Radeon ProRender and Radeon Rays in a Gaming Rendering Workflow

Takahiro Harada, AMD

2017/3

Agenda

- Introduction
  - Radeon ProRender & Radeon Rays
- Radeon Rays
  - Unity + Radeon Rays
  - Integration to real time applications
- Radeon ProRender

Introduction

Ray Tracing Solution from AMD

RADEON RAYS

- A GPU accelerated ray triangle intersection engine
- For low level engine developers
- OpenCL, Vulkan, C++ backends
- Full open source

RADEON PRORENDER

- A GPU accelerated light transport simulator
  - Computes global illumination using Monte Carlo ray tracing (path tracing)
- Intersection, shading, lighting, sampling, all in
- High level API
- Set up a scene, call render()
  - Returns you a nice render
- For high level engine developers
- OpenCL, C++ backend
- Open source planned

AMD’s Approach

- Not locking users to AMD platform
- Trying to make it run as many platforms as possible
- Using OpenCL 1.2, industry standard API
- We implement at least
  - GPU optimized OpenCL code
  - CPU optimized C++ code
    - better control, optimization compared to relying on OpenCL to run on the CPU
- Our solutions are competitive if compared on a CPU based solution
- As OpenCL is dynamically loaded, OCL isn’t necessary
  - If it cannot find OCL, it’ll fall back to the CPU implementation
- Most likely they run on your machine as they are

AMD's Approach

- Support multiple vendors, multiple OSes (Windows, Linux, MacOS)
  - No initial investment is necessary to use our solution
  - It does run on CPU too
- If you have an AMD GPUs, it is better
  - Better performance
  - Better experience
  - We do full testing on AMD GPUs
- Non AMD platforms, it depends on the vendor's OpenCL implementation
  - We do crash test on some vendor's GPUs
  - We disable some vendor's GPUs unfortunately because of their OpenCL bug (compiler, runtime)

This Talk

- How Radeon Rays, Radeon ProRender are used in game development process

RADEON
RAYS

RADEON
PRORENDER

Radeon Rays

Radeon Rays

- Can be used as a building block of a renderer
  - Global illumination renderer
  - Sound renderer (True Audio)
  - AI
- Comes with a reference renderer
- It could be used for lightmap baking and light probe calculation
  - Uses ray casting
  - There are a few game companies integrating Radeon Rays
  - We integrated Radeon Rays into Unity

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

Using Radeon Rays

- Simple C++ API

```
// Find closest intersection
void QueryIntersection(Buffer const* rays, int numrays, Buffer* hitinfos,
                       Event const* waitevent, Event** event) const;
// Find any intersection.
void QueryOcclusion(Buffer const* rays, int numrays, Buffer* hitresults,
                    Event const* waitevent, Event** event) const;
```

- Passing an array of rays and number of rays
- It fills hit results

Using Radeon Rays

- Embree is popular, but using Radeon Rays gives you more
- With Radeon Rays
  - It uses Embree for the CPU backend => Same performance is guaranteed
  - You can turn on the GPU backend => Performance improvements when you have a GPU

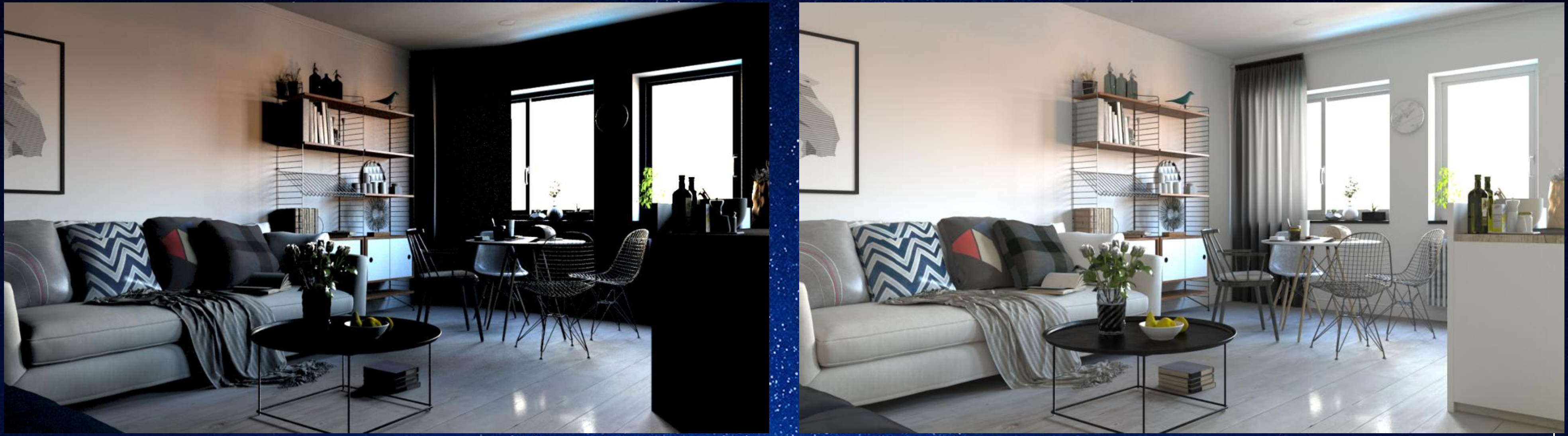
Unity + Radeon Rays

Global Illumination

- Lightmap is a solution for global illumination
- Global Illumination is
  - Essential to get realism
  - Computationally expensive
- Real time global illumination is still a research topic
  - No obvious solution using rasterization yet

