MULTIPLATFORM GPU RAY-TRACING SOLUTIONS WITH FIRERENDER AND FIRERAYS

TAKAHIRO HARADA, DMITRY KOZLOV

3/16/2016

AMD

MULTIPLATFORM GPU RAY-TRACING SOLUTIONS

- FireRender
  - All in package (ray casting, shading)
  - For renderer users
  - Output - Rendered image
  - Physically based rendering library
- FireRays
  - For renderer developers
  - Output - Intersections
  - Ray intersection library
- Implemented using OpenCL

AGENDA

- FireRender
  - Features
  - Architecture
  - Examples
- FireRays
  - Introduction
  - Technology
  - Examples

FIRERENDER

FIRERENDER

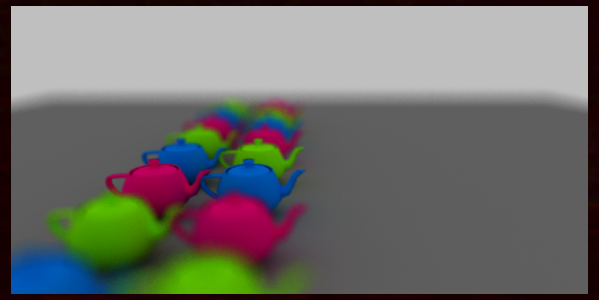
- “Fast high quality rendering everywhere”

  GPU Path tracing OpenCL

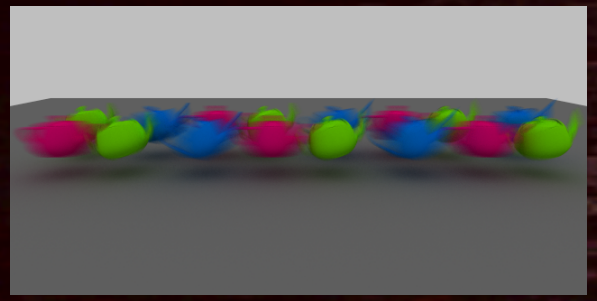
- C API
- OpenCL 1.2
- Multi platform solution
  - OS (Windows, Linux)
  - Vendor (AMD,…)

FEATURES

- Camera



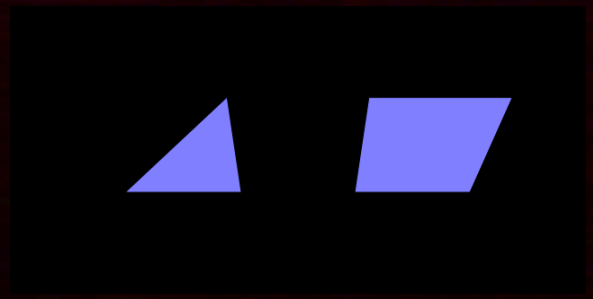
DOF



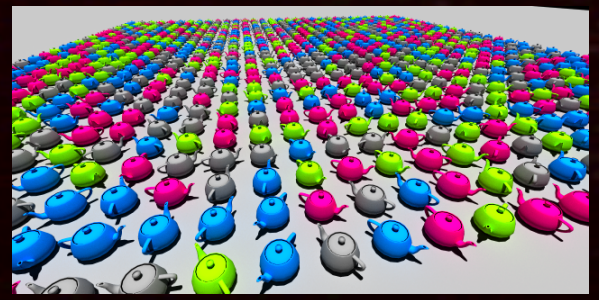
Motion Blur

Bokeh

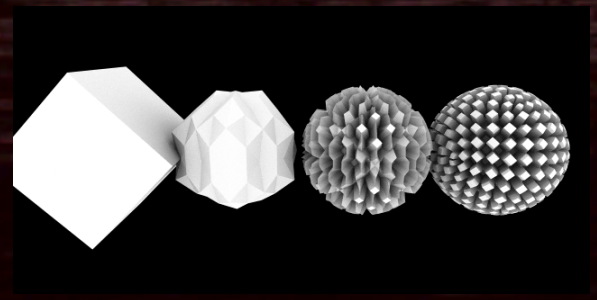
- Geometry



Mesh

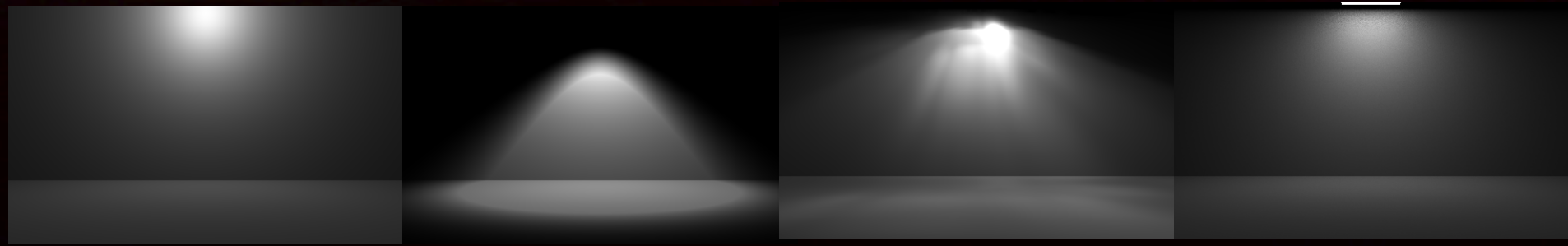


Instancing



Subdivision

- Lights



Point

Spot

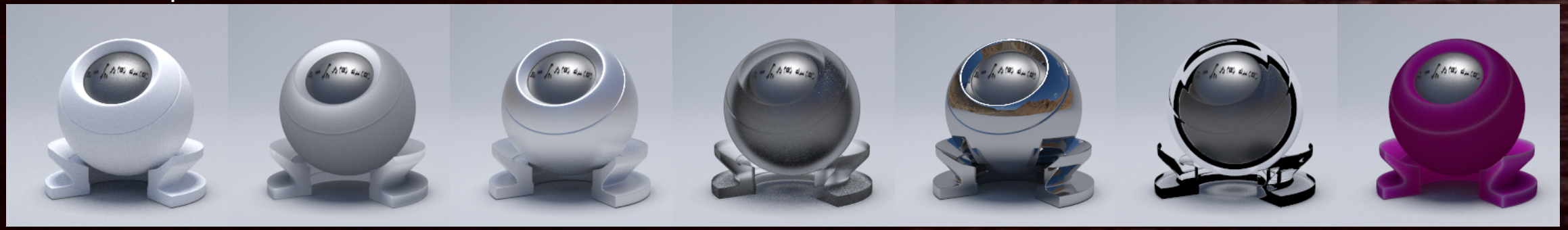
IES

Area

FEATURES

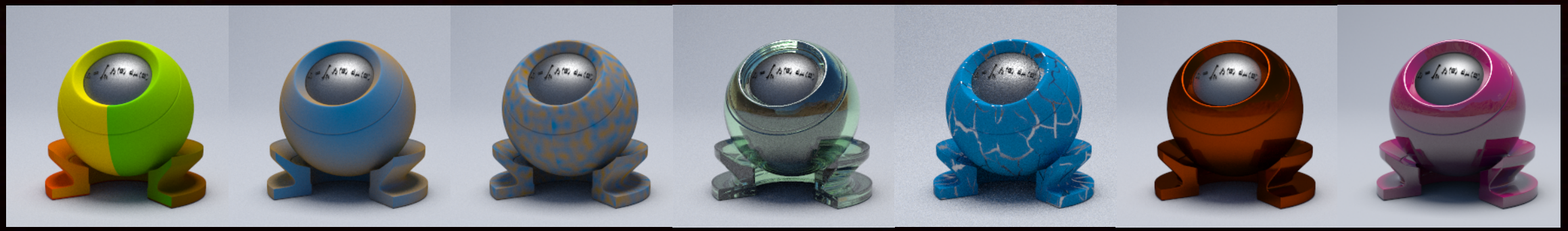
MATERIALS

- BSDFs
  - Basic components



Diffuse reflection | Diffuse refraction | Glossy reflection | Glossy refraction | Spec. reflection | Spec. refraction | SSS

- Shader graph
  - Arbitrary connection of shader nodes for flexible shading system



Input Lookup | Arithmetic | Procedural | Blend BSDFs | Example | Example | Example

