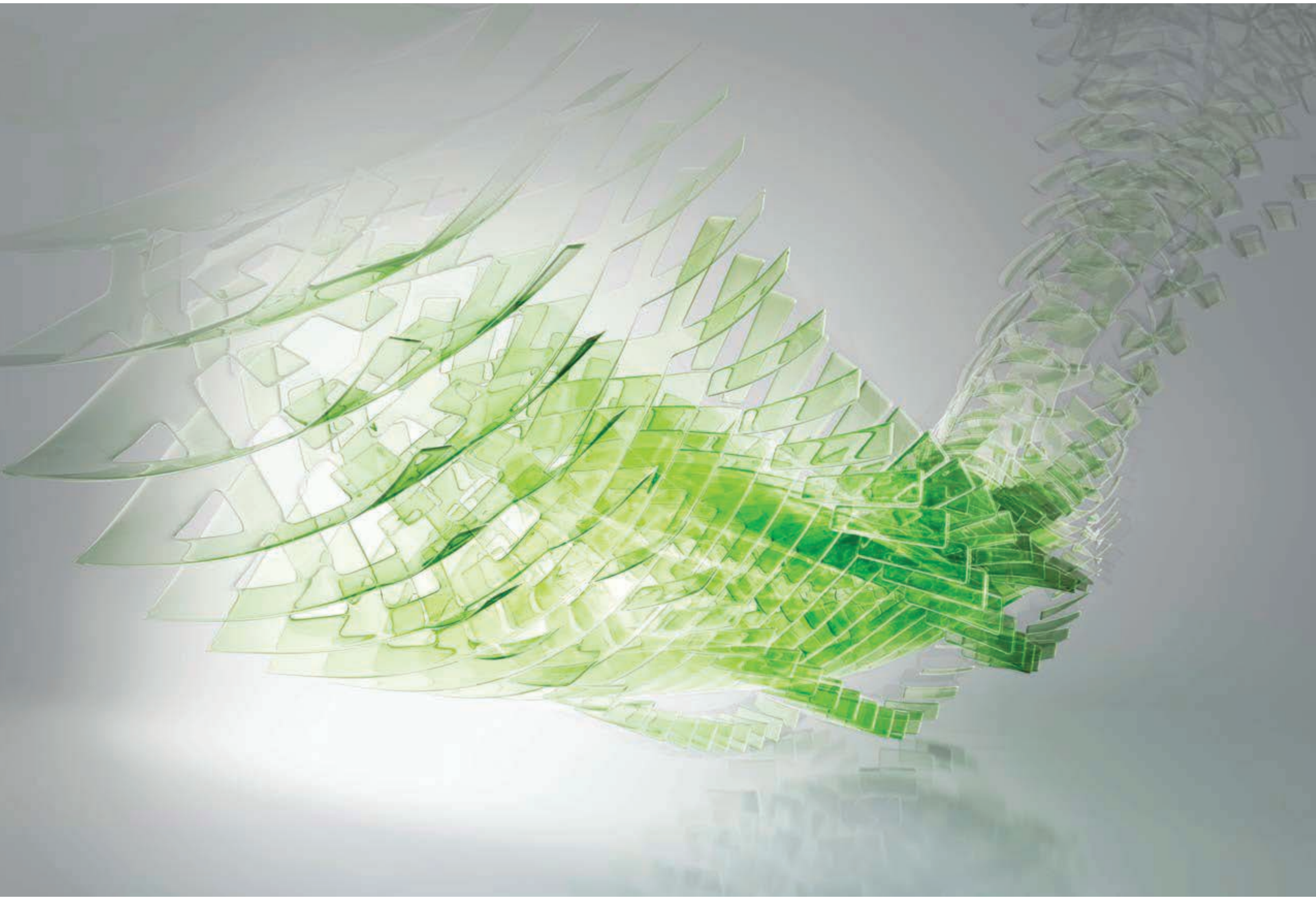
AUTODESK®
FLAME® PREMIUM



Shader Builder API Guide

AUTODESK.

Contents

Chapter 1 Creating Your Own Shader 1

Chapter 2 Creating a Matchbox Shader 3
Matchbox Shader Examples 6

Chapter 3 Creating a Lightbox Shader 9
Lightbox Shader Examples 16

Chapter 4 About shader_builder XML 17

Chapter 5 Shader API Documentation 31

Creating Your Own Shader

1

A great benefit of working with customized shaders in Autodesk Flame, Flare, Flame Assist or Smoke is being able to create your own effects, depending on your particular needs. Creating a custom shader can be as simple as copying and pasting GLSL code snippets, or can be complex multi-pass effects with multiple inputs and dozens of user interface elements. Custom shaders can also be encrypted for distribution.

Here's the high-level workflow to follow when creating custom shaders:

1. Write or copy/paste GLSL fragment shader code.
2. Use the provided shader_builder command line tool to test the shader and generate an XML file for the user interface displayed in the application.
3. Fix any errors in the GLSL code.
4. Edit the XML output from the shader_builder to customize the user interface.
5. Package and maybe encrypt the XML and GLSL code together.

Matchbox and Lightbox availability depends on the application being used.

Product	Lightbox Availability	Matchbox in Action	Matchbox in Batch/BFX	Matchbox in Timeline
Flame	Available	Available	Available	Available
Flare	Available	Available	Available	Available (as source in Batch)
Flame Assist	Not available	Not available	Available	Available
Smoke	Not available	Not available	Available (ConnectFX)	Available

What is GLSL?

GLSL stands for OpenGL Shading Language, and is a language based on C syntax. GLSL, as its name indicates, is a low-level API of OpenGL, the API used to render graphics on graphical processing units.

GLSL was created to facilitate the manipulation of the graphics pipeline by developers, without requiring deep machine code knowledge. In GLSL, normals, texture maps, complex vector and matrices operations are all central to language, making the creation of shaders simpler. The specifications for OpenGL and GLSL can be found at the *OpenGL Foundation website*.

Lightbox and Matchbox shaders are all GLSL-created shaders.

You will also find resources on the web, such as *Lighthouse 3D*. When looking for online resources, keep in mind that on Linux the supported version of GLSL is 130. If you want your Lightbox (or Matchbox) snippet to run on Mac OS X, you must use GLSL version 120.

Creating a Matchbox Shader

2

Here are the contents of a simple Add effect as a Matchbox shader:

```
uniform float adsk_result_w, adsk_result_h;

void main()
{
   vec2 coords = gl_FragCoord.xy / vec2( adsk_result_w, adsk_result_h );
   vec3 sourceColor1 = texture2D(input1, coords).rgb;
   vec3 sourceColor2 = texture2D(input2, coords).rgb;

   gl_FragColor = vec4( sourceColor1+sourceColor2, 1.0 );
}
```

Writing and Testing Matchbox Shader Code

A Matchbox shader is always structured as:

- A forward declaration where all the uniforms and APIs are declared.
- A `main()` function where the magic happens. Matchbox shaders are not part of an established shading pipeline like Lightbox shaders are, hence the use of `main()`.

You can re-purpose existing fragment shader code, or create an effect specific to your needs.

NOTE On Linux, the supported version of GLSL is 130. If you want your Matchbox code to run on Mac OS X, you must use GLSL 120. Also, Action does comply with OpenGL core profile.

Once you're done coding the shader, use shader_builder to validate and debug your code, and help you design user interface elements via a sidecar XML file. Once validated, you can encrypt and package your custom shader. shader_builder also has an extensive set of uniforms (including a number of Autodesk custom uniforms) and XML entities that allow you to:

- Define the UI layout.
- Define parameter names.
- Define numerical intervals for specified parameters.
- Disable or hide a button based on the status of another button.
- Define a web link, to direct users to documentation on the shader you are providing.

Because Matchbox shaders can be used in Batch/Batch FX/ConnectFX, there are some Matchbox-only uniforms:

- Setting an Input Type for a Matchbox Input socket (Front, Back, Matte).
- Defining custom colours for a Matchbox Input Socket.

- Limiting the number of Input Sockets displayed on the Matchbox node to the number of Textures required by the current shader.

