Game Rendering
Game Rendering

- 3D or 2D?
Game Rendering

● 3D or 2D?

● Realistic?
  ○ Photorealistic rendering
    ■ Physically-based rendering

Game Rendering

- 3D or 2D?
- Realistic?
  - Photorealistic rendering
  - Physically-based rendering
  - Non-photorealistic rendering

https://en.wikipedia.org/wiki/Non-photorealistic_rendering
Game Rendering

● 3D or 2D?
● Realistic?
  ○ Photorealistic rendering
    ■ Physically-based rendering
  ○ Non-photorealistic rendering
● Real-time?
  ○ Performance, performance, and performance
Ray tracing

Model

Mesh

- Vertices
  - Positions, Normals, UVs, ...

- Faces
  - Triangles or Quads?
Graphics hardware

Hardware

CPU, RAM, GPU, VRAM, Disk, ...
Graphics hardware

Hardware: CPU, RAM, **GPU**, **VRAM**, Disk, ...
Graphics hardware

- OS: Windows, Android, iOS, Linux, macOS, Orbis, ..
- Hardware: CPU, RAM, **GPU**, **VRAM**, Disk, ..
Graphics hardware

3D Graphics API: OpenGL, Direct3D, Vulkan, Metal, ...

OS: Windows, Android, iOS, Linux, macOS, Orbis, ...

Hardware: CPU, RAM, GPU, VRAM, Disk, ...
Graphics hardware

Game Engine

3D Graphics API

OS

Hardware

Unity, Unreal, Cocos2d-x, Godot, ...

OpenGL, Direct3D, Vulkan, Metal, ...

Windows, Android, iOS, Linux, macOS, Orbis, ..

CPU, RAM, **GPU**, VRAM, Disk, ...
Model

Mesh

- Vertices
  - Positions, Normals, UVs, ...

- Faces
  - Triangles (hardware)
  - Quads (artist)

http://3drender.com/jbirn/hippo/hairyhipponose.html
Mesh Filter / Mesh Renderer
"How many vertices / faces can we use?"

https://www.zbrushcentral.com/t/headshot-series/360990
Mesh simplification

Ray tracing

[Image: Ray_trace_diagram.svg]

Ray-triangle intersection

Ray-triangle intersection (cont'd)
Ray-triangle intersection (cont'd)
Space-partitioning data structures
Case study: NVIDIA RTX™ Platform

MDL and USD

OptiX | DXR | Vulkan

Rasterization (Graphics Pipeline)  Ray Tracing (RT Core)  Compute (CUDA)  AI (Tensor Core)

NVIDIA RTX Platform

https://developer.nvidia.com/rtx
Case study: DXR (DirectX Raytracing)
Ray Tracing with HDRP

Getting started with ray tracing

The High Definition Render Pipeline (HDRP) includes preview ray tracing support from Unity 2019.3. Ray tracing is a feature that allows you to access data that is not on screen. For example, you can use it to request position data, normal data, or lighting data, and then use this data to compute quantities that are hard to approximate using classic rasterization techniques.

While film production uses ray tracing extensively, its resource intensity has limited its use to offline rendering for a long time. Now, with recent advances in GPU hardware, you can make use of ray tracing effect in real time.

This document covers:

- Hardware requirements.
- Integrate ray tracing into your HDRP Project.
- HDRP effects that use ray tracing.

Hardware requirements

Ray tracing hardware acceleration is only available on certain graphics cards. The graphics cards with full support are:

- NVIDIA Volta (Titan X)
- NVIDIA Turing (2060, 2070, 2080, and their Ti variants)

https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.1/manual/Ray-Tracing-Getting-Started.html
Ray tracing

Material
Material

\[ f(\omega_i, \omega_r) \]
Material

\[ f(\omega_r, \omega_i) \]
Material (cont'd)

http://collagefactory.blogspot.com/2010/04/brdf-for-diffuseglossyspecular.html
Material (cont'd)