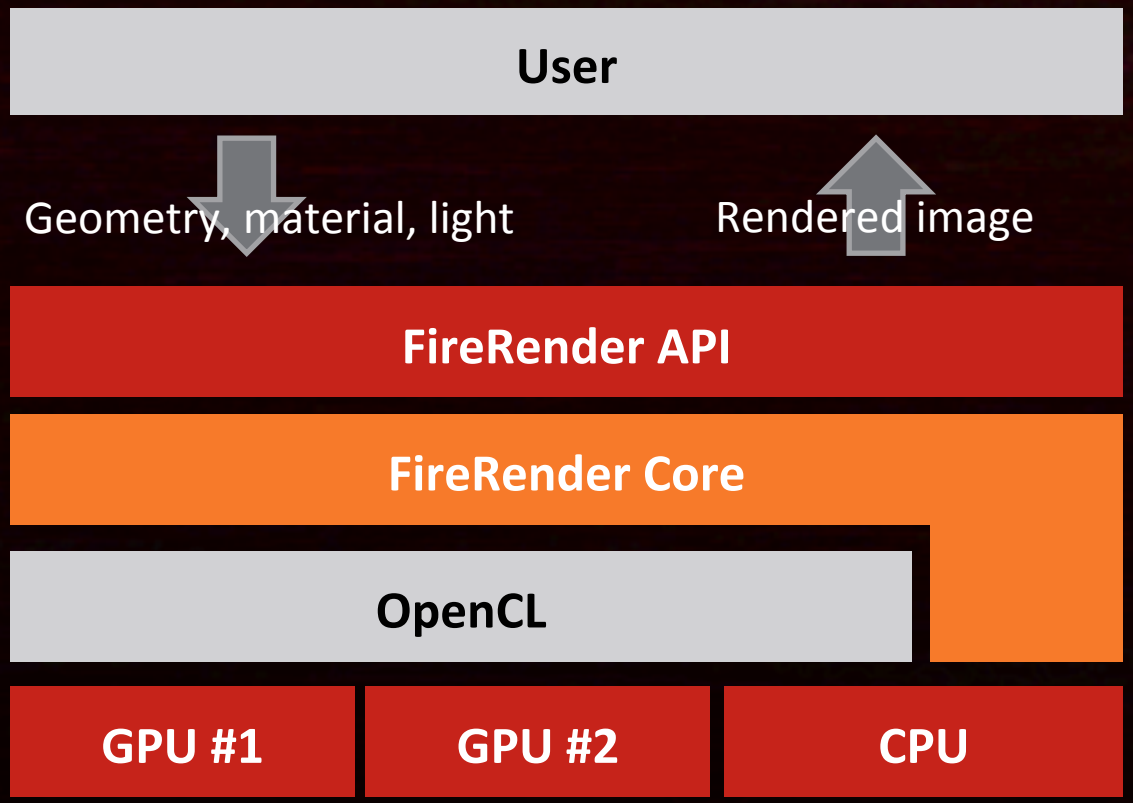
RADEON
TECHNOLOGIES GROUP

FIRERENDER ARCHITECTURE

HOW FIRERENDER IS BUILT

AMD

ARCHITECTURE OVERVIEW



- Input: scene info
- Output: rendered image

- Runs on
  - Single GPU
  - Multiple GPUs
  - CPUs
  - Mix of these

FIRERENDER CORE

AMD

IMPLEMENTATION

- Split kernel implementation
  - Modular
    - Easy to modify, extend, debug
    - Change behavior by replacing a kernel
      - Ray casting kernel => FireRays, Vector displacement, Out of core support
      - Camera => Bake camera
  - High GPU utilization
    - Less GPR usage
    - Better performance

IMPLEMENTATION

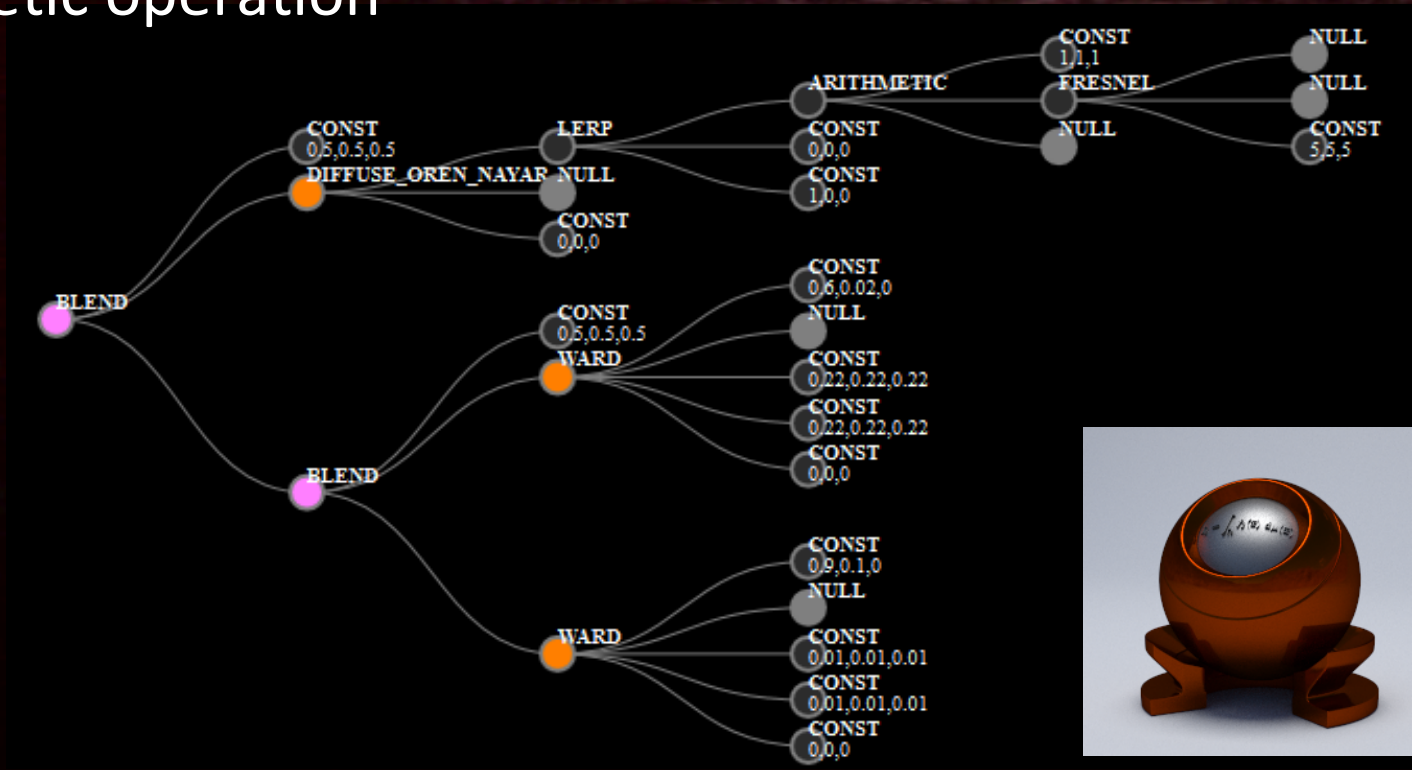
- Textures
  - Manually managed
  - No limit to the number of textures (up to the memory limit)
- Lights
  - Many lights are challenging
    - Slow convergence
  - Optimized for many lights
    - GPU optimized stochastic light culling

6,000 area lights

IMPLEMENTATION

MATERIAL SYSTEM

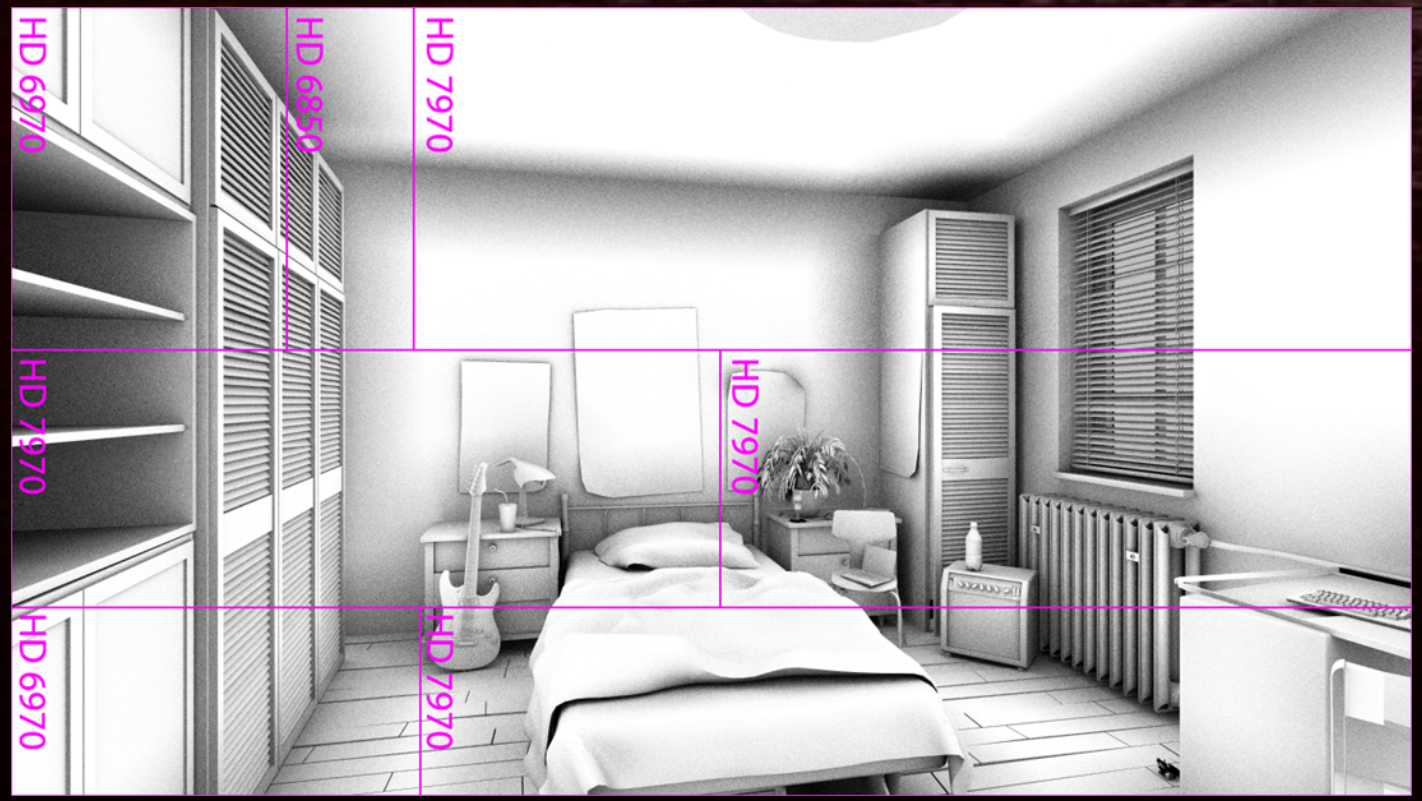
- Material == shader graph
  - –Arbitrary node connection
  - –BSDFs (Closures), textures, arithmetic operation
- For each shading point
  - –Traverse the graph
    - => BSDF + Parameters
  - –Evaluate BSDF
  - –Sample using BSDF



IMPLEMENTATION

MULTI GPU SUPPORT

- Make single frame render faster
- Screen split
- Support heterogeneous GPUs
- Load balancing
  - Minimize the idle time on GPU



7 GPUs

FIRERENDER API

HOW TO USE FIRERENDER

AMD

FIRERENDER API EXAMPLE (1/4)

