Real-Time Ray-Tracing Techniques for Integration into Existing Renderers

TAKAHIRO HARADA, AMD

3/2018

AGENDA

AMD

- Radeon ProRender, Radeon Rays update
- Unity GPU Lightmapper using Radeon Rays (by Jesper)
  - Helping the game content creator to make better assets
- Radeon ProRender + Universal Scene Description
  - Real-time preview of assets
- Radeon ProRender Real-time Rendering
  - Hybrid ray tracing is a stepping stone to a fully ray traced future, as the same path was followed with production movie rendering. Our solution provides a was to fully path traced rendering with Radeon pro render

RADEON PRORENDER, RADEON RAYS

AMD's Ray tracing solutions

RADEON PRORENDER, RADEON RAYS

AMD'S RAY TRACING SOLUTIONS

- Radeon ProRender
  - A complete renderer (ray casting, shading)
  - Physically based rendering library
  - Output - Rendered image
  - For renderer users, or developers

- Radeon Rays
  - For developers
  - Ray intersection library
  - Output - Intersections

RADEON PRORENDER

- For developers
  - SDK available today on request
  - Bruno.Stefanizzi@amd.com
- C API
- OpenCL 1.2, Metal 2
- Multi platform solution
  - OS (Windows, MacOs, Linux)
  - Vendors (AMD,…)
- For content creators
  - https://pro.radeon.com/en/software/prorender/
- Plugins
  - Maya, 3DS Max, Blender, Rhino, SolidWorks
- Direct integration
  - Cinema4D (Maxon, R19~)
  - Modo (The Foundry, Beta)

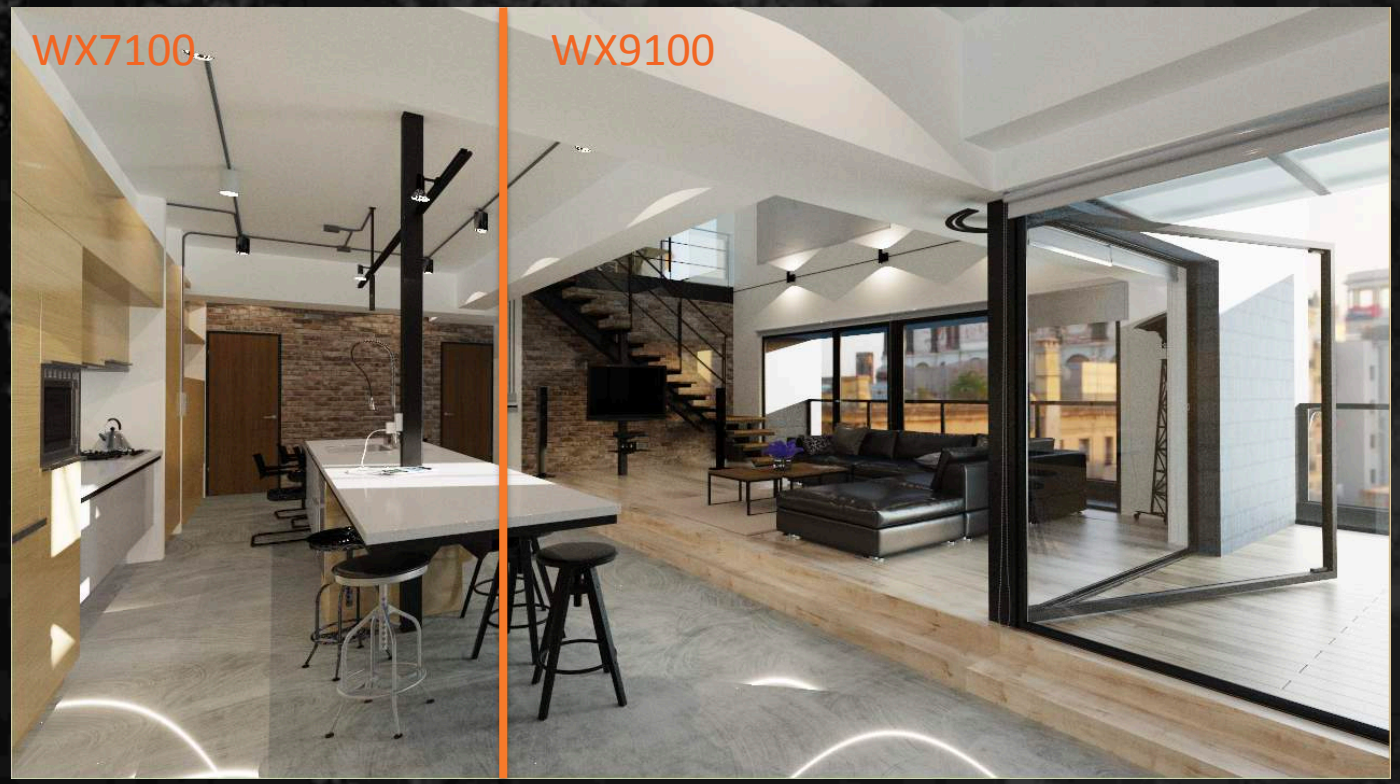
RPR for Blender on MacOs

(BMW from Mike Pan)

RADEON PRORENDER

FEATURE HIGHLIGHTS

- Heterogeneous device support
  - GPU+GPU, GPU+CPU
  - Less latency for interactive render
- No limit for the texture usage
  - Out of core texture
  - Use system memory or disk
- MacOs Metal support
  - Maya, Blender betas, C4D available today
  - https://pro.radeon.com/en/radeon-prorender-macos-get-beta-now/



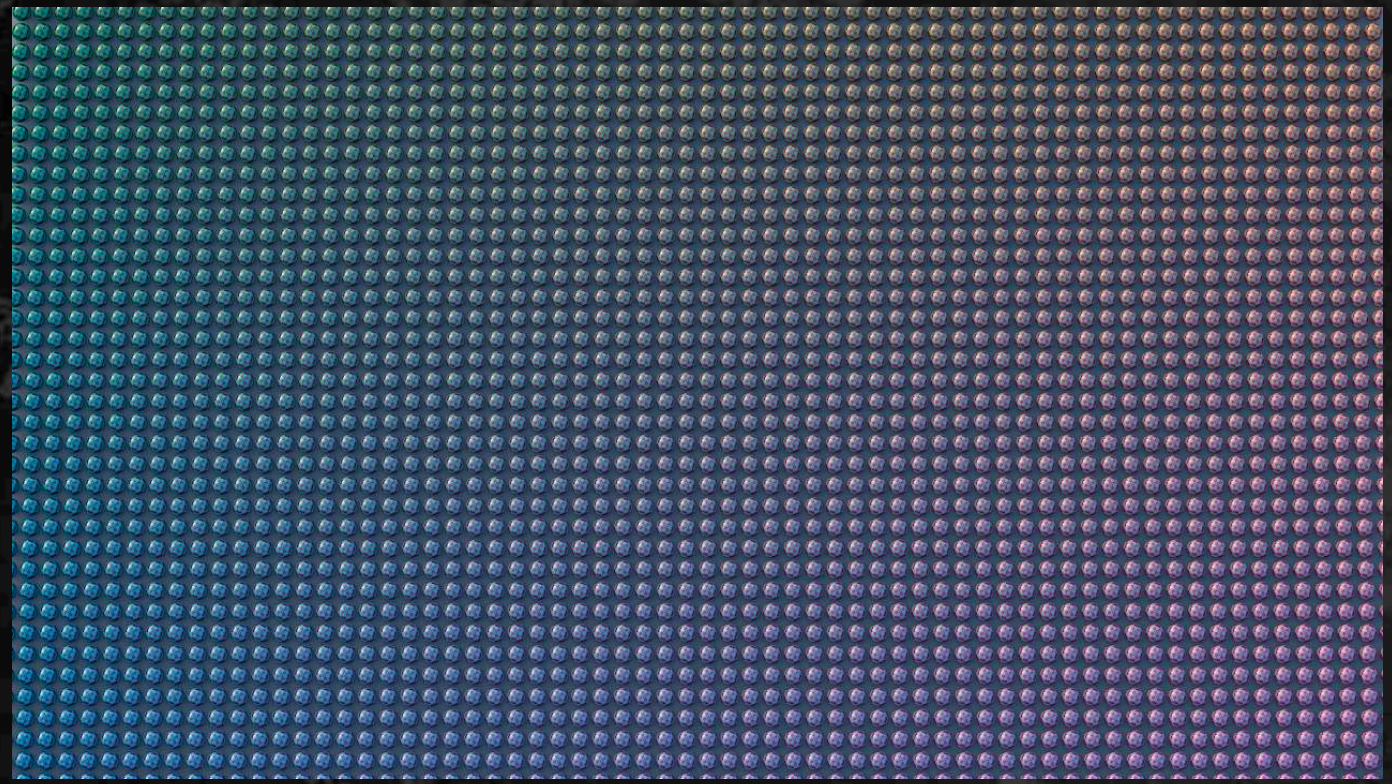
Multi GPU render

WX7100 + WX9100

RADEON PRORENDER

FEATURE HIGHLIGHTS

- Heterogeneous device support
  - GPU+GPU, GPU+CPU
  - Less latency for interactive render
- No limit for the texture usage
  - Out of core texture
  - Use system memory or disk
- MacOs Metal support
  - Maya, Blender betas, C4D available today
  - https://pro.radeon.com/en/radeon-prorender-macos-get-beta-now/



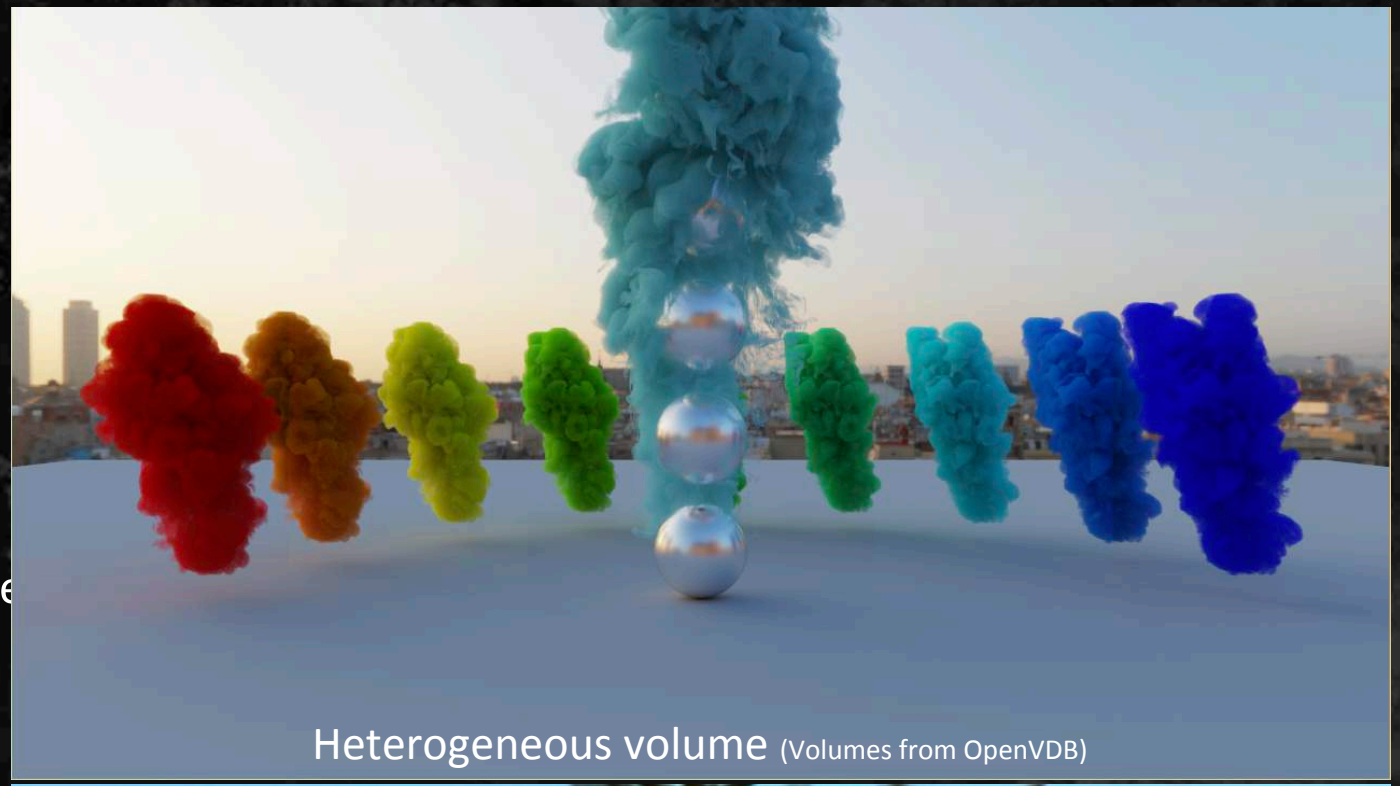
4.4k x 1k textures (4.4G texels) on WX7100 (8GB)

Texture size (~16GB) is larger than 8GB VRAM size!!

