

# Game Rendering



Ken-Yi Lee

**Game Programming, Fall 2020 @ National Taiwan University**



DEVOTION

<https://shop.redcandlegames.com/#game>

# Game Rendering

- 3D or 2D ?

# Game Rendering

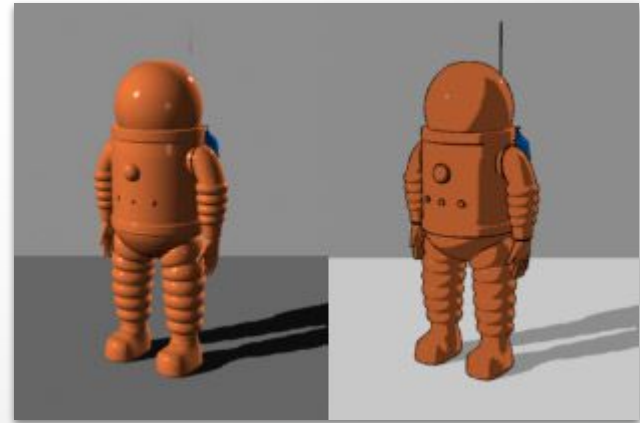
- 3D or 2D ?
- Realistic ?
  - Photorealistic rendering
    - Physically-based rendering



[https://en.wikipedia.org/wiki/Unbiased\\_rendering](https://en.wikipedia.org/wiki/Unbiased_rendering)

# Game Rendering

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



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



Unreal Paris 2020 on Steam

[https://store.steampowered.com/app/1280890/Unreal\\_Paris\\_2020/](https://store.steampowered.com/app/1280890/Unreal_Paris_2020/)



Unreal Paris 2020 on Steam

[https://store.steampowered.com/app/1280890/Unreal\\_Paris\\_2020/](https://store.steampowered.com/app/1280890/Unreal_Paris_2020/)

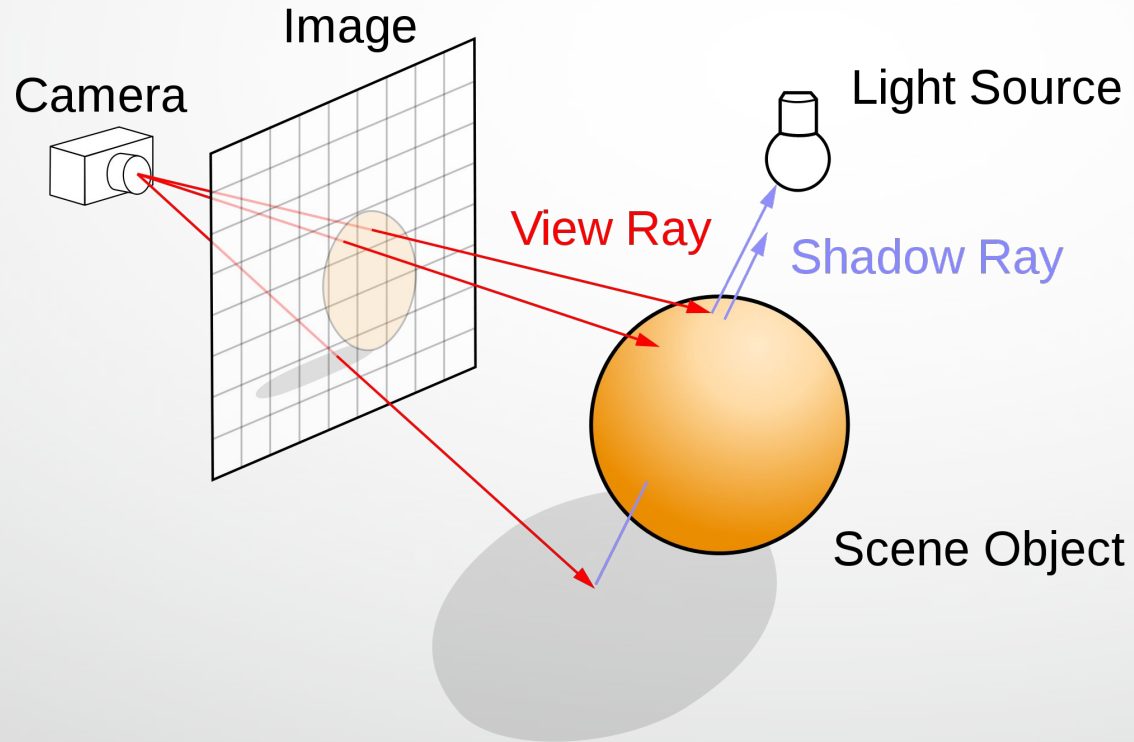




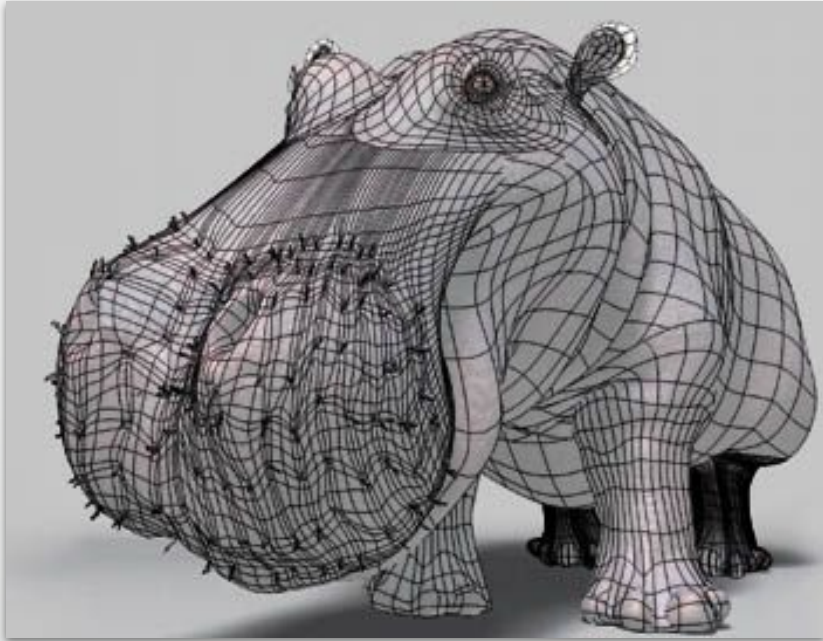
Unreal Paris 2020 on Steam

[https://store.steampowered.com/app/1280890/Unreal\\_Paris\\_2020/](https://store.steampowered.com/app/1280890/Unreal_Paris_2020/)

# Ray tracing



# Model



<http://3drender.com/jbirm/hippo/hairyhipponose.html>

## Mesh

- Vertices
  - Positions, Normals, UVs, ...
- Faces
  - Triangles or Quads ?

# Graphics hardware

Hardware

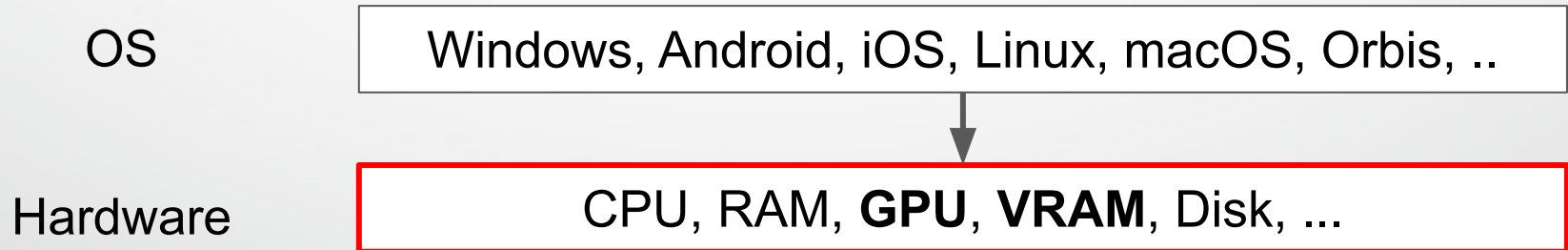
CPU, RAM, GPU, VRAM, Disk, ...

# Graphics hardware

Hardware

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

# Graphics hardware



# Graphics hardware

3D Graphics API

OpenGL, Direct3D, Vulkan, Metal, ...