SIMPLE RENDER

Initialization

```
// Create OpenCL context using a single GPU
fr_context context;
fr_material_system matsys;
fr_scene scene;

frCreateContext(FR_API_VERSION, FR_CONTEXT_OPENCL, FR_CREATION_FLAGS_ENABLE_GPU0, NULL, NULL, &context);
frContextCreateMaterialSystem(context, 0, &matsys);
frContextCreateScene(context, &scene);
frContextSetScene(context, scene);
```

Camera Set up

```
// Create camera
fr_camera camera;
frContextCreateCamera(context, &camera);
frCameraLookAt(camera, 5, 5, 20, 0, 0, 0, 0, 1, 0);
frCameraSetFocalLength(camera, 75.f);
frSceneSetCamera(scene, camera);
```

FIRERENDER API EXAMPLE (2/4)

SIMPLE RENDER

Create Point Light

```
// Create point light
fr_light light;
frContextCreatePointLight(context, &light);
frLightSetTransform(light, FR_TRUE, &lightm(0,0));
frPointLightSetRadiantPower3f(light, 255, 241, 224);
frSceneAttachLight(scene, light);
```

Create Mesh

```
// Create plane mesh
fr_shape plane;
frContextCreateMesh(context,...);
frSceneAttachShape(scene, plane);
```

Create shader

```
// Create simple diffuse shader
fr_material_node diffuse;
frMaterialSystemCreateNode(matsys, FR_MATERIAL_NODE_DIFFUSE, &diffuse);
frMaterialNodeSetInputF(diffuse, "color", 0.5f, 0.5f, 0.5f, 1.f);
frShapeSetMaterial(plane, diffuse);
```

FIRERENDER API EXAMPLE (3/4)

SIMPLE RENDER

Create frame buffer

```
// Create framebuffer to store rendering result
fr_framebuffer_desc desc;
desc.fb_width = 800;
desc.fb_height = 600;

fr_framebuffer_format fmt = {4, FR_COMPONENT_TYPE_FLOAT32};
fr_framebuffer frame_buffer;
frContextCreateFrameBuffer(context, fmt, &desc, &frame_buffer);
frFrameBufferClear(frame_buffer);
frContextSetAOV(context, FR_AOV_COLOR, frame_buffer);
```

Render & Save

```
// Render
frContextRender(context);

// Save the result to file
frFrameBufferSaveToFile(frame_buffer, "simple_render.png");
```

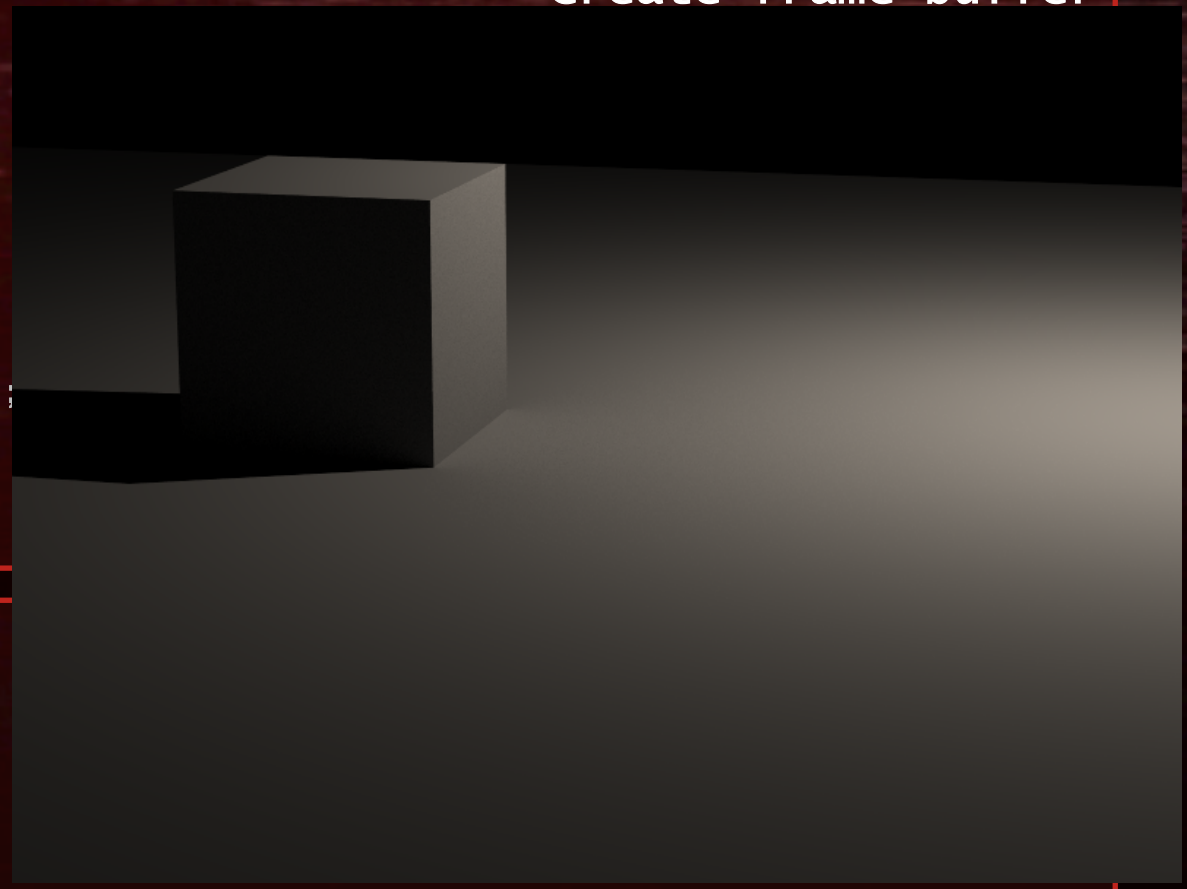
FIRERENDER API EXAMPLE (3/4)

SIMPLE RENDER

```
// Create framebuffer to store rendering result
fr_framebuffer_desc desc;
desc.fb_width = 800;
desc.fb_height = 600;

fr_framebuffer_format fmt = {4, FR_COMPONENT_TYPE_FLOAT32};
fr_framebuffer frame_buffer;
frContextCreateFrameBuffer(context, fmt, &desc, &frame_buffer);
frFrameBufferClear(frame_buffer);
frContextSetAOV(context, FR_AOV_COLOR, frame_buffer);
```

```
// Render
frContextRender(context);

// Save the result to file
frFrameBufferSaveToFile(frame_buffer, "simple_render.png");
```

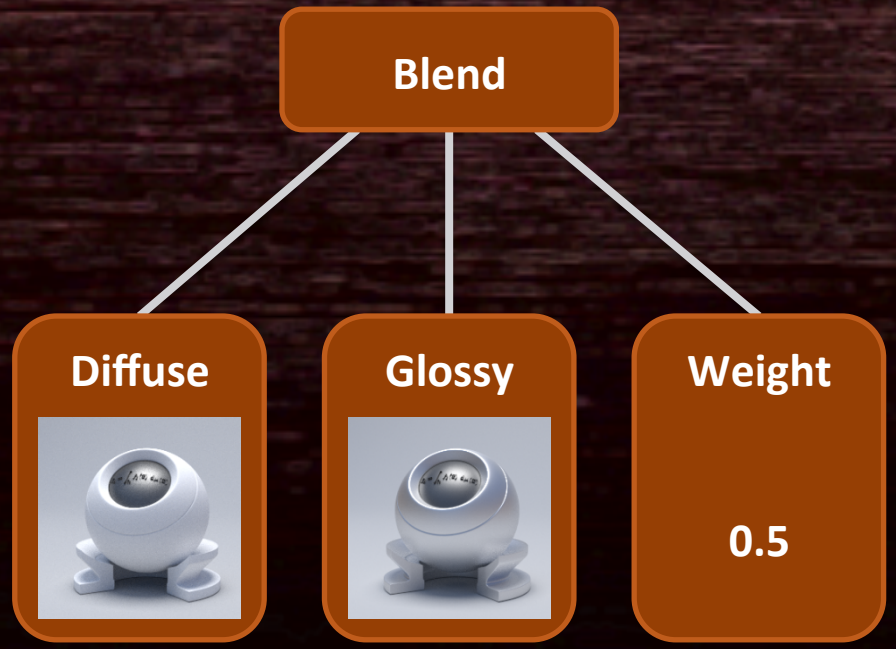
Create frame buffer



FIRERENDER API EXAMPLE (4/4)

LAYERED SHADER

```
// Create BXDFs
fr_material_node base, top;
{
    frMaterialSystemCreateNode(matsys, FR_MATERIAL_NODE_DIFFUSE, &base);
    frMaterialNodeSetInputF(base, "color", r0, g0, b0, 1.f);

    frMaterialSystemCreateNode(matsys, FR_MATERIAL_NODE_MICROFACET, &top);
    frMaterialNodeSetInputF(top, "color", r1, g1, b1, 1.f);
    frMaterialNodeSetInputF(top, "roughness", 0.1f, 0.f, 0.f, 1.f);
}
// Create a layered shader
fr_material_node layered;
{
    frMaterialSystemCreateNode(matsys, FR_MATERIAL_NODE_BLEND, &layered);
    frMaterialNodeSetInputN(layered, "color0", base);
    frMaterialNodeSetInputN(layered, "color1", top);
    frMaterialNodeSetInputF(layered, "weight", 0.5f, 0.5f, 0.5f, 1.f);
}
```