RADEON PRORENDER

SDK UPDATES

- Improved heterogeneous volume
  - Efficient sampling, less memory
- Metal support on MacOs
  - Requires MacOs High Sierra (10.13.3)
- Nested Dielectrics
- More AOVs
  - Per BRDF AOVs (diffuse, reflect, refract, volume
- Color LUTs
  - Color correction using .cube file
- Procedural UVs (Decal projection, triplanar)
- Easier to use Uber Material (Closer to Disney)
- Performance improvements
- Real time denoiser



Heterogeneous volume (Volumes from OpenVDB)

Improved uber material

RADEON PRORENDER

HELLO PRORENDER

Only Change to run on Metal

```
  1 rpr_int tahoePluginID = rprRegisterPlugin("Tahoe64.dll");
  2 rpr_int plugins[] = { tahoePluginID };
  3 rprCreateContext(RPR_API_VERSION, plugins, 1,
#if MACOS_METAL
  4                            RPR_CREATION_FLAGS_ENABLE_GPU0 | RPR_CREATION_FLAGS_ENABLE_METAL,
#else
  4                            RPR_CREATION_FLAGS_ENABLE_GPU0,
#endif
  5                            NULL, NULL, &context) ;
  6 rprContextSetActivePlugin(context, plugins[0]);
  7
  8 rpr_material_system matsys;
  9 rprContextCreateMaterialSystem(context, 0, &matsys);
 10
 11 // Create a scene
 12 rpr_scene scene;
 13 rprContextCreateScene(context, &scene);
 14 rprContextSetScene(context, scene);
 15
 16 // Create cube mesh
 17 rpr_shape cube;
 18 rprContextCreateMesh(context, ... );
 19 rprSceneAttachShape(scene, cube);
 20
```

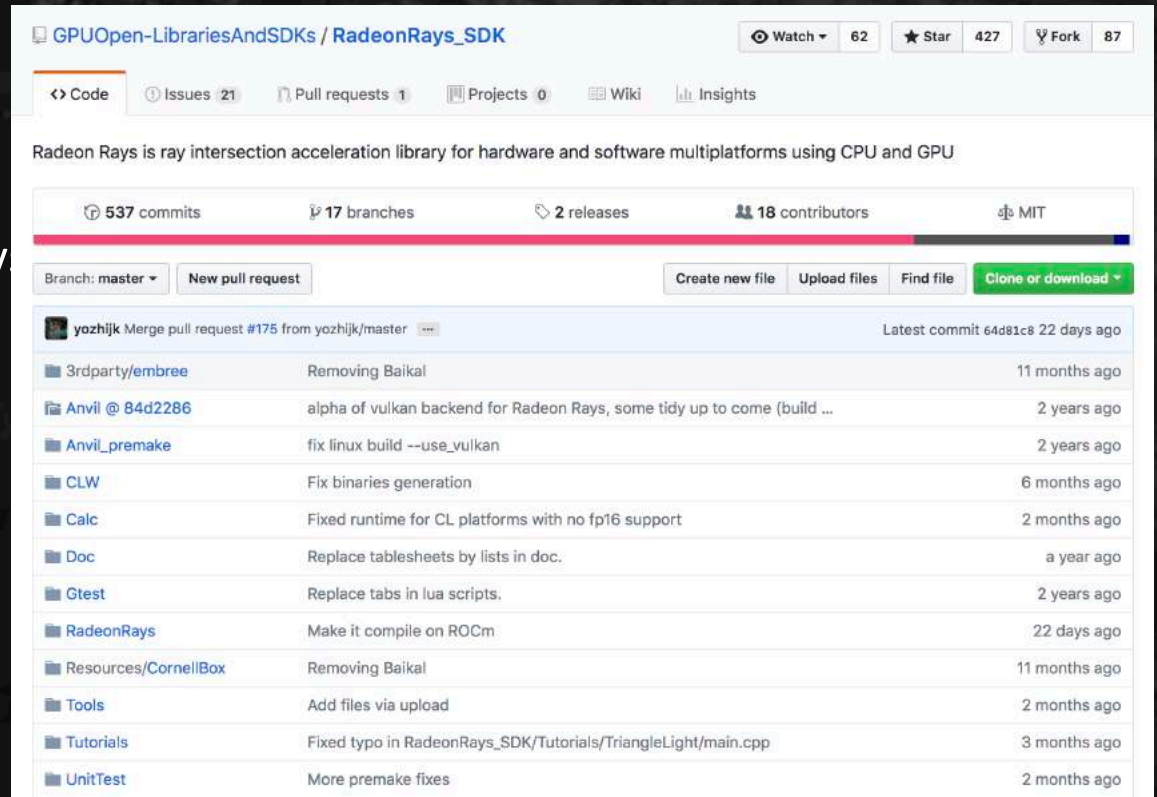
RADEON PRORENDER

HELLO PRORENDER

```
21 // Create camera
22 rpr_camera camera;
23 rprContextCreateCamera(context, &camera);
24 rprCameraLookAt(camera, 5, 5, 20, 0, 0, 0, 0, 1, 0);
25 rprSceneSetCamera(scene, camera);
26
27 // Create point light
28 rpr_light light;
29 rprContextCreatePointLight(context, &light);
30 rprPointLightSetRadiantPower3f(light, 100,100,100);
31 rprSceneAttachLight(scene, light);
32
33 // Create framebuffer to store rendering result
34 rpr_framebuffer_desc desc;
35 desc.fb_width = 800; desc.fb_height = 600;
36 rpr_framebuffer_format fmt = {4, RPR_COMPONENT_TYPE_FLOAT32};
37 rpr_framebuffer frame_buffer;
38 rprContextCreateFrameBuffer(context, fmt, &desc, &frame_buffer);
39 rprFrameBufferClear(frame_buffer);
40 rprContextSetAOV(context, RPR_AOV_COLOR, frame_buffer);
41
42 rprContextRender(context);
43
44 // Save the result to file
45 rprFrameBufferSaveToFile(frame_buffer, "rprRender.png");
```

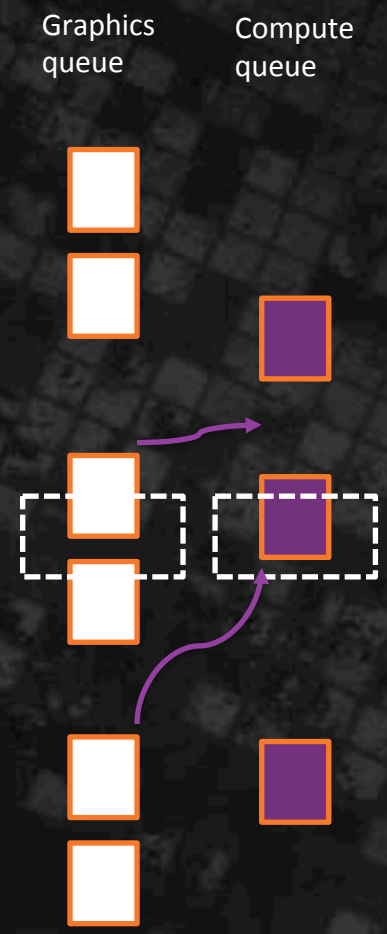
WHAT'S NEW IN RADEON RAYS

- Revised BVH builder
  - Up to **10x** faster builds
    - Manual vectorization
    - Multithreading
  - Lower memory overhead
- Improved BVH layouts
  - Using less GPU memory bandwidth
  - Up to 20% performance improvement (for secondary ray
- Available today at GPUOpen
- Vulkan version!



RADEON RAYS VULKAN

- We have done proper Vulkan version (available in soon)
  - Simplified C API
  - Designed around flexible interop with graphics
    - Commit calls now return VK command buffers
    - Intersect calls now return VK command buffers
    - Can run asynchronously with graphics
    - Application uses VK semaphores to setup dependencies



RADEON RAYS

HELLO WORLD IN RADEON RAYS

```
int nativeidx = -1;
// Always use OpenCL
IntersectionApi::SetPlatform(DeviceInfo::kOpenCL);

for (auto idx = 0U; idx < IntersectionApi::GetDeviceCount(); ++idx)
{
    DeviceInfo devinfo;
    IntersectionApi::GetDeviceInfo(idx, devinfo);
    if (devinfo.type == DeviceInfo::kGpu && nativeidx == -1)
        nativeidx = idx;
}

IntersectionApi* api = IntersectionApi::Create(nativeidx);

//adding triangle to tracing scene
Shape* shape = api->CreateMesh(g_vertices, 3, 3 * sizeof(float), g_indices,     0, g_numfaceverts, 1);
api->AttachShape(shape);
api->Commit();

// prepare rays for intersection
ray rays[3] = {...};
auto ray_buffer = api->CreateBuffer(3 * sizeof(ray), rays);

```

RADEON RAYS

HELLO WORLD IN RADEON RAYS

```
24 // prepare intersection data
25 Intersection isect[3];
26 auto isect_buffer = api->CreateBuffer(3 * sizeof(Intersection), nullptr);
27
28 //intersection
29 api->QueryIntersection(ray_buffer, 3, isect_buffer, nullptr, nullptr);
30
31 //get results
32 Event* e = nullptr;
33 Intersection* tmp = nullptr;
34 api->MapBuffer(isect_buffer, kMapRead, 0, 3 * sizeof(Intersection), (void**)    &tmp, &e);
35 //RadeonRays calls are asynchronous, so need to wait for calculation to comp    lete.
36 e->Wait();
37 api->DeleteEvent(e);
```

AMD

UNITY GPU LIGHTMAPPER

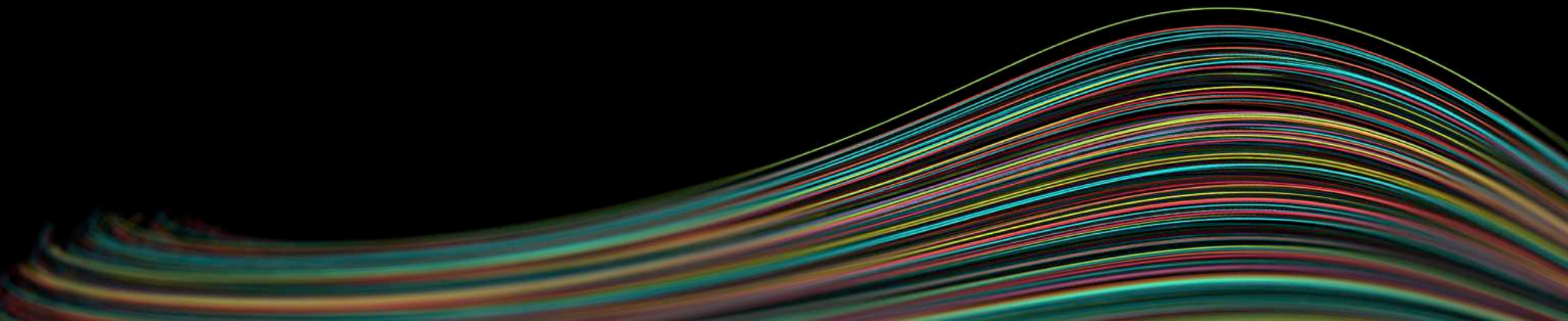
Unity + Radeon Rays

Jesper Mortensen
Unity


unity
Unity
at GDC

Jesper Mortensen

Lead Graphics Engineer, Unity Technologies



GPU Progressive Lightmapper

So what's up?

- Who are we?
- Why do we need lightmap baking at all?
- What’s the problem with baking?
- Progressive Lightmapping
- Integration with AMD RadeonRays
- Some results
- Live demo
- Questions

Who are we?

500+ R&D Developers

Massive community of game devs

50%
- all new mobile games

60%
- all new AR/VR

20+ billion yearly installs

on 3+ billion unique devices

Why lightmaps?

Why lightmaps?

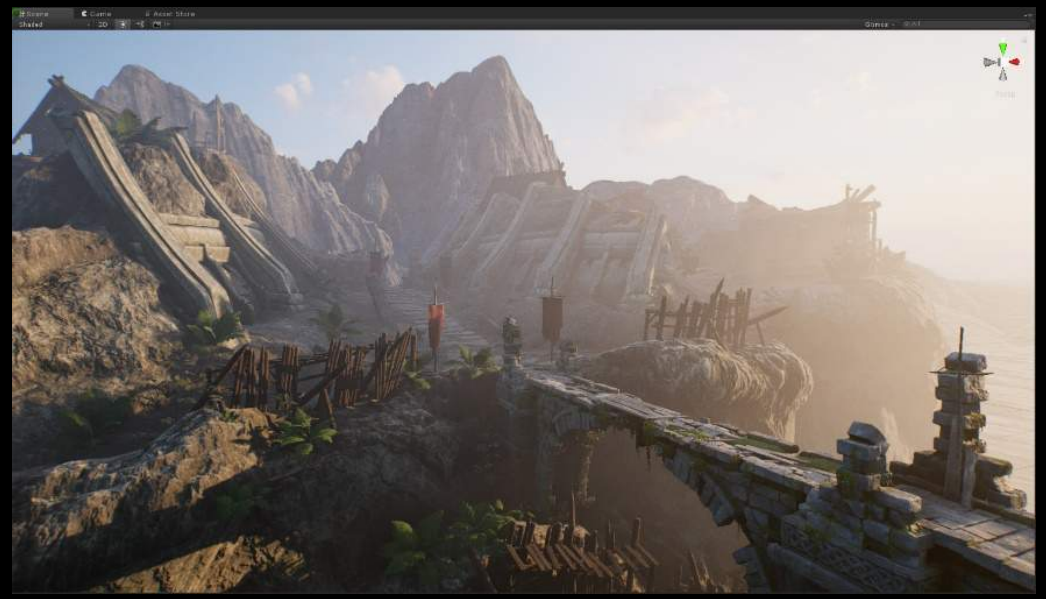
- Need high fidelity physically based GI
- Must be performant
  - Consoles and PC
  - Mobile and VR
- Mix and match
  - Realtime direct / shadowmasks / baked direct
  - Realtime GI / baked GI
  - Realtime AO / baked AO

the problem with lightmapping...