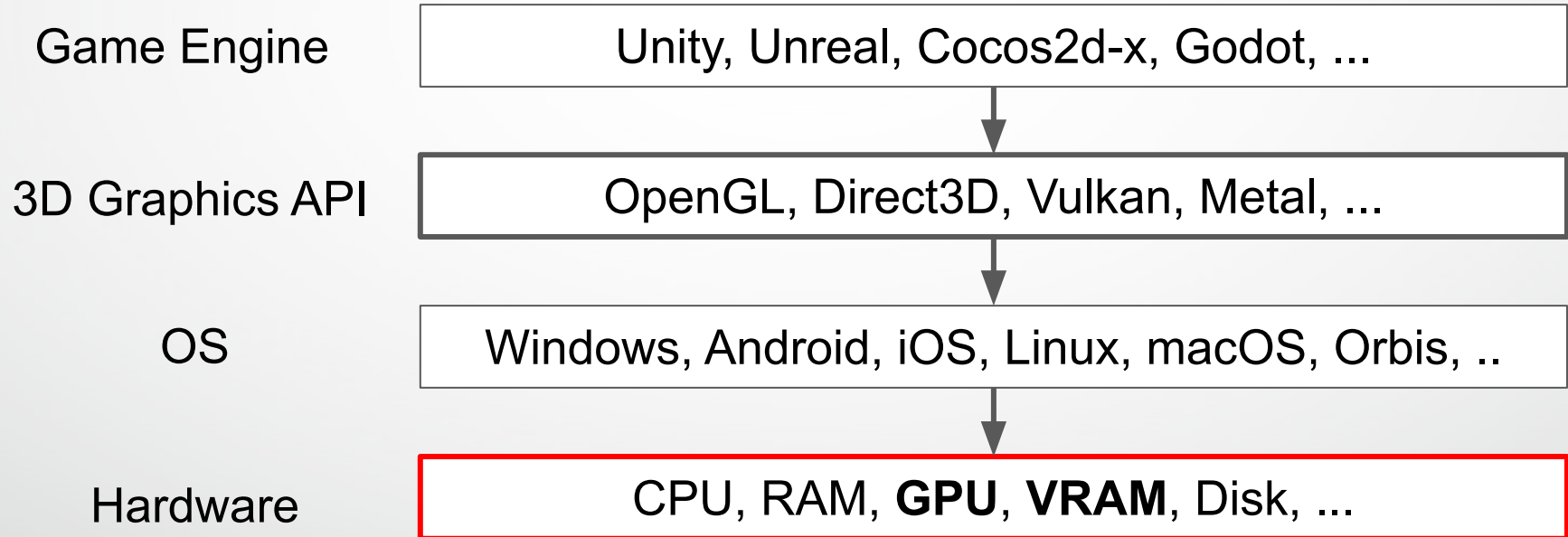
OS

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

Hardware

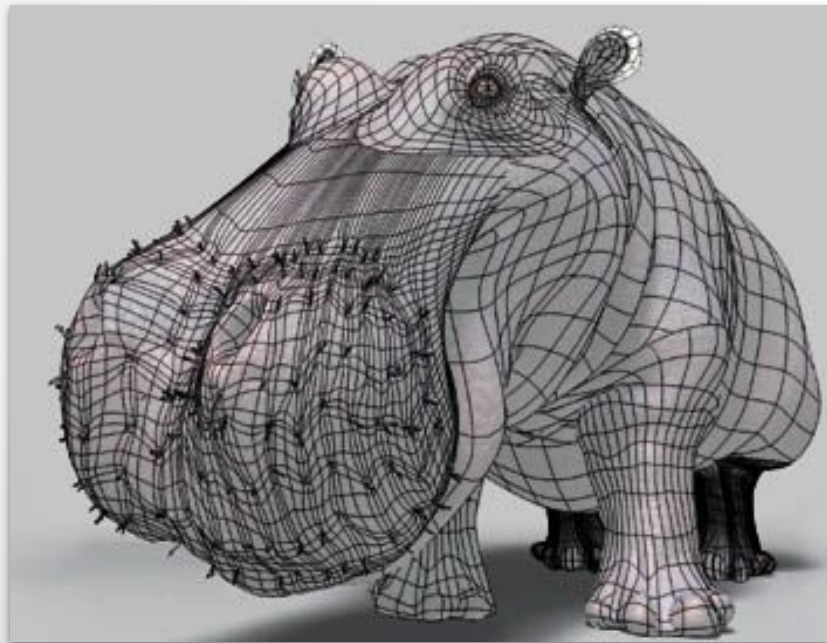
CPU, RAM, **GPU**, **VRAM**, Disk, ...

# Graphics hardware





# Model



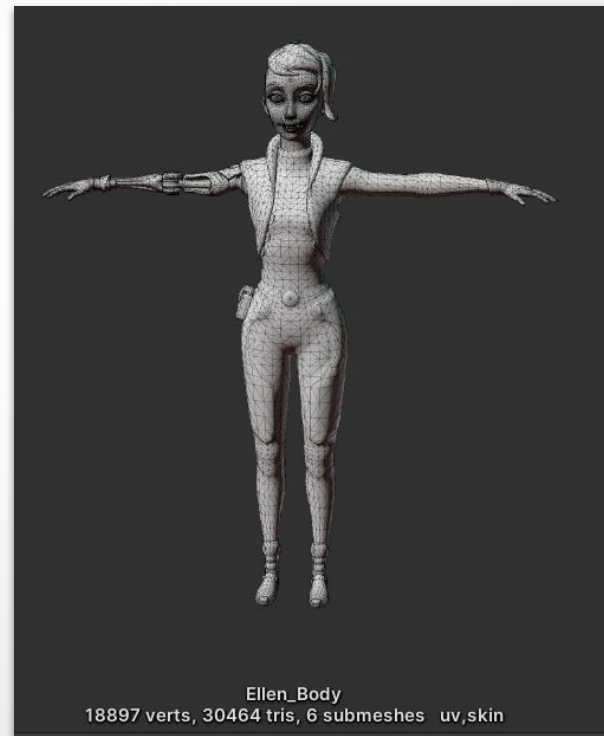
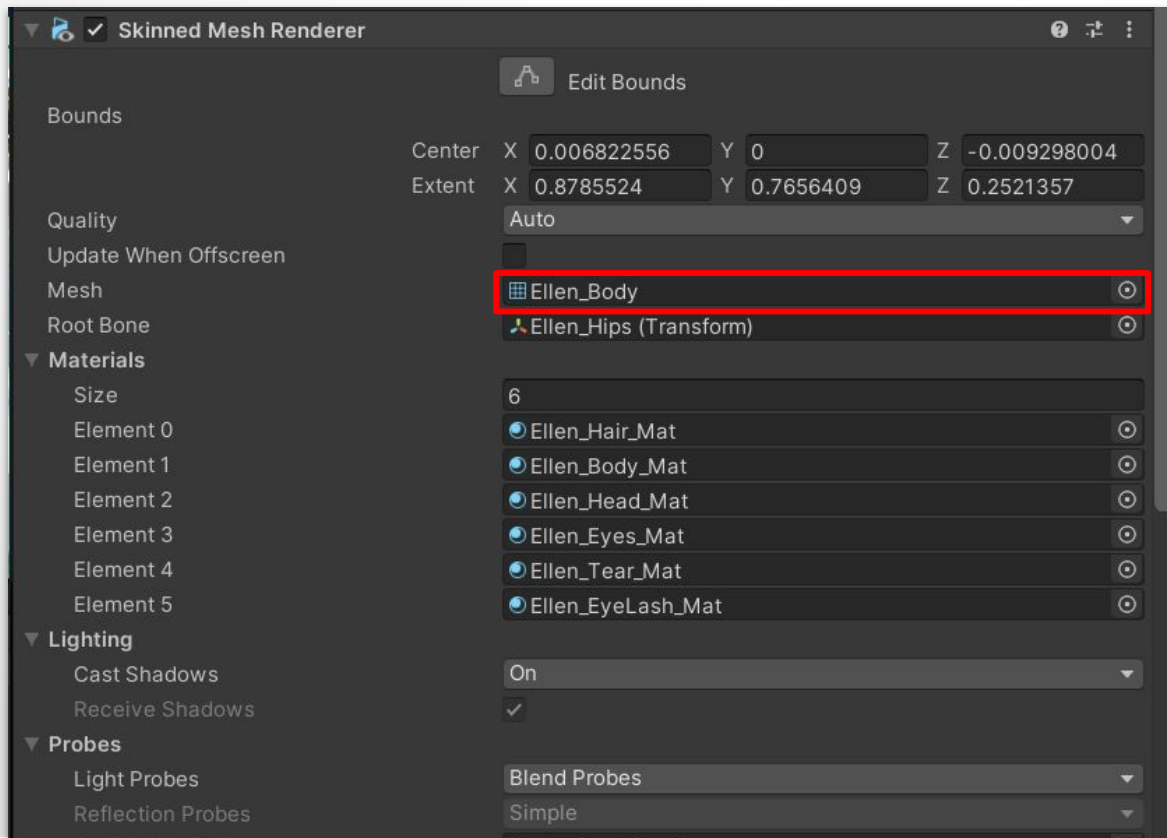
<http://3drender.com/jbirm/hippo/hairyhipponose.html>

## Mesh

- Vertices
  - Positions, Normals, UVs, ...
- Faces
  - Triangles (hardware)
  - Quads (artist)



# Mesh Filter / Mesh Renderer

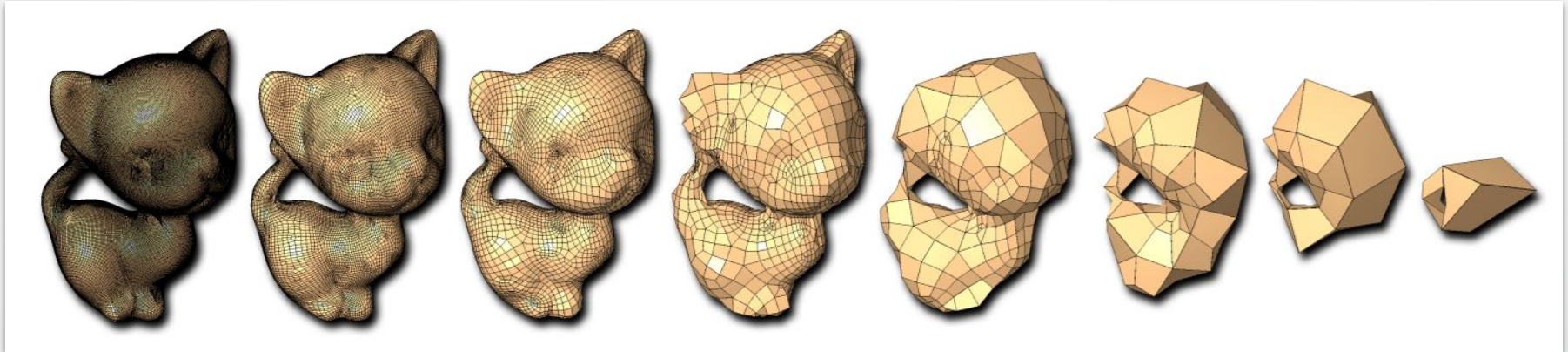


"How many vertices / faces can we use ? "