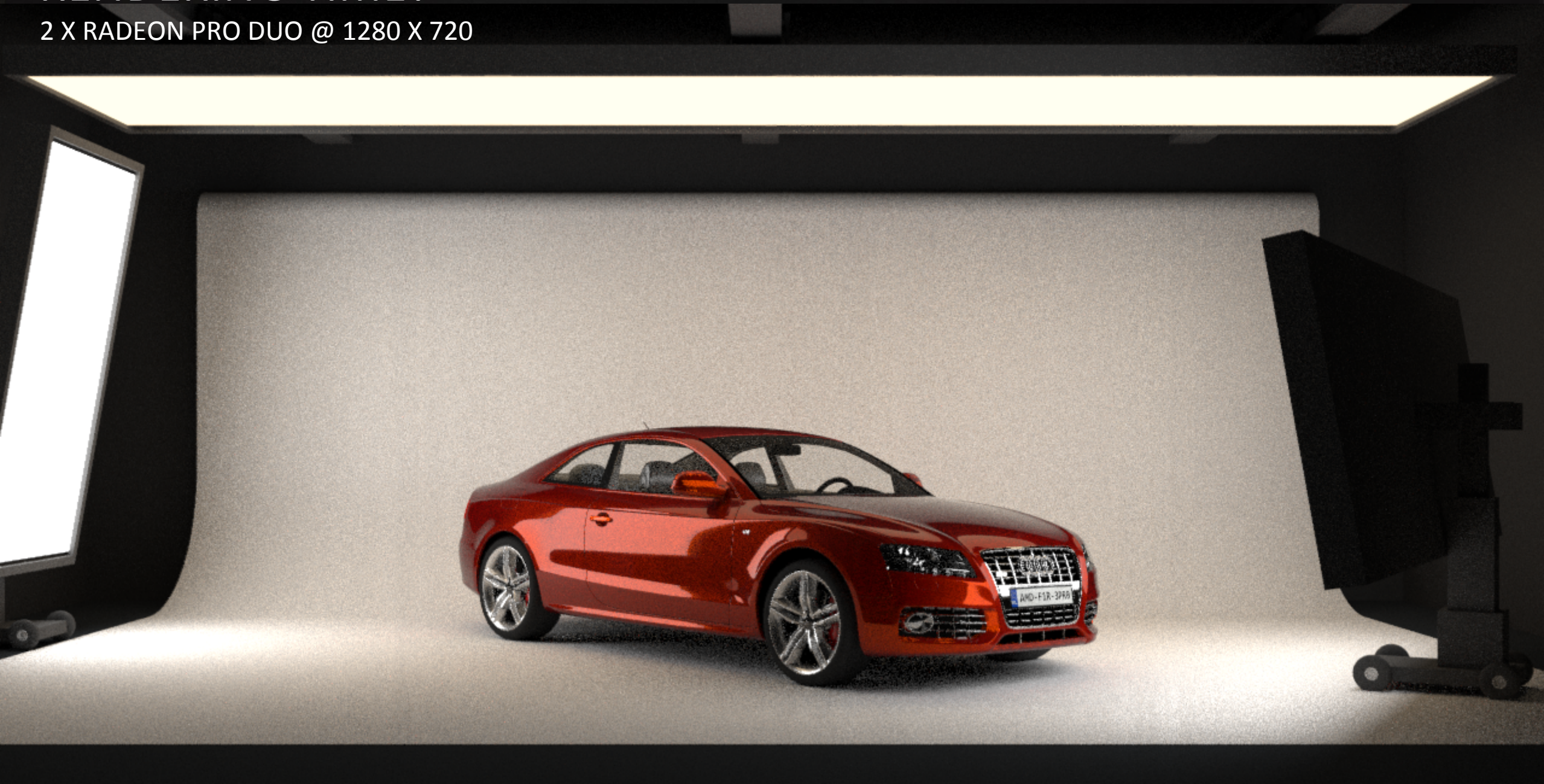
RADEON
TECHNOLOGIES GROUP

PERFORMANCE

AMD

RENDERING TIME?

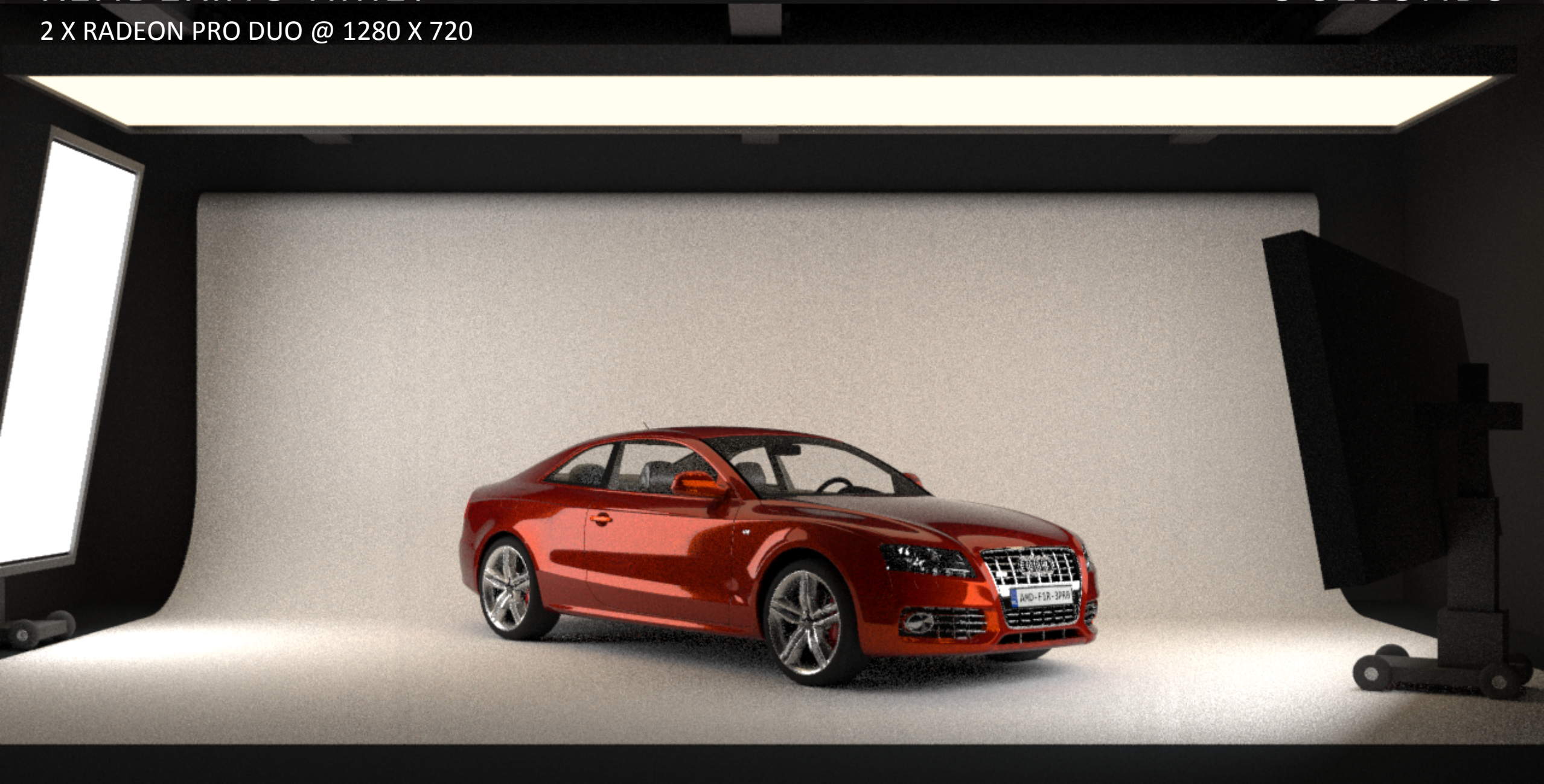
2 X RADEON PRO DUO @ 1280 X 720



RENDERING TIME?