The shader_builder utility is found in /usr/discreet/<product_home>/bin. To access its Help file, from the bin directory, type shader_builder --help. All this information is also available in About shader_builder XML (page 17).

The main command to process a Matchbox shader is `shader_builder -m <name of shader>.glsl`. You can use the following switches with this command:

- `-m` to signify to shader_builder that the .glsl file must be read as a Matchbox shader.
- `-l` to signify to shader_builder that the .glsl file must be read as a Lightbox shader.
- `-x` to generate a sidecar XML file.
- `-p` to encrypt and package the shader.
- `-o` to specify a specific directory to output the shader.
- `-u` to output the shader to the current directory.

These switches can be combined in one command. For example: `shader_builder -m -x <name of shader>.glsl`.

To create and test a fragment shader:

1 Write or copy your fragment shader code in a text editor. For example, here is a scaling effect:

```
uniform float size;
uniform sampler2D myInputTex;

void main (void) {
        vec4 tex0 = texture2D(myInputTex, gl_TexCoord[0] * size);
        gl_FragColor = vec4 (tex0.rgb, 1.0);
}
```

2 Save the file with the extension .glsl. For the purpose of this example, it is named `scale.glsl`.

3 Run your code through the test utility and generate a sidecar XML file.

`shader_builder -m -x scale.glsl` tests the above shader and outputs the results in the shell, while generating the sidecar XML file named `scale.xml`. But produces the following result:

```
Shader pass 0:
Fragment info
-------------
0(5) : warning C7011: implicit cast from "vec4" to "vec2"
compiling shader file scale.glsl ... [OK]
all shaders compiled ... [OK]
generating XML interface ...
creating XML file (scale.xml) ... [OK]
```

In the example above, the fourth line displays a compilation warning that you might want to fix, indicating the number of the line where the error is located. In other cases, you'll receive errors that need to be fixed; `shader_builder` builds the XML file only if it encounters no errors while testing the .glsl file.

4 Fix any errors, and rerun the code through the `shader_builder` utility.

5 Use the XML file generated by `shader_builder` to help you set up the UI of the effect. This can be especially useful if different users are going to be working with these effects. In our example, you can edit `scale.xml` to add default values, better names for inputs and other UI elements, and even tooltips to help the user.

In the example below, updated values are in bold-italics.

```
<ShaderNodePreset
   SupportsAdaptiveDegradation="False" SupportsAction="False"
   SupportsTransition="False" SupportsTimeline="False"
   TimelineUseBack="False" MatteProvider="False" ShaderType="Matchbox"
   SoftwareVersion="2016.0.0" LimitInputsToTexture="True"
   Version="1" Description="" Name="Next Generation Scaling">

   <Shader OutputBitDepth="Output" Index="1">
        <Uniform Index="0" NoInput="Error"
                 Tooltip="" DisplayName="Front"
                 InputColor="67, 77, 83" Mipmaps="False"
                 GL_TEXTURE_WRAP_T="GL_REPEAT"
                 GL_TEXTURE_WRAP_S="GL_REPEAT"
                 GL_TEXTURE_MAG_FILTER="GL_LINEAR"
                 GL_TEXTURE_MIN_FILTER="GL_LINEAR"
                 Type="sampler2D" Name="myInputTex">
        </Uniform>

        <Uniform ResDependent="None"
                 Max="100.00"
                 Min="-100.00"
                 Default="0.0"
                 Inc="0.01"
                 Tooltip="Displays the percentage of scaling applied."
                 Row="0" Col="0" Page="0"
                 Type="float"
                 DisplayName="Scale"
                 Name="size">
        </Uniform>
    </Shader>

    <Page Name="Page 1" Page="0">
        <Col Name="Effect Settings" Col="0" Page="0">
        </Col>
    </Page>

</ShaderNodePreset>
```

6 Add the .glsl and sidecar .xml file to the same directory, with an optional .p or .png file to be used as a thumbnail in the Matchbox browser. The existing shader files are located in `/usr/discreet/presets/<application version>/matchbox/shaders/`.

7 If you want to encrypt and package the shader in the .mx format, from that location, run `shader_builder -m -p <name of shader>.glsl`.

8 Try your effect in Smoke or Flame.

If you cannot see your shader in the Matchbox browser, try toggling the Matchbox box filter between Matchbox and GLSL.

To help you in creating custom shaders, example shaders (page 6) are available in `/usr/discreet/presets/<application_version>/matchbox/shaders/EXAMPLES/`.

Forward Declaration

Lightbox requires that you forward declare your uniforms and used APIs. This ensures that shader_builder provides accurate file locations for warnings and errors.

Creating Multipass Shaders

In order to build more efficient, complex, or sophisticated effects, you can split your effects into multiple passes. In order to do this, you separate your effect into multiple .glsl files using advanced adsk_ uniforms. For example, the existing Pyramid Blur filter preset consists of `PyramidBlur.1.glsl` and `PyramidBlur.2.glsl`. In this case, when selecting this effect from the Load Shaders browser in the application, you need to select the root group `PyramidBlur.glsl` file to incorporate all of the passes as one effect. Use the `-p` switch to package everything into a single file, making it less confusing to load from the browser.

About Cross-Version Compatibility

Due to updates to both the Matchbox API and the schema of the sidecar XML, you cannot use a Matchbox created for 2016 in a previous version. But you can use a Matchbox created in a previous version in 2016.

Optional: Create a Browser Proxy Files

Along with the .glsl and .xml files that comprise a fragment shader, you can also create a .png file that displays a proxy of your effect in the Matchbox browser.

To use a .png file as the proxy:

1. Create an 8-bit, 126 pixels wide by 92 pixels high .png
2. Place the .png file in the same folder as the .glsl and .xml files of the same name.

Matchbox Shader Examples

To help you in creating custom Matchbox shaders, example shaders are available in `/usr/discreet/presets/<application_version>/matchbox/shaders/EXAMPLES/`.

The following example Matchbox shaders are available:

Blending Shows an example of both the usage of int popup, as well as the usage of the blend mode API.

BuildList Only shows the int as popup XML feature.

ColourWheel Shows how to use the Colour Wheel widget in both the xml and glsl files.

ConditionalUI Shows how to use the conditional UI xml functionality.

Curves Shows how to use the Curve widget with two different curve types, in both the xml and glsl files.

ImageLighting Shows how to do a light in Matchbox.

ImagePosition Shows how to do image X and Y transforms.

ImageRotation Shows how to do image rotation.

ImageScaling Shows how to do image X and Y scaling.

InputSockets Shows how to define Input socket type and colour in the xml.

MatchboxAPI Shows how to use the different Matchbox API calls.

Mipmaps Shows how to use mipmaps to perform a blur in Matchbox.

PyramidBlur Shows how to do the equivalent of a fast Gaussian blur in Matchbox.

TemporalSampling Shows how to use previous and next frame input in both the xml and glsl files.

TransitionShader Shows how to create a Transition Matchbox shader to be used on the timeline.

UIOnly Shows how to create Matchbox that are only used to do UI for expression usage.

Creating a Lightbox Shader

3

Here are the contents of a simple invert color effect as a Lightbox shader:

```
vec4 adskUID_lightbox(vec4 source)
{
    source.rgb = vec3(1.0) - source.rgb;
    return source;
}
```

Writing and Testing Lightbox Shader Code

A Lightbox shader is always structured as:

- A forward declaration where all the uniforms and APIs are declared.
- A `vec4 adskUID_lightbox(vec4 source)` function, where the magic happens. Lightbox shaders are part of the Action shading pipeline.

You can re-purpose existing fragment shader code, or create an effect specific to your needs.

NOTE On Linux, the supported version of GLSL is 130. If you want your Lightbox (or Matchbox) snippet to run on Mac OS X, you must use GLSL 120. Also, Action does NOT comply with OpenGL core profile.

Once you're done with coding the shader, use shader_builder to validate and debug your code, and help you design user interface elements via a sidecar XML file. Once validated, you can encrypt and package your custom shader. The shader_builder utility also has an extensive set of uniforms (including a number of Autodesk custom uniforms) and XML entities that allow you to:

- Define the UI layout.
- Define parameter names.
- Define numerical intervals for specified parameters.
- Disable or hide a button based on the status of another button.
- Define a web link, to direct users to documentation on the shader you are providing.