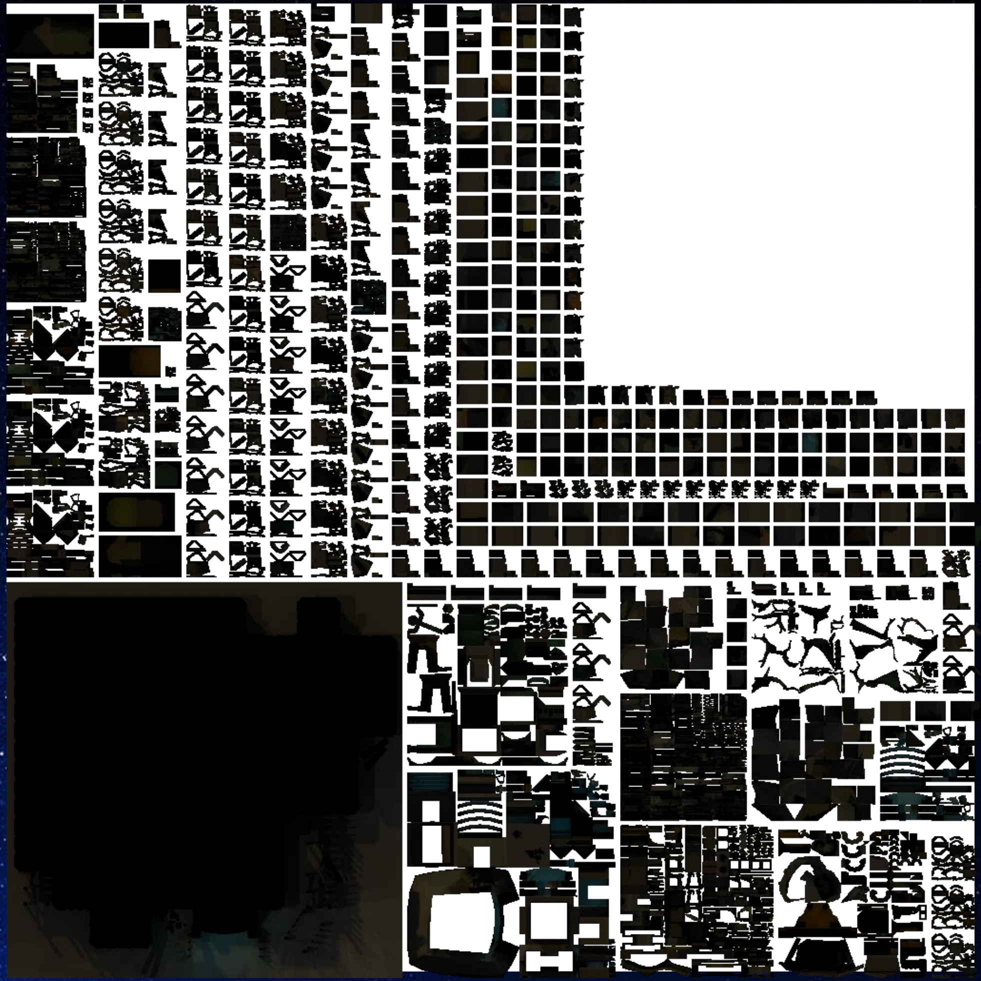
Global Illumination

- Monte Carlo ray tracing is a way to compute global illumination
  - Too computationally intensive for game runtime
- GPU accelerated ray tracing is a hot topic these days
  - Still not ready for real time game
  - Potential in content creation (Radeon ProRender)
- Lightmap is solution for real-time global illumination



Lightmap

- Many games today uses lightmaps
- Lightmap
  - Texture storing global illumination
  - Although there are some limitations, it’s widely used
- Precompute global illumination
  - Ray traced global illumination
  - Saved in texture “lightmap”
- At runtime, simply put it as a texture, fetch it
- The precomputation takes forever for a complex game scene
  - Hours to days
- Radeon Rays can help you from this pain



Lightmap Baker using Radeon Rays

- A fast lightmap baking solution
- Runs on GPU
- 10 – 20x performance improvement
  - Before 1 day baking => 1 hour with Radeon Rays
- Faster solution => Faster iteration => Better content creation

Unity Lightmap Baker using Radeon Rays

- Collaboration of Unity & AMD
- Implemented in a branch of Unity 5.X
- Based on the existing CPU lightmap baker
  - Using infrastructure for lightmap baking in Unity

- The logic needs to be changed to fill the GPU better
  - Before: for each lightmap, for each texel, execute
  - After: for each lightmap, execute all the texels in the lightmap in parallel

- Implemented 2 modes
  - Ambient occlusion and Global illumination

Ambient Occlusion Mode

- Using Unity’s lightmap G buffer rendering functionality
  - World position
  - Surface normal