~ 5 SECONDS

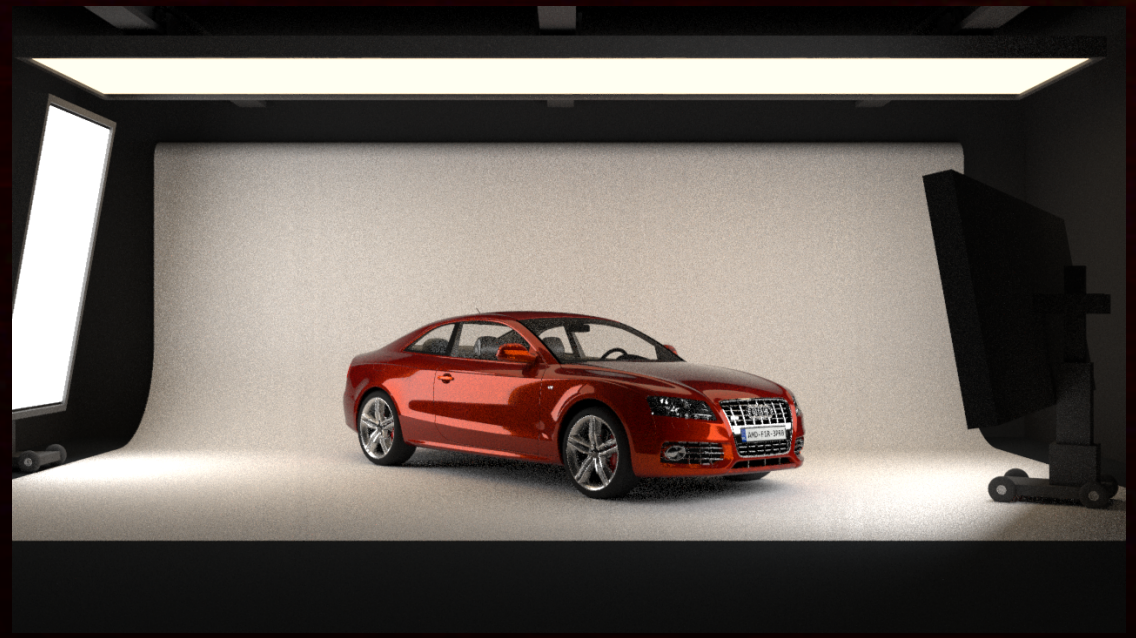
2 X RADEON PRO DUO @ 1280 X 720



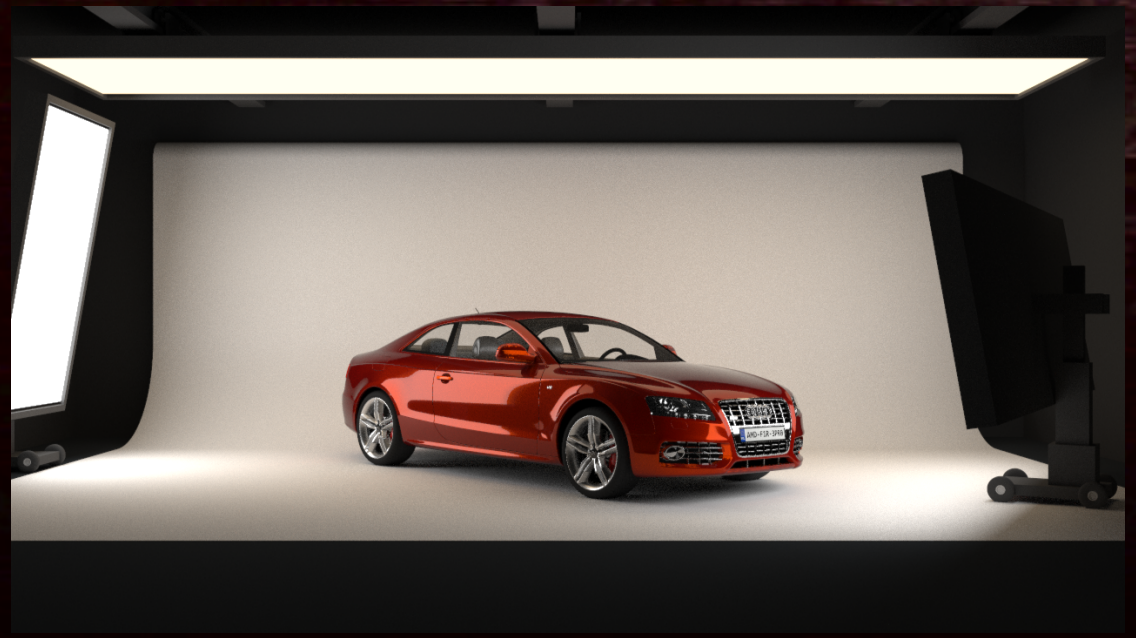
PERFORMANCE

2 X RADEON PRO DUO @ 1280 X 720

5s



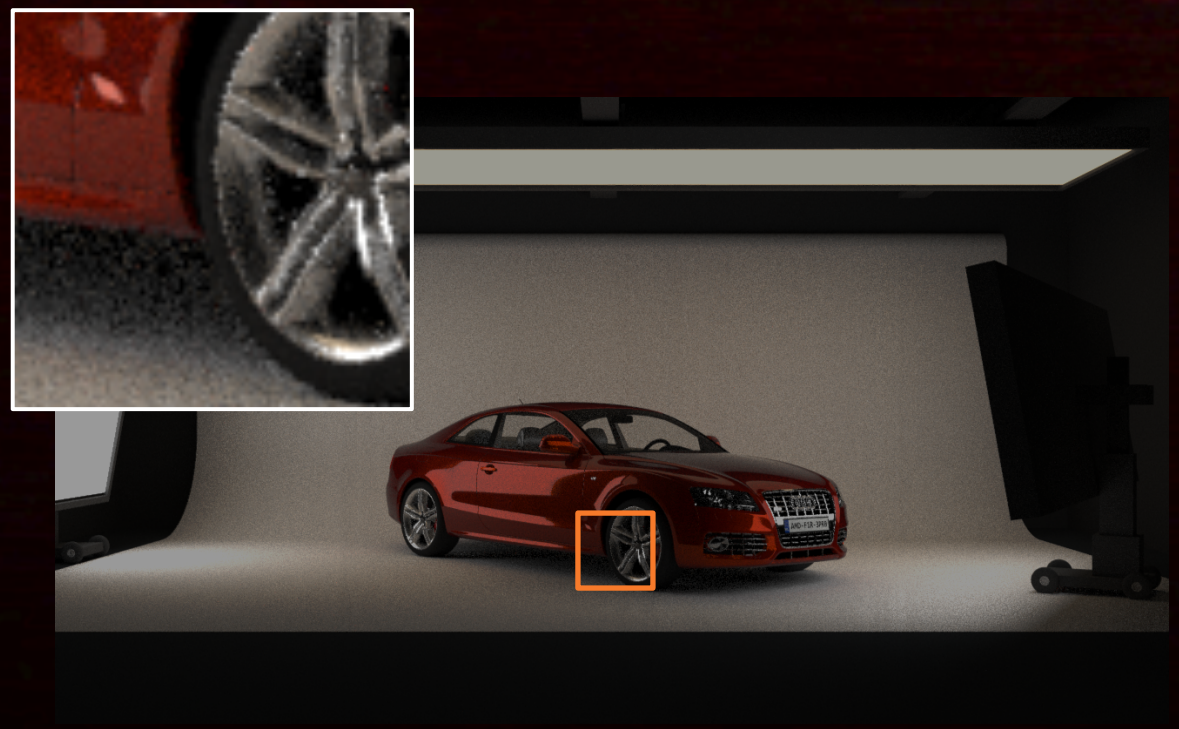
30s



PERFORMANCE

2 X RADEON PRO DUO @ 1280 X 720

5s



30s

PERFORMANCE

2 X RADEON PRO DUO @ 1280 X 720

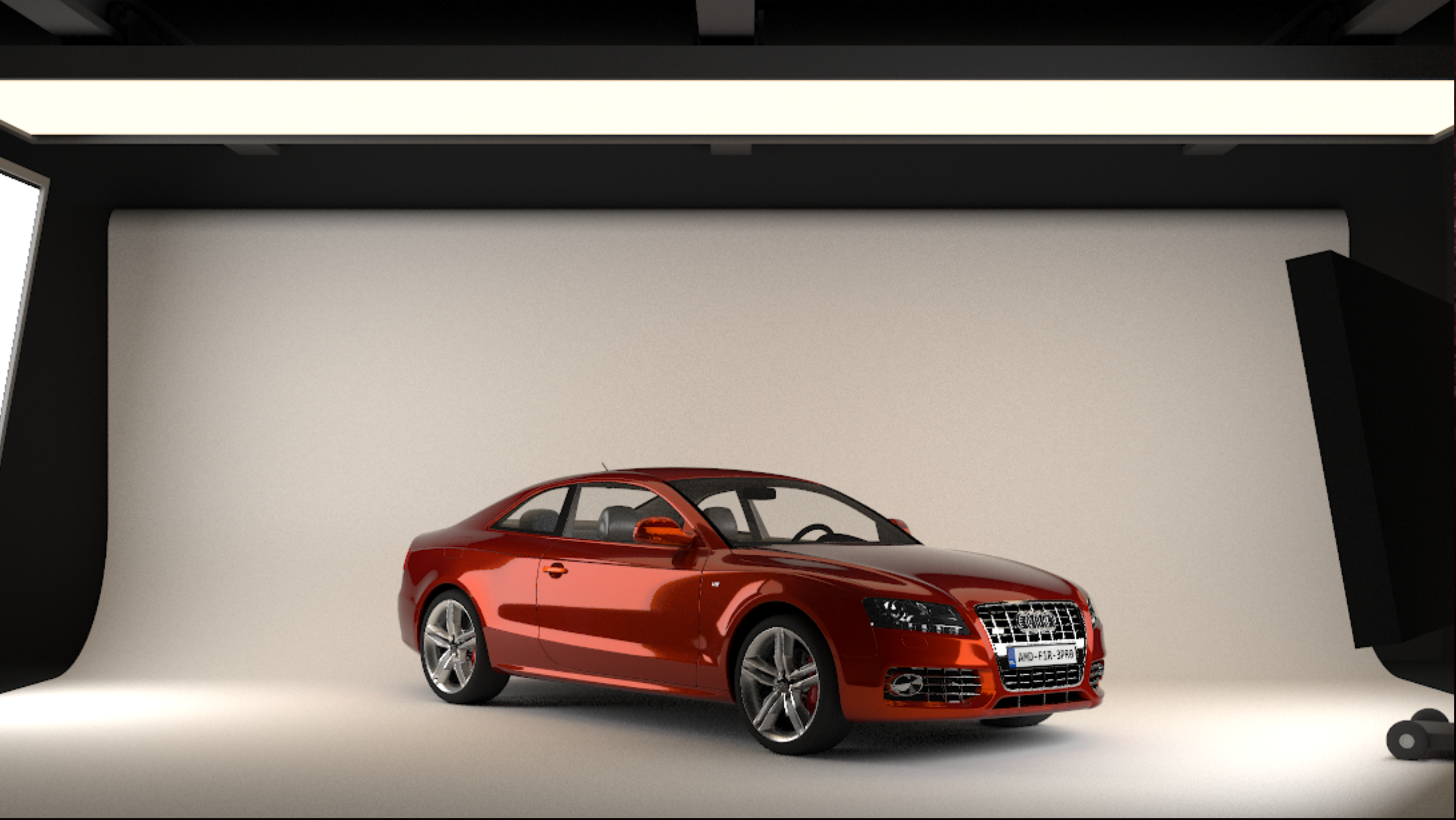
5s

30s

RADEON
TECHNOLOGIES GROUP

EXAMPLES

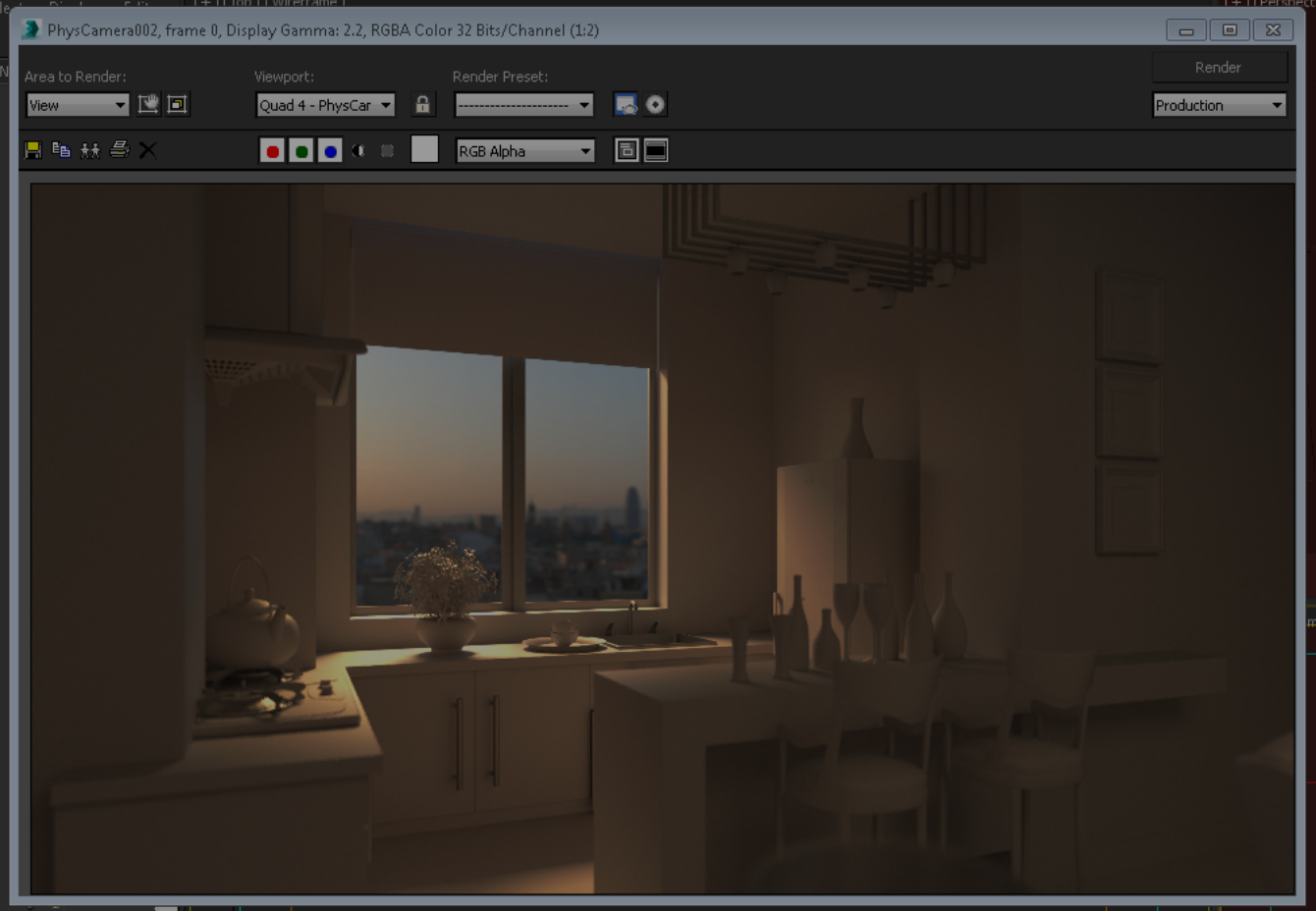
AMD

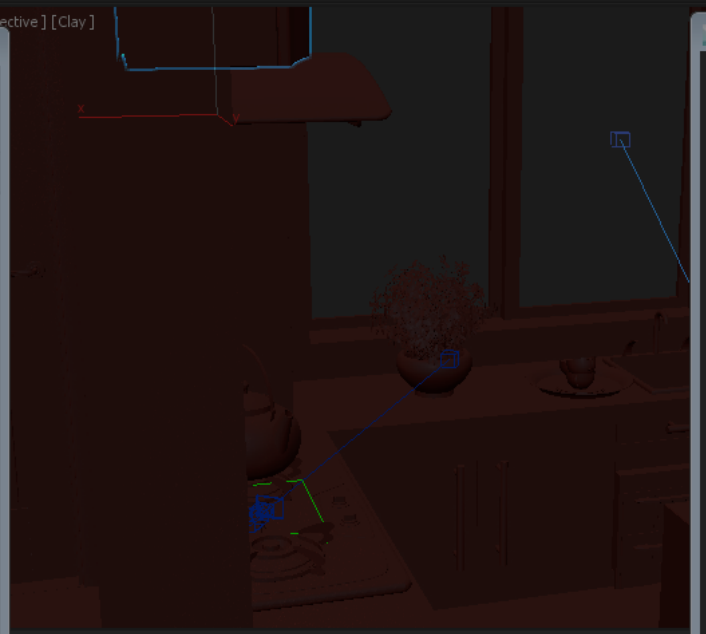

AMD-F1R-3PRR