BSSRDF (Bidirectional scattering distribution function)

\[ f_{\text{BSSRDF}}(x, \omega_i, \omega_r) \]

http://collagefactory.blogspot.com/2010/04/brdf-for-diffuseglossyspecular.html
Material (cont'd)

BSSRDF (Bidirectional scattering distribution function)

\[ f_{\text{BSSRDF}}(x, \omega_i, \omega_r) \]

http://collagefactory.blogspot.com/2010/04/brdf-for-diffuseglossyspecular.html

Material (cont'd)

BSDF (Bidirectional scattering distribution function) =
BRDF (Bidirectional reflectance distribution function) +
BTDF (Bidirectional transmittance distribution function)

http://collagefactory.blogspot.com/2010/04/brdf-for-diffuseglossyspecular.html

\[
f_{BRDF}(\omega_i, \omega_r) \quad f_{BTDF}(\omega_i, \omega_r)
\]
BRDF (Bidirectional reflectance distribution function)

https://en.wikipedia.org/wiki/Bidirectional_reflectance_distribution_function
**BRDF** (Bidirectional reflectance distribution function)
BRDF (Bidirectional reflectance distribution function)
BRDF (Bidirectional reflectance distribution function)
BRDF (Bidirectional reflectance distribution function)
BRDF (Bidirectional reflectance distribution function)
BRDF (Bidirectional reflectance distribution function)

- Different Normals
BRDF (Bidirectional reflectance distribution function)

- **Diffuse**
- **Glossy**
- **Mirror**

- Different **Normals**
BRDF (Bidirectional reflectance distribution function)
BRDF Measurement

https://graphics.stanford.edu/projects/gantry/
BRDF Measurement

https://graphics.stanford.edu/projects/gantry/
BRDF Measurement

https://graphics.stanford.edu/projects/gantry/

https://www.merl.com/brdf/
Analytic BRDF (Bidirectional reflectance distribution function)

Analytic BRDF (Bidirectional reflectance distribution function)

\[ = f_D(\text{Diffuse}) + f_M(\text{Mirror}) + f_G(\text{Glossy}) \]
Analytic BRDF (Bidirectional reflectance distribution function)

\[ \text{BRDF} = f_D(0.8) + f_M(2) + f_G(0.1) \]
Analytic BRDF (Bidirectional reflectance distribution function)

\[ \text{BRDF} = f_\text{D}(0.8) + f_\text{M}(2) + f_\text{G}(0.1) \]

Phong illumination model

\[ I_{\text{Ambient}}(k_a) + I_{\text{Diffuse}}(k_d) + I_{\text{Specular}}(k_s) = I \]
Disney Principled BRDF

Github:

Ray tracing

Image

Camera

Light Source

View Ray

Shadow Ray

Scene Object

Material
Material

Use **Shader** (a program) to calculate the result

**Given:**
- Light
- View
- Normal
- BRDF parameters
- Position
- ...
Material (cont'd)

All triangles/quads have the same Material?

All triangles/quads have the same BRDF?
Texture
Texture
Texture

Ellen_Body
18897 verts, 30464 tris, 6 submeshes, uv, skin
Material

New Material

Shader Standard

Rendering Mode
Opaque

Main Maps
Albedo
Metallic
Smoothness
Source
Metallic Alpha

Normal Map
Height Map
Occlusion
Detail Mask

Emission
Tiling X 1 Y 1
Offset X 0 Y 0

Secondary Maps
Case Study: Concept Art
Shader

Given:
- Light
- View
- Normal
- BRDF parameters
- Position
- UVs
- Textures
- ...
Graphics hardware

| Game Engine | Unity, Unreal, Cocos2d-x, Godot, ...
|-------------|----------------------------------|
| 3D Graphics API | OpenGL, Direct3D, Vulkan, Metal, ...
| OS | Windows, Android, iOS, Linux, macOS, Orbis, ..
| Hardware | CPU, RAM, **GPU**, **VRAM**, Disk, ...
Writing Shaders

Shaders in Unity can be written in one of three different ways:

**Surface Shaders**

Surface Shaders are your best option if your Shader needs to be affected by lights and shadows. Surface Shaders make it easy to write complex Shaders in a compact way - it's a higher level of abstraction for interaction with Unity's lighting pipeline. Most Surface Shaders automatically support both forward and deferred lighting. You write Surface Shaders in a couple of lines of Cg/HLSL, and a lot more code gets auto-generated from that.