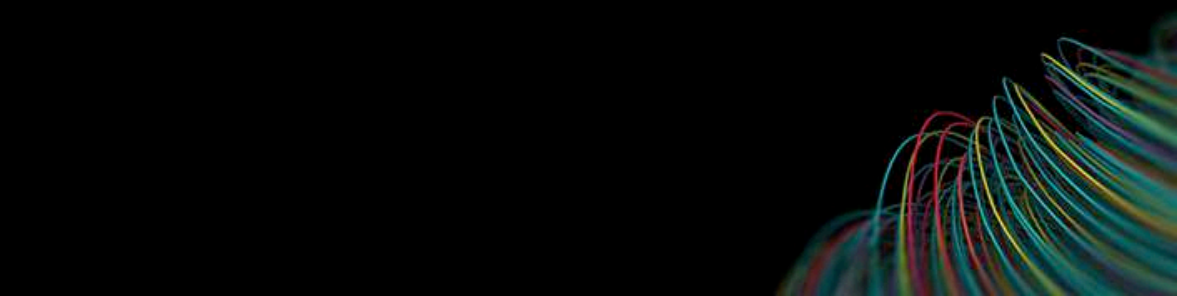
press

Generate Lighting

wait...

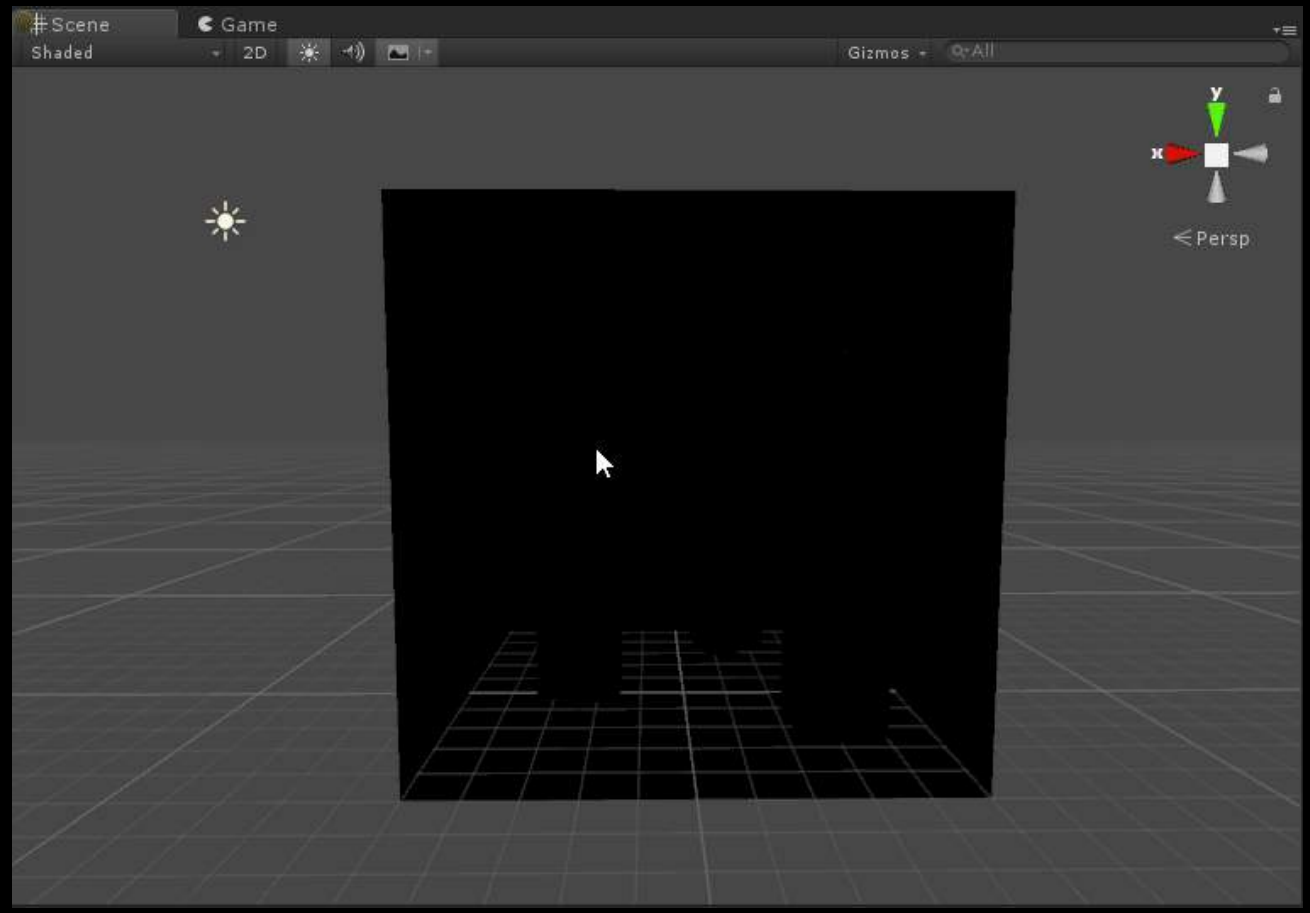


500 msec

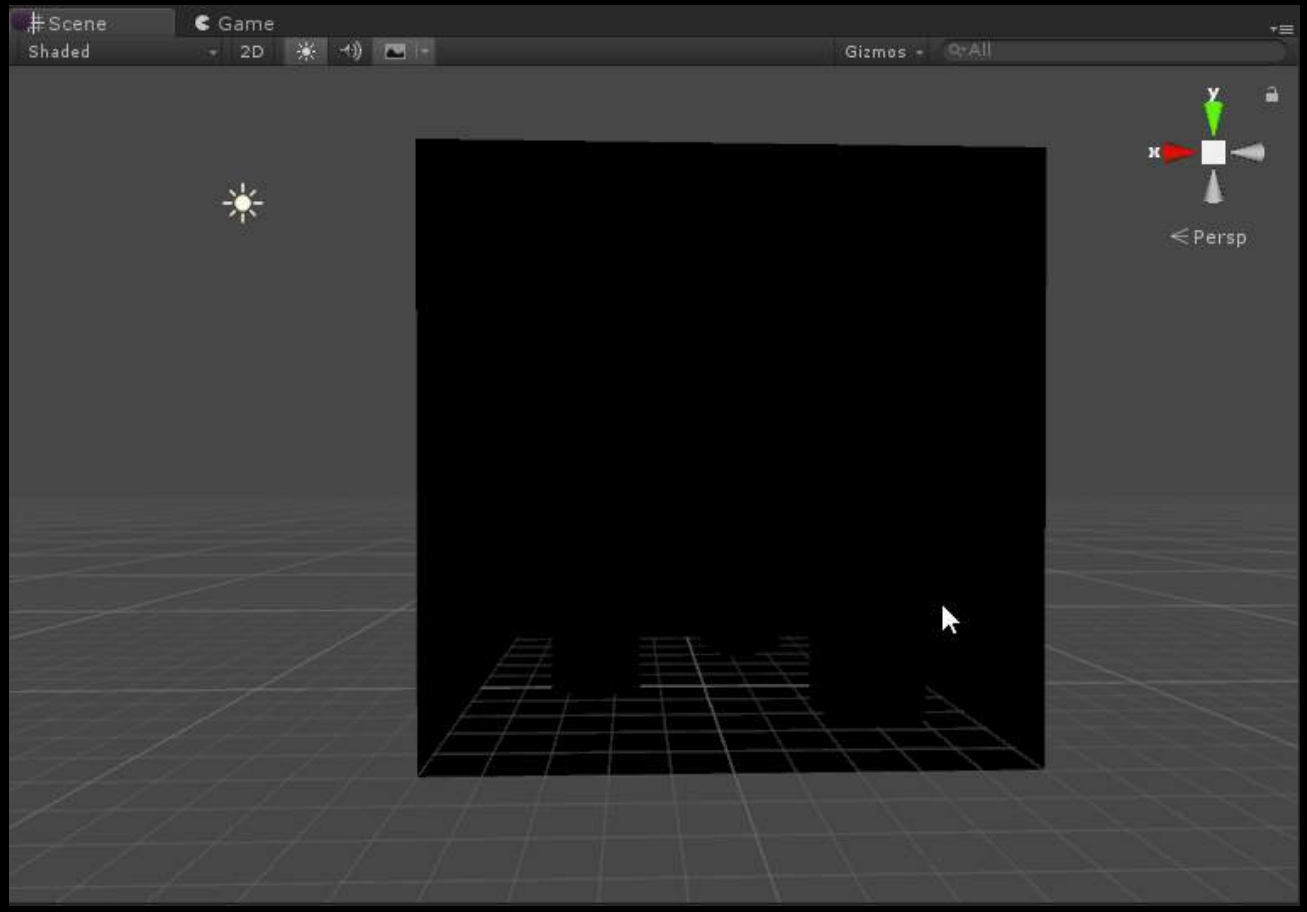


Progressive Lightmapper

Progressive updates



Prioritize view

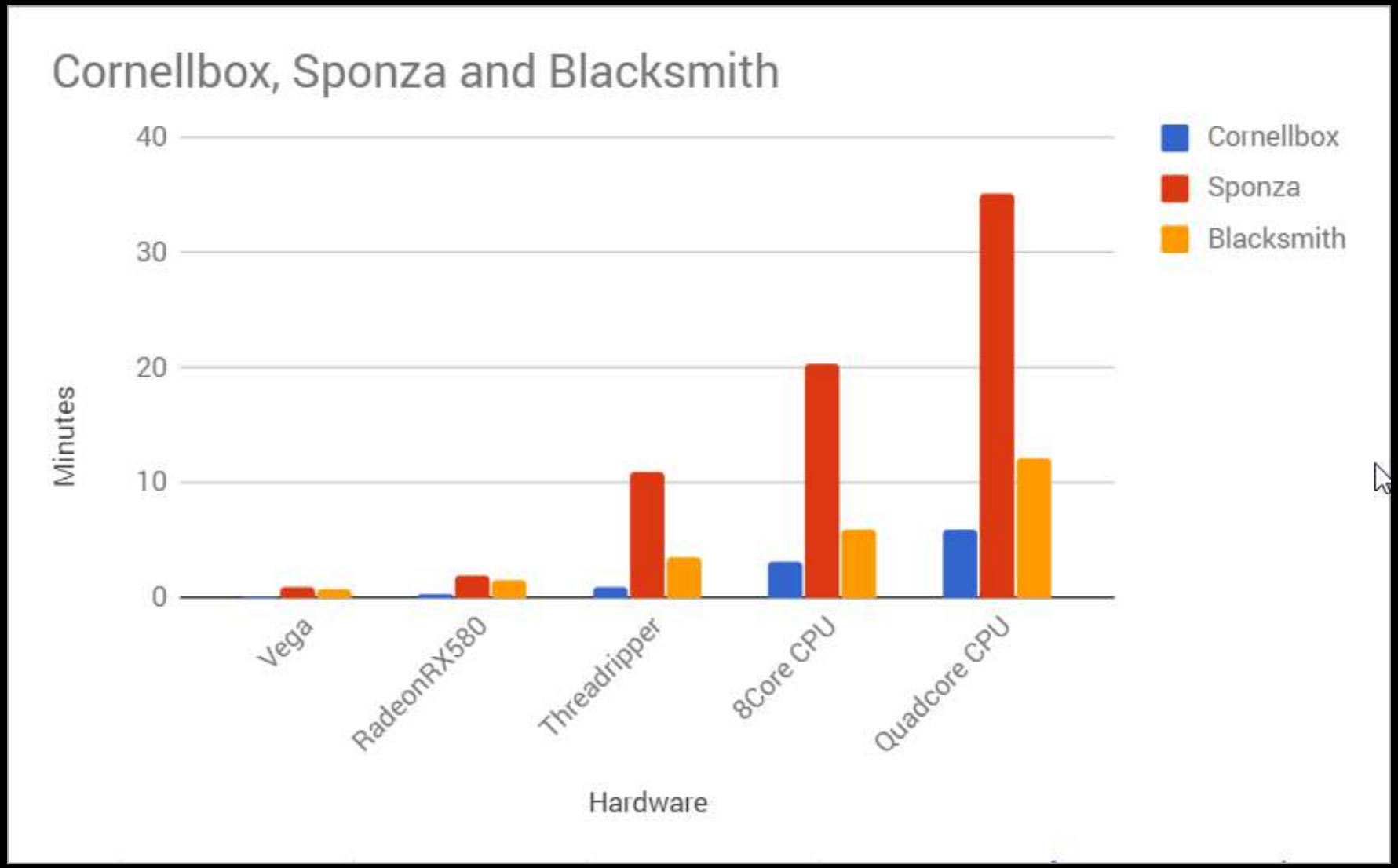


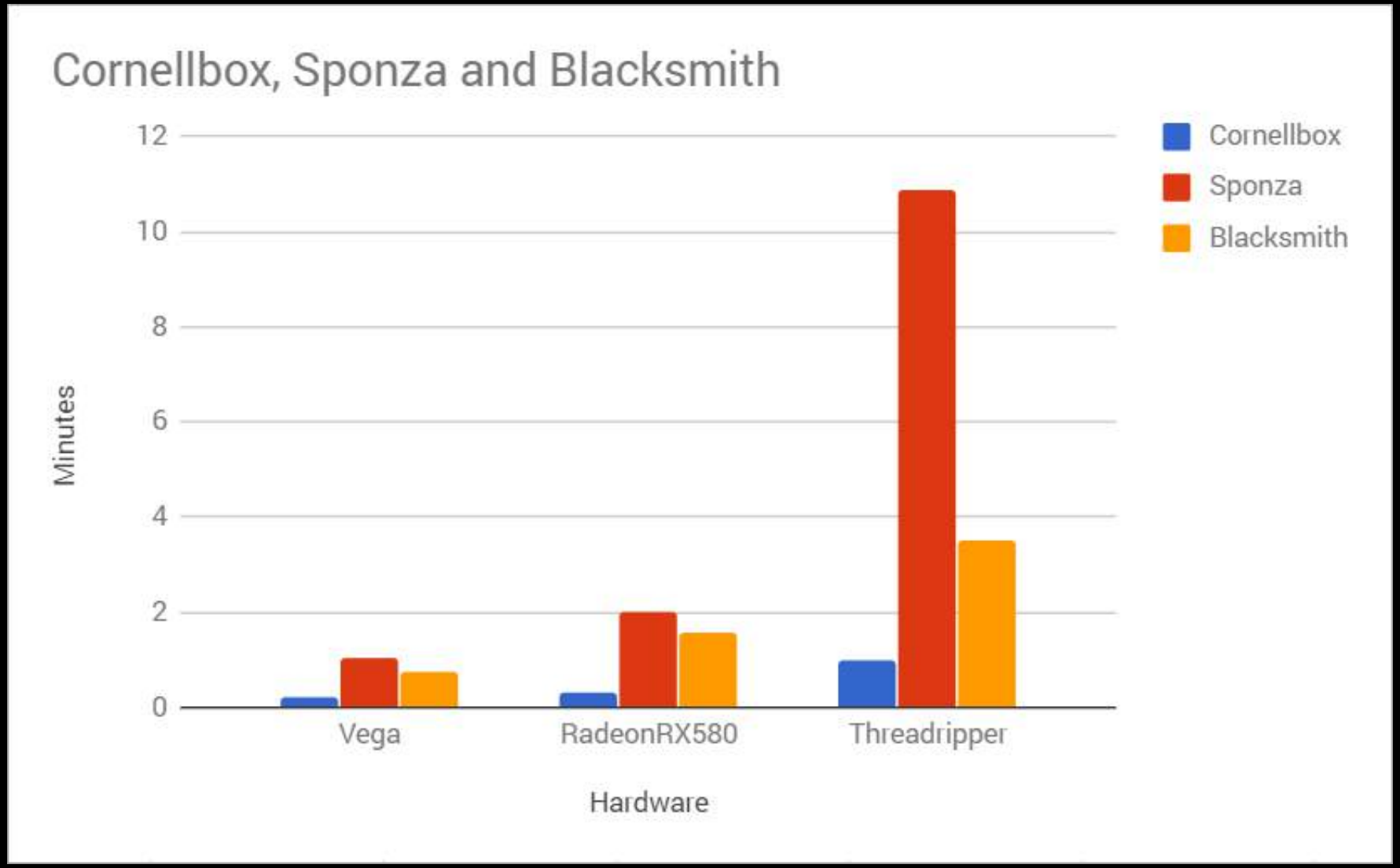
Integration with RadeonRays
aka
GPU Progressive Lightmapper

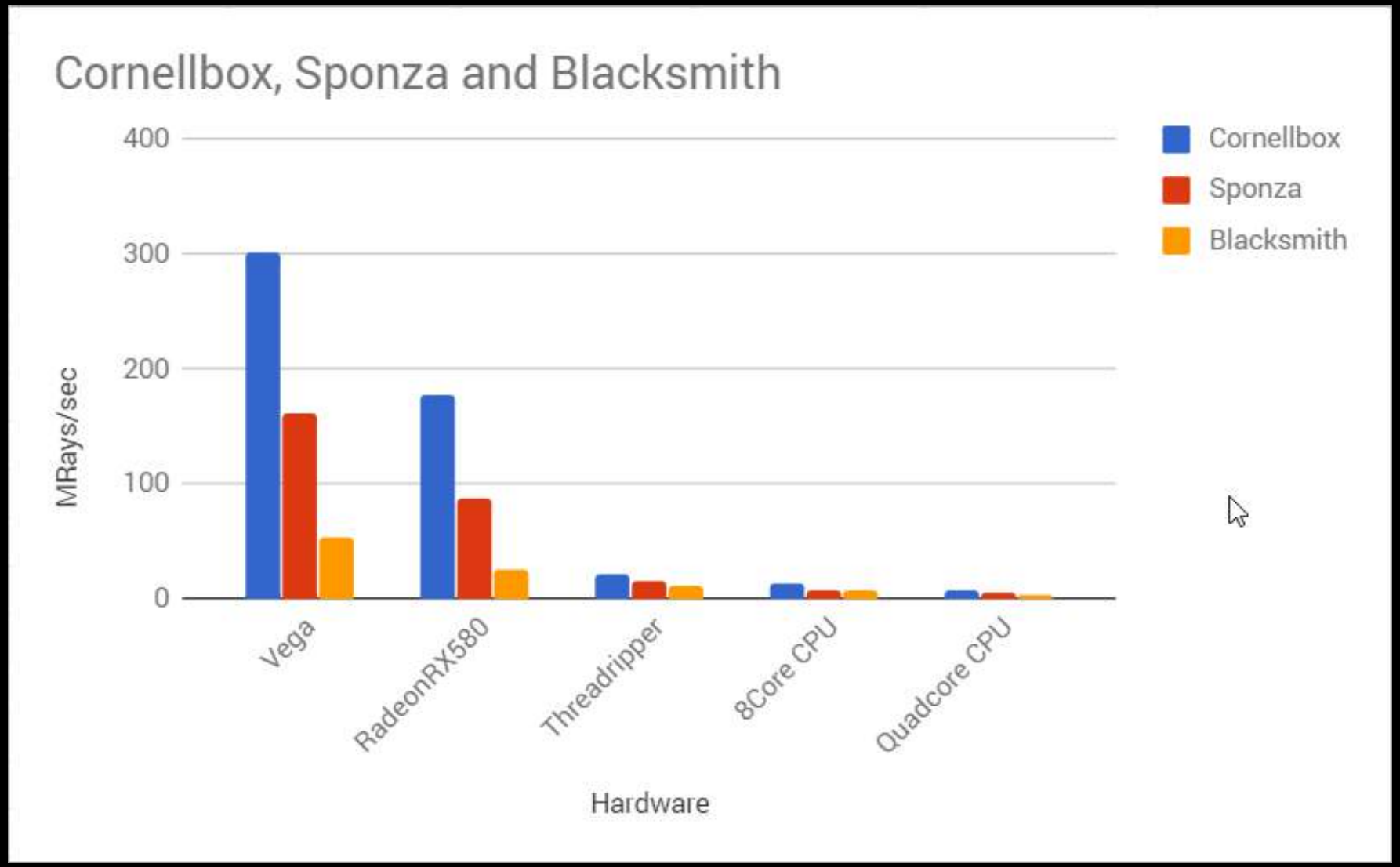
OpenCL + RadeonRays