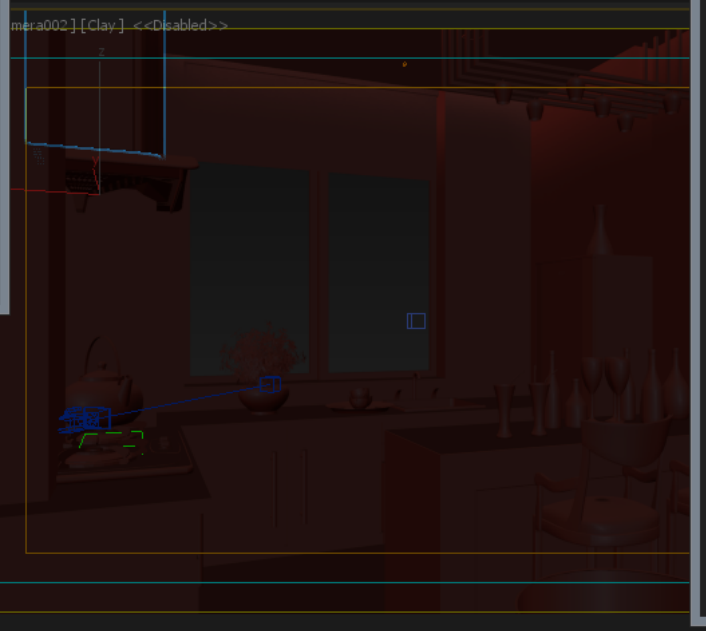

IRL
AMD-F1R-3PR0

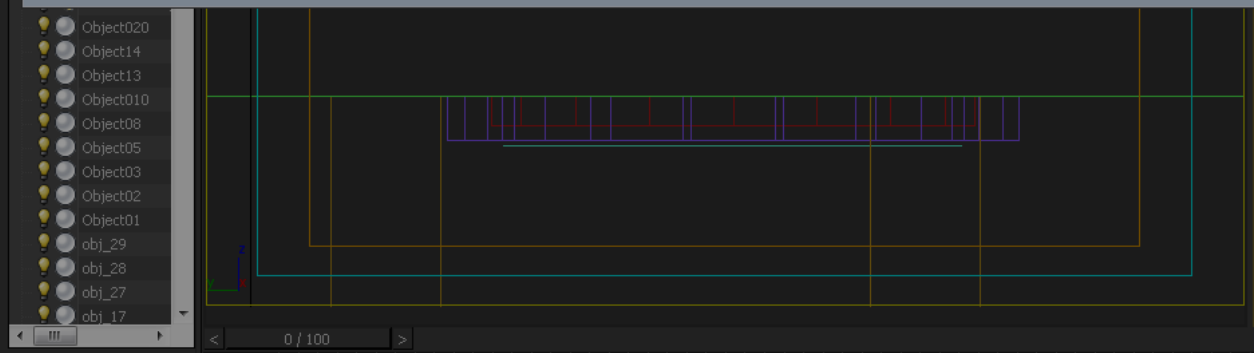

classe 9
max
min

3DS MAX PLUGIN DEMO









FIRERENDER SDK

- Open to registered developers

- http://developer.amd.com/tools-and-sdks/graphics-development/firepro-sdk/amd-firerender-technology/
- FirePro.Developers@amd.com

FIRERAYS

MOTIVATION

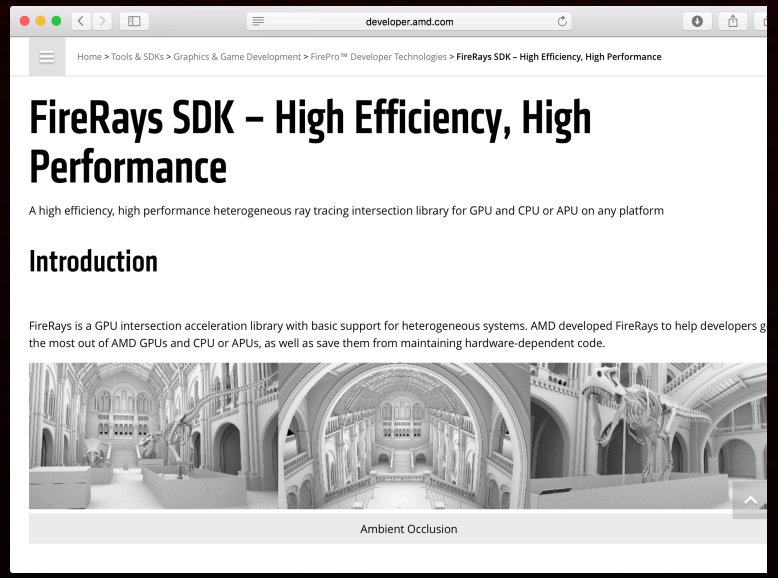
- Photorealistic rendering
- Collision detection
- Particle physics
- Game AI
- Medical imaging
- Predictive rendering
- Sound propagation
- Real-time effects