Do not use Surface Shaders if your Shader is not doing anything with lights. For post-processed effects or many special-effect Shaders, Surface Shaders are a suboptimal option, since they do a bunch of lighting calculations for no good reason.

**Vertex and Fragment Shaders**

Vertex and Fragment Shaders are required if your Shader doesn't need to interact with lighting, or if you need some very exotic effects that the Surface Shaders can't handle. Shader programs written this way are the most flexible way to create the effect you need (even Surface Shaders are automatically converted to a bunch of Vertex and Fragment Shaders), but that comes at a price: you have to write more code and it's harder to make it interact with lighting. These Shaders are written in Cg/HLSL as well.

**Fixed Function Shaders**

Fixed Function Shaders are legacy Shader syntax for very simple effects. It is advisable to write programmable Shaders, since that allows much more flexibility. Fixed Function Shaders are not recommended.
Shader

Considers:
- Light
- View
- Normal
- BRDF
- Position
- UVs
- Textures
- ...
Point light

\[ \omega_i \rightarrow \omega_r \]

\[ n \]
Point light / Directional light
Spot light

\[ \omega_i \rightarrow n \rightarrow \omega_r \]
Area light?
Lots of point lights? integral?
Point light
Directional light
Area light
Shadow?

Shadow ?

[Diagram showing the concept of shadow rays in ray tracing]

Shadow?

Camera → Image

Light Source → Light Ray → Shadow → Scene Object

View Ray

Shadow?

Camera → Image

Light Source → View Ray → Shadow Ray

Scene Object


shadow rays?
Soft shadows

https://en.wikipedia.org/wiki/Shadow
Direct illumination

- Camera
- Image
- Light Source
- View Ray
- Shadow Ray
- Scene Object

Number of lights
Hard / Soft shadows (Light type)

Indirect illumination

[Diagram showing light source, camera, and scene object with rays and shadows]

Direct vs. indirect illumination

Direct + Indirect = Final
Global illumination = Indirect illumination

[Image: Ray_trace_diagram.svg]
Global illumination = Indirect illumination

Lots of ray intersection tests

We'll talk about this later.
Camera?
Camera

- Anti-aliasing
- Depth of field
- Auto exposure
- White balance
- Color adjustments
- Lens distortion
- Grain

- Motion blur
- HDR (High-dynamic-range)
  - Bloom
  - Tonemapping
- ...

90
Anti-aliasing
Anti-aliasing (cont'd)

1) **Supersampling**: casting multiple rays/samples in a pixel

https://en.wikipedia.org/wiki/Supersampling

Anti-aliasing (cont'd)

1) **Supersampling**: casting multiple rays/samples in a pixel
2) **Approximation**: smoothing undesirable jagged edges

Depth of Field

1) Lots of rays/samples
2) Approximation
Depth of Field (cont'd)

https://www.nintendo.com/games/detail/octopath-traveler-switch/
Bloom

https://en.wikipedia.org/wiki/Bloom_(shader_effect)
Post Processing (before)
Post Processing (after)

https://docs.unity3d.com/Manual/PostProcessingOverview.html
Post Processing (package for Built-in RP)

Post-processing is the process of applying full-screen filters and effects to a camera's image buffer before it is displayed to screen. It can drastically improve the visuals of your product with little setup time.

You can use post-processing effects to simulate physical camera and film properties.