The shader_builder utility is found in `/usr/discreet/<product_home>/bin`. To access its Help file, from the bin directory, type `shader_builder --help`. All this info is also available in About shader_builder XML (page 17).

The main command to process a Lightbox shader is `shader_builder -l <name of shader>.glsl`. You can use the following switches with this command:

- `-l` to signify to shader_builder that the .glsl file must be read as a Lightbox shader.
- `-m` to signify to shader_builder that the .glsl file must be read as a Matchbox shader.

- `-x` to generate a sidecar XML file.
- `-p` to encrypt and package the shader.
- `-o` to specify a specific directory to output the shader.
- `-u` to output the shader to the current directory.

These switches can be combined in one command. For example: `shader_builder -l -x <name of shader>.glsl`.

To create and test a fragment shader:

1. Write or copy your fragment shader code in a text editor, making sure to include the main `function vec4 adskUID_lightbox ()` .

   For example, here is the contents of a gain effect:

   ```
   1 uniform float adskUID_gain;
   2 vec4 adskUID_lightbox( vec4 source )
   3 {
   4       source.rgb = source.rgb * adskUID_gain;
   5       return vec4( source.rgb, source.a )
   6 }
   ```

2. Save the file with the extension `.glsl` .

   For the purpose of this example, it is named `gain.glsl`.

3. Run your code through the test utility and generate a sidecar XML file.

   `shader_builder -l -x gain.glsl` tests the above shader and outputs the results in the shell, while generating the sidecar XML file named `gain.xml` . But produces the following result:

   ```
   uniform float adskUID_gain;
   vec4 adskUID_lightbox(vec4 source)
   {
           source.rgb = source.rgb * adskUID_gain;
           return vec4( source.rgb, source.a)
   }
   0(6) : error C0000: syntax error, unexpected '}', expecting ',' or ';' at token "}"
   0(2) : error C1110: function "adskUID_lightbox" has no return statement
   ```

   In this case, instead of the expected xml output, we get a compilation error: line 5 in our glsl fragment is missing an ending semi-colon. Errors must be fixed for the glsl fragment to work properly in Flame Premium. But in other cases, you will just get a compilation warning: whether such a warning can be ignored or not depends on the circumstances.

4. Fix any errors, and rerun the code through shader_builder. In the gain.glsl example, simply add a semi-colon at the end of line 5:

   ```
   return vec4( source.rgb, source.a);
   ```

5. Use the XML file generated by shader_builder to help you set up the UI of the effect. This can be especially useful if different users are going to be working with these effects. In our example, you can edit scale.xml to add default values, better names for inputs and other UI elements, and even tooltips to help the user. Updated values are in bold-italics.

   In the example below, updated values are in bold-italics.

   ```
   <ShaderNodePreset
      SupportsAdaptiveDegradation="False"
      OverrideNormals="False" CommercialUsePermitted="True"
      ShaderType="Lightbox" SoftwareVersion="2016.0.0"
      LimitInputsToTexture="True" Version="1"
   ```

```
   Description=""
   Name="Simple Gain">
   <Shader OutputBitDepth="Output" Index="1">
      <Uniform ResDependent="None"
               Max="100.00"
               Min="-100.00"
               Default="0.0" Inc="0.01"
               Tooltip="Defines the RGB multiplier."
               Row="0" Col="0" Page="0"
               Type="float"
               DisplayName="Gain"
               Name="adskUID_gain">
      </Uniform>
   </Shader>

   <Page Name="Page 1" Page="0">
      <Col Name="Effect Settings" Col="0" Page="0">
      </Col>
   </Page>
</ShaderNodePreset>
```

6. Add the .glsl and sidecar .xml file to the same directory, with an optional .png file to be used as a thumbnail in the Lightbox browser. The existing shader files are located in `/usr/discreet/presets/<application_version>/action/lightbox/`.
7. If you want to encrypt and package the shader in the .lx format, from that location, run `shader_builder -l -p <name of shader>.glsl` .
8. Try your effect in Flame or Flare.

To help you in creating custom shaders, example shaders (page 16) are available in the `/usr/discreet/presets/<application_version>/action/lightbox/EXAMPLES` .

Optional: Create a Browser Proxy Files

Along with the .glsl and .xml files that comprise a fragment shader, you can also create a .png file that displays a proxy of your effect in the Lightbox browser.

To use a .png file as the proxy:

1. Create an 8-bit, 126 pixels wide by 92 pixels high .png,
2. Place the .png file in the same folder as the .glsl and .xml files of the same name.

Forward Declaration

Lightbox requires that you forward declare your uniforms and used APIs. This ensures that shader_builder provides accurate file locations for warnings and errors.

About Lightbox Structure and the Action Shading Pipeline

Some contextual information about Action's rendering pipeline.

Action's fragment shader is split in 3 blocks:

1. Lighting loop, where shading/IBL and shadows are rendered
2. Fog
3. Blending

The Lightbox framework allows you to extend Action's lighting capabilities. By defining a function `adskUID_lightbox` in a GLSL file, Lightbox appends the shader snippet to Action's fragment shader, and can then call it within the lighting loop.

A simple Lightbox example:

```
uniform vec3 adskUID_lightColor;

vec4 adskUID_lightbox( vec4 source )
{
   vec3 resultColor = source.rgb + adskUID_lightColor;
   return vec4( resultColor, source.a );
}
```

The adskUID_ prefix is required for any global symbol in the shader (functions, uniforms, constants...) because namespaces aren't available in GLSL. You must identify uniquely every global symbol in the Lightbox code with `adskUID_` to avoid symbol clashes when the shader snippet is appended to the Action shader string, or users will not be able to load more than one copy of your Lightbox in Action; symbol clashes actually disable all loaded Lightboxes.

Also, when figuring out the input of your shader, remember that the Priority Editor in Action allows the user to reorder lights and their Lightboxes: the order in which lights are applied now matters.

Finally, keep in mind that some Lightbox options are defined in the parented light and affect the behaviour of the attached Lightbox:

Pre-/Post- When the light is active, the attached Lightboxes can render the pixel without the light effect (Pre), or the Lightboxes can render the pixel with the light effect (Post).

Lightbox Normals Even with the light inactive, the user can decide to feed the normals information to the Lightbox shader. Or not.

Additive Light/Solo Light This defines the contribution of the light and its attached Lightboxes. The default is Additive. In the API, `bool adsk_isLightAdditive()` gives you its status.

- Additive: The input of the light is the output of the light listed before it in the Priority Editor.
- Solo: The input of the light is the diffuse value, creating a "punch through" effect.

Lightbox Change Shader mix The `vec4source` is the incoming fragment rendered by the previous light or Lightbox in the shading pipeline. The return value is mixed according to the alpha returned by the function and the mix amount from the Lightbox's Shader tab. This alpha blending is the reason why every Lightbox must return a valid `source.a` . Actually, you can override the shape of the light by overriding the alpha component of `return vec4` with an alpha of your own, computed within the Lightbox.

Lightbox Processing Pipeline

Below is commented pseudocode that should help you understand the Lightbox processing pipeline.

The scene ambient light if enabled is the first light to be processed. IBL environment maps are then processed next for all other lights, the light loop compute shadings and lightbox sorted according to priority editor.

```
vec3 computeShading()
{
   vec3 shadedColor = computeSceneAmbientLight();
   shadedColor = applyIBL( shadedColor );
   for ( int i = 0; i < nbLights; ++i )
   {
```

The light UI settings are:

    [light : {active, inactive},

lightbox rendering : {pre, post lighting},
light blend mode: {additive, solo}]

The different combinations that can affect the input colors to lightbox:

1. [inactive, n/a, additive] : the current fragment (un-shaded by the current light)
2. [active, post-light, additive] : the current fragment (shaded by the current light)
3. [inactive, n/a, solo] : simply the diffuse value (un-shaded)
4. [active, post-light, solo] : the diffuse value shaded by the current light.