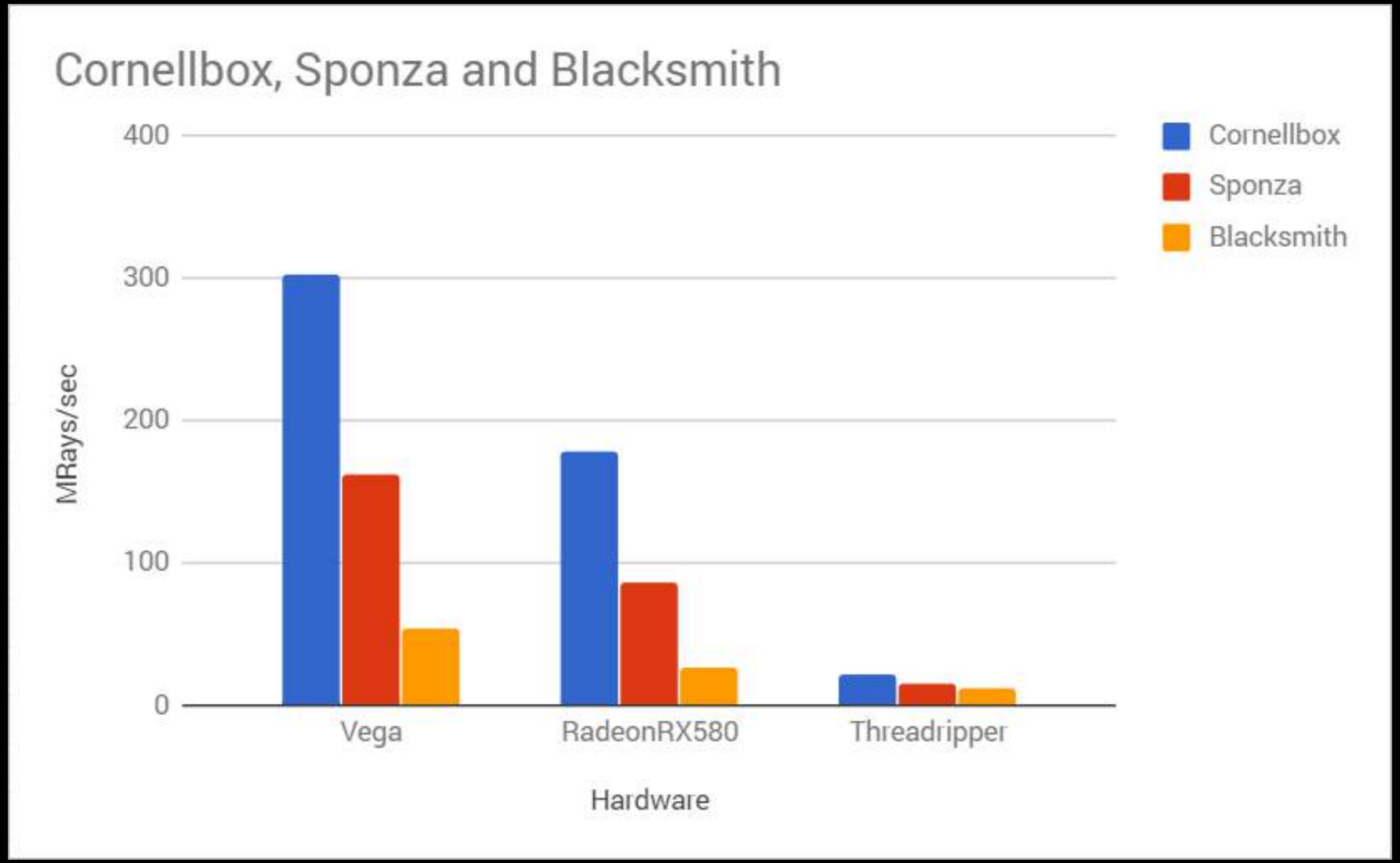
- Cross platform (Editor)
- Vendor agnostic
- Wavefront compute based
  - Kernels operate on lightmaps
  - Compaction removes empty areas
- Very little RadeonRays code
  - It’s a lean and mean interface
- Up to 10x faster than CPU

Results

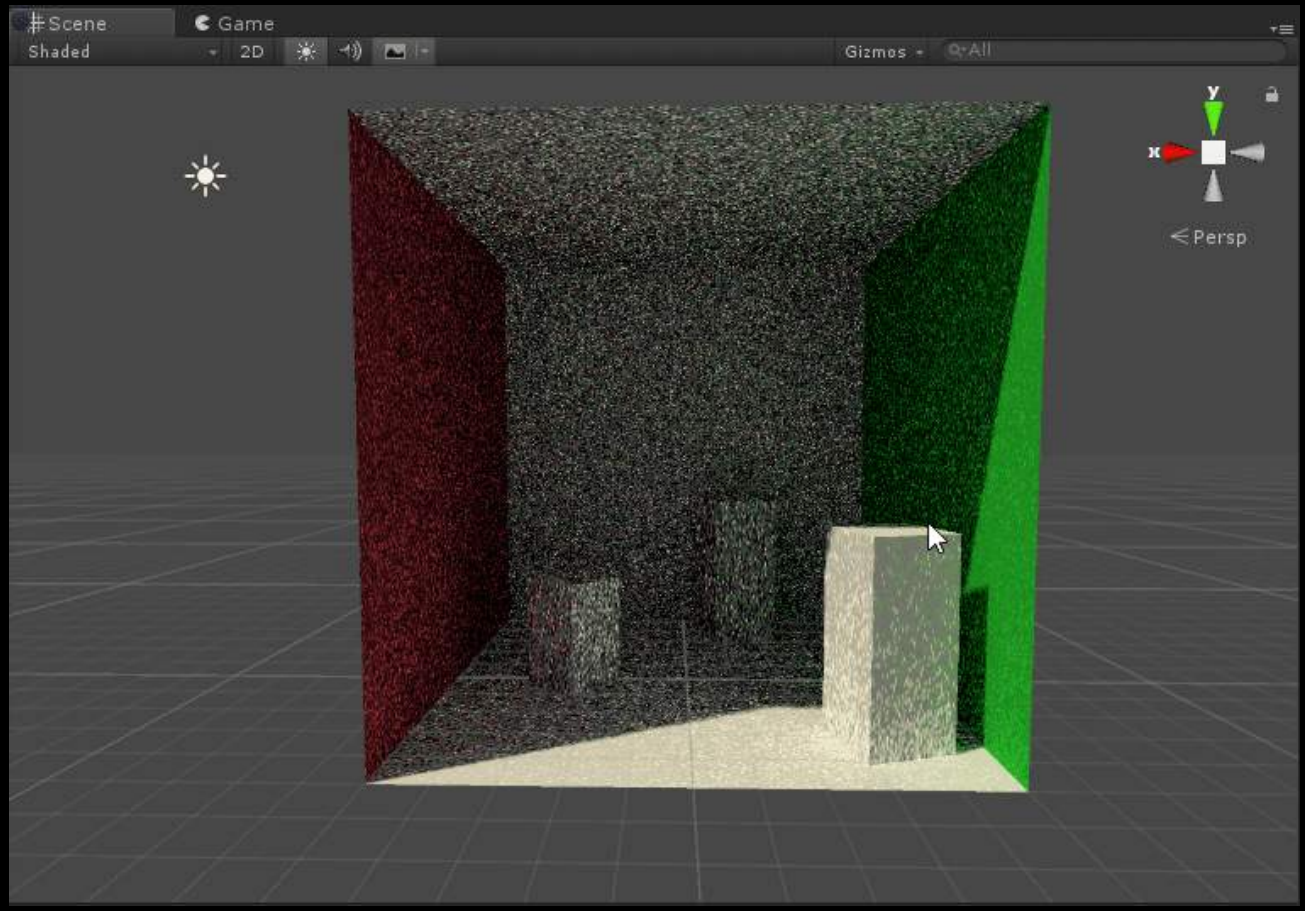

Cornellbox, Sponza and Blacksmith
40
30
20
10
0
Minutes
Cornellbox
Sponza
Blacksmith
Vega
RadeonRX580
Threadripper
8Core CPU
Quadcore CPU
Hardware