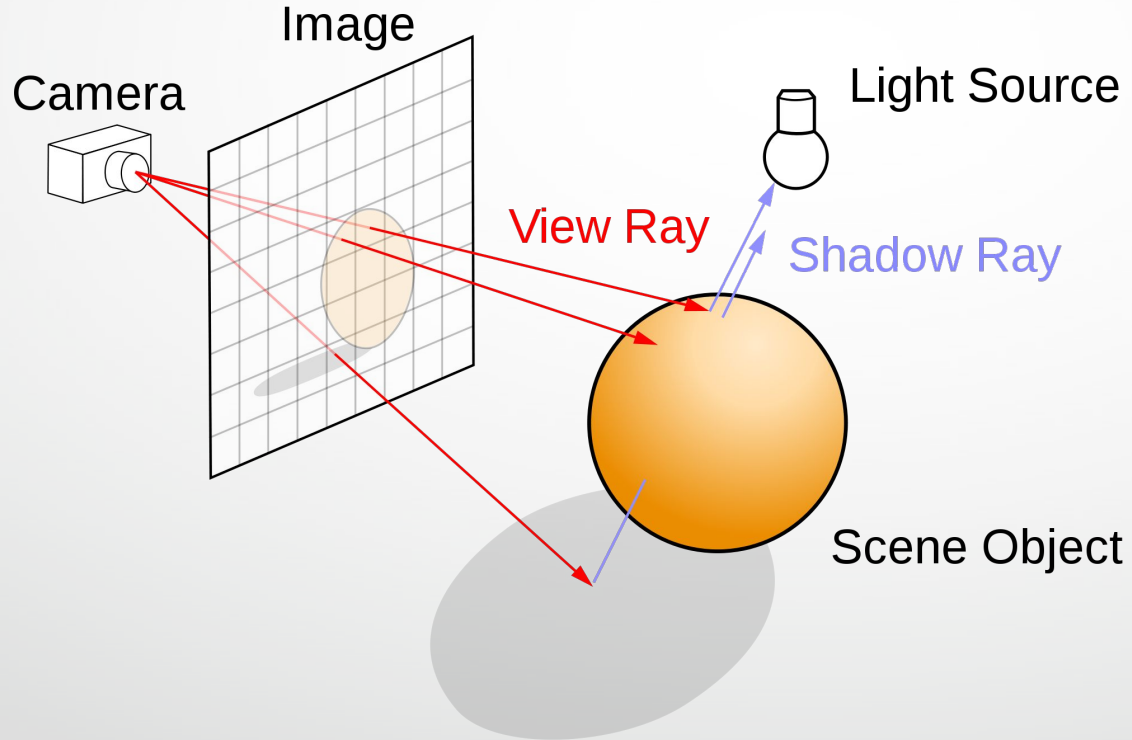
<https://www.zbrushcentral.com/t/headshot-series/360990>