Compute the light alpha: GMask alpha * decay * spotlightAttenuation

```
float alpha = getLightAlpha( i );
```

The diffuse color is the diffuse material color modulated by the diffuse map.

```
vec3 lightboxInputColor = isLightSolo( i ) ? getDiffuseColor() : shadedColor;
```

Light is active and renders before lightbox

```
if ( isLightActive( i ) && isLightPre( i ) )
{
```

computeLight performs the standard (non PBR) shading equation and could be additive or solo. Solo replaces the current color with the computed light color.

```
   lightboxInputColor = computeLight( i, lightboxInputColor );
}
```

If available process lightboxes.

```
float nbLightboxes = getNbLightboxes( i );
vec4 lightboxColor = vec4( lightboxInputColor, alpha );
for ( int lx = 0; lx < nbLightboxes; ++lx )
{
   lightboxColor = computeLightbox( lx, lightboxColor );
}

vec4 lightboxResultColor = lightboxColor;
```

If the "use lightbox" option is enabled, the combined lightbox effects for this light can be applied according to the Lightbox effect.

```
if ( useLightboxNormals( i ) ) {
   lightboxResultColor.a = mix( lightboxResultColor.a,
                                lightboxResultColor.a * nDotL,
                                // you can dose the shading of a lightbox.
                                // light's lightbox panel effect numeric.
                                getLightboxEffect( i ) );
}
```

Light is active and renders after lightbox.

```
if ( isLightActive( i ) && !isLightPre( i ) )
{
   lightboxResultColor.rgb = computeLight( i, lightboxColor );
}
shadedColor = mix( lightboxInputColor, lightboxResultColor, lightboxResultColor.a );
```

```
    }
    return shadedColor;
}

vec4 computeLightbox( int lxInstance, vec4 inputColor )
{
```

adskUID is replaced by UID. Given the lx instance we find the corresponding adskUID_lightbox signature.

```
    vec4 lightboxColor = adskUID_lightbox( inputColor );
```

Each lightbox has mix numeric in its UI. Allows you to dose the lightbox color and NOT the alpha.

```
    return vec4( mix( inputColor.rgb, lightboxColor, adskUID_mix ), lightboxColor.a );
}
```

Action fragment shader.

```
void main()
{
    vec4 finalColor = vec4( computeShading(), 1.0 );
    applyFog( finalColor );
    applyBlending( finalColor );
    gl_FragColor = finalColor;
}
```

Why Create a Lightbox

Lightbox is a unique blend of concepts inherited from 3D lighting, color grading, and Photoshop adjustment layers, delivering an entirely new take on look development for images and 3D geometry. The Lightbox framework provides users the ability to define custom color lighting effects within Action, Flame's 3D compositing engine.

When working within Lightbox, defining the task to accomplish will help you measure the complexity of the task at hand.

Starting User

You want to create simple effects to relight your scene; inverse the colours of a specific object, in the scene using all the familiar light controls and behaviors, taking into account the characteristics of the object such as its specularity and normals. For example, the following Lightbox adds red colours wherever it is pointed.

```
vec4 adskUID_lightbox( vec4 source )
{
     vec3 red = vec3 (1.0, 0.0 ,0.0 );
     return vec4( source.rgb + red , source.a );
}
```

It is a very simple Lightbox, but it goes to show that you do not need complex mathematics to create a Lightbox.

Or you can leverage the available Lightbox API to change colour space, as in the following example, where the lighted area is converted from RGB to YUV.

```
vec4 adskUID_lightbox( vec4 source )
{
     vec3 converted_to_yuv = adsk_rgb2yuv( source.rgb )
     return vec4( converted_to_yuv , source.a );
}
```

But whereas Matchbox can use the MatteProvider xml property to define whether a children node should use its matte or not, Lightbox must always return a matte. And this brings us to the second type of user.

Intermediate User

You need to modify the shape or other parameters of the light, such as decay, and this means actually editing the alpha of the Lightbox.

In the example below, the Lightbox queries the Action scene to do more advanced processing while still using the Lighting framework.

```
vec3 adsk_getLightPosition(vec3 LightPos);
vec3 adsk_getVertexPosition(vec3 VertexPos);

//Used to create the near clipping plane,
//from a UI element created by the Lightbox.
uniform float adskUID_near;

//Used to create the far clipping plane,
//from a UI element created by the Lightbox.
uniform float adskUID_far;

//This value is from a UI element created by the Lightbox.
uniform vec3 adskUID_tint;
vec4 adskUID_lightbox( vec4 source )
{
    vec3 colour = source.rgb;

  //We are using the Light position and vertex position information
  //to determine the distance in between the two to then allow
  //to modulate the color based on Light versus vertex distance
    float alpha = clamp(length(adsk_getLightPosition() - adsk_getVertexPosition())
                  / (adskUID_far-adskUID_near), 0.0, 1.0);
    vec3 result = colour * adskUID_tint;
    colour = mix(result, colour, alpha);

    return vec4 (colour, source.a);
}
```

In another example, a simple grayscale effect is applied to a target offset, defined by the end-user using adskUID_targetPosition.

```
vec3    adsk_getVertexPosition();
float   adsk_getLuminance( vec3 rgb );
uniform float adskUID_distanceMax;
uniform vec3 adskUID_targetPosition;

vec4 adskUID_lightbox( vec4 source )
{
    vec3 vertexPos = adsk_getVertexPosition();

  // Get the radius of the effect
    float distMax = max( adskUID_distanceMax, 1.0 );

  // Get the distance from the centre of the effect
    float dist = length( vertexPos - adskUID_targetPosition );
    float alpha;
    if ( dist > distMax )
        alpha = 0.0; // outside the range of the effect
    else
        alpha = clamp( dist / distMax, 0.0, 1.0 );
  //The effect is simply a conversion to grayscale
    vec3 rgbOut = vec3( adsk_getLuminance(source.rgb) );

    //Since we edited the alpha in this Lightbox, we need to make sure
  //not to override the alphas that was calculated by the previous
  //light in the shading pipeline.
    return vec4( rgbOut, source.a * alpha );
}
```

NOTE Be careful to not white out the alpha, or you will lose the whole scene information whenever you use that Lightbox. Unless you are an advanced user, whenever you edit the alpha, make sure to always multiply your result by the entering alpha, as in the example above: `source.a * alpha`.

Advanced User

The advanced user wants to actually modify the rendering pipeline, ignoring all that was computed upstream from the light, and substitute their own render calculations. This is possible because lighting calculations happen at the end of the rendering pipeline, and means . While this is an interesting proposition that allows truly unique renders, it is also a very expensive one, computationally speaking. The Action render pipeline is static, and substituting your own methods by using a Lightbox essentially means rendering a scene twice: once by Action, and once again by the Lightbox.

While creating such a Lightbox is possible, it is way beyond the scope of this document. But there is an expansive, un-optimized example available, providing an alternative to the current IBL implementation. See `/usr/discreet/presets/<application_version>/action/lightbox/EXAMPLES/GGXIBLExample.glsl`.

Lightbox Shader Examples

To help you in creating custom Lightbox shaders, examples are available in the `/usr/discreet/presets/<application_version>/action/lightbox/EXAMPLES` .

The following example Lightbox shaders are available:

GGXIBLExample Similar to GGXMaterial, but it doesn't hijack the Material node, and all controls are present in the Lightbox itself. Also it doesn't support object texture and only a BaseColor which is set in the Lightbox itself.

GGXMaterial This shader performs a rendering of the scene using a physically based shading algorithm which requires three distinct values from each material in the scene; the Specular, Metallic, and Roughness component values. These components are not available natively in the Material settings, so they are mapped as followed in the GGXMaterial Lightbox shader: Specular Red Channel for the Specular value, the Specular Green Channel for the Metallic value and the Shine parameter to define the Roughness.

LightboxAPISimple This shader shows all possible API calls to the Lightbox API and shows a simple decay example using vertex and light position.

LightboxBasics This is the most basic usage of Lightbox, only applying a Gain without having to use the API at all.

SimpleLight This is an example re-creating an Action light in Ligthbox using the API.