Cornellbox, Sponza and Blacksmith
12
10
8
6
4
2
0
Minutes
Cornellbox
Sponza
Blacksmith
Vega
RadeonRX580
Threadripper
Hardware


Cornellbox, Sponza and Blacksmith
400
300
200
100
0
MRays/sec
Cornellbox
Sponza
Blacksmith
Vega
RadeonRX580
Threadripper
8Core CPU
Quadcore CPU
Hardware


Cornellbox, Sponza and Blacksmith
400
300
200
100
0
MRays/sec
Cornellbox
Sponza
Blacksmith
Vega
RadeonRX580
Threadripper
Hardware

GPU bake



Live demo

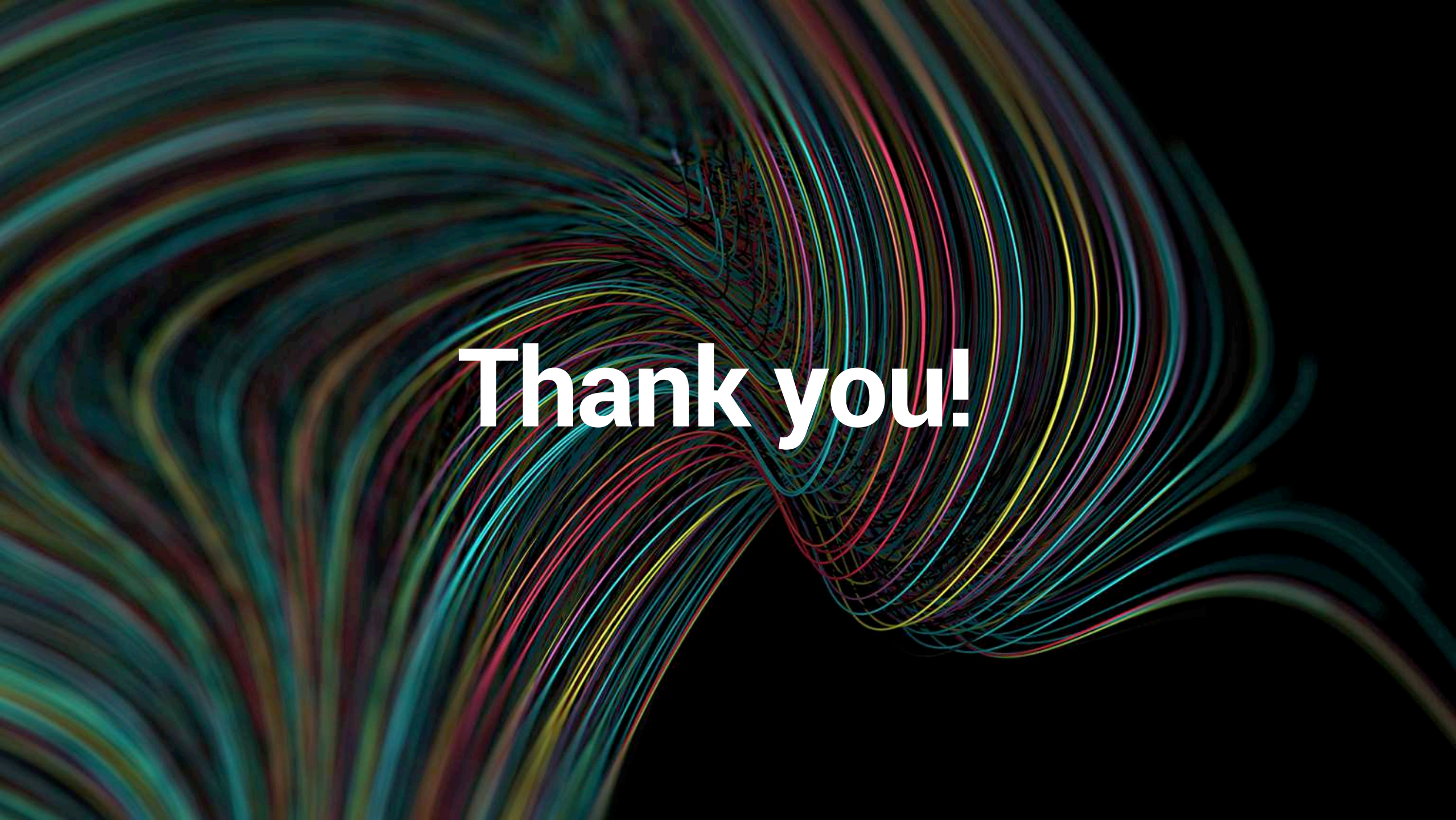
What’s next

- Use the rays better
  - MIS
  - Light power sampling
- Shoot fewer rays
  - Denoising
- Exploit coherence
- Multi GPU

Acknowledgements

- AMD RadeonRays team
- Special thanks
  - Dmitry Kozlov
  - Guillaume Boisse
  - Bruno Stefanizzi
  - Bikram Singh

Questions?

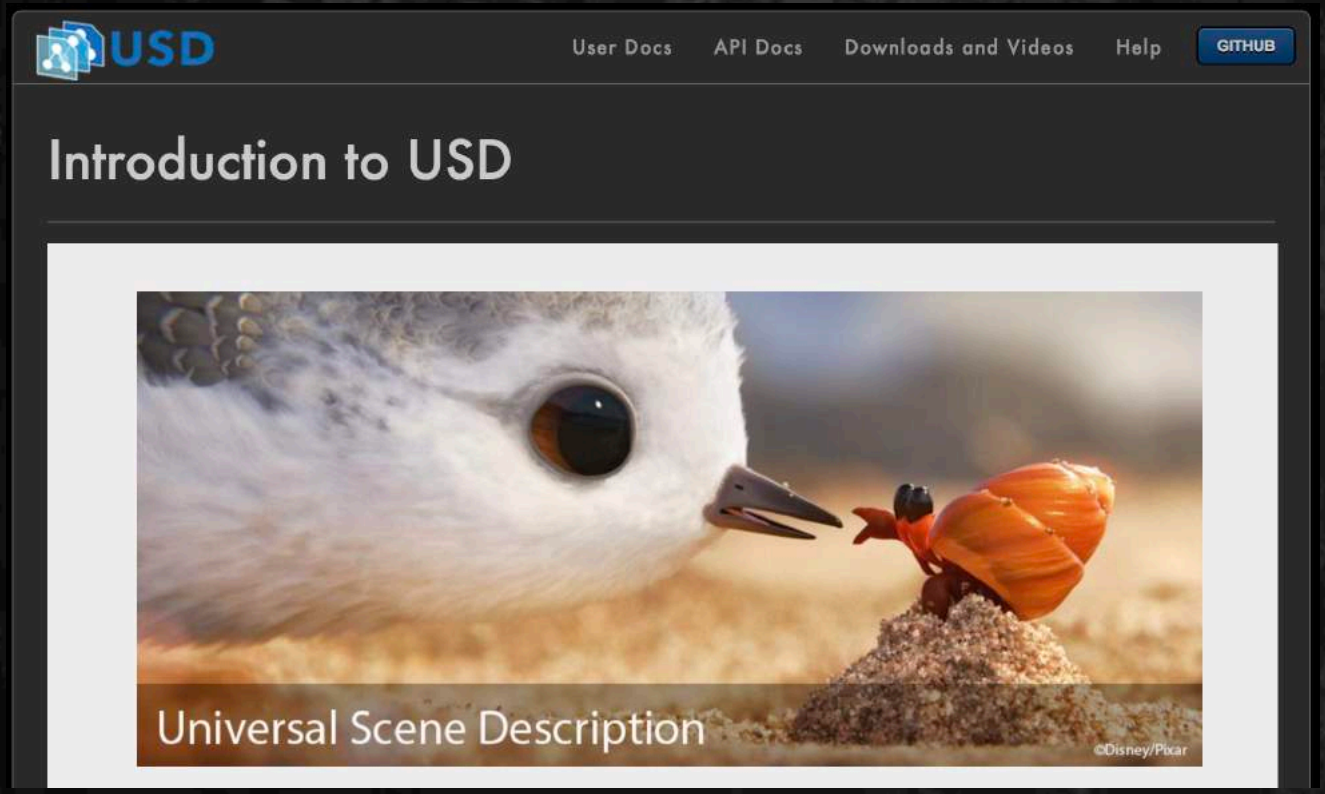

Thank you!

PRORENDER + USD

Real-time preview of assets

UNIVERSAL SCENE DESCRIPTION

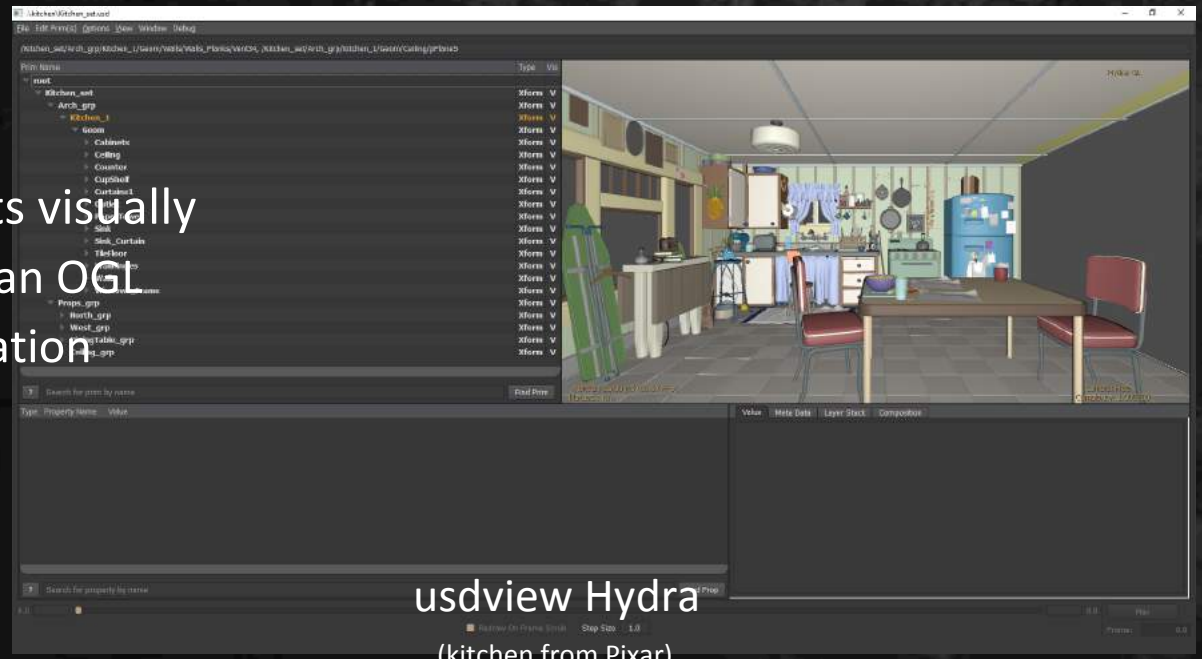
- Scene description from Pixar
- Interchange between applications
- Used in production VFX and animation
- Quickly becoming standard in VFX industry



USDVIEW, HYDRA

- One of the tools comes with USD
- Handy for investigation of a USD file
- Comes with the high performance Hydra OGL renderer
  - Visual debugging
  - Scalability
  - OpenSubdiv support
  - Designed for multiple back-ends, front-ends

- Hydra OGL isn’t designed to investigate materials, lights visually
  - Nicer to visualize the work in real time closer to final than OGL
  - Computationally expensive to solve light transport equation
    - (Embree backend)
  - Radeon ProRender can help