# Mesh simplification

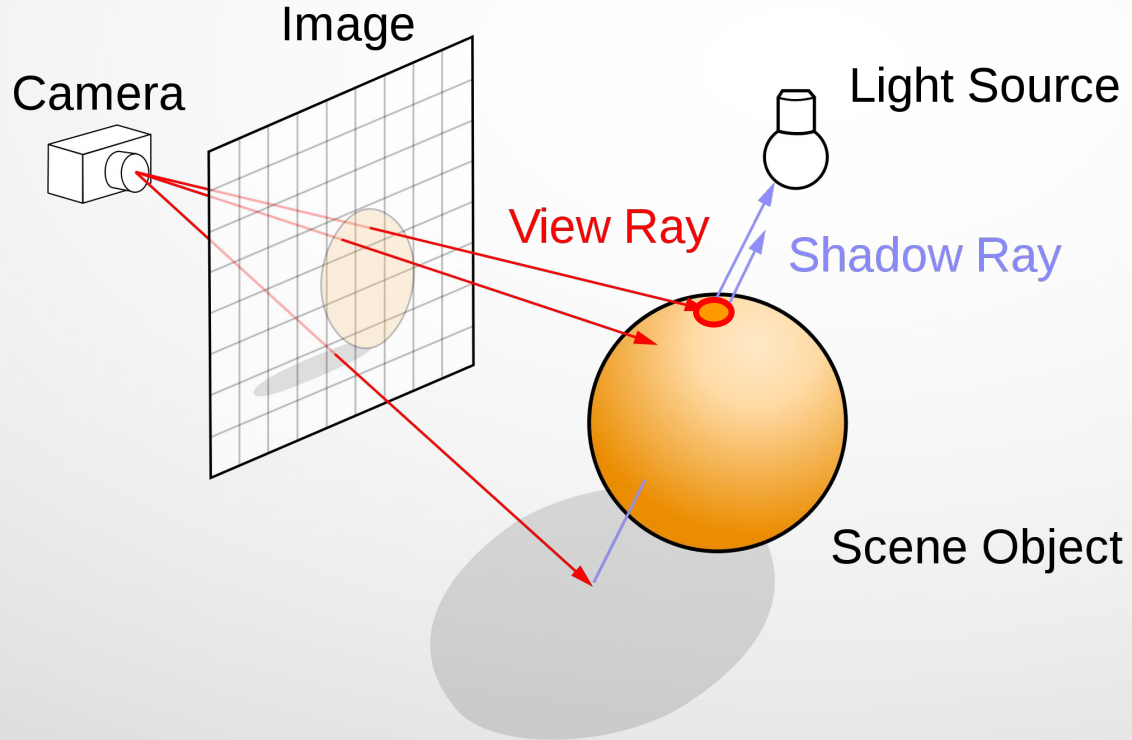


<https://vgc.poly.edu/~csilva/papers/siggraph-asia2008.pdf>

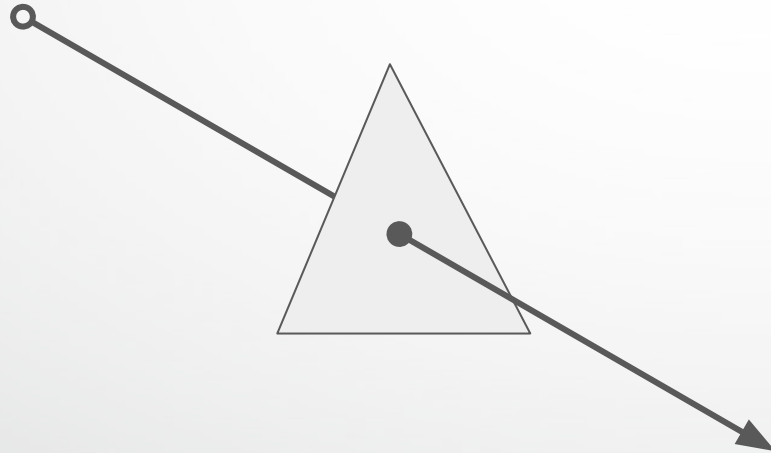
# Ray tracing



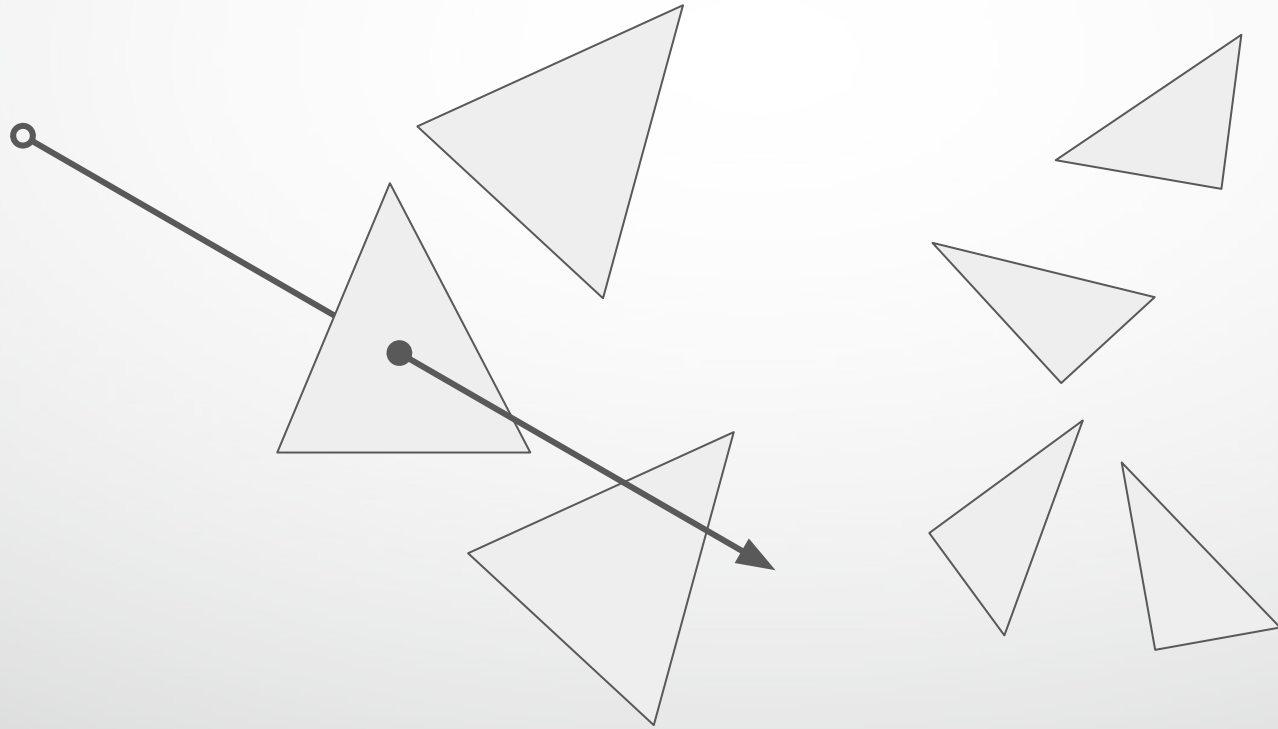
# Ray-triangle intersection



# Ray-triangle intersection (cont'd)

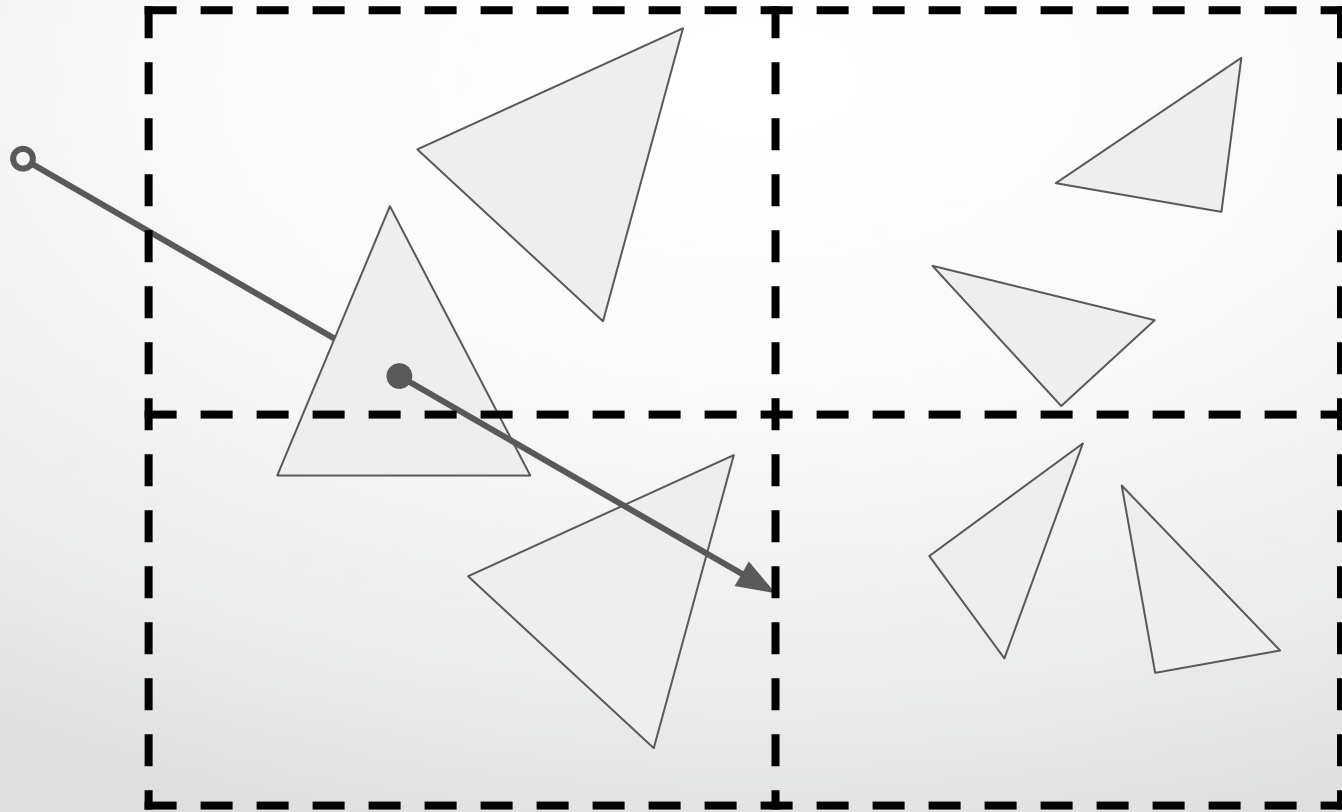


# Ray-triangle intersection (cont'd)

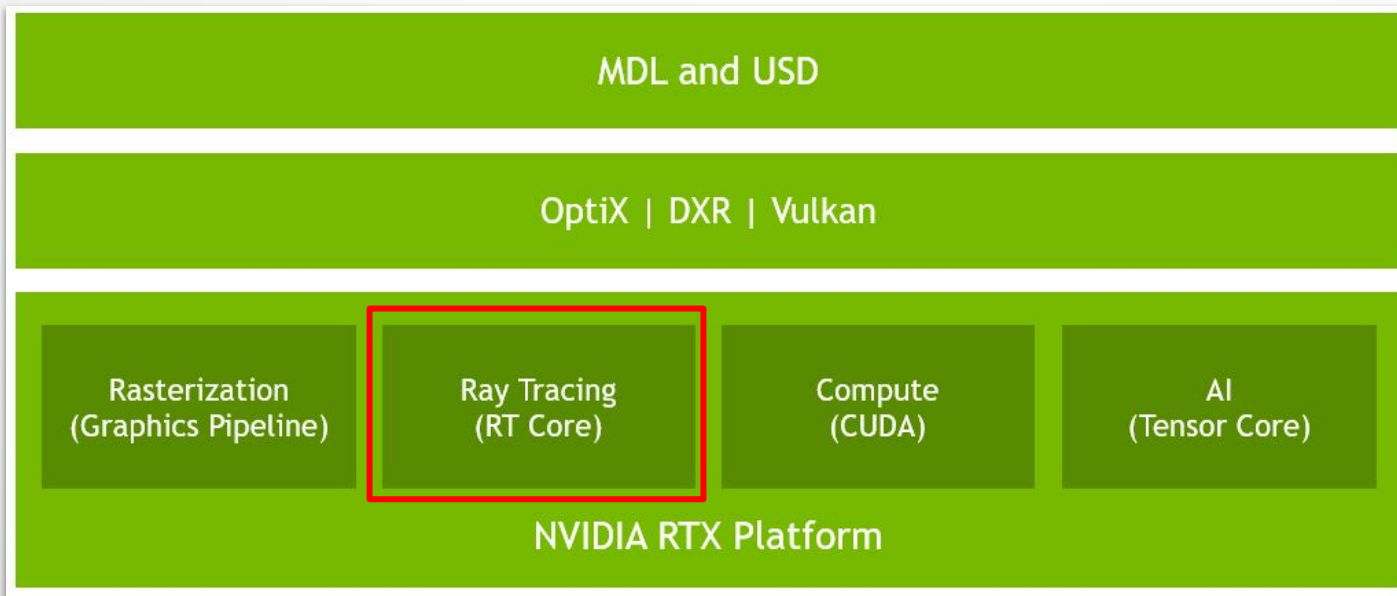




# Space-partitioning data structures

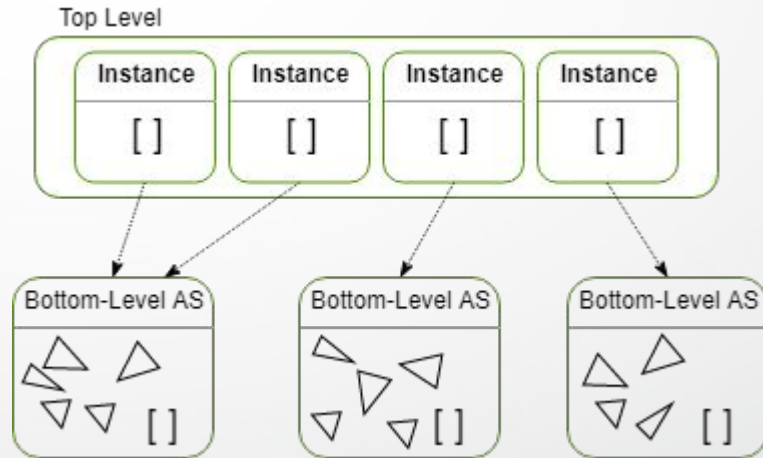
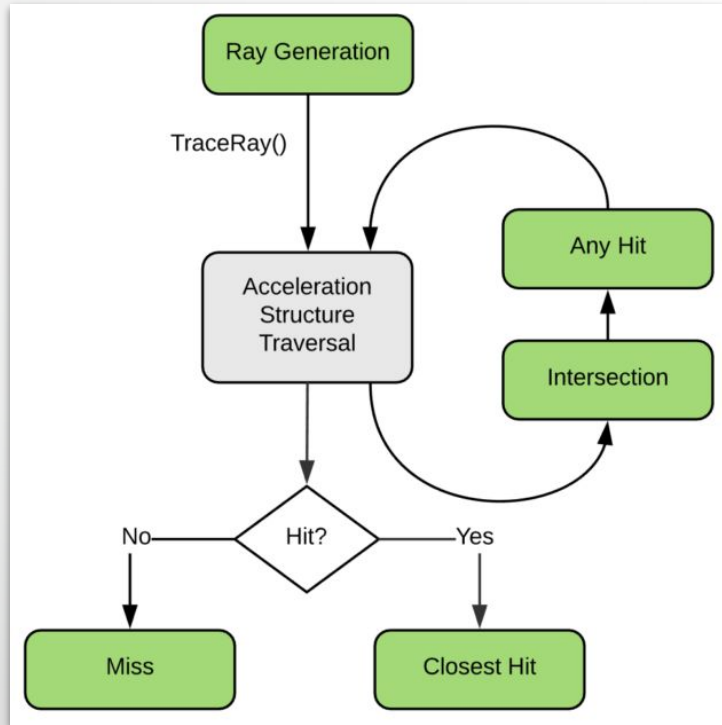


# Case study: NVIDIA RTX™ Platform



<https://developer.nvidia.com/rtx>

# Case study: DXR (DirectX Raytracing)





ILMxLAB

IMMERSIVE ENTERTAINMENT



# Ray Tracing with HDRP

The screenshot shows the Unity documentation website. At the top, there is a navigation bar with the Unity logo, links for Manual, Scripting API, Changelog, and License, a search box, and the URL docs.unity3d.com. Below the navigation bar, the page title is 'High Definition RP 7.1.8'. A left sidebar contains a search filter and a table of contents with categories like Features, Getting started, Upgrading HDRP between Unity Versions, Volume Framework, Render Pipeline Settings, Materials, Lighting, Camera, Post-processing, and Ray Tracing. The main content area is titled 'Getting started with ray tracing' and includes an introduction to HDRP ray tracing, a list of topics covered in the document, and a section on hardware requirements. A right sidebar titled 'IN THIS ARTICLE' lists sub-topics like hardware requirements, integration into HDRP projects, and overview of effects.

unity Manual Scripting API Changelog License Search docs.unity3d.com →

High Definition RP 7.1.8 ▾

Enter here to filter...

High Definition Render Pipeline

- + Features
- + Getting started
- + Upgrading HDRP between Unity Versions
- + Volume Framework
- + Render Pipeline Settings
- + Materials
- + Lighting
- + Camera
- + Post-processing
- Ray Tracing
  - Getting Started with Ray Tracing
  - + Effects and Volume Overrides