About shader_builder XML

4

In its most basic form, a Matchbox or Lightbox is only a .glsl file that contains all the shader information. That single file is enough for the application to generate both the shading result and a rough UI for the end user to manipulate the shader parameters.

But to create a useful UI, you need a sidecar .xml file. This file is generated by the running shader_builder with the `-x` switch. Once it's been created by shader_builder, you can edit the xml to tweak the UI to your needs, such as substituting curves for numeric fields.

NOTE Any uniform with a name that starts with "adsk_" is not shown in the user interface and not considered by shader_builder.

XML Structure

```
<!DOCTYPE ShaderNodePreset [

<!ELEMENT ShaderNodePreset (Shader+, Page+)>
<!ELEMENT Shader (Uniform+)>
<!ELEMENT Uniform (SubUniform* | PopEntry* | Duplicate*)>
<!ELEMENT SubUniform EMPTY>
<!ELEMENT PopEntry EMPTY>
<!ELEMENT Duplicate EMPTY>
<!ELEMENT Page (Col+)>
<!ELEMENT Col EMPTY>
]>
```

<ShaderNodePreset>

Parent

- None. ShaderNodePreset is the root element.

Children

- <Shader> : At least one for a Matchbox. Only one for a Lightbox . See <Shader> usage definition.
- <Page>: At least one.

Usage

- Root element. Mandatory.

Attribute	Description	Allowed Values	Default
Description	A description of the Preset. The description appears in the Matchbox node Note available from the Schematic.	Character data	EMPTY
Name	The name of the Preset. The name appears in the Name field of the Node interface. The Name attribute overrides the filename, so if the .xml file is present, even if the user loads the .glsl file, the Name attribute is displayed in the UI.	Character data	Name of the preset
HelpLink	Allows you to specify an URL that can be used for external documentation (only valid XML characters are allowed in the URL). A Help button is added to the Matchbox interface when this token is detected.	Character data	EMPTY
LimitInputsToTexture	Allows you to limit the number of Input Sockets on the Matchbox node to the number of textures required by your shader.	■ True ■ False	False
MatteProvider	*Matchbox only* Specifies if the Matchbox shader provides a matte to its child node, or if it is a pass through, where the Matchbox's child node uses the matte from the Matchbox's parent. If the Matchbox operates on the alpha and you want that alpha to be passed on the child node, you must set MatteProvider = True.	■ True ■ False	False
SupportsAction	*Matchbox only* Hints to the usage of the Matchbox, in this case as an Action node. Batch is always supported.	■ True ■ False	False
SupportsTransition	*Matchbox only* Hints to the usage of the Matchbox, in this case as a transition in	■ True ■ False	False

Attribute	Description	Allowed Values	Default
	the timeline. Batch is always supported.		
SupportsTimeline	*Matchbox only* Hints to the usage of the Matchbox, in this case the timeline. Batch is always supported.	■ True ■ False	False
OverrideNormals	*Lightbox only* Flag that specifies if the geometry of the normals affects the effect's intensity.	■ True ■ False	False
ShaderType	Defined by shader_builder, determines the type of shader, Lightbox or Matchbox.	Matchbox / Lightbox	Value depends on the .glsl file used to create the XML.
SoftwareVersion	Defined by shader_builder, indicates the version of the application used to build the .xml.	Character data	shader_builder defined
SupportsAdaptiveDegradation	*Matchbox only* Defines if the Matchbox node supports Batch adaptive degradation. Use in combination with the adsk_degrade uniform to create a less costly data path, or even completely bypass the Matchbox to enhance Batch interactivity. Note that the Matchbox must be coded with alternate paths using the adsk_degrade for adaptive degradation to work: simply setting SupportsAdaptiveDegradation = True does not make the node turn off when adaptive degradation is turned on.	■ True ■ False	False
TimelineUseBack	*Matchbox only* Defines if the Matchbox uses the background of the timeline (True), or not (False). Only used in context of the timeline.	■ True ■ False	True
Version	Version of shader. Metadata which is user-defined and not processed by the application.	Character data	1
CommercialUsePermitted	Indicates if the shader should be used for commercial purposes.	■ True ■ False	True

Attribute	Description	Allowed Values	Default
	This attribute is not enforced by the application. Set by the creator of the shader to indicate if the shader can be used for commercial purposes. The user of the shader is responsible for verifying this attribute and using the shader accordingly.		

<Shader>

Parent

- <ShaderNodePreset>

Children

- <Uniform> : At least one.

Usage

- Mandatory.
- There is one <Shader> for each shader pass defined in the .glsl file: a Matchbox can have multiple shader passes, but a Lightbox can only ever have one. <Shader> lists the uniforms required for the associated shader pass.

Attribute	Description	Allowed Values	Default
Index	The index of the associated shader pass. Indexed from 1.	Integer	1
OutputBitDepth	Bit depth of the render target for this render pass.	■ Output ■ 8 ■ 16 ■ Float16 ■ Float32	Output

<Uniform>

Parent

- <Shader>

Children

- <SubUniform> : As many as required by the uniform, as defined by the `Type` attribute, and if `ValueType <> PopUp`.
- <PopEntry> : Only if attribute `ValueType = PopUp`, in which case there can be as many as required.
- <Duplicate>: Only one, only if parent <Uniform> is a duplicate. See `<Duplicate>` definition.

Usage

- One for each uniform in the current pass.

Attribute	Description	Allowed Values
Type	Type of uniform.	float vec2 vec3 vec4 int ivec2 ivec3 ivec4 bool bvec2 bvec3 bvec4 mat2 mat3 mat4 mat2x3 mat2x4 mat3x2 mat3x4 mat4x2 mat4x3 sampler2D
Tooltip	The text for this uniform tooltip.	Character data
DisplayName	The name displayed for this uniform in the user interface.	Character data
Name	The name of the uniform used in the GLSL file, set by shader_builder. It cannot be changed as this is used to indicate which uniform in the .glsl file uses this UI element.	Character data
Row	The row where the uniform is positioned in the UI.	0-4
Col	The column where the uniform is positioned in the UI.	0-3
Page	The page where the uniform is positioned in the UI.	0-6
Default	The default value of this uniform.	■ Values allowed based on type of the uniform: ■ True or False: bool, bvec2, bvec3, bvec4 ■ An Integer: int, ivec2, ivec3, ivec4 ■ A Float: float, vec2, vec3, vec4, mat2, mat3, mat4, mat2x3, mat2x4, mat3x2, mat3x4, mat4x2, mat4x3
Inc	The stepping used when dragging in the uniform.	■ Values allowed based on type of the uniform: ■ An Integer: int, ivec2, ivec3, ivec4 ■ A Float: float, vec2, vec3, vec4, mat2, mat3, mat4, mat2x3, mat2x4, mat3x2, mat3x4, mat4x2, mat4x3
Max	Maximum allowed value for the uniform.	■ Values allowed based on type of the uniform: ■ An Integer: int, ivec2, ivec3, ivec4

Attribute	Description	Allowed Values
		■ A Float: float, vec2, vec3, vec4, mat2, mat3, mat4, mat2x3, mat2x4, mat3x2, mat3x4, mat4x2, mat4x3
Min	Minimum allowed value for the uniform.	■ Values allowed based on type of the uniform: ■ An Integer: int, ivec2, ivec3, ivec4 ■ A Float: float, vec2, vec3, vec4, mat2, mat3, mat4, mat2x3, mat2x4, mat3x2, mat3x4, mat4x2, mat4x3
IconType	*vec2/vec3 only* Defines the type of icon used to control the uniform in the Viewport. **NOTE** Only "None" and "Pick" are available in the Timeline Player. "Light" and "Axis" are automatically replaced by Pick in that case.	■ None ■ Pick (Pick in the Viewport) ■ Axis (Axis icon in the Viewport) ■ Light (Light icon in the Viewport)
ResDependant	*float/vec2/vec3/vec4 only* Defines which resolution dependance this uniform has, if any.	■ None ■ Width (values are scaled based on width) ■ Height (values are scaled based on height)
ValueType	*vec3, ivec2, ivec3, ivec4 and int only* XYZ/RGB/Popup modes.	■ Position: Values are presented as numerics. ■ Colour: Values are presented as a colour, with a colour pot. ■ ColourWheel: Values are presented as a HGS colour wheel. You can also name the channels with the tokens: AngleName, IntensityName1, IntensityName2. The bool HueShift tag controlls if the outer wheel rotates with the chosen Hue. ■ PopUp: Values are presented as a popup menu. You must add a <PopupEntry> element within this <Uniform> to specify the names and their associated Int values. ■ Curve: Values are presented as curves. The int (ivec2, ivec3 or ivec4) value is a handle on 1 (2,3 or 4) curve(s). The curve(s) can be evaluated using the function adskEvalDynCurves taking as arguments the handle and a float (vec2, vec3 or vec4) representing the point to evaluate and returning a float (vec2, vec3 or vec4) representing the output(s) of the curve(s)