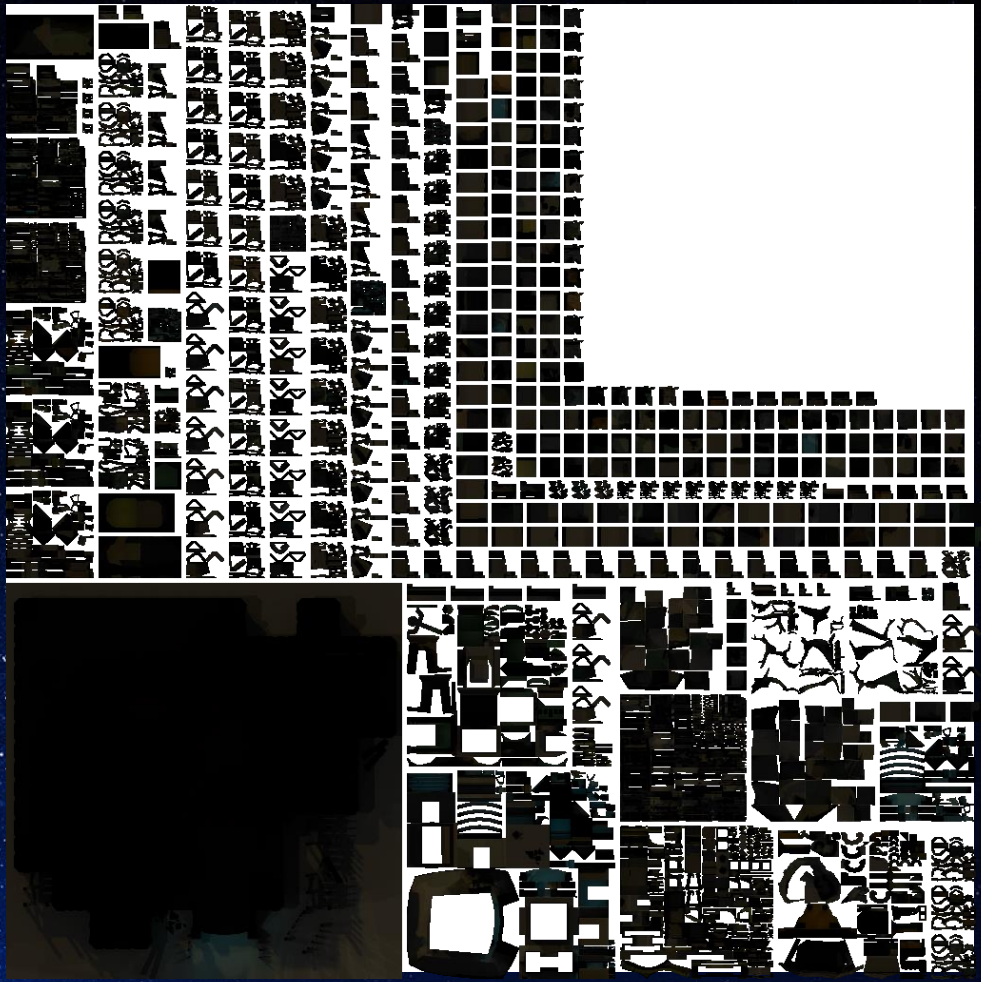
- These are enough to do AO computation

- Primary rays are generated by cosine weighted sampling
  - Makes the integration simple (simply count without any PDF computation)

- AO is calculated as
  - 1 – [# of occluded rays] / [# of casted rays]
  - 1 – sum(weight(hit distance)) / [# of casted rays]

Global Illumination Mode

- AO ray doesn’t bounce, but it does in GI
- Maximum bounces is a user defined parameter
  - Ray termination
- Supported light types
  - Point light
  - Spot light
  - Directional light
  - Area light
  - Emissive shader
  - IBL



Global Illumination Mode

- Surface properties are filled at lightmap G buffer rendering stages
  - World position
  - Surface normal (with normal maps)
  - Diffuse albedo
    - Necessary for color bleeding
  - Emission

- View dependent effect are ignored
  - glossy, specular reflections

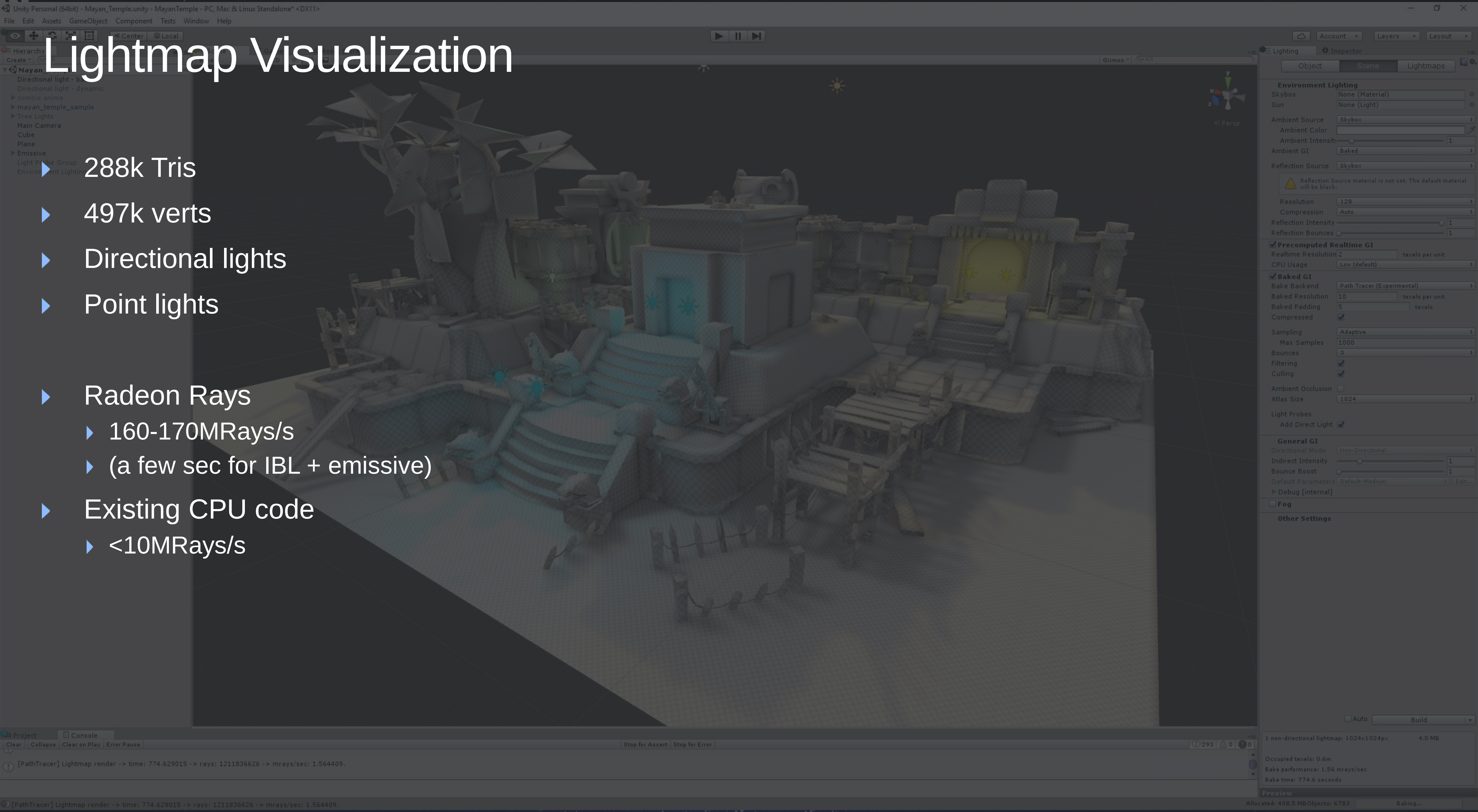
Global Illumination Mode

```
for lightmap in lightmaps
    ray = generatePrimaryRay( lightmap )
    for bounce < maxBounce
        hit = RR::intersect( ray )
        // emissive
        texel += evaluateEmissive( hit )
        // ibl
        shadowRay = generateRayIBL( hit )
        shadowHit = RR:intersect( shadowRay )
        texel += evaluateIBL( hit, shadowHit )
        for light in lights // point, spot, directional
            shadowRay = generateRayLight( hit, light )
            shadowHit = RR:intersect( shadowRay )
            texel += directIllumination( shadowHit, light )
        ray = generateNextRay( ray, hit )
```