Manual / Ray Tracing / Getting Started with Ray Tracing

## 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

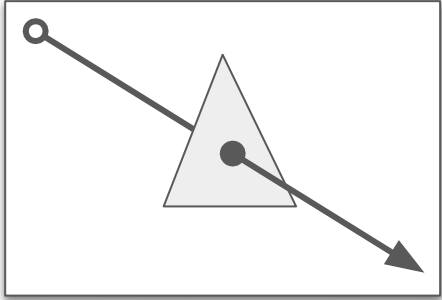
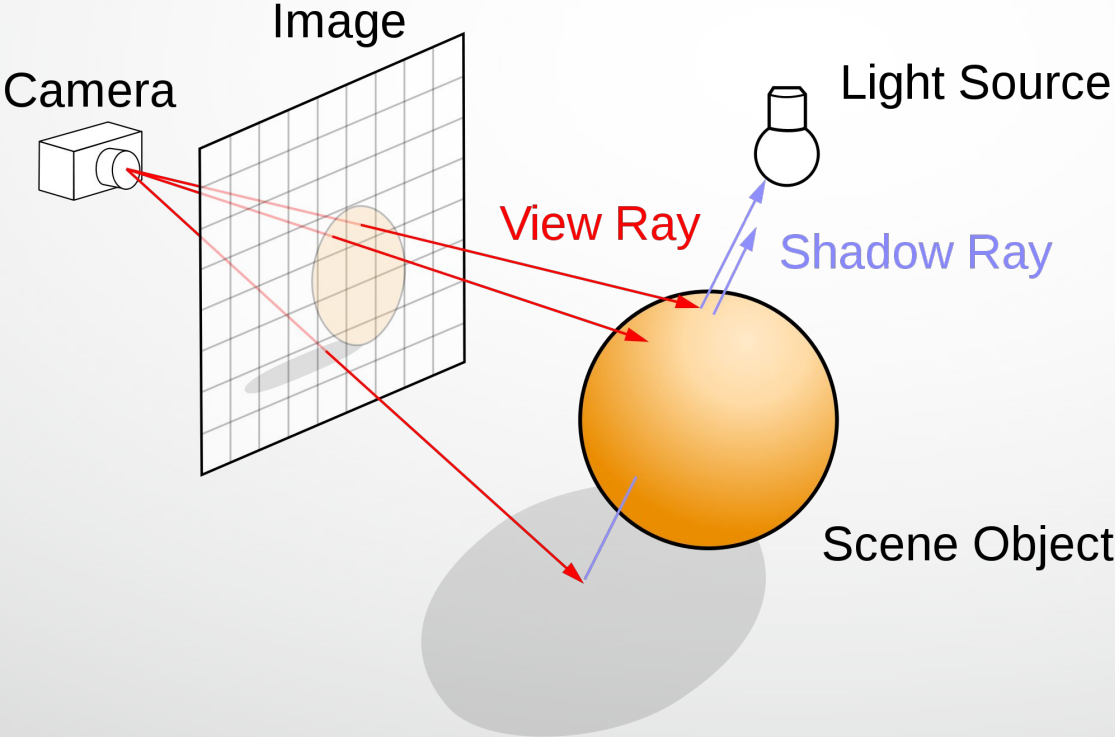
Ray tracing hardware acceleration is only available on certain graphics cards. The graphics cards with full support are:

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

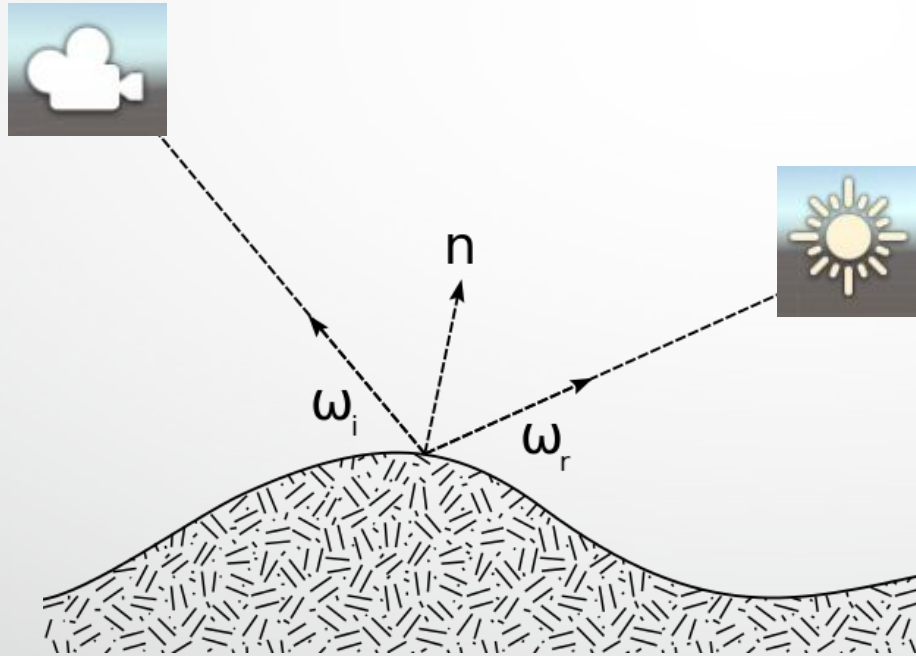
**IN THIS ARTICLE**

- Hardware requirements
- Integrating ray tracing into your HDRP Project
- Ray tracing effects overview
- Ray tracing project
- Advice and supported feature of preview ray tracing