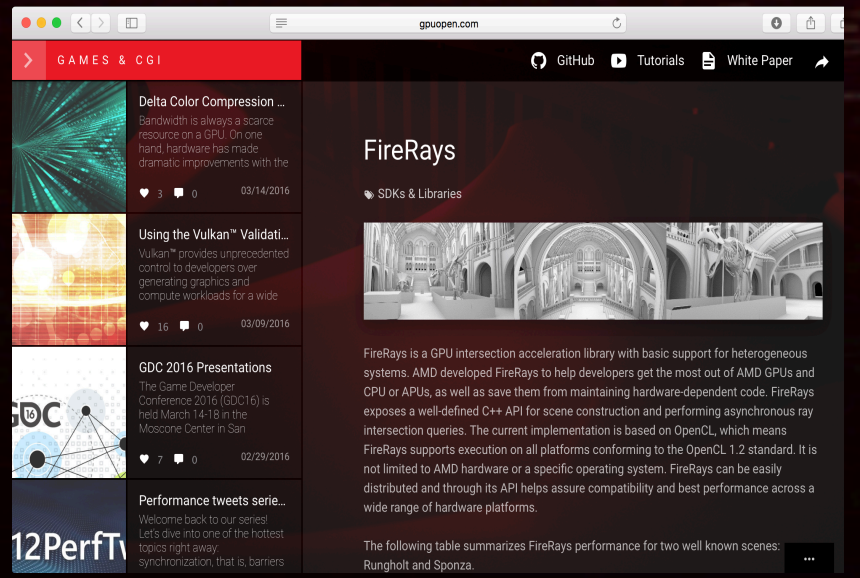
GPU ACCELERATION

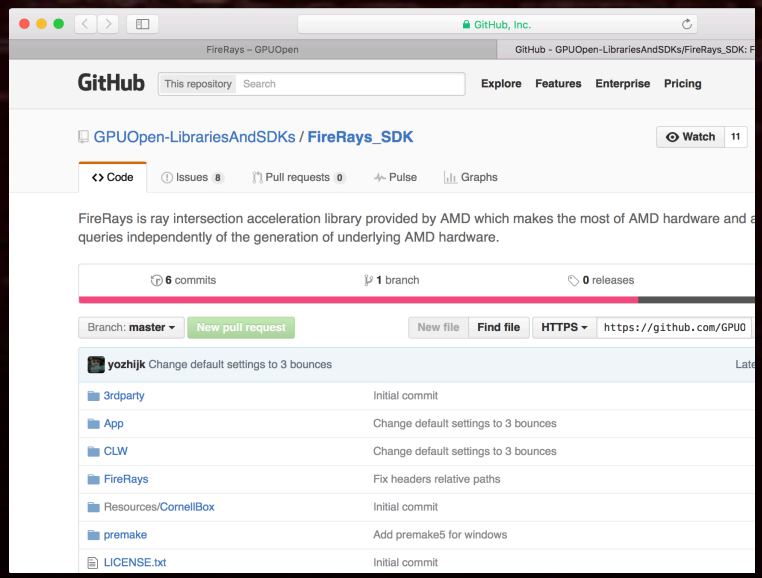
- Massively parallel architecture
- Naïve approach does not work on GPUs
- Variety of hardware to support
- Rapidly changing software
  - –DX/GL
  - – OpenCL
  - – Vulkan/DX12
  - – HCC

GPU ACCELERATION: FIRERAYS

- Fast intersection API:
  - Best perf on AMD
  - CPU/GPU/MGPU
- Cross-platform
  - Windows / Linux / OSX
- Cross-vendor
- Open-source renderer
- Hosted on GPUOpen

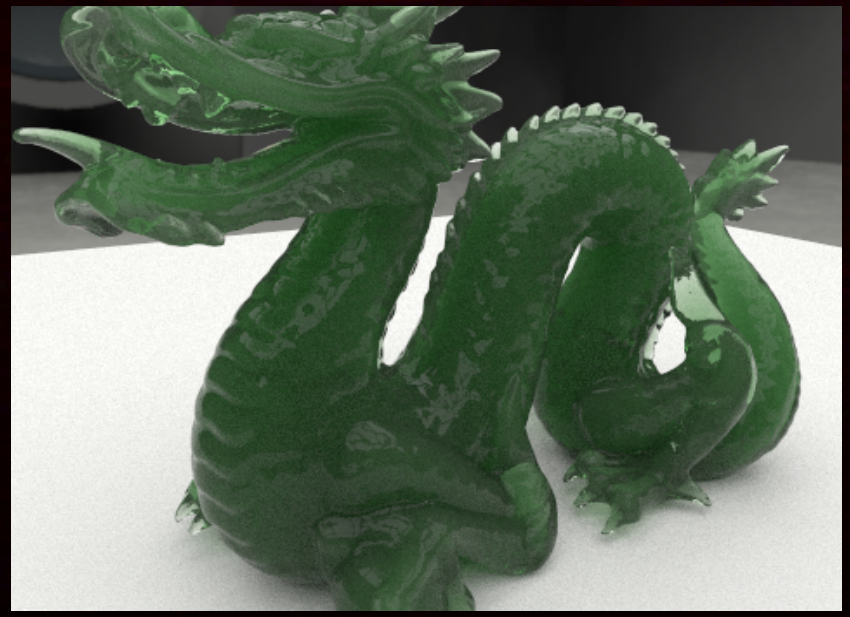






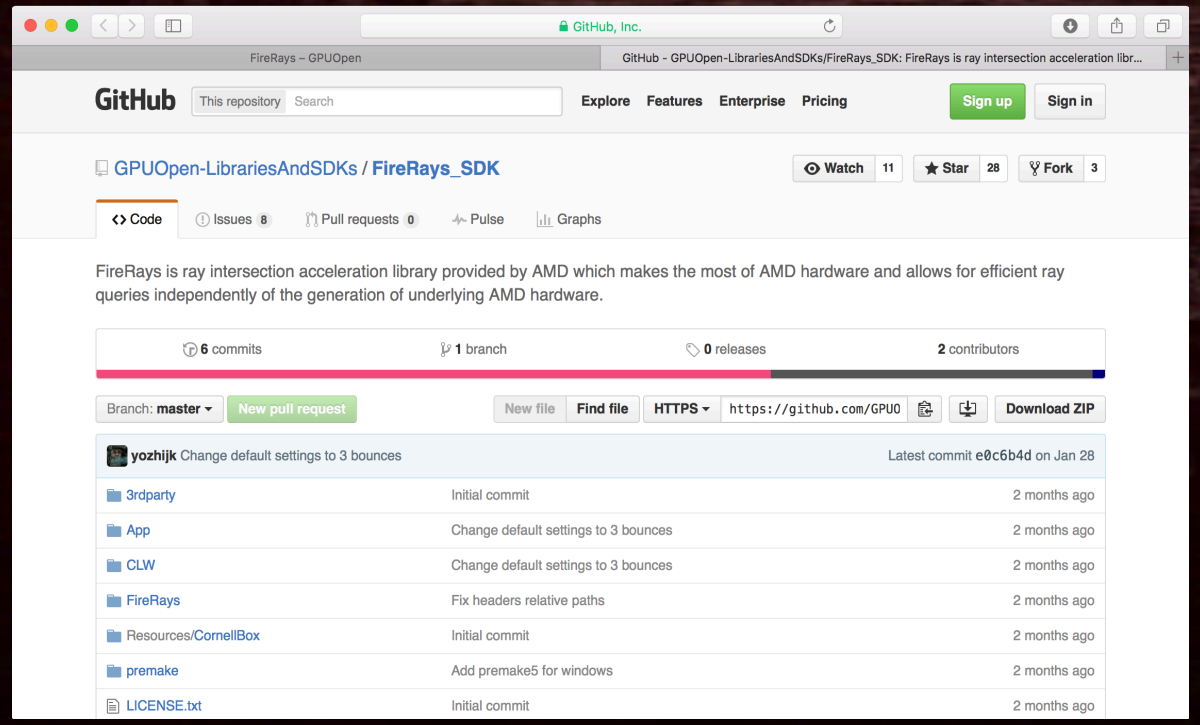
GPU ACCELERATION: FIRERAYS

- Triangle meshes
- Instancing support
- Fast traversal
- Ray masking
- CPU/GPU BVH
- OpenCL interop



FIRERAYS: SDK

- SDK is hosted on GitHub
- Library binaries:
  - Windows
  - Mac OS (coming soon)
  - Linux (coming soon)
- Open-source sample renderer
  - Efficient streaming path-tracer
  - Illustrates FireRays usage
- Benchmark

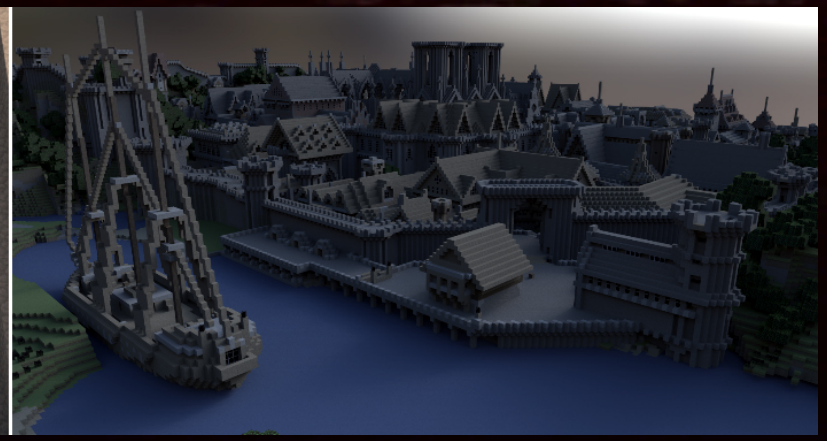


https://github.com/GPUOpen-LibrariesAndSDKs/FireRays_SDK

FIRERAYS: TECHNOLOGY

- High quality CPU SBVH
  - Takes longer to build
  - Fastest traversal
  - Stackless and short LDS stack
  - Good for final rendering
- GPU HLBVH
  - Fastest build
  - Slower traversal (stacked LDS)
  - Good for animation & preview





FIRERAYS: EXAMPLE RENDERER

- Fully open-source
- Reference path tracer
- Microkernel based
- Efficient re-compaction
- Supports multiple GPUs
- Implemented using FireRays OpenCL interop

- Compound materials
  - Diffuse
  - Reflection / refraction
  - Microfacet Blinn / GGX / Beckmann
  - Blend
  - Bump
  - Volume scattering
  - SSS

FIRERAYS: APPLICATIONS

- Renderer developers
  - Easy integration
  - Cross-platform
- Physics researchers
  - Easy to use API
  - Not specific to CG
- CG researchers
  - Fast prototyping
  - Open-source renderer
  - Going fully open-source!
- Game developers
  - Light baking
  - Future effects
  - We are going Vulkan!