Lightmap Visualization

- 288k Tris
- 497k verts
- Directional lights
- Point lights

- Radeon Rays
  - 160-170MRays/s
  - (a few sec for IBL + emissive)
- Existing CPU code
  - <10MRays/s


Unity Personal (64bit) - Mayan_Temple.unity - MayanTemple - PC, Mac & Linux Standalone* <DX11>
File Edit Assets GameObject Component Tests Window Help
Center Local
Account Layers Layout
Hierarchy
Create
Directional light - dynamic
zombie anime
mayan_temple_sample
Tree Lights
Main Camera
Cube
Plane
Emissive
Gizmos
Persp
Lighting Inspector
Object Scene Lightmaps
Project Console
Clear Collapse Clear on Play Error Pause Stop for Assert Stop for Error
Preview

Environment Lighting
Skybox: None (Material)
Sun: None (Light)
Ambient Source: Skybox
Ambient Color
Ambient Intensity: 1
Ambient GI: Baked
Reflection Source: Skybox
Reflection Source material is not set. The default material will be black.
Resolution: 128
Compression: Auto
Reflection Intensity: 1
Reflection Bounces: 1

☑ Precomputed Realtime GI
Realtime Resolution: 2 texels per unit
CPU Usage: Low (default)

☑ Baked GI
Bake Backend: Path Tracer (Experimental)
Baked Resolution: 10 texels per unit
Baked Padding: 5 texels
Compressed: ☑
Sampling: Adaptive
Max Samples: 1000
Bounces: 3
Filtering: ☑
Culling: ☑
Ambient Occlusion: □
Atlas Size: 1024
Light Probes
Add Direct Light: ☑

General GI
Directional Mode: Non-Directional
Indirect Intensity: 1
Bounce Boost: 1
Default Parameters: Default-Medium Edit...
Debug [internal]

□ Fog

Other Settings

□ Auto Build

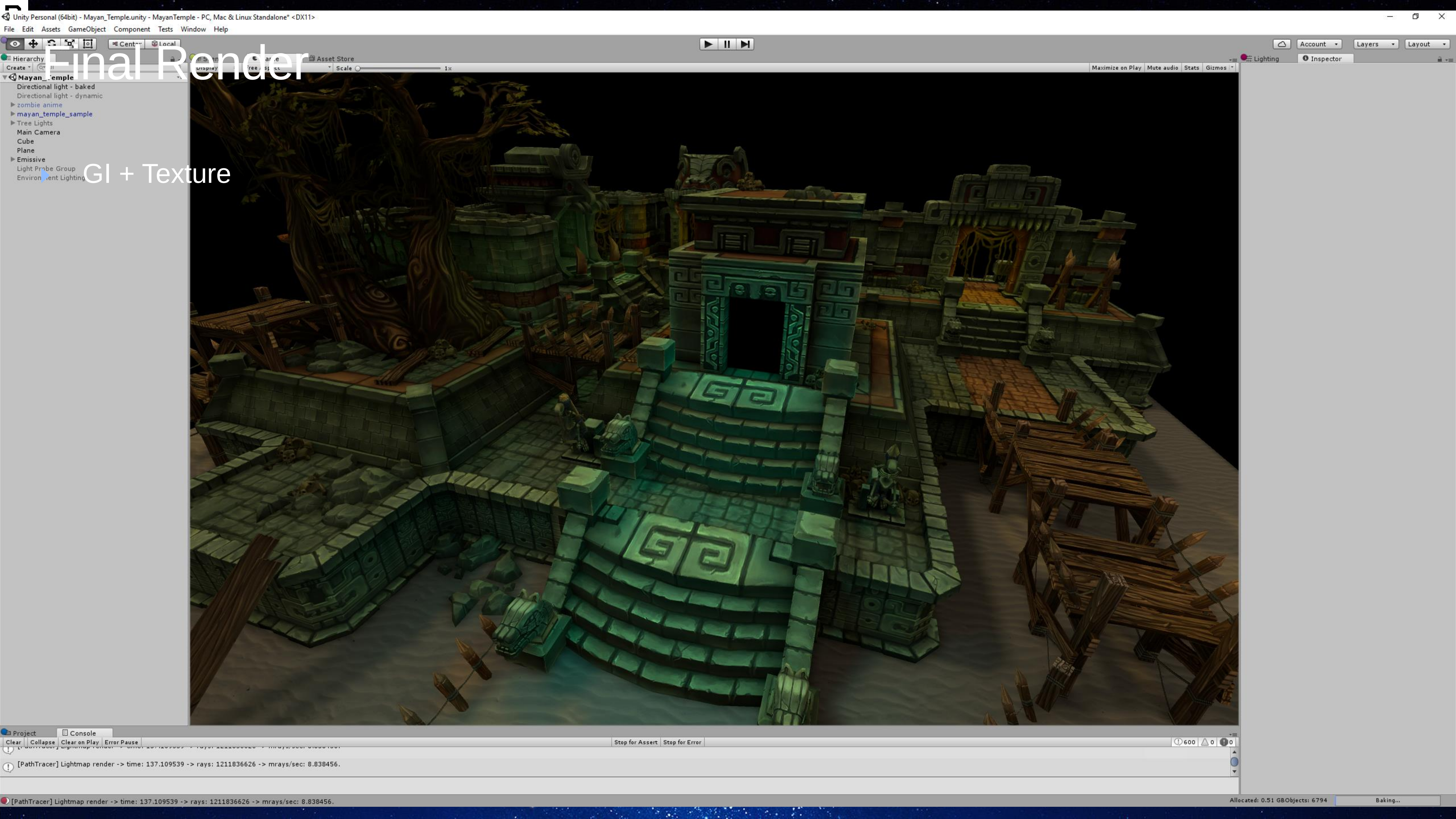
1 non-directional lightmap: 1024x1024px 4.0 MB

Occupied texels: 0.6m
Bake performance: 1.56 mrays/sec
Bake time: 774.6 seconds

```
[PathTracer] Lightmap render -> time: 774.629015 -> rays: 1211836626 -> mrays/sec: 1.564409.
```

```
[PathTracer] Lightmap render -> time: 774.629015 -> rays: 1211836626 -> mrays/sec: 1.564409.
```

Allocated: 408.5 MBObjects: 6783 Baking...