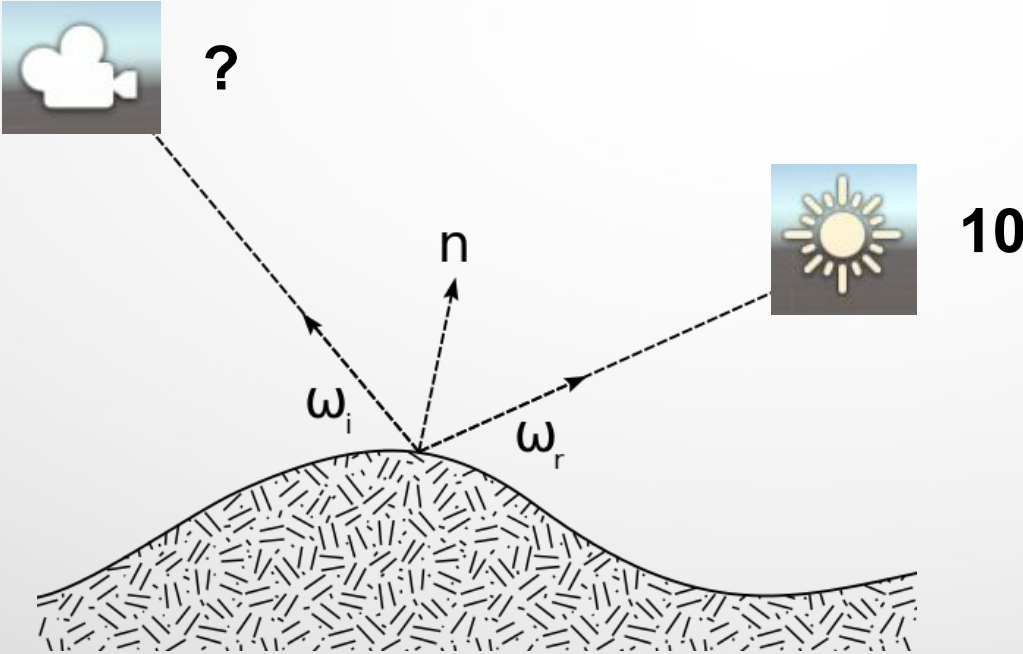
# Ray tracing



# Material



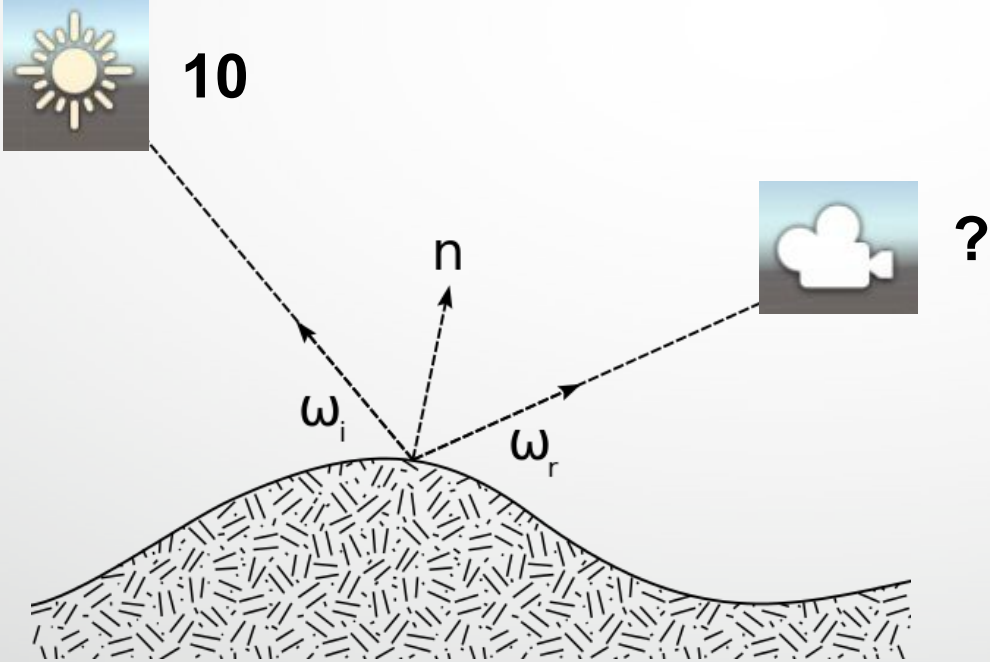
# Material



$$\mathbf{f}(\omega_i, \omega_r)$$

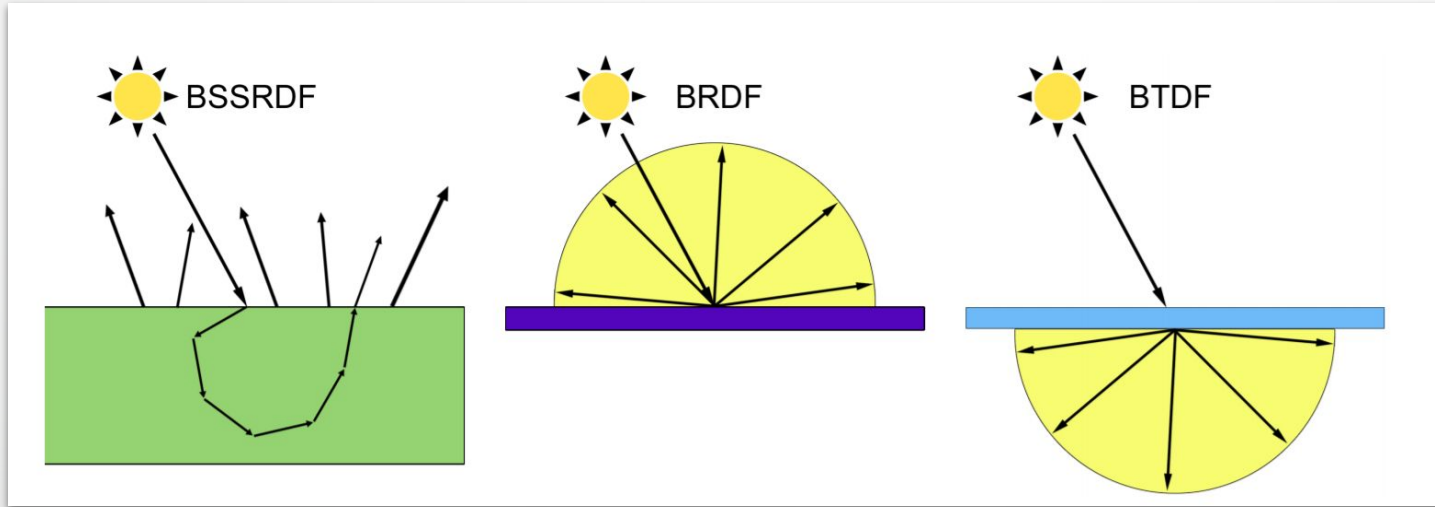


# Material



$$\mathbf{f}(\omega_r, \omega_i)$$

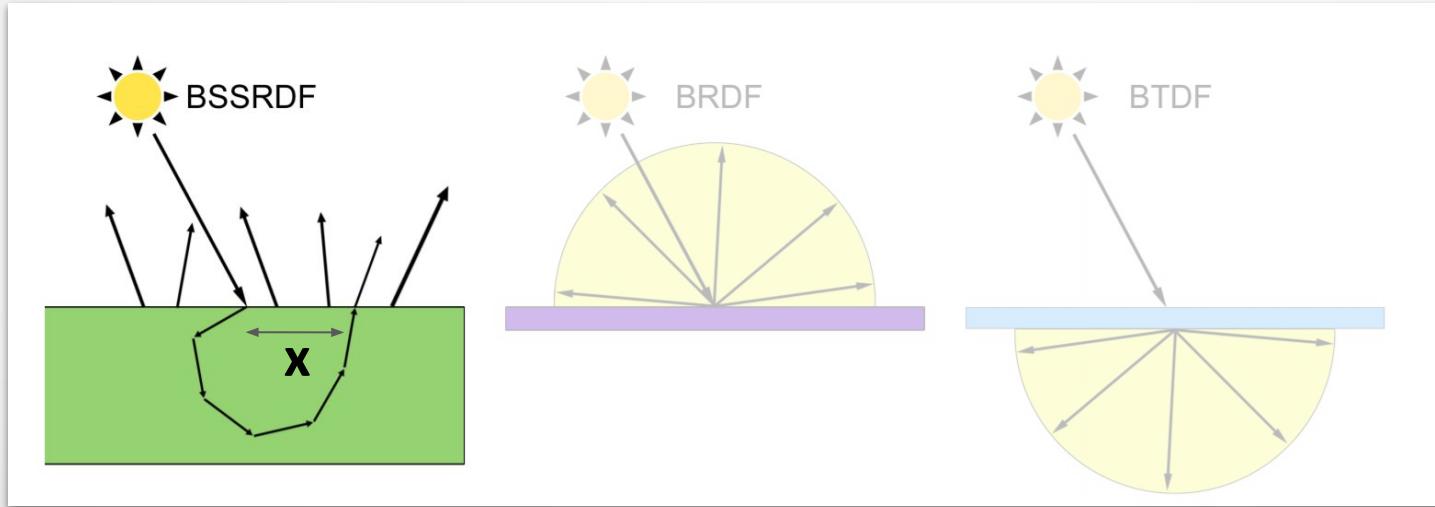
# Material (cont'd)



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

# Material (cont'd)

BSSRDF (Bidirectional scattering distribution function)

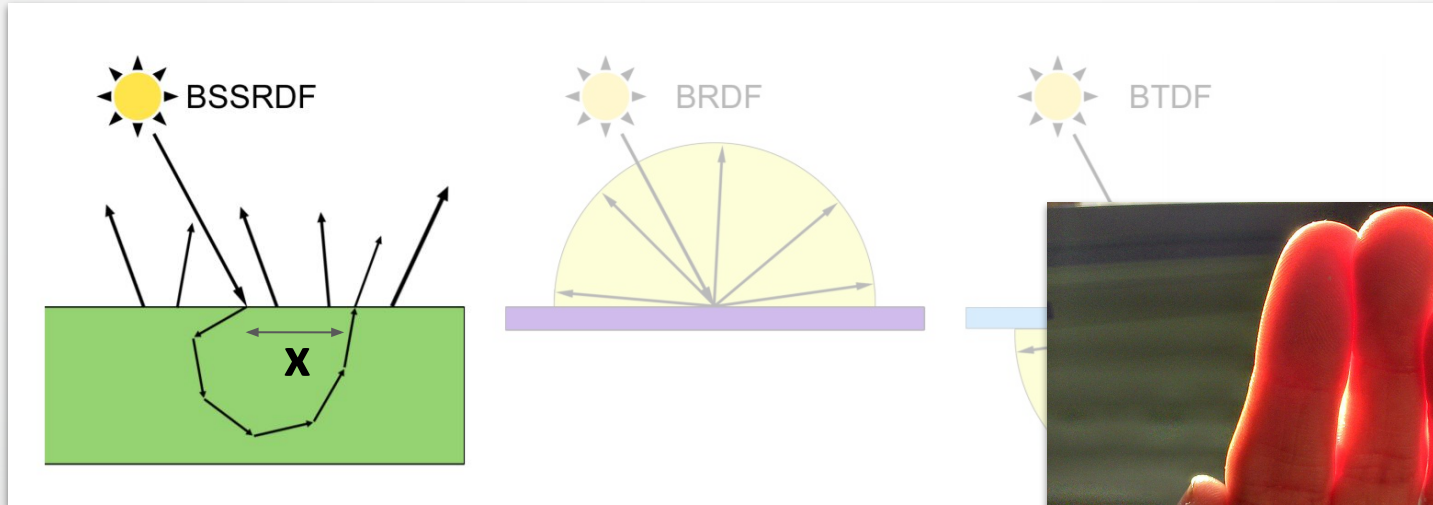


<http://collagefactory.blogspot.com/2010/04/brdf-for-diffuseglossyspecular.html>

$$\mathbf{f}_{\text{BSSRDF}}(\mathbf{x}, \omega_i, \omega_r)$$

# Material (cont'd)

BSSRDF (Bidirectional scattering distribution function)



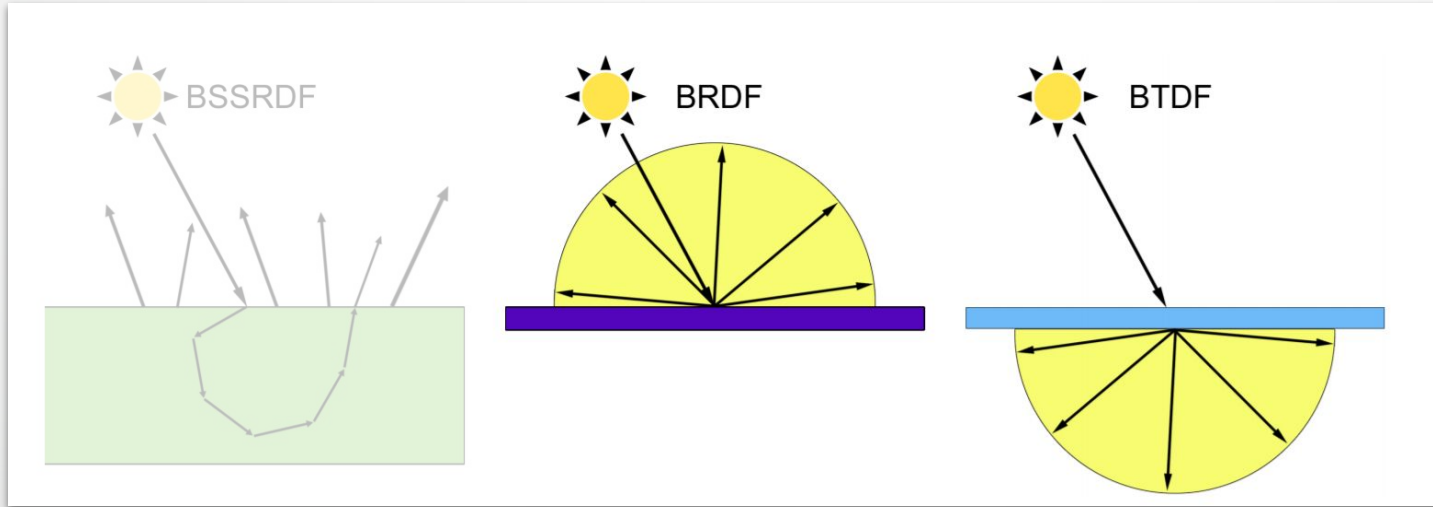
<http://collagefactory.blogspot.com/2010/04/brdf-for-diffuseglossyspecular.html>

$$\mathbf{f}_{\text{BSSRDF}}(\mathbf{x}, \omega_i, \omega_r)$$

[https://en.wikipedia.org/wiki/Subsurface\\_scattering](https://en.wikipedia.org/wiki/Subsurface_scattering)

# Material (cont'd)

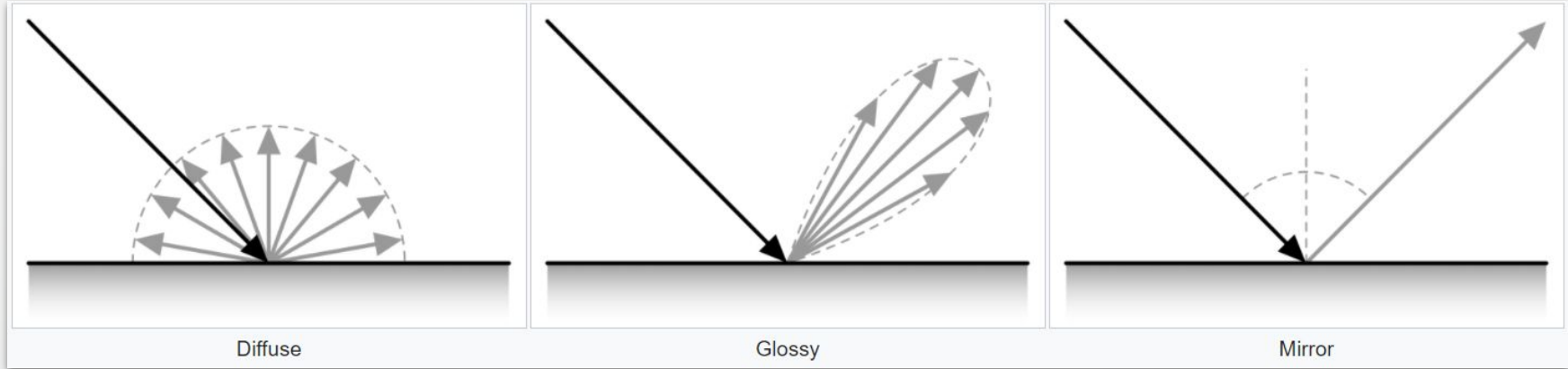
BSSDF (Bidirectional scattering distribution function) =  
BRDF (Bidirectional reflectance distribution function) +  
BTDF (Bidirectional transmittance distribution function)



