithm produces beautiful smooth results and no longer requires using and tweaking a shadow "samples" parameter. Shadow maps now produce much nicer results with transparent and motion blurred objects.

- Vortex now supports the indirect and occlusion prepasses.

- Faster true displacement

- A new attribute controls whether an object is sampled during an indirect or occlusion prepass:

  ```
  Attribute "indirect" "integer prepass" [1]
  ```

- New option to set the rendering process priority on Windows

- New `__threshold` tag for blobbies

- A new `$AIRHOME/usershaders` directory in the AIR home directory (included in the default search path) has been added to provide a location for custom shaders that are not part of the standard AIR distribution.

- A new `$AIRHOME/procedurals` directory is in the default procedural search path

- `ArchiveInstance` will now attempt to create an instance from an archive file if an existing instance is not found.

- For shadeop DSOs AIR now looks for a DLL with an abbreviated name. E.g., for shadeop `MyTest`, AIR will look for `MyTest.dll`, `MyTes.dll`, `MyTe.dll`, and `MyT.dll` in that order.

- Basic texture statistics are now reported in the stats output.

- Faster ambient occlusion (about 5%)
- The `gather()` instruction now accepts a "distribution" parameter that specifies the distribution of samples - either "uniform" or "cosine".
- New shaders: VGlass, VSkin, VTexturedBump, VTranslucent, VTranslucentMarble
- A link to Massive has been added to the Plugins and Companions page
- Bug fixes:
  - Fix for a rare condition in blobbies
  - Fix for user attributes and Procedural primitives
  - Fix for case where a freed prim was queried for its motion position
  - Fixed token handling for Procedural DelayedLoad DLLs
  - Fixed bug where displaced surfaces were incorrectly culled