Final Render

GI + Texture

```
[PathTracer] Lightmap render -> time: 137.109539 -> rays: 1211836626 -> mrays/sec: 8.838456.
```

```
[PathTracer] Lightmap render -> time: 137.109539 -> rays: 1211836626 -> mrays/sec: 8.838456.
```

Finally

- This project is still in progress
- We are going to improve to make it
  - Robust
  - Better convergence
- Progressive rendering, so that it can run async with other work
  - A big advantage over CPU

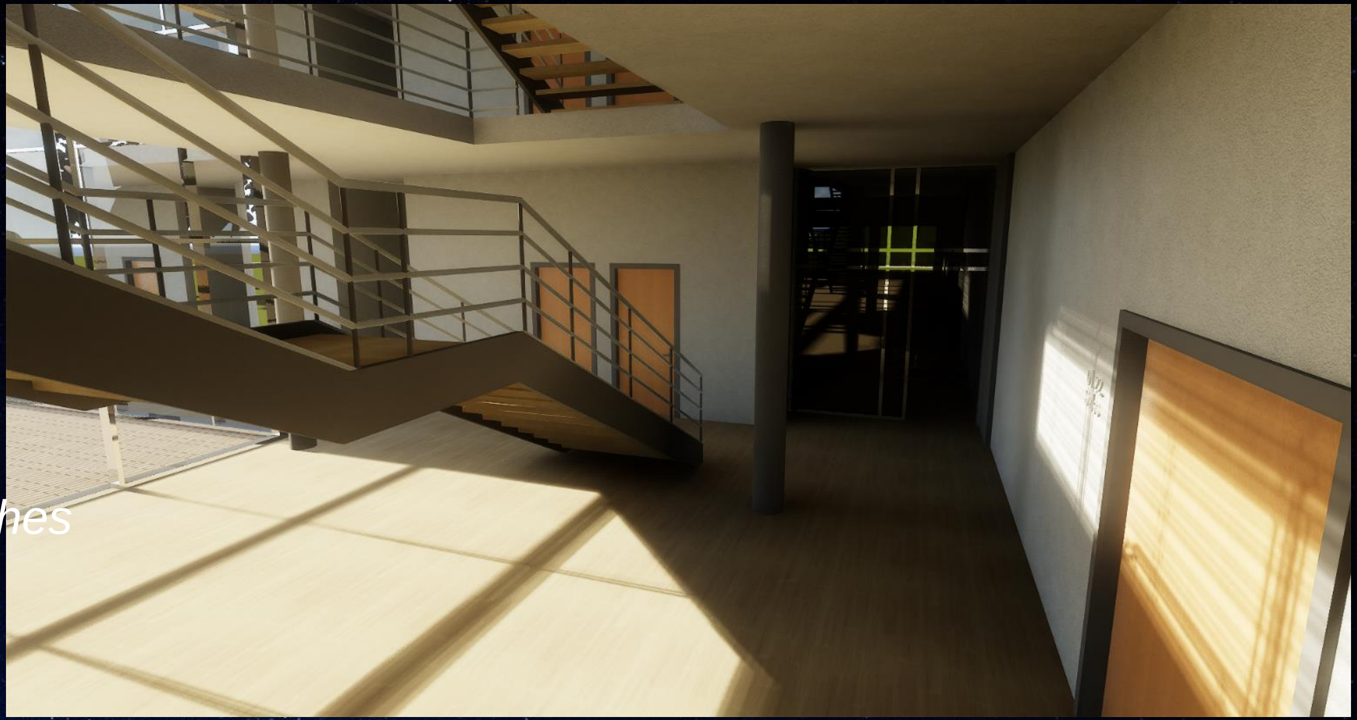
Other Radeon Rays Adaptions

ENSCAPE™

- Real-time rendering plugin for Autodesk Revit
  - Exploring the model with high quality rendering
- Use of custom fork of Radeon Rays

ENSCAPE™

- Real-time rendering plugin for Autodesk Revit
  - Exploring the model with high quality rendering
- Use of custom fork of Radeon Rays
- Radeon Rays is used to compute *illumination caches*
- Hybrid global illumination solution
  - Hierarchy of illumination caches
  - Screen space ray tracing
  - World space ray tracing as a last resort
  - BVH streaming



Others and More

- Radeon Rays integration
  - Some game studios

- Radeon Rays integration is not for everybody
- If you don’t need the fine control in baking, Radeon ProRender might be the solution for you
- Radeon ProRender has not only ray intersection, but all the logic necessary for GI (shading, sampling etc) are there
- You only need to set up the scene and call rprContextRender()
  - Lightmap render
  - Light probe render
  - Interactive preview