FIRERAYS: API USAGE

- Pass your meshes into FireRays
- Setup ray buffers
  - From the host
  - Or CL kernel
- Query intersections
- Handle intersections

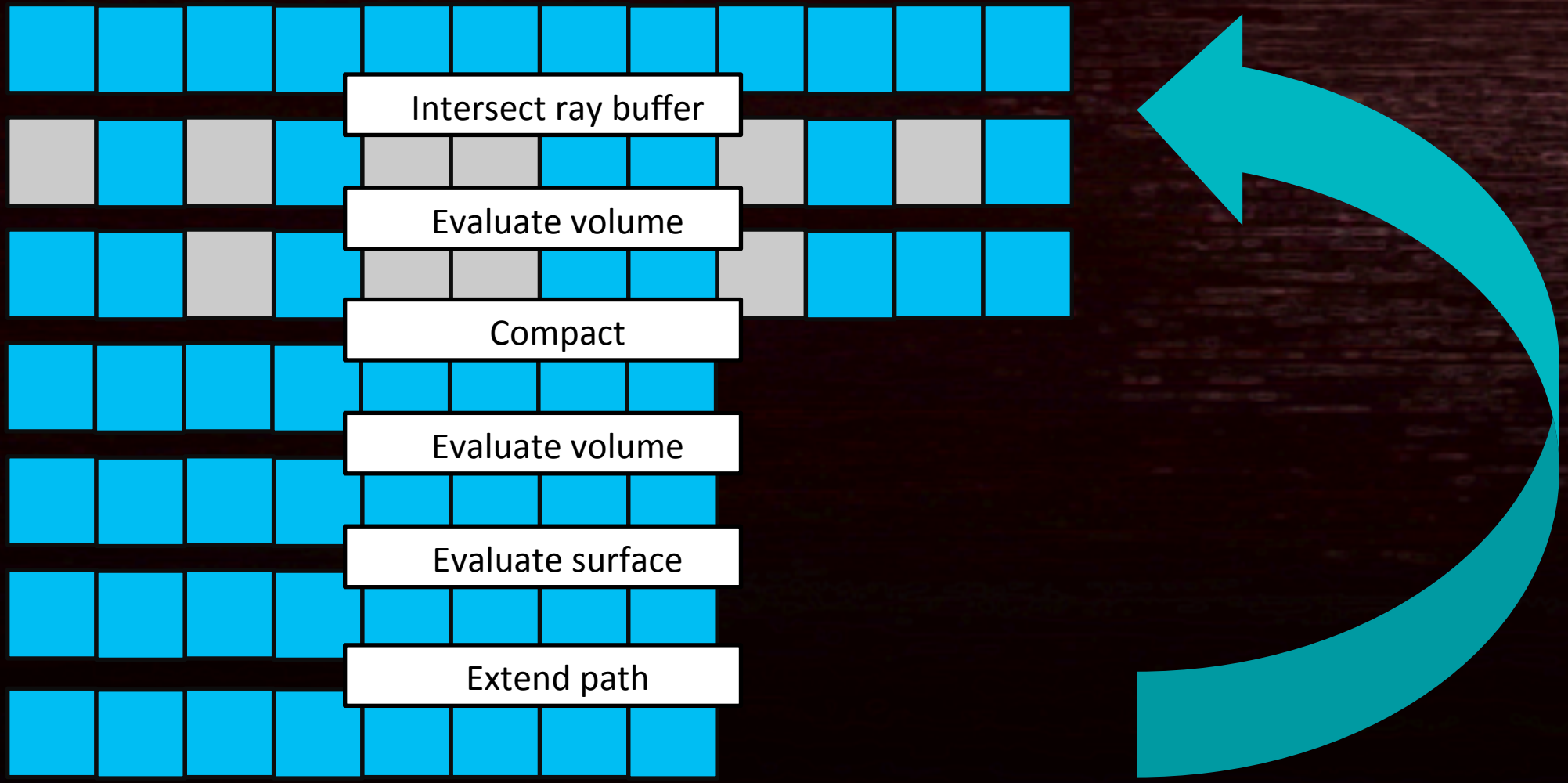
```
 // Enumerate all shapes in the scene
for (int i = 0; i < (int)shapes.size(); ++i)
{
        Shape* shape = nullptr;
        shape = api->CreateMesh(vertices, numvertices, sizeof(float3),
                                indices, 0, nullptr, numprims);

        shape->SetTransform(shapes[i].m, inverse(shapes[i].m));
        api->AttachShape(shape);
}


api->Commit();
```

```
// Intersect ray batch
api->IntersectBatch(rays[pass & 0x1], hitcount[pass & 0x1], maxrays, intersections, nullptr, nullptr);
```

FIRERAYS: BEST PRACTICES

- Use two buffers ping-pong
  - Helps to handle many bounces
- Compact the work
  - Helps to avoid divergence
  - Better memory access
- Do not use mega kernels
  - Helps to reduce VGPR usage
- Try to generate coherent rays
- Use multiple API instances for mGPU
- Let GPUs work asynchronously
- Monte-Carlo is good to distribute to multiple GPUs
- Use lock-less interaction

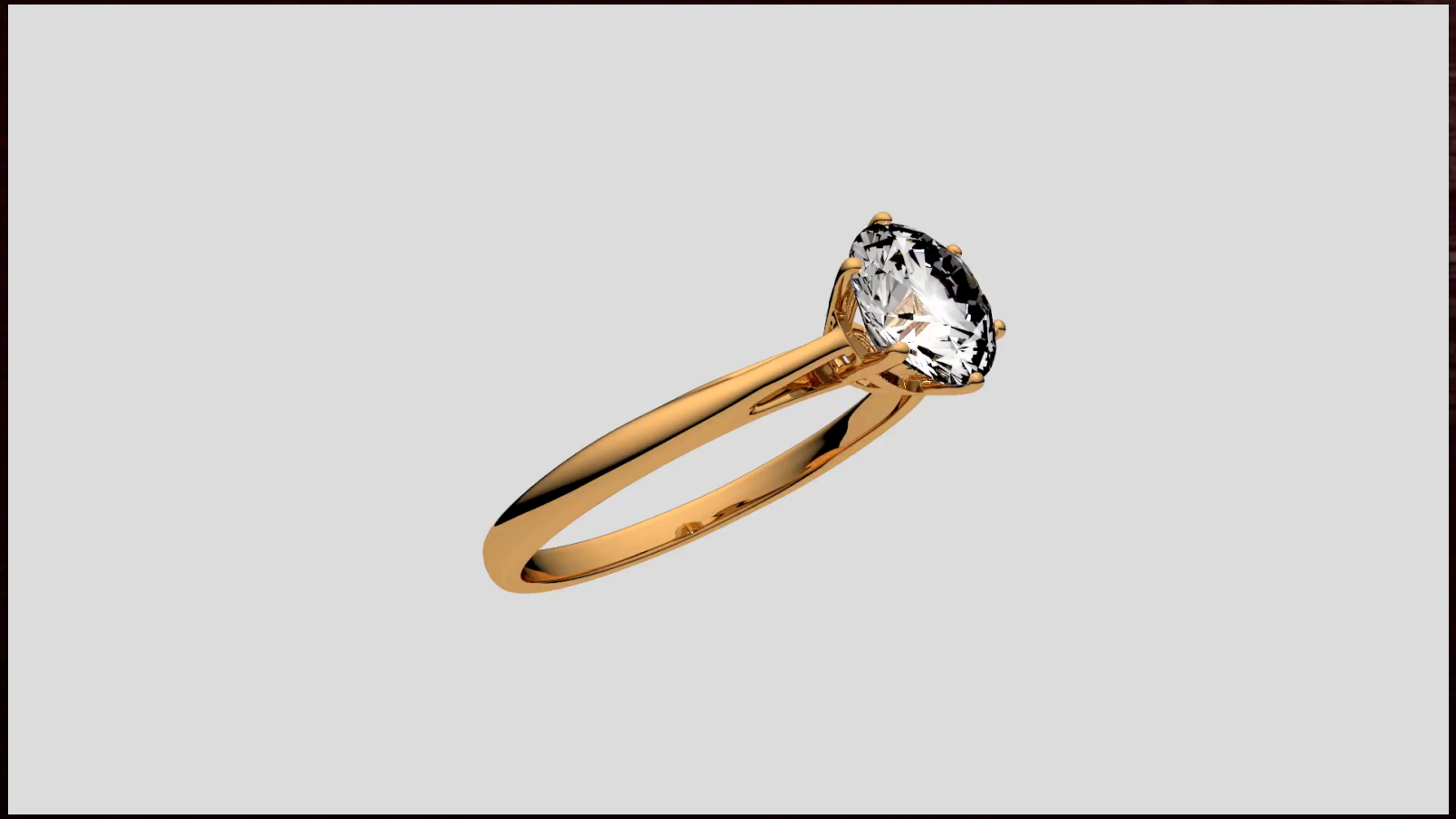
FIRERAYS: EXAMPLE RENDERER



FIRERAYS: USECASE

- Web interface
  - Interactive web product viewer
  - Interactive 3D configurator
- Specific requirements
  - Mainly jewelry rendering
  - RGB is not enough
  - Need real-time post effects

FIRERAYS: STAR-SHAPED FILTER



Bague Lady

Gemmyo © 2016

FIRERAYS: SPECTRAL RENDERING

Bague Lady

Gemmyo © 2016

FIRERAYS: WHAT'S NEXT?

- Go fully open-source on GPUOpen!
- Vulkan port
  - Gaming applications
  - Flexible 3D-compute interop
  - Asynchronous operation
- Non-batched mode
- FireRender backend
- Features
  - Faster builds / traversal
  - Subdivision & displacement
  - Out of core
  - Improved motion blur