The images below demonstrate a scene with and without post-processing.
Ray tracing

Image

Camera

View Ray

Light Source

Shadow Ray

Scene Object

Global illumination = Indirect illumination

[Diagram showing ray tracing concepts with labels for Camera, Image, Light Source, View Ray, Shadow Ray, Scene Object, and a note about lots of ray intersection tests.

Global illumination = Indirect illumination

Lots of ray intersection tests

hardware acceleration?
Global illumination = Indirect illumination

- Camera
- Image
- Light Source
- View Ray
- Shadow Ray
- Scene Object

Lots of ray intersection tests

hardware acceleration?

Challenge: divergence of ray paths

Direct illumination

- Camera
- Image
- Light Source
- View Ray
- Shadow Ray
- Scene Object

Lots of ray intersection tests

hardware acceleration?

Direct illumination

Lots of ray intersection tests

hardware acceleration?

GPU & VRAM

Case study: NVIDIA RTX™ Platform

https://developer.nvidia.com/rtx
Case study: NVIDIA RTX™ Platform

https://developer.nvidia.com/rtx
Rasterization
Rasterization (cont'd)
Rasterization (cont'd)
Rasterization (cont'd)
3D Graphics Rendering Pipeline
3D Graphics Rendering Pipeline

Which triangle/mesh needs to be drawn first?
Painter's algorithm

https://en.wikipedia.org/wiki/Painter%27s_algorithm
Painter's algorithm

https://en.wikipedia.org/wiki/Painter%27s_algorithm
Painter's algorithm (cont'd)

https://en.wikipedia.org/wiki/Painter%27s_algorithm
Z (Depth)-buffering

3D Graphics Rendering Pipeline

Are all triangles visible?
Culling

http://people.csail.mit.edu/fredo/PUBLI/surveyTVCG.pdf
View-frustum culling

- Yes, Unity applies frustum culling to every renderer.
  - But, you can also use the CullingGroup API
Occlusion culling
Are all triangles opaque?
Transparency?

- Fog
- Particles
- Visual effects
- ...

http://cs248.stanford.edu/winter19/lecture/pipeline/slide_055
Transparency?

Opaque

Depth Buffer

Transparent

http://cs248.stanford.edu/winter19/lecture/pipeline/slide_055
Transparency?

Does the order of transparent objects matter?
Render Queue
How many draw calls are required?
Draw call

Needs to have The same rendering order and material
Frame debugger

https://docs.unity3d.com/Manual/FrameDebugger.html
Frame debugger

https://docs.unity3d.com/Manual/FrameDebugger.html
Case study: RenderDoc

https://renderdoc.org/
Unity render pipeline stages

Spotlight On
Render pipeline stages

1. Culling
List objects to render

culled objects

2. Rendering
Draw objects

a pixel

3. Post-processing
Apply additional image effects

* These images are simplified representations. The actual number of pixels is much higher.
Rasterization vs. Ray tracing

Hard / soft shadows?

Indirect illumination?

Shadow and global illumination

- **Shadow**
  - Shadow mapping, ambient occlusions (post-processing)

- **Lightmap**
  - Baked indirect lighting (static objects)
  - Baked ambient occlusions

- **Light Probes**
  - Baked indirect lighting (dynamic objects)

- **Reflection Probes**
How about 2D / GUI?

https://docs.unity3d.com/Manual/Sprites.html
How about 2D / GUI?

Rectangle?

AABB (Axis-Aligned Bounding Box)

https://docs.unity3d.com/Manual/Sprites.html
How about 2D / GUI?

Rectangle?

AABB (Axis-Aligned Bounding Box)

Drawing order: painter's algorithm?

https://docs.unity3d.com/Manual/Sprites.html
How about 2D / GUI?

- Rectangle?
- AABB (Axis-Aligned Bounding Box)
- Drawing order: painter's algorithm?
- Opaque vs. Transparent

https://docs.unity3d.com/Manual/Sprites.html
How about 2D / GUI? (cont'd)
Draw call

Needs to have The same rendering order and material
Pack as many textures as possible?
Q & A
Deferred shading vs. Forward shading