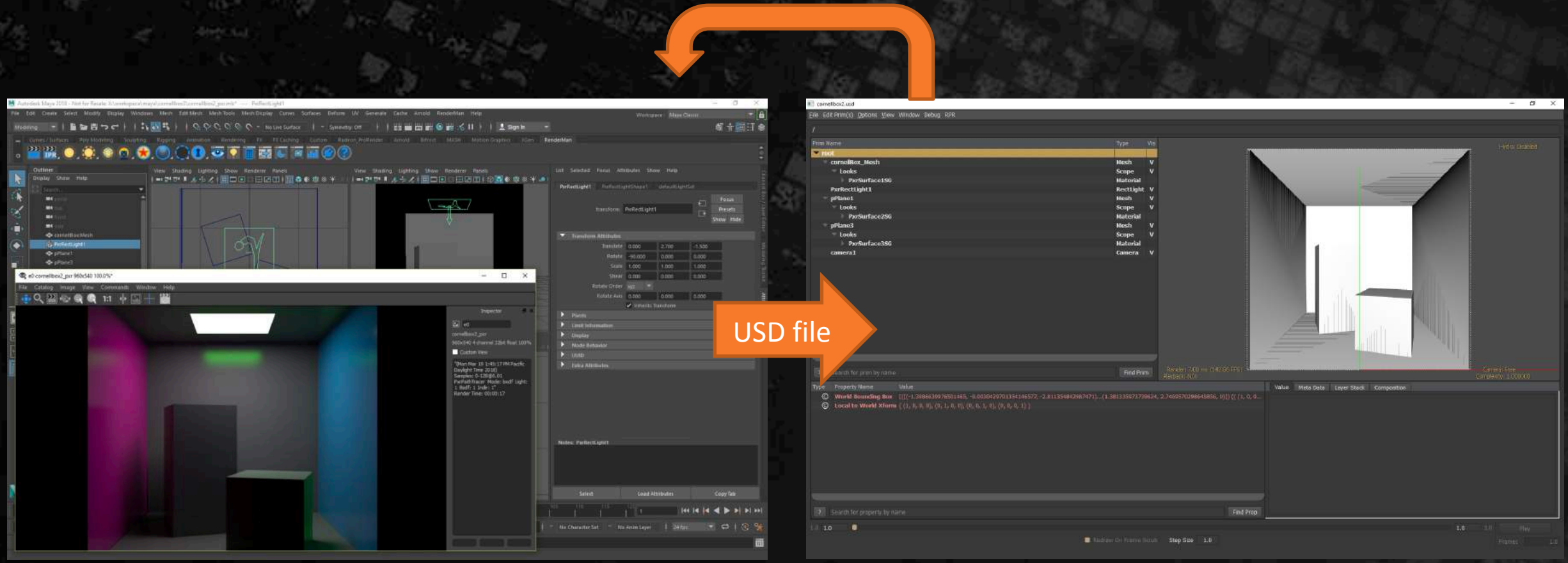


usdview Hydra
(kitchen from Pixar)

CURRENT WORKFLOW

OGL BACKEND

- Export from Maya
- Debug display in usdview

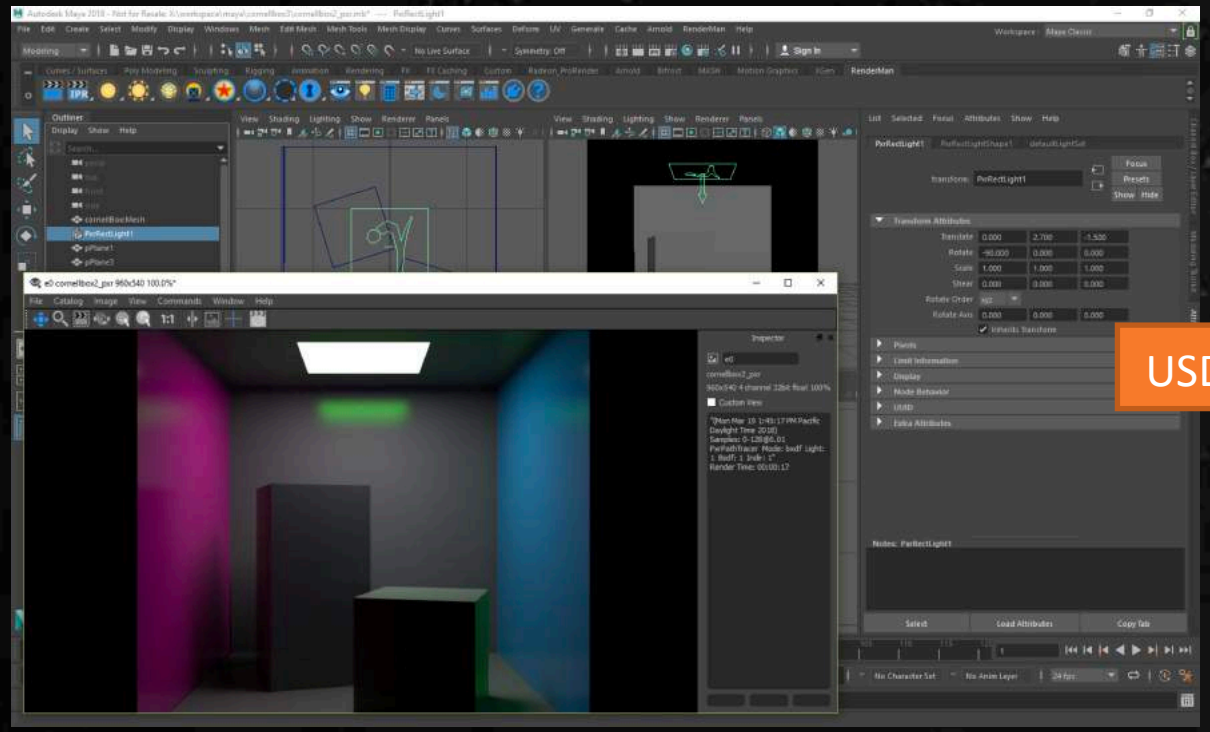


WORKFLOW WITH RPR

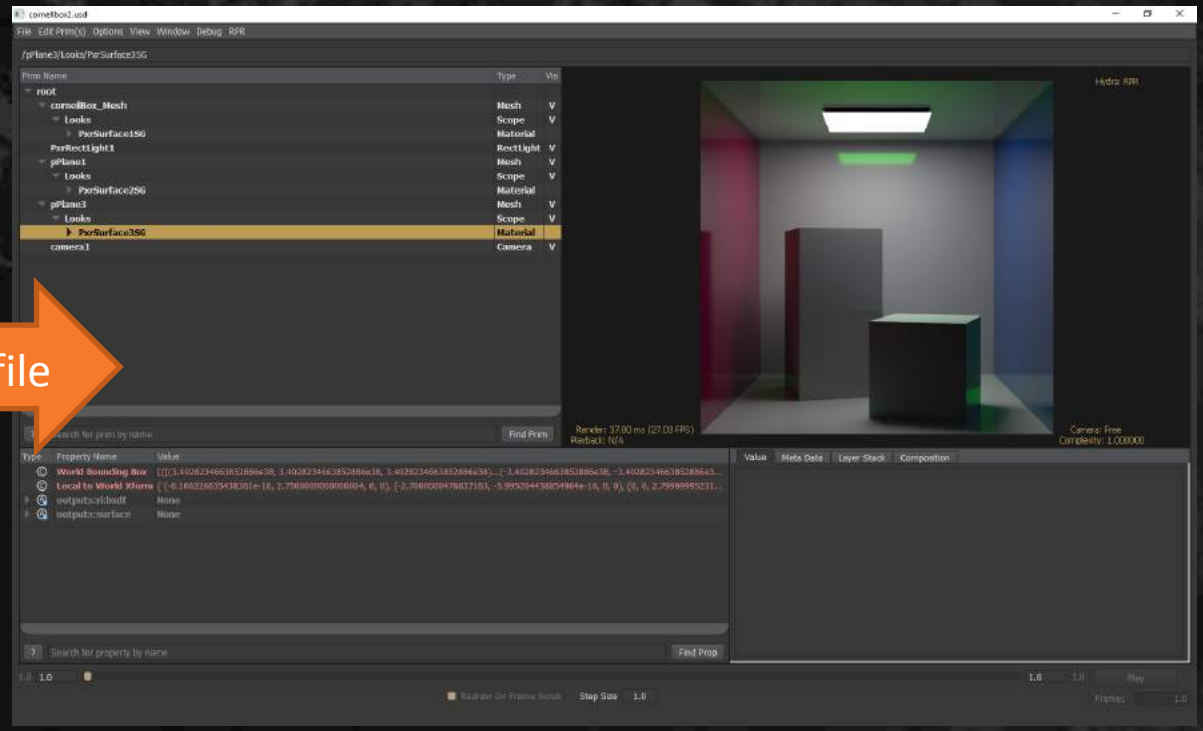
PRORENDER BACKEND

- Export from Maya
- See the lighting and shading in real time in usdview with RPR

ProRender Backend



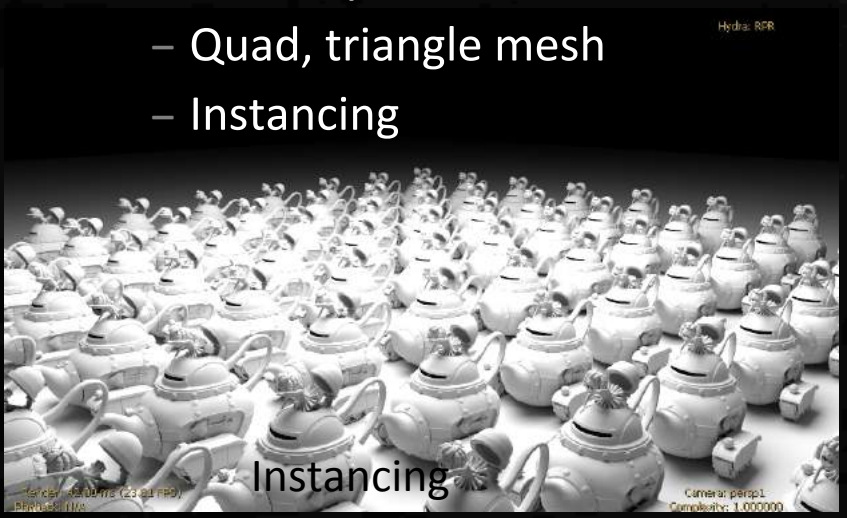
USD file



PRORENDER + USD

PROTOTYPE IMPLEMENTATION FEATURE LIST

- Light
  - Rect light
  - Dome light
- Material
  - PxrSurface, RPR
  - Image Textures
- Geometry
  - Quad, triangle mesh
  - Instancing



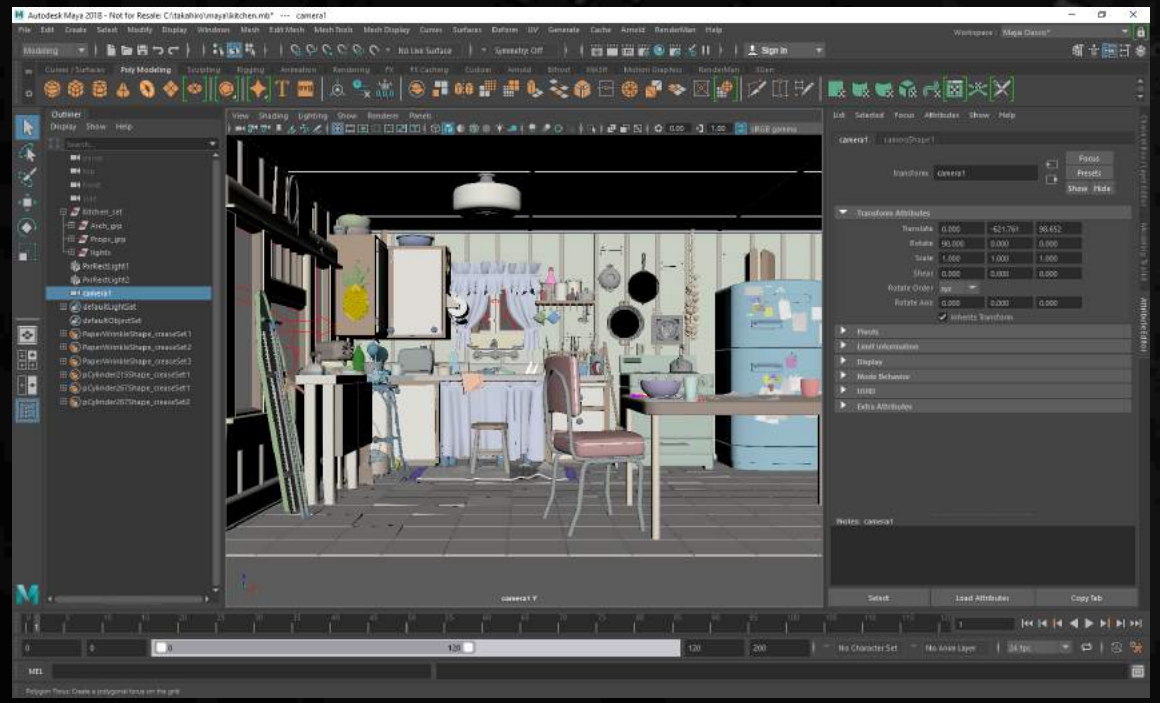
(Rolling Teapot from Pixar)

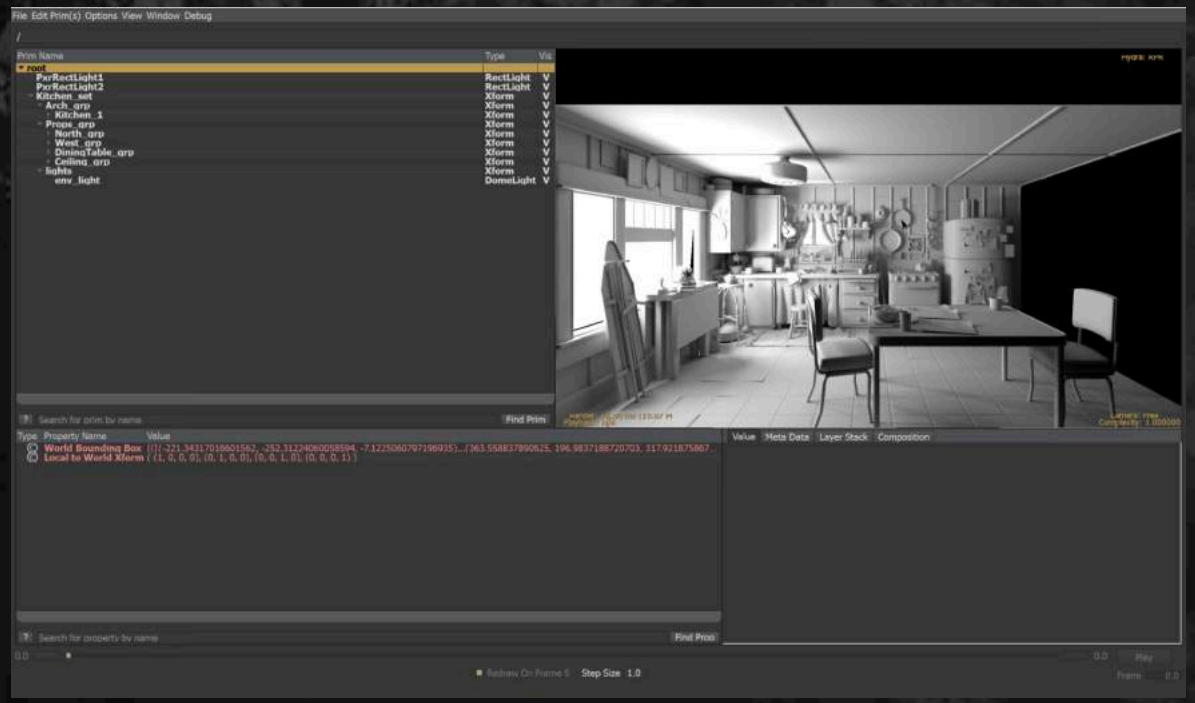
Demo

WHY IMPLEMENTED A HYDRA BACKEND?