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

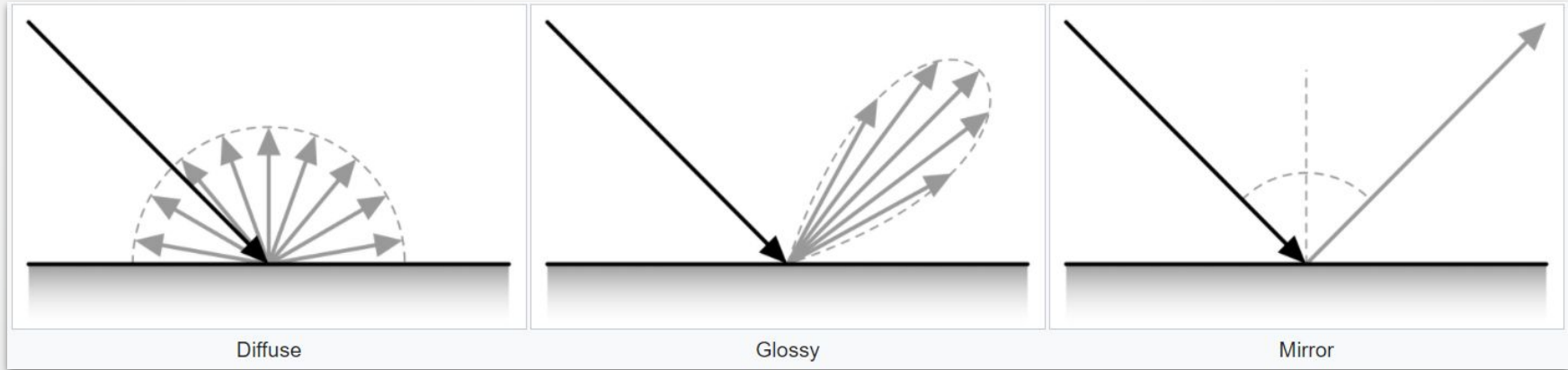
$$\mathbf{f}_{\text{BRDF}}(\omega_i, \omega_r) \quad \mathbf{f}_{\text{BTDF}}(\omega_i, \omega_r)$$

# BRDF (Bidirectional reflectance distribution function)

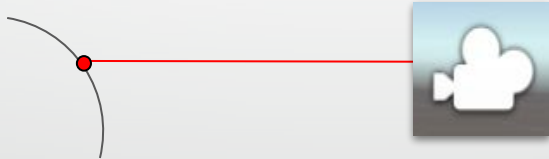
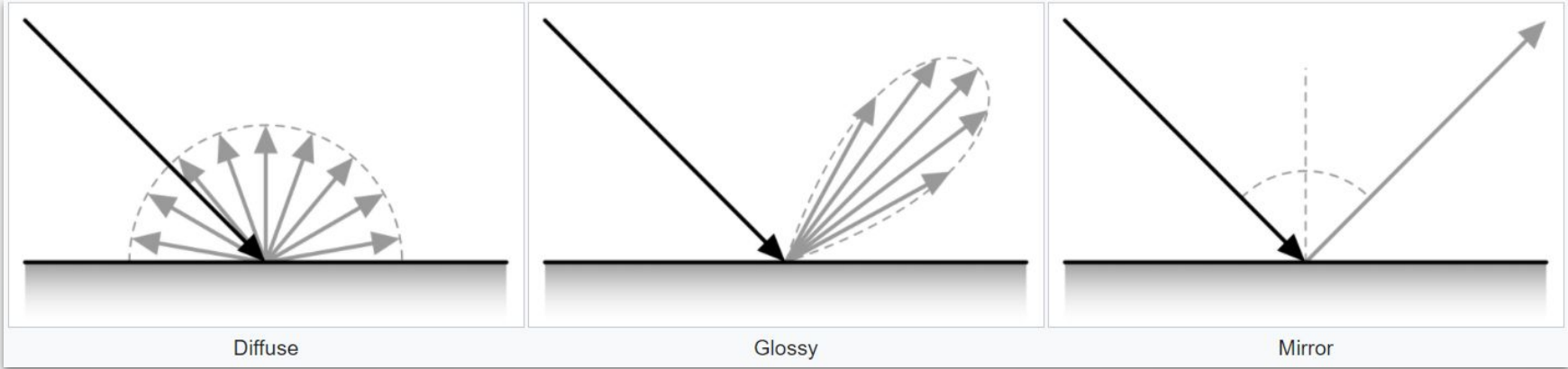


[https://en.wikipedia.org/wiki/Bidirectional\\_reflectance\\_distribution\\_function](https://en.wikipedia.org/wiki/Bidirectional_reflectance_distribution_function)

# BRDF (Bidirectional reflectance distribution function)

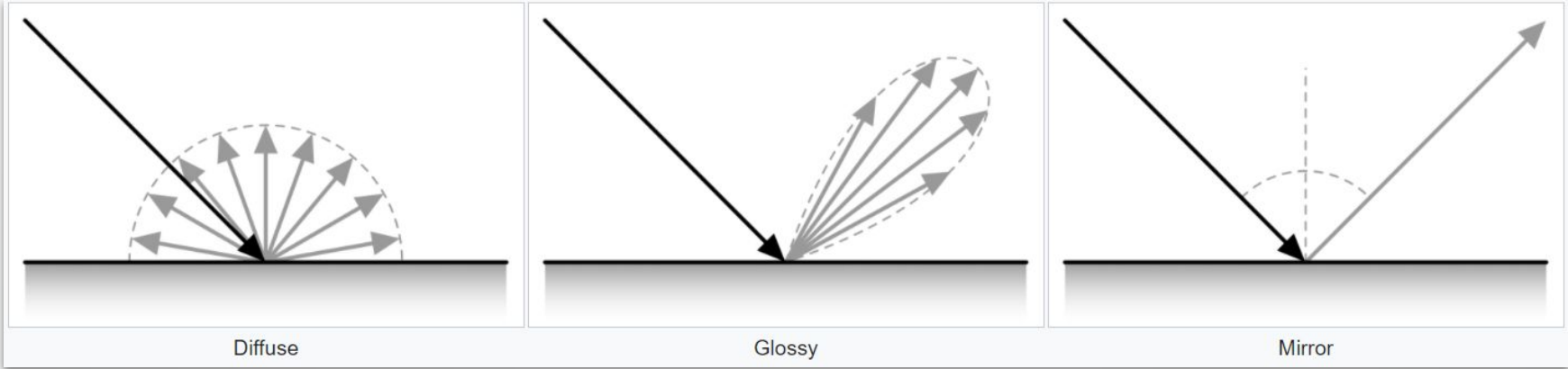


# BRDF (Bidirectional reflectance distribution function)

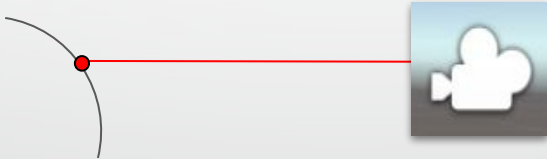




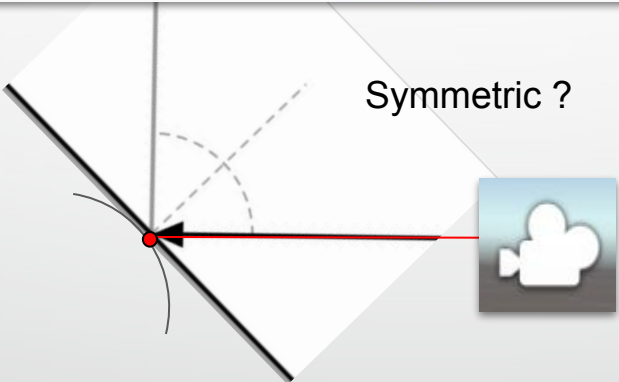
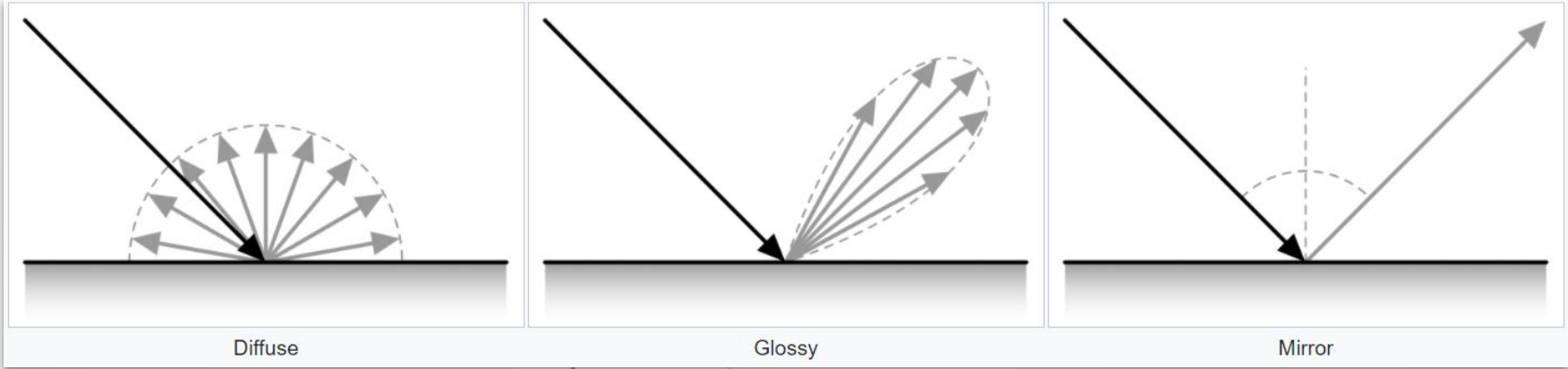
# BRDF (Bidirectional reflectance distribution function)