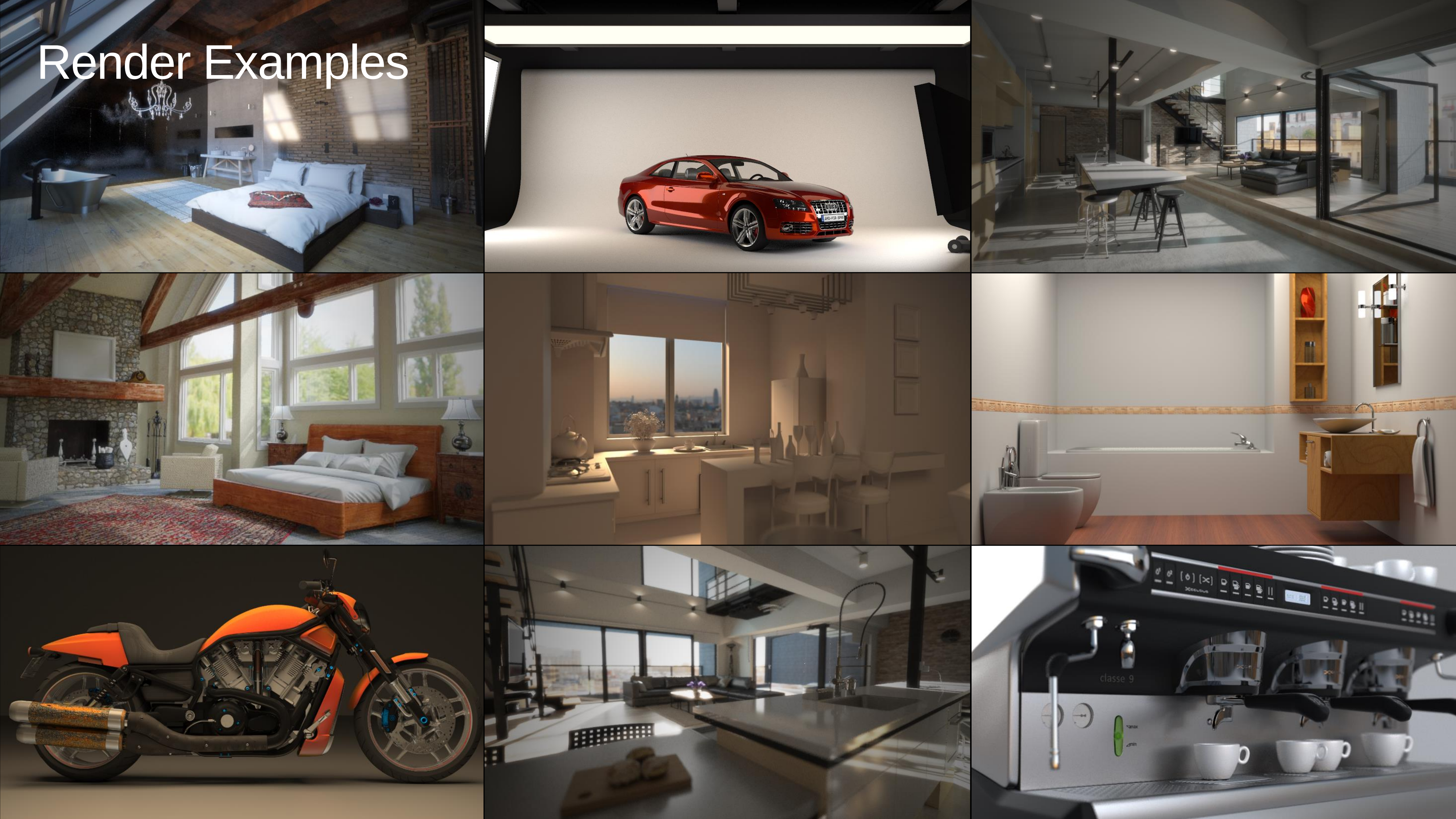
Radeon ProRender

What I have talked about are

- A workflow where we bake, apply, then you can see global illumination
- Could be wasteful
  - Texture resolution is too high
- Could be insufficient
  - Texture resolution is too low
- Optimal sampling rate is difficult with lightmap solution

- Interactive global illumination solution with Radeon ProRender is alternative
  - Single click “Render”
    - Simpler workflow
  - Progressive global illumination refinement

Render Examples



VRay Material Converter

VRay Material Converter

ROAD
TO
HOME
28 WALLACE
28 SYLVAN
28 FREYSVILLE
28 LIBHART
28 EMIG
28 PICKING
28 KEW GREEN
EXIT
28 ALTLA
28 FREYS
28 INDIAN
28 GREE
28 EDGE
28 LIBHA
28 SUNDA
28 SCOTT
28 LIBHAR

Radeon ProRender Demo

- https://www.youtube.com/watch?v=z9wArygtwlI

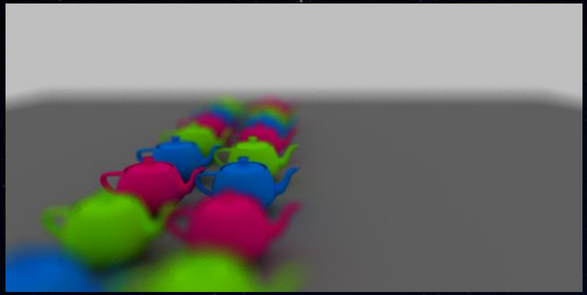
Radeon ProRender is

- A fast GPU accelerated global illumination renderer
- Not fast enough for game runtime
- There is a potential in game content creation acceleration

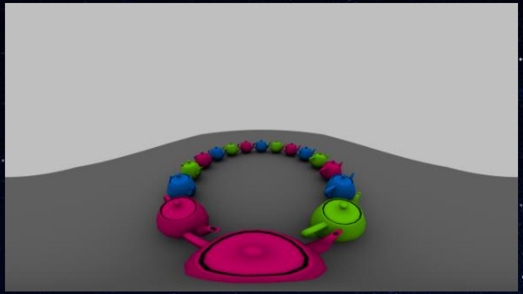
- Provided as
  - SDK for developers (C API)
  - Plugins for creators