RPR IN 3DCC TOOLS

- RPR is implemented as a usdImaging plugin
- Some applications are integrating USD Hydra as the main viewport renderer
  - You can get ProRender viewport automatically
- Multiplatform support





RADEON PRORENDER REAL TIME RAY TRACING

Bridging the gap

PRO GRAPHICS VIEWPORT RENDER

AMD

- In Pro Graphics (e.g., 3DCC tools)
  - Viewport is using mostly OpenGL
- 2 issues
  - Scalability
  - Quality of the rendering
- Announce 2 solutions
  - V-EZ
    - Better performance from Vulkan without going through API complexity
  - Radeon ProRender real-time ray tracing
    - Bringing the viewport to the next level

V-EZ

VULKAN MIDDLEWARE SOLUTION FROM AMD

AMD

- Problem
  - Vulkan API adoption among ProGraphics ISVs slow
  - Vulkan API difficult to learn relative to OpenGL
  - Inordinate amounts of code relative to OpenGL
  - ISVs see no compelling reasons to migrate from OpenGL
  - Vulkan middleware layers and libraries exist but not being adopted
  - Vulkan missing required CAD features (ex: line stipple)
- Objectives
  - Provide a simplified layer on top of Vulkan
  - Be a stepping stone between OpenGL and Vulkan
  - Maintaining existing API semantics
  - Allow ISVs to learn Vulkan API without the explicit responsibilities
  - Allow for interop with native Vulkan
  - Make GLSL a first class citizen again

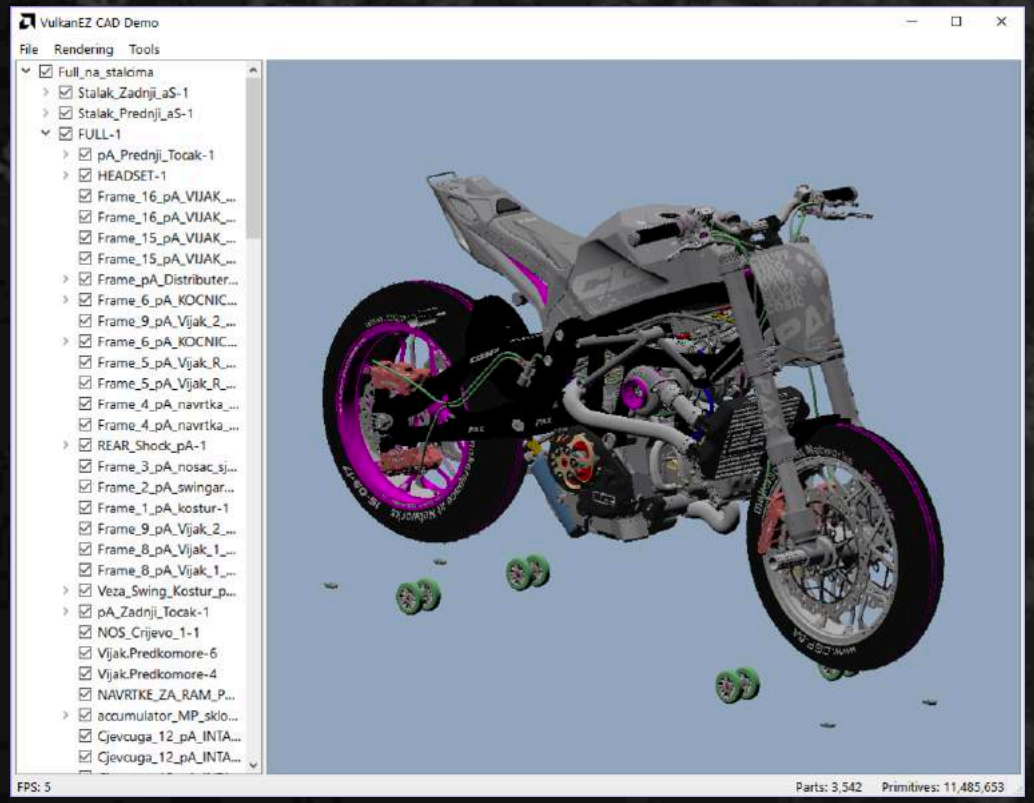
V-EZ

VULKAN MIDDLEWARE SOLUTION FROM AMD

AMD

- Solution
  - A slimmed down Vulkan API that still exposes the strengths of Vulkan
    - Multi-threaded command buffer recording
    - Asynchronous compute
    - Asynchronous transfers
    - Multi-gpu
  - Alleviates responsibilities from application:
    - Swapchain management
    - Memory management
    - Command pools
    - Descriptor pools
    - Descriptor sets
    - Pipeline permutations
    - Render pass management
    - Render pass compatibility
    - Pipeline barriers
    - Image layout transitions
    - SPIR-V compilation

- Additional benefits:
  - Vulkan interop
  - GLSL and SPIR-V reflection
  - Line stipple support



REAL TIME RAY TRACING

MOTIVATION

Offline Renderers

- Photo real
- Long render time
- Fully physically based

?

Game Engine Renderers

- Good quality
- Real time
- Relaxed physically based
- Fakes

REAL TIME RAY TRACING

MOTIVATION

Offline Renderers

- Photo real
- Long render time
- Physically accurate

ProRender Real-time Ray Tracing

- Better quality
- Adjustable computational cost
- Lerp(accurate, fake, your flavor)
- We take care the complexity in the API
- Add physically based effect on **your raster renderer**

Game Engine Renderers

- Good quality
- Real time
- Relaxed physically based
- Fakes

REAL TIME RAY TRACING

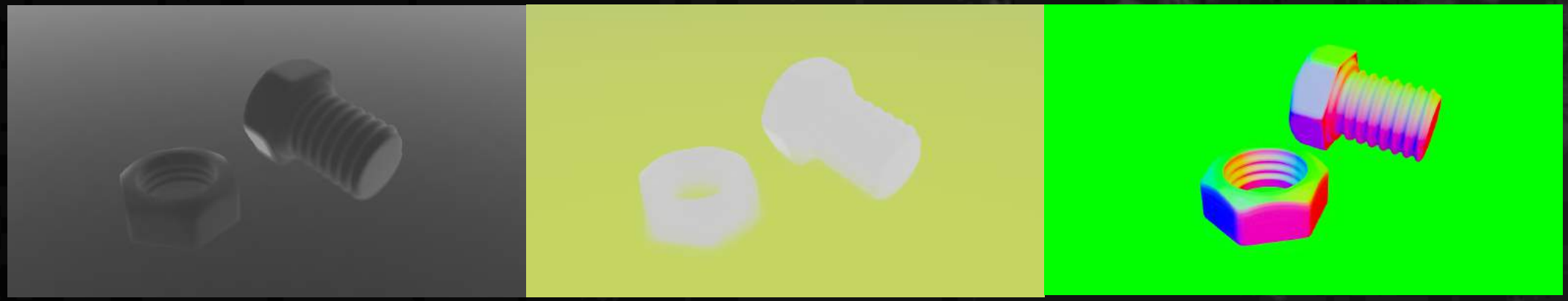
DETAIL

- Add physically based effect on your rasterization based renderer
- Implemented using Vulkan
- Asynchronous compute in mind
- Dispatch the ray tracing effect kernels at the back of the graphics tasks
- Adjust the amount of ray tracing effect depends on the target (HW and frame rate)
- Built in denoiser to produce less noise image for effects using Monte Carlo integration

PRORENDER REAL-TIME RAY TRACING

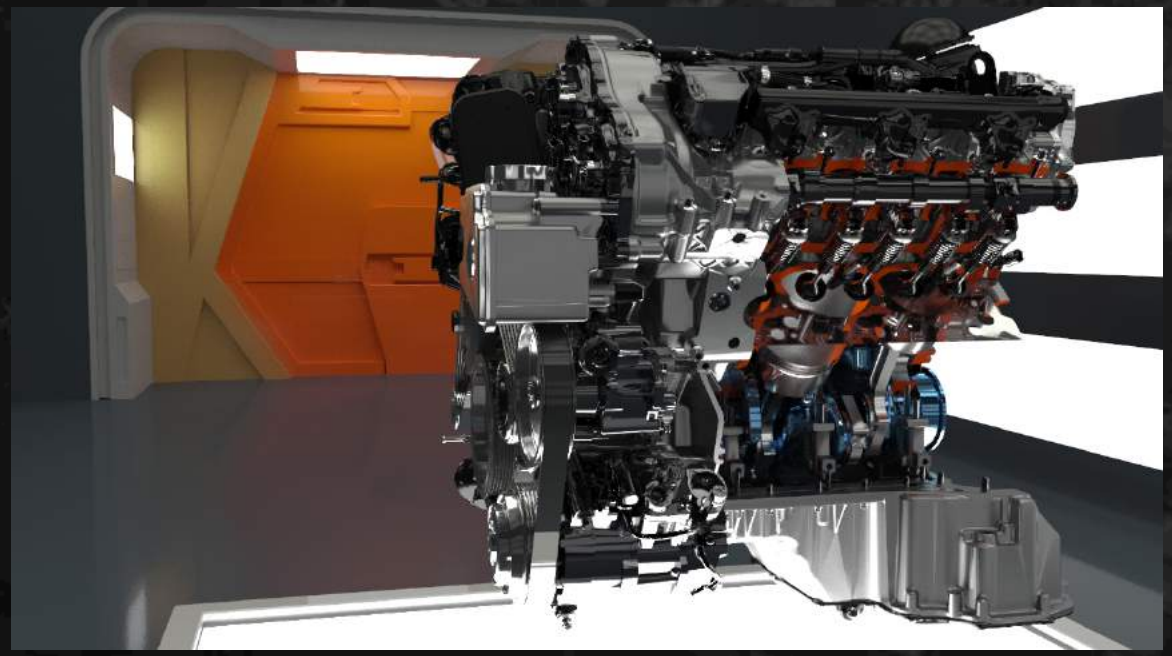
AMD

- Rasterization for primary visibility and lighting
  - No noise in primary
  - Fast feedback
- Deferred shading
- First step is render G-buffers
  - Normal and depth
  - Albedo and transparency
  - Roughness, metallicity and motion vectors



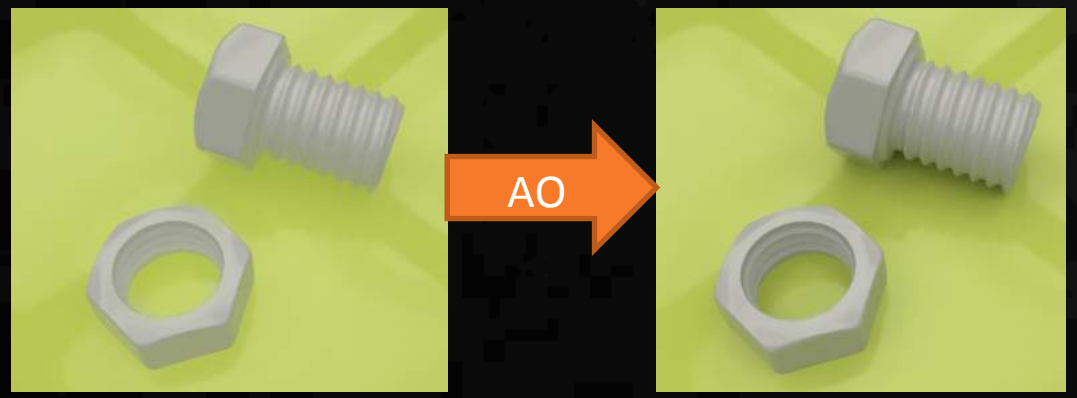
PRORENDER REAL-TIME RAY TRACING

- Rasterization for primary visibility and lighting
  - No noise in primary
  - Fast feedback
- Asynchronous ray tracing for secondary and complex effects
  - Based on RadeonRays
- You choose
  - Ambient occlusion
  - Glossy reflections
  - Diffuse global illumination
  - Area lighting
- Effects can be turned on/off based on HW capabilities
- MC-based effects are denoised using wavelet filter