Attribute	Description	Allowed Values
Index	*sampler2D only* The position of the sampler in the user interface.	Integer
GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER	*sampler2D only* Interpolation modes.	■ GL_NEAREST ■ GL_LINEAR ■ GL_NEAREST_MIPMAP_NEAREST ■ GL_LINEAR_MIPMAP_NEAREST ■ GL_NEAREST_MIPMAP_LINEAR ■ GL_LINEAR_MIPMAP_LINEAR
GL_TEXTURE_WRAP_T, GL_TEXTURE_WRAP_S	*sampler2D only* Wrapping modes.	■ GL_CLAMP ■ GL_CLAMP_TO_BORDER ■ GL_CLAMP_TO_EDGE ■ GL_REPEAT ■ GL_MIRRORED_REPEAT
NoInput	*sampler2D only* Defines the behaviour of a missing input.	■ Error (Generates an error) ■ Black (Replaces missing input with Black) ■ White (Replaces missing input with White)
Mipmaps	*sampler2D only* Enable to use mipmaps on this sampler. GL_TEXTURE_MIN_FILTER must be set to one of the following: GL_NEAREST_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_LINEAR.	■ True ■ False
InputType	Specifies the colour of an Input socket on the Matchbox node.	■ Front (Red) ■ Back (Green) ■ Matte (Blue)
InputColor	Allows you to specify a custom RGB colour. If InputType is also specified, InputType takes precedence over InputColor. Refer to `/usr/discreet/presets/<application_version>/matchbox/shaders/EXAMPLES/InputSockets.xml` for an example.	r,g,b format, where each component is 0-255.

Attribute	Description	Allowed Values
UIConditionSource	The name of the uniform used to conditionally disable or hide the current uniform in the user interface.	Name of a uniform
UIConditionValue	The value of the `UIConditionSource` required for the uniform to be active.	Character data
UIConditionInvert	Allows you to invert the logic of `UIConditionValue`.	■ True ■ False
UIConditionType	Allows you to choose between disabling or hiding the current uniform, based on the `UIConditionSource` status. Refer to `/usr/discreet/presets/<application_version>/matchbox/shaders/EXAMPLES/ConditionalUI.xml` for an example.	■ Disable ■ Hide

The following attributes are only used to define a curve-based UI, when `<Uniform ValueType = Curve>`. For an example in context, see `/usr/discreet/presets/<application_version>/matchbox/shaders/EXAMPLES/Curves.glsl`.

Attribute	Description	Allowed Values	Default
CurveMinX, CurveMaxX, CurveMinY, CurveMaxY	Define the x/y range of the curves.	Float.	■ CurveMinX and CurveMinY: 0.0 ■ CurveMaxX and CurveMaxY: 1.0
CurveBackground	Defines the type of background for the curve editor.	■ 0 : empty background ■ 1 : hue gradient background ■ 2 : intensity gradient background	0
CurveWrapArround	Defines the behaviour of the curve(s) at the border.	■ 0 : no wraparound ■ 1 : wraparound	0
CurveShape	Defines the default shape of the curves.	■ 0 : linear shape from (CurveMinX,CurveMinY) to (CurveMaxX,CurveMaxY). ■ 1 : linear shape from (CurveMinX,CurveMaxY) to (CurveMaxX,CurveMinY). ■ 2 : S shape. 3 : inverse S shape.	0

Attribute	Description	Allowed Values	Default
		- 4 : constant with MaxY value. - 5 : constant with MinY value. - 6 : constant with MinY + 0.5*(MaxY-MinY) value.	
CurveR	Defines the R colour of all the curves attached to this uniform.	Integer	0
CurveG	Defines the G colour of all the curves attached to this uniform.	Integer	0
CurveB	Defines the B colour of all the curves attached to this uniform.	Integer	0

<SubUniform>

Parent

- <Uniform>

Child

- EMPTY

Usage

- There is one `<SubUniform>` element for each value required by the `<Uniform>` parent, based on the Value attribute: `vec3` requires 3 `<SubUniform>`, `float` requires none. How this gets translated in the UI depends on the `ValueType` defined: Colour displays a colour pot, which means that the SubUniform defines the starting RGB values for that uniform, whereas `Position` with a `vec3` displays three numeric fields defined by the attributes of 3 `<SubUniform>`.

Attribute	Description	Allowed Values
Default	The default value of the parent uniform.	- Values allowed based on type of the uniform: - True or False: bool, bvec2, bvec3, bvec4 - An Integer: int, ivec2, ivec3, ivec4 - A Float: float, vec2, vec3, vec4, mat2, mat3, mat4, mat2x3, mat2x4, mat3x2, mat3x4, mat4x2, mat4x3
Inc	The stepping used when dragging in the SubUniform.	- Values allowed based on type of the uniform: - An Integer: int, ivec2, ivec3, ivec4 - A Float: float, vec2, vec3, vec4, mat2, mat3, mat4, mat2x3, mat2x4, mat3x2, mat3x4, mat4x2, mat4x3

Attribute	Description	Allowed Values
Max	Maximum allowed value for the SubUniform.	■ Values allowed based on type of the uniform: ■ An Integer: int, ivec2, ivec3, ivec4 ■ A Float: float, vec2, vec3, vec4, mat2, mat3, mat4, mat2x3, mat2x4, mat3x2, mat3x4, mat4x2, mat4x3
Min	Minimum allowed value for the SubUniform.	■ Values allowed based on type of the uniform: ■ An Integer: int, ivec2, ivec3, ivec4 ■ A Float: float, vec2, vec3, vec4, mat2, mat3, mat4, mat2x3, mat2x4, mat3x2, mat3x4, mat4x2, mat4x3
ResDependant	*float/vec2/vec3/vec4 only* Defines which resolution dependance this uniform has, if any.	■ None Width (values are scaled based on width) Height (values are scaled based on height)

The following attributes are only used to define a curve-based UI, when `<Uniform ValueType = Curve>`. These values override the more general values from the parent <Uniform>.

Attribute	Description	Allowed Values	Default
CurveMinX, CurveMaxX, CurveMinY, CurveMaxY	Define the x/y range of the curves.	Float.	■ CurveMinX and CurveMinY: 0.0 ■ CurveMaxX and CurveMaxY: 1.0
CurveBackground	Defines the type of background for the curve editor.	■ 0 : empty background ■ 1 : hue gradient background ■ 2 : intensity gradient background	0
CurveWrapArround	Defines the behaviour of the curve(s) at the border.	■ 0 : no wraparound ■ 1 : wraparound	0
CurveShape	Defines the default shape of the curves.	■ 0 : linear shape from (CurveMinX,CurveMinY) to (CurveMaxX,CurveMaxY).	0

Attribute	Description	Allowed Values	Default
		■ 1 : linear shape from (CurveMinX,CurveMaxY) to (CurveMaxX,CurveMinY). ■ 2 : S shape. 3 : inverse S shape. ■ 4 : constant with MaxY value. ■ 5 : constant with MinY value. ■ 6 : constant with MinY + 0.5*(MaxY-MinY) value.	
CurveName	The name of the curve represented by this SubUniform.	Character data	
CurveR	Defines the R colour of this SubUniform's curve.	Integer	0
CurveG	Defines the G colour of this SubUniform's curve.	Integer	0
CurveB	Defines the B colour of this SubUniform's curve.	Integer	0

<Duplicate>

Parent

- <Uniform>

Children

- EMPTY

Usage

- When a uniform is present in more than one shader pass, it is only shown once in the user interface. This is indicated by the `<Duplicate>` element.