Features

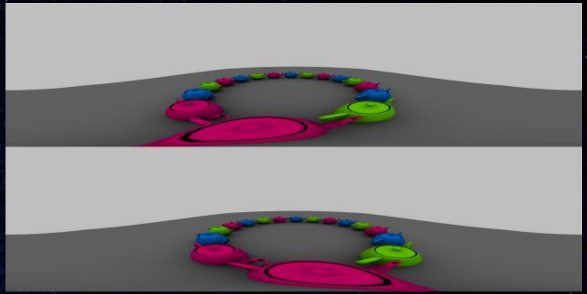
- Camera



Perspective

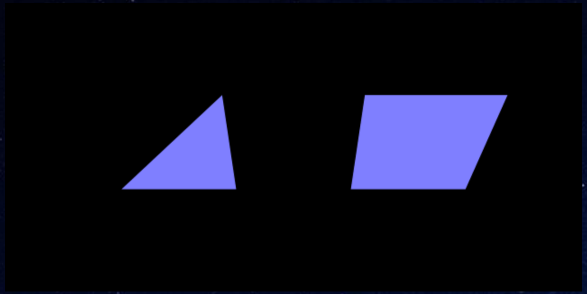


360

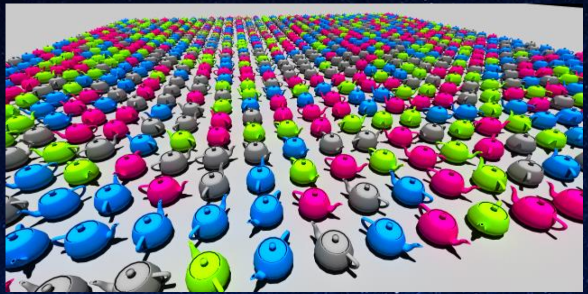


VR

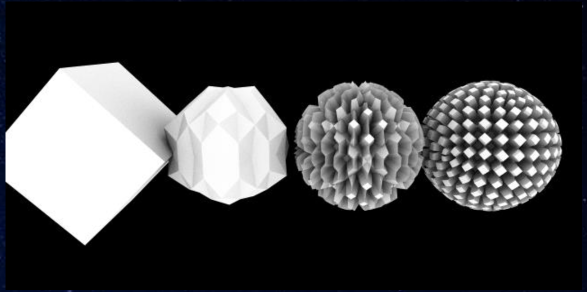
- Geometry



Mesh



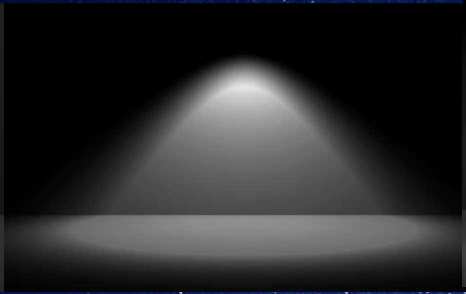
Instancing



Subdivision

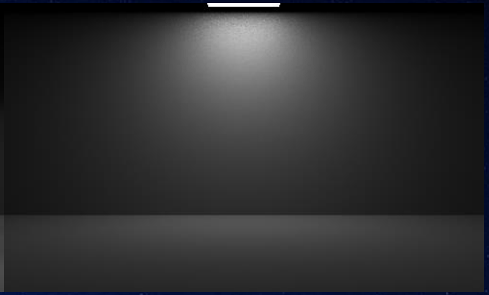
- Lights

Point



Spot

IES

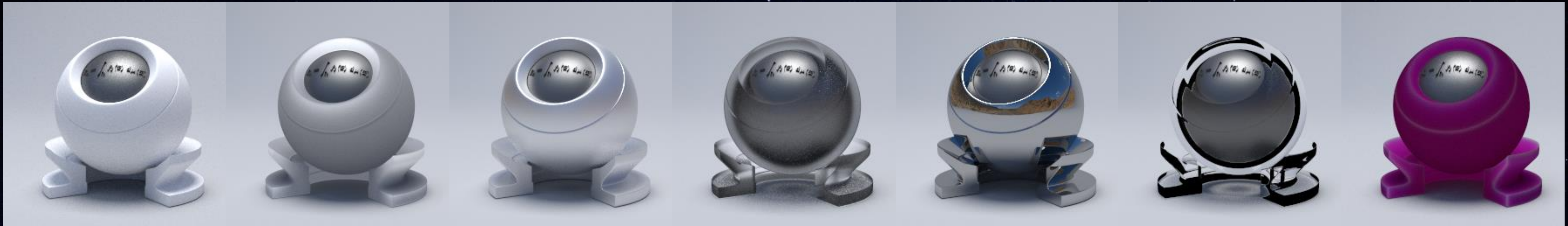


Area

Features

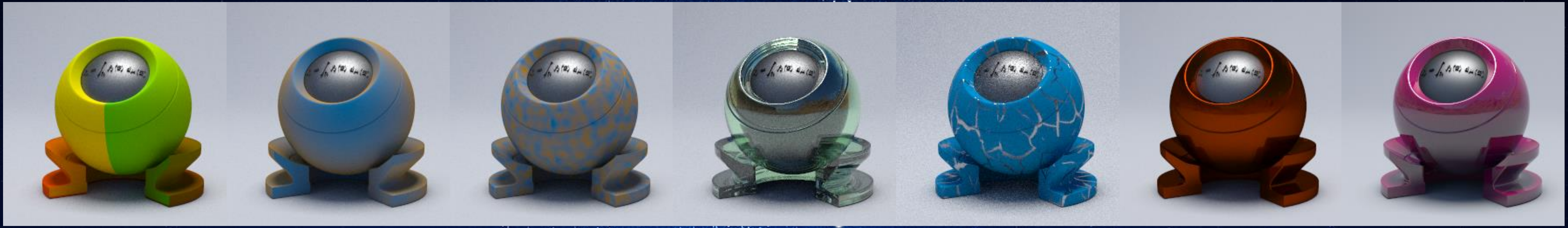
MATERIALS

- 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

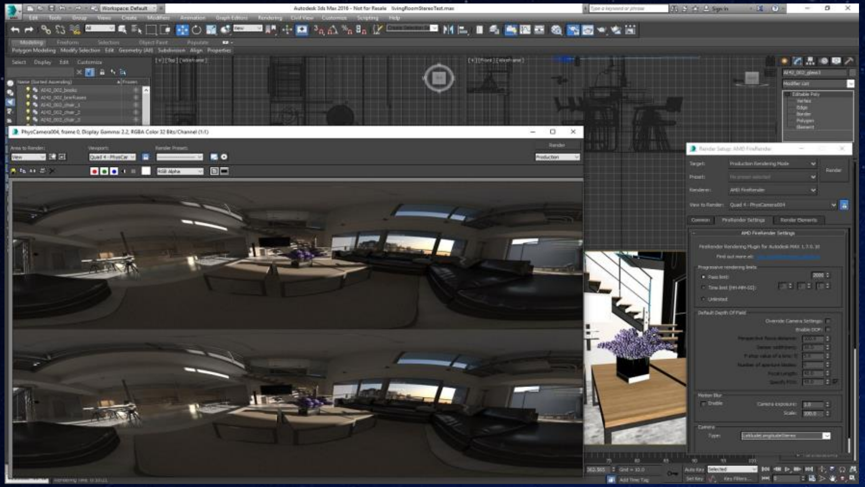
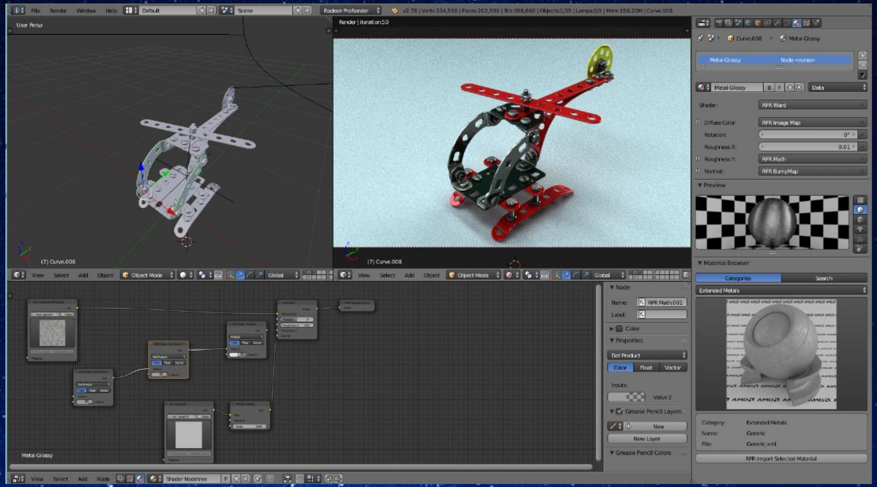
Radeon ProRender Plugins