AMBIENT OCCLUSION

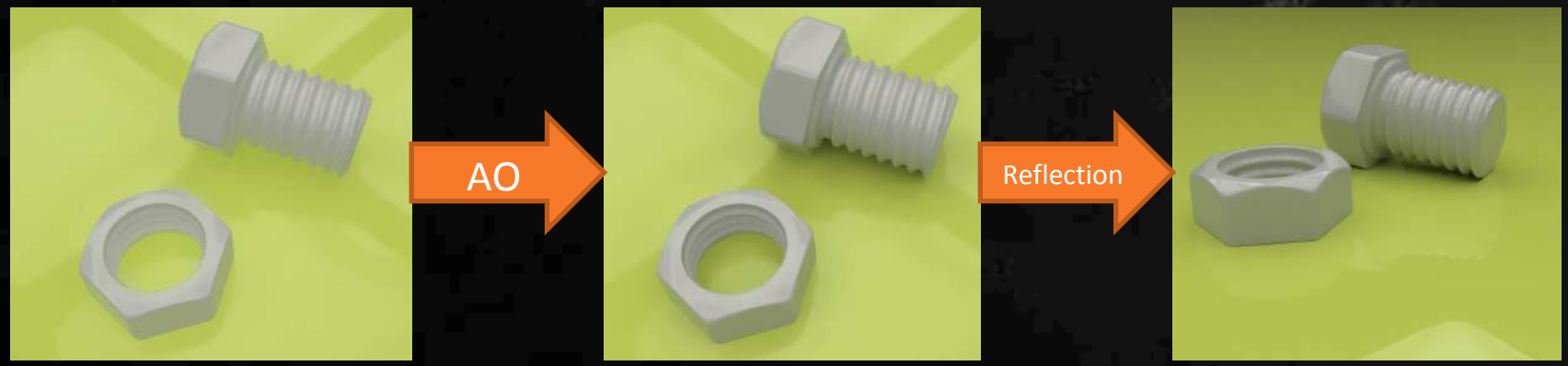
- True ray traced ambient occlusion (shadow from an IBL)
- Compute shader generates AO rays based on G-buffer position and normal
- RadeonRays traces rays asynchronously
- Ambient occlusion is applied to an IBL component of a direct illumination
- Performance:
  - ~500-600 MRays/s for moderate scenes*



*Depends on the number of pixels marked for AO / visibility complexity

GLOSSY REFLECTIONS

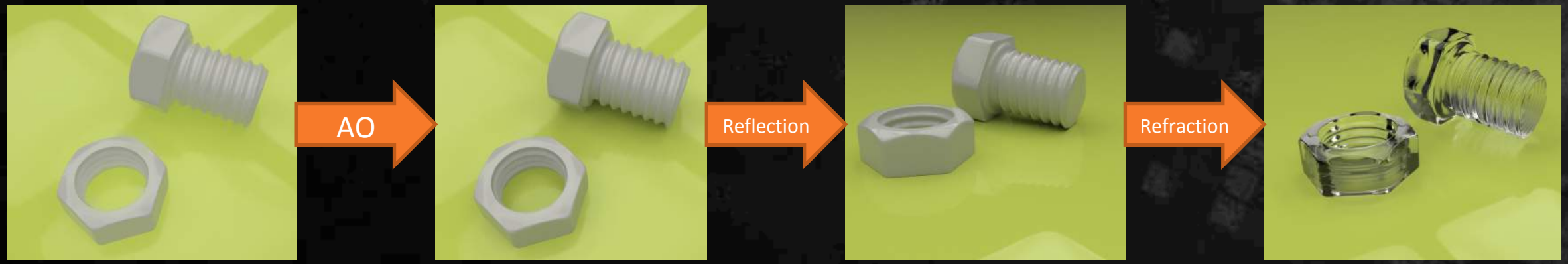
- True ray traced reflections (multiple bounces)
- Compute shader handles Gbuffer
  - Generates reflection rays for pixels marked for reflection
- RadeonRays traces rays asynchronously
- Resolve kernel calculates illumination
- Performance:
  - ~500-600 MRays/s for moderate scenes*



*Depends on the number of pixels marked for reflection

GLOSSY REFRACTION

- Ray traced refractions
- Compute shader handles Gbuffer
  - Generates refraction rays for pixels marked for refraction
- RadeonRays traces rays asynchronously
- Resolve kernel calculates illumination
- Performance:
  - ~1-1.5GRays/s for moderate scenes*
- If you are not satisfied with these…



*Depends on the number of pixels marked for refraction

FULL GI

TURNING IT TO 11

- True ray traced reflections (# of bounces, your choice)
- Compute shader starts reflection or diffuse rays
- RadeonRays traces rays asynchronously
- Resolve kernel calculates illumination
- Performance:
  - ~300 MRays/s for moderate scenes*

*Depends on the number of pixels marked for AO/visibility complexity

Demo

CONCLUSION

AMD

- Gave latest updates on Radeon ProRender, Radeon Rays
- Showed the Unity GPU Lightmapper using Radeon Rays improves the game contents creation pipeline
- Showed Radeon ProRender + USD extends the capability of the hydra renderer, added lighting preview functionality
- Empower the Pro Graphics viewport by 2 new solutions, V-EZ and Radeon ProRender real-time rendering

FOR SDK ACCESSES

AMD

- Bruno.Stefanizzi@amd.com

DISCLAIMER & ATTRIBUTION

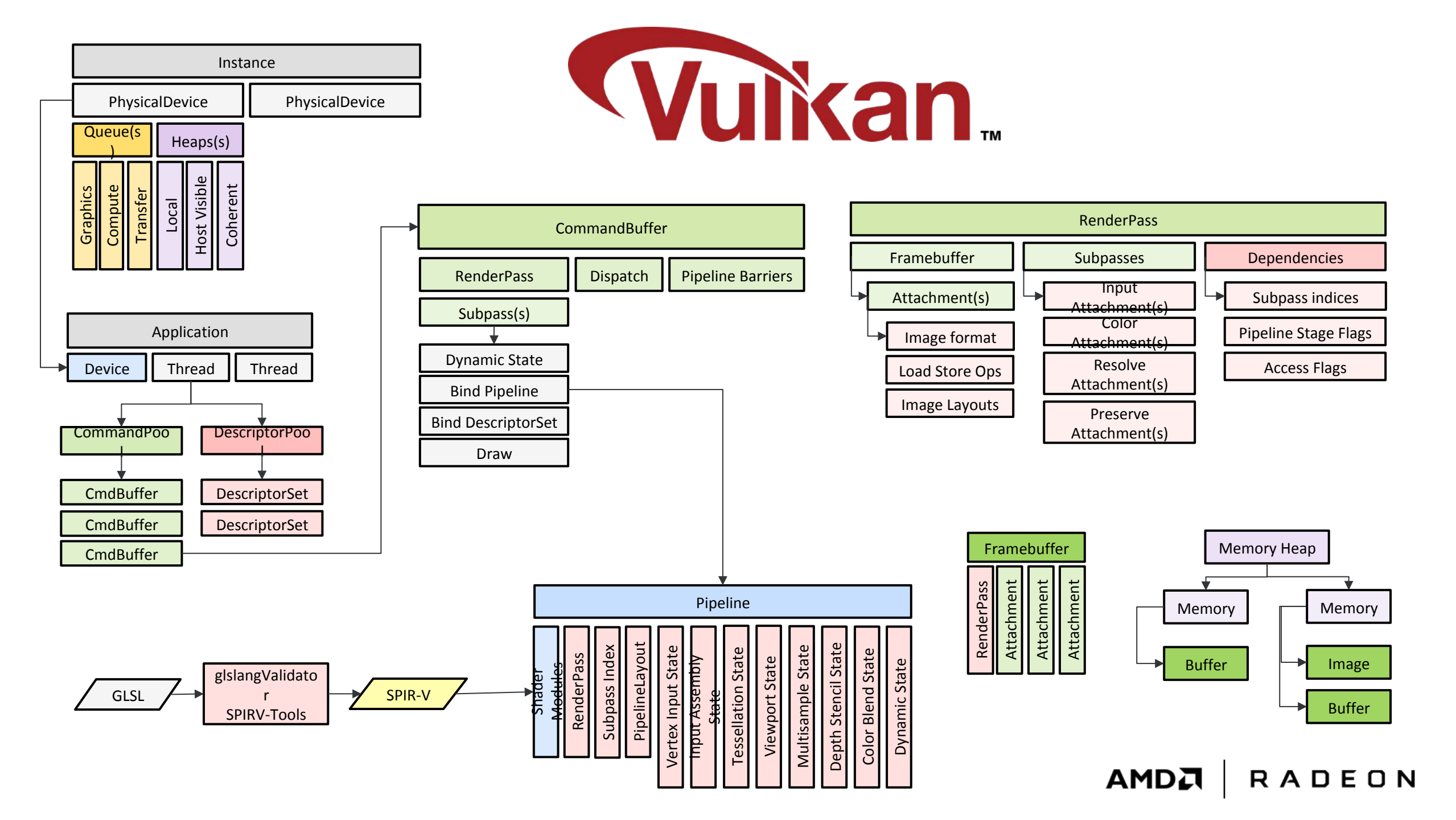
The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

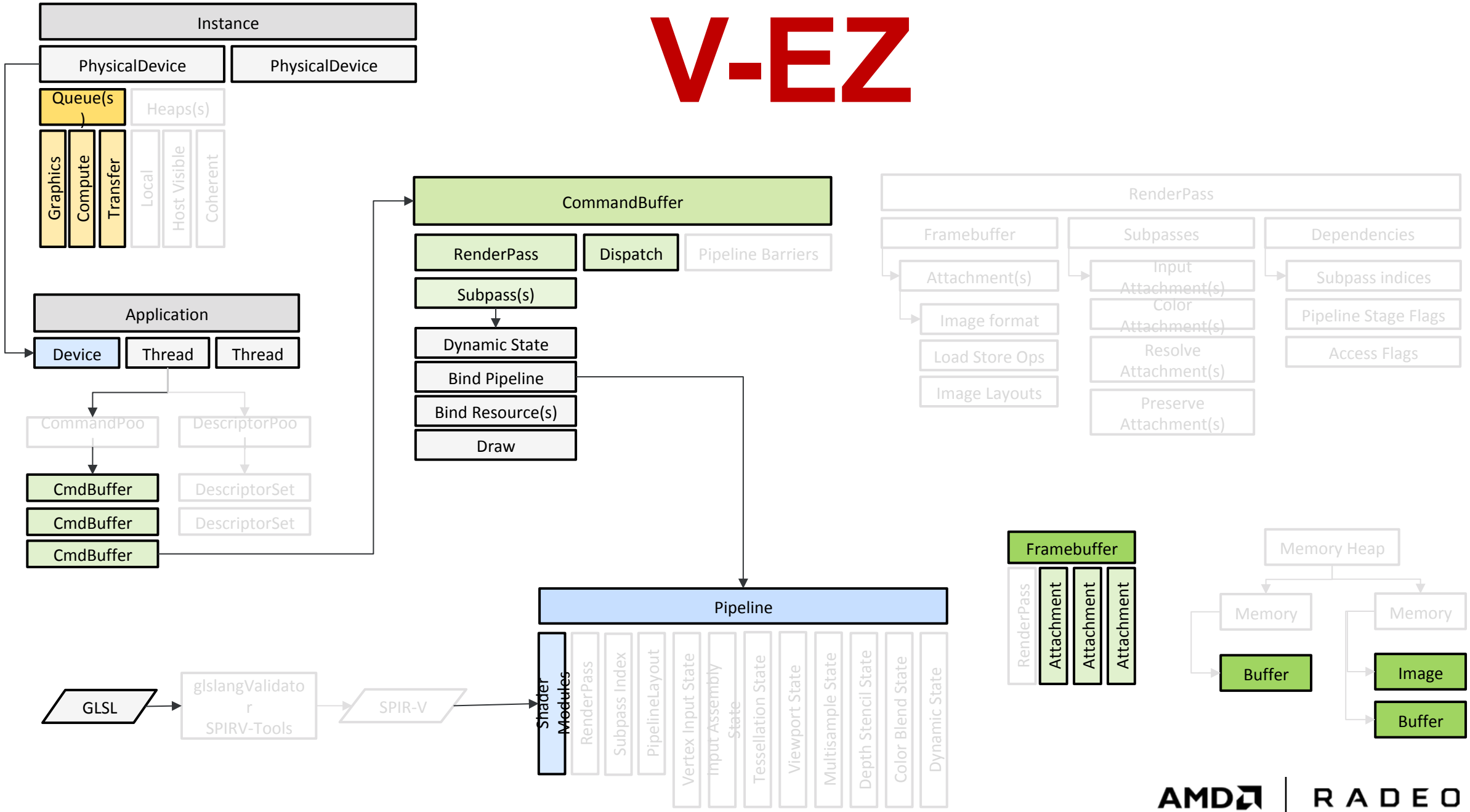
AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION


Instance
PhysicalDevice
PhysicalDevice
Queue(s)
Heaps(s)
Graphics
Compute
Transfer
Local
Host Visible
Coherent
Application
Device
Thread
Thread
CommandPoo
DescriptorPoo
CmdBuffer
CmdBuffer
CmdBuffer
DescriptorSet
DescriptorSet
CommandBuffer
RenderPass
Dispatch
Pipeline Barriers
Subpass(s)
Dynamic State
Bind Pipeline
Bind DescriptorSet
Draw
RenderPass
Framebuffer
Attachment(s)
Image format
Load Store Ops
Image Layouts
Subpasses
Input Attachment(s)
Color Attachment(s)
Resolve Attachment(s)
Preserve Attachment(s)
Dependencies
Subpass indices
Pipeline Stage Flags
Access Flags
Pipeline
Shader Modules
RenderPass
Subpass Index
PipelineLayout
Vertex Input State
Input Assembly State
Tessellation State
Viewport State
Multisample State
Depth Stencil State
Color Blend State
Dynamic State
GLSL
glslangValidator
SPIRV-Tools
SPIR-V
Framebuffer
RenderPass
Attachment
Attachment
Attachment
Memory Heap
Memory
Memory
Buffer
Image
Buffer
AMD
RADEON

V-EZ



AMD | RADEON