  For instance, a two-pass preset uses a bool uniform named filtering. The XML file defines it in the first `<Shader>` element:

  ```
  <Uniform Row="0" Col="0" Page="0" Default="True" Tooltip="" DisplayName="Filtering"
  Type="bool" Name="filtering">
  </Uniform>
  ```

  In the second `<Shader>` element, it is defined as a duplicate.

  ```
  <Uniform Type="bool" Name="filtering">
  <Duplicate>
  </Duplicate>
  </Uniform>
  ```

  If the second uniform isn't defined as a duplicate, it appears twice in the node UI.

Attributes

- NONE

<PopEntry>

Parent

- <Uniform>

Children

- EMPTY

Usage

- Only used if `<Uniform ValueType = Popup>`. Each `<PopEntry>` is an entry of the drop-down list.

```
<PopupEntry Title="ReplaceMe" Value="0">
</PopupEntry>
```

Refer to `/usr/discreet/presets/<application_version>/matchbox/shaders/EXAMPLES/BuildList.xml` for an example.

Attribute	Description	Allowed Values
Title	The entry displayed in the drop-down list.	Character data
Value	The actual value passed to the uniform.	Character data

<Page>

Parent

- <ShaderNodePreset>

Children

- <Col> : Up to 4 children.

Usage

- Defines the name of a page.

Attribute	Description	Allowed Values	Default
Name	Name of the page to be displayed in the UI.	Character data	Page 1
Page	Index of the page.	Indexed from 0, max value of 6.	0

<Col>

Parent

- <Page>

Children

- EMPTY

Usage

- Defines the name of a column within a page.

Attribute	Description	Allowed Values	Default
Name	Name of the column to be displayed in the UI.	Character data	Column 1
Col	Index of the column.	Indexed from 0, max value of 3.	0
Page	Index of the <Page> where the column appears.	Indexed from 0, max value of 6.	0

Shader API Documentation

5

In the following API, all positions and directions are in camera space: their local positions are multiplied by the modelView matrix. Apart from that, most of the calls are self- explanatory or are mapped directly to the UI.

NOTE Any uniform with a name that starts with `adsk_` is not shown in the user interface.

Lighting and Shading (Lightbox only)

vec3 adsk_getNormal(); Returns unaltered interpolated normal, before any maps are applied. Use adsk_getComputedNormal to get the value computed after the maps. The normal, binormal and tangent form an orthonormal basis where the origin is at the vertex position.

vec3 adsk_getComputedNormal(); Returns the computed normal once the maps are applied. Use adsk_getNormal to get the value before the maps.

vec3 adsk_getBinormal(); Returns the interpolated binormal.

vec3 adsk_getTangent(); Returns the interpolated tangent.

vec3 adsk_getVertexPosition(); Returns the interpolated vertex position for the current fragment after all maps have been processed.

vec3 adsk_getCameraPosition(); Returns the camera position.

bool adsk_isLightActive(); Returns the activity status of the parent light.

bool adsk_isLightAdditive(); Returns TRUE if the light attached to the Lightbox is set as additive, or FALSE if the light is set to solo.

vec3 adsk_getLightPosition(); Returns the position vector of the light attached to the Lightbox. ??This is not a position vector, right? It is not directional, it is only a point in space, just a position without vector.

vec3 adsk_getLightColour(); Returns the colour of the light parented to the Lightbox, but modulated by the intensity (lightColour * intensity).

vec3 adsk_getLightDirection(); Returns the direction of the light attached to the Lightbox, if that light is directional. The direction is computed whether the light is in target or free mode.

vec3 adsk_getLightTangent(); Given the light direction, the light tangent vector is an orthogonal vector where the origin is at the light position. For example, given the light direction looking down negative z-axis. The light tangent would be the x-axis.

void adsk_getLightShadowDecayType (out int lightDecayType, out int shadowDecayType); Returns the type of decay selected by the user for the light parented to the Lighbox, using Profile > Decay Type box.

- const int NO_DECAY = 0;

- const int LINEAR_DECAY = 1;
- const int QUADRATIC_DECAY = 2;
- const int CUBIC_DECAY = 3;
- const int EXP_DECAY = 4;
- const int EXP2_DECAY = 5;

float adsk_getLightDecayRate(); Returns the decay rate selected by the user for the light parented to the Lighbox, using Profile > Decay field.

bool adsk_isSpotlightFalloffParametric(); Returns TRUE if the user set Profile > Falloff Model box to Parametric for the light parented to the Lightbox. Returns FALSE if the user set the Falloff Model box to Custom.

float adsk_getSpotlightParametricFalloffIn(); Returns the value selected by the user for Profile > Falloff In field. Only enabled if adsk_isSpotlightFalloffParametric returns TRUE.

float adsk_getSpotlightParametricFalloffOut(); Returns the value selected by the user for Profile > Falloff Out field. Only enabled if adsk_isSpotlightFalloffParametric returns TRUE.

float adsk_getSpotlightSpread(); Returns the spread angle selected by the user for Basics > Spread field.

float adsk_getLightAlpha(); Returns the precomputed alpha of the light attached to the Lightbox, calculated from its cutoff, decay, and GMask.

bool adsk_isPointSpotLight(); Returns TRUE if the user sets Basics > Light Type box to Point/Spot for the light parented to the Lightbox.

bool adsk_isDirectionalLight(); Returns TRUE if the user sets Basics > Light Type box to Directional for the light parented to the Lightbox .

bool adsk_isAmbientLight(); Returns TRUE if the user sets Basics > Light Type box to Ambient for the light parented to the Lightbox .

bool adsk_isAreaRectangleLight(); Returns TRUE if the user sets Basics > Light Type box to Rectangle Area for the light parented to the Lightbox .

bool adsk_isAreaEllipseLight(); Returns TRUE if the user sets Basics > Light Type box to Ellipse Area for the light parented to the Lightbox.

float adsk_getAreaLightWidth(); Returns the width of the light attached to the Lightbox. Only applicable if adsk_isAreaRectangleLight or adsk_isAreaEllipseLight is TRUE.

float adsk_getAreaLightHeight(); Returns the height of the light attached to the Lightbox. Only applicable if adsk_isAreaRectangleLight or adsk_isAreaEllipseLight is TRUE.

bool adsk_isLightboxRenderedFromDiffuse(); Returns FALSE if the Lightbox is rendered from the currently lit fragment. Returns TRUE if Lightbox is being applied on the unlit raw diffuse value.

bool adsk_isLightboxRenderedBeforeLight(); Returns FALSE if the Lightbox is rendered after the attached light.

vec3 adsk_getComputedDiffuse(); Returns the computed diffuse value for the current fragment, before shading but after all the maps have been processed.

float adsk_getShininess(); Returns the shininess material property.

vec3 adsk_getComputedSpecular(); Returns the computed specular value for the current fragment, before shading but after all the maps have been processed.

Map Access (Lightbox only)

The getMapValue functions give raw access to the texture maps. The getMapCoord functions return the interpolated coordinates used to fetch in those texture maps. The coordinates need to be safe divided (to avoid divisions by 0) by the returned z value to get a UV texture coordinate.

vec4 adsk_getComputedDiffuseMapValue (in vec3 vertexPos); Returns the computed diffuse map value once all maps have been processed.