From AMD

- 3DS Max
- Maya
- Solidworks
- Blender

From third party

- Coming soon!!

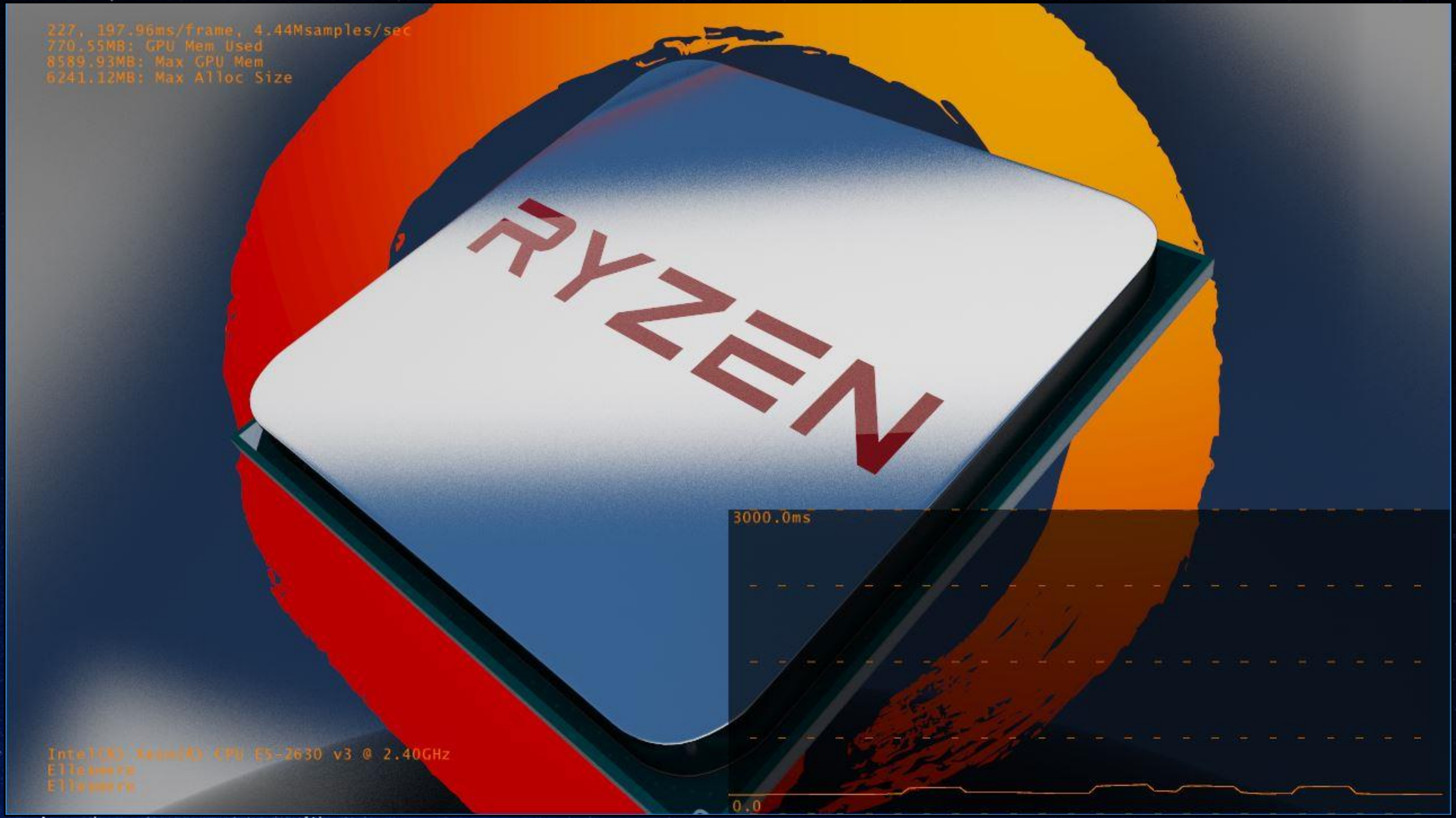



3DS Max Plugin New Features

- Portal
- Displacement mapping
- CPU + GPU
- VRay Material Converter

3DS Max Plugin New Features

- Portal
- Displacement mapping
- CPU + GPU
- VRay Material Converter



```
227, 197.96ms/frame, 4.44Msamples/sec
770.55MB: GPU Mem Used
8589.93MB: Max GPU Mem
6241.12MB: Max Alloc Size
```

```
Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
Ellesmere
Ellesmere
```

3DS Max Plugin New Features

- Portal
- Displacement mapping
- CPU + GPU
- VRay Material Converter

ROAD
TO
HOME
EXIT
28 WALLACE
28 SYLVAN
28 FREYSVILLE
28 LIBHART
28 EMIG
28 PICKING
28 KEW GREEN

Acknowledgement

- Thanks to **Nicholas Timmons, Dmitry Kozlov** for Unity integration