The following return the raw map values. Use the getMapCoord functions below to compute the texture map coordinates (texCoord) required for the current fragment.

vec4 adsk_getDiffuseMapValue (in vec2 texCoord);
vec4 adsk_getEmissiveMapValue (in vec2 texCoord);
vec4 adsk_getSpecularMapValue (in vec2 texCoord);
vec4 adsk_getNormalMapValue (in vec2 texCoord);
vec4 adsk_getReflectionMapValue (in vec2 texCoord);
vec4 adsk_getUVMapValue (in vec2 texCoord);
vec4 adsk_getParallaxMapValue (in vec2 texCoord);

The following return the interpolated coordinates that will need to be safe divided by z to fetch the corresponding map value for the current fragment.

vec3 adsk_getDiffuseMapCoord();
vec3 adsk_getEmissiveMapCoord();
vec3 adsk_getSpecularMapCoord();
vec3 adsk_getNormalMapCoord();
vec3 adsk_getParallaxMapCoord();

vec2 adsk_getReflectionMapCoord(in vec3 vrtPos, in vec3 normal);

Transforms (Lightbox only)

mat4 adsk_getModelViewMatrix(); Converts local coordinates to camera space.

mat4 adsk_getModelViewInverseMatrix(); Converts camera space to local coordinates.

Image Based Lighting (Lightbox only)

NOTE There is significant performance degradation with shaders that use an IBL-based API call if that shader is used in combination with Hardware Anti-Aliasing and high resolution maps. To mitigate this, disable HWAA, or use lower-resolution maps.

int adsk_getNumberIBLs(); Returns the supported the number of IBL maps.

bool adsk_isCubeMapIBL(in int idx); Returns TRUE of the IBL map with the `idx` index is Cubic. FALSE if the map is Angular or Cylindrical. If both `adsk_isAngularMapIBL` and `adsk_isCubeMapIBL` are FALSE, the map is Cylindrical.

bool adsk_isAngularMapIBL(in int idx); Returns TRUE of the IBL map with the `idx` index is Angular. FALSE if the map is Cubic or Cylindrical. If both `adsk_isAngularMapIBL` and `adsk_isCubeMapIBL` are FALSE, the map is Cylindrical.

vec3 adsk_getCubeMapIBL(in int idx, in vec3 coords, float lod); Where lod is the Level of Detail of the IBL with the `idx` index.

vec3 adsk_getAngularMapIBL(in int idx, in vec2 coords, float lod); Where lod is the Level of Detail of the IBL with the `idx` index.

bool adsk_isAmbientIBL(in int idx); Returns TRUE if the IBL mapping option is set to Ambient, FALSE if it is set to Reflection.

float adsk_getIBLDiffuseOffset(in int idx); The Diffuse Offset defined in the UI for the IBL with the `idx` index.

mat4 adsk_getIBLRotationMatrix(in int idx); The rotation matrix defined in the UI for the IBL with the `idx` index.

Material Information (Lightbox only)

vec4 adsk_getMaterialDiffuse(); Material node's Diffuse property.

vec4 adsk_getMaterialSpecular(); Material node's Specular property.

vec4 adsk_getMaterialAmbient(); Material node's Ambient property.

Colour Management and Colour Space Conversion (Lightbox and Matchbox)

The following return the value of a curve uniform in the UI (dynCurveId) read at x.

float adskEvalDynCurves(in int dynCurveId, in float x);
vec2 adskEvalDynCurves(in ivec2 dynCurveId, in vec2 x);
vec3 adskEvalDynCurves(in ivec3 dynCurveId, in vec3 x);
vec4 adskEvalDynCurves(in ivec4 dynCurveId, in vec4 x);

vec3 adsk_scene2log(in vec3 src); Converts src from scene linear colour space to Cineon log colour space. Inverse operation of adsk_log2scene.

vec3 adsk_log2scene(in vec3 src); Converts src from Cineon log colour space to scene linear colour space. Inverse operation of adsk_scene2log.

vec3 adsk_rgb2hsv(in vec3 src); Converts src from RGB colour space to HSV colour space. Inverse operation of adsk_hsv2rgb.

vec3 adsk_hsv2rgb(in vec3 src); Converts src from HSV colour space to RGB colour space. Inverse operation of adsk_rgb2hsv.

vec3 adsk_rgb2yuv(in vec3 src); Converts src from RGB colour space to YUV colour space. Inverse operation of adsk_yuv2rgb.

vec3 adsk_yuv2rgb(in vec3 src); Converts src from YUV colour space to RGB colour space. Inverse operation of adsk_rgb2yuv.

vec3 adsk_getLuminanceWeights();

float adsk_getLuminance(in vec3 color);

float adsk_highlights(in float pixel, in float halfPoint); Parametric blending curves suitable for blending effects for Shadows, Midtones, Highlights. The parameters are:

- pixel: the colour of the pixel.
- halfpoint: the X-axis location where the curve is 50% .

To get the midtone weight, you compute for the same pixel colour: `1 - adsk_shadows - adsk_highlights`

float adsk_shadows(in float pixel, in float halfPoint); Parametric blending curves suitable for blending effects for Shadows (of the Shadows-Midtones-Highlights triumvirate). The parameters are:

- pixel: the colour of the pixel.
- halfpoint: the X-axis location where the curve is 50%.

To get the midtone weight, you compute for the same pixel colour: `1 - adsk_shadows - adsk_highlights`

Shadows (Lightbox only)

vec3 adsk_getComputedShadowCoefficient(); Returns a coefficient which is computed from all the shadow parameters. The coefficient needs to be multiplied against the light colour. The parent light must be a shadow caster. See the example `/action/lightbox/EXAMPLES/SimpleLight.glsl`.

Blending (Lightbox and Matchbox)

vec4 adsk_getBlendedValue(int blendType, vec4 srcColor, vec4 dstColor); Blends srcColor and dstColor using the blendType.

- Add = 0
- Sub = 1
- Multiply = 2
- LinearBurn = 10
- Spotlight = 11
- Flame_SoftLight = 13
- HardLight = 14
- PinLight = 15
- Screen = 17
- Overlay = 18
- Diff = 19
- Exclusion = 20
- Flame_Max = 29
- Flame_Min = 30
- LinearLight = 32
- LighterColor = 33

General Purpose (Lightbox only)

bool adsk_isSceneLinear(); Return TRUE if the Action scene is in linear colour space, or FALSE if Log.

General Purpose (Lightbox and Matchbox)

float adsk_getTime(); Returns the current time as a frame number. 1-based float.

General Purpose (Matchbox only)

The following float uniforms are unconditionally sent to each pass. You can use them in the shader you are creating, if needed. Unless indicated otherwise, the following uniforms are only available to Matchbox shaders.

The following can be used to set the result resolution.

adsk_result_w
adsk_result_h
adsk_result_frameratio
adsk_result_pixelratio

adsk_results_pass{pass_index}(); Can be used to refer to a previous pass result texture. For instance, adsk_results_pass1 refers to the first pass result and can be used in any subsequent pass. the sampler must be declared and used normally.

adsk_previous_frame_{sampler_name}(); adsk_next_frame_{sampler_name}(); Not available if the Matchbox is loaded in Action.

Can be used to fetch the texture of the previous or next {sampler_name} frame. All uniform sampler parameters for previous and next will come from the {sampler_name} uniform xml entry. For example, to use both current and previous frames given a sampler input1 the XML entries would look like this.

```
<Uniform Type="sampler2D" Name="adsk_previous_frame_input1"> xml entry.
<Uniform Index="0" NoInput="Error" Tooltip="" DisplayName="input1" Mipmaps="False"
GL_TEXTURE_WRAP_T="GL_REPEAT" ....>
```

NOTE Requires that the sampler "sampler_name" for the current frame be declared and used in the fragment shader. Refer to /usr/discreet/presets/<application_version>/matchbox/shaders/EXAMPLES/TemporalSampling.xml for an example.

bool adsk_degrade(); This boolean uniform can be used to obtain the process degradation setting. When True, a degraded frame was requested. When False, a full rendered frame or a preview frame was requested.

float adsk_{sampler_name}_w (); float adsk_{sampler_name}_h (); This uniform gives access to each texture's resolution. For instance, adsk_frontTex_w & adsk_frontTex_h refer to the resolution of the sampler2D named frontTex.

float adsk_{sampler_name}_frameratio This gives access to texture frame ratio. For instance, adsk_frontTex_frameratio refers to the frame ratio of the sampler2D named frontTex.

float adsk_{sampler_name}_pixelratio This gives access to texture pixel ratio. For instance, adsk_frontTex_pixelratio refers to the pixel ratio of the sampler2D named frontTex.