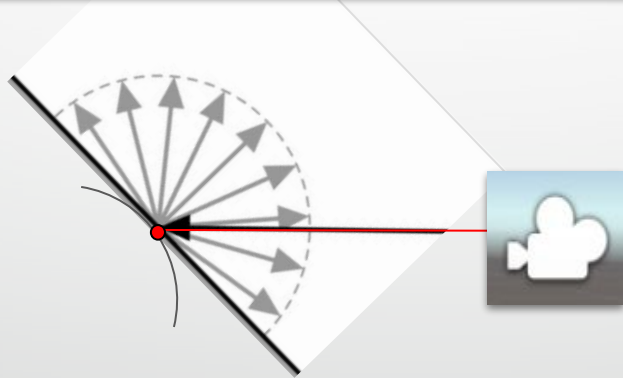
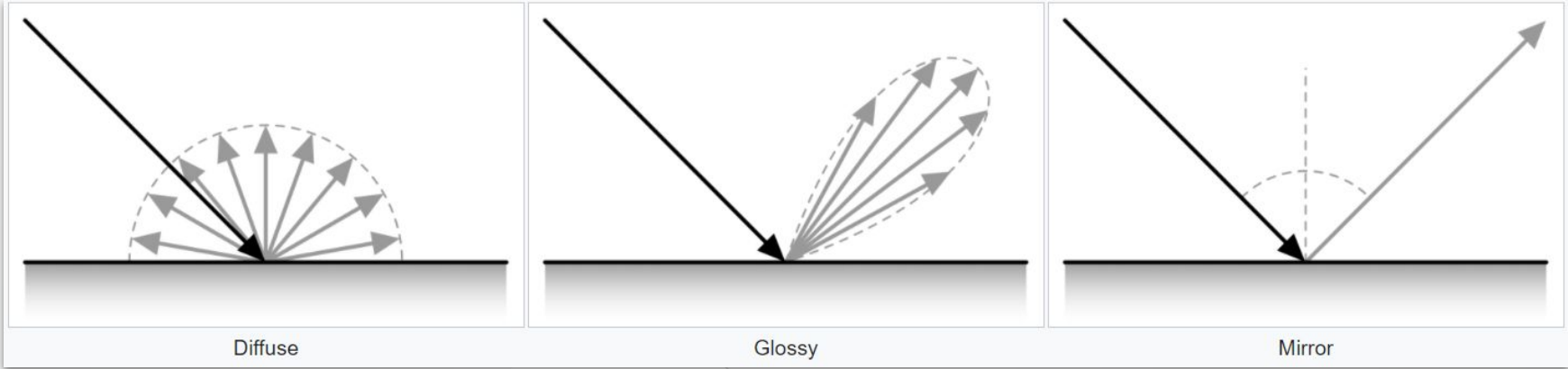
Emissive ?



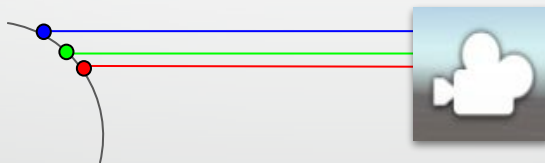
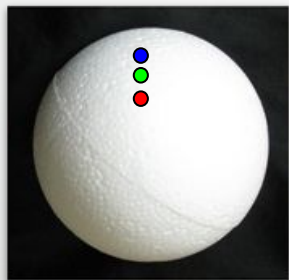
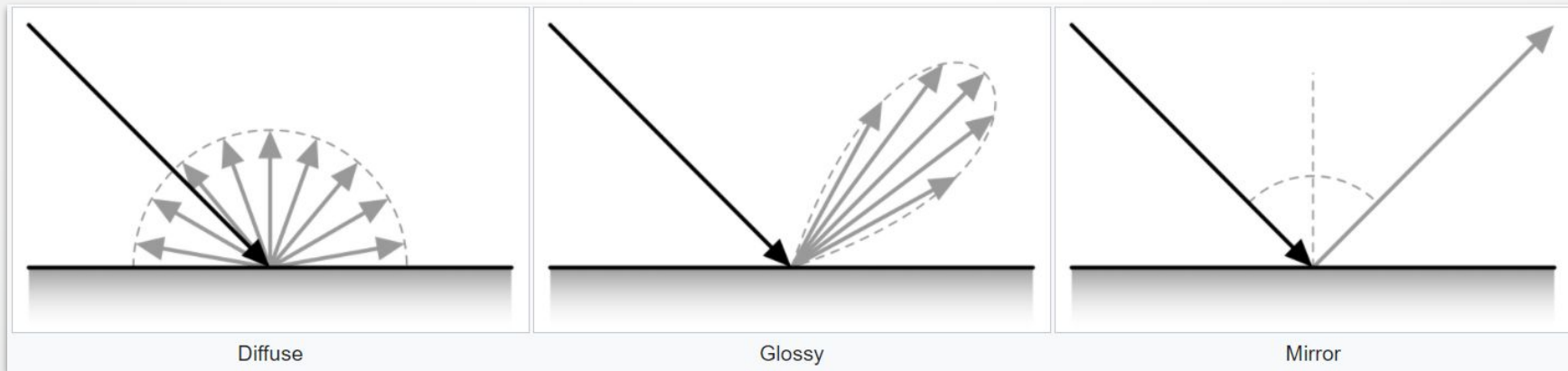
# BRDF (Bidirectional reflectance distribution function)



# BRDF (Bidirectional reflectance distribution function)

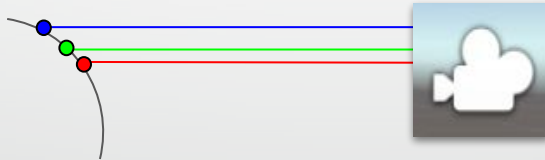
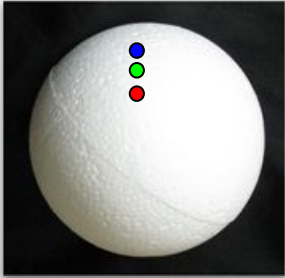
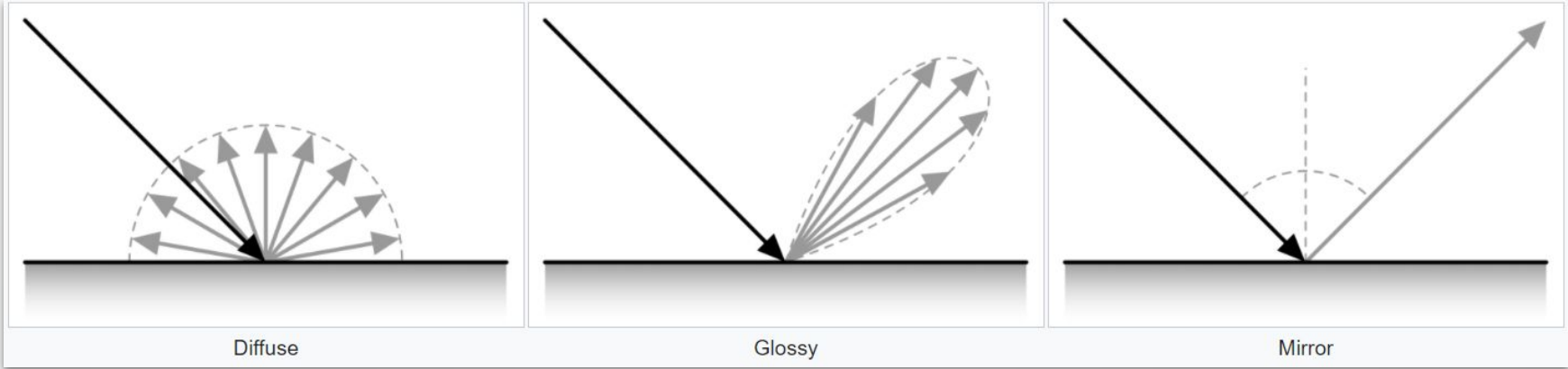


# BRDF (Bidirectional reflectance distribution function)

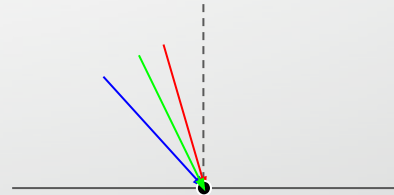


- Different **Normals**

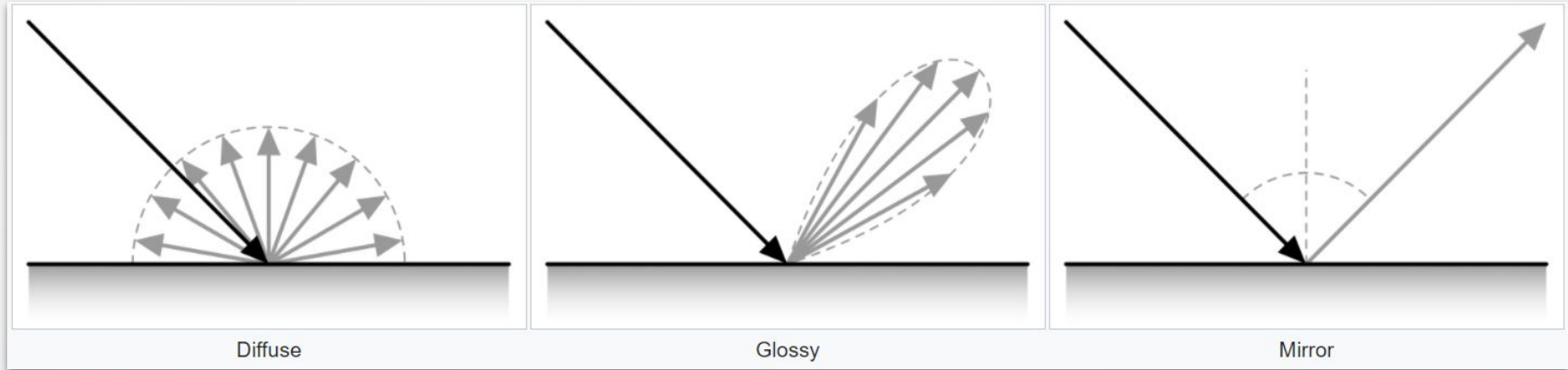
# BRDF (Bidirectional reflectance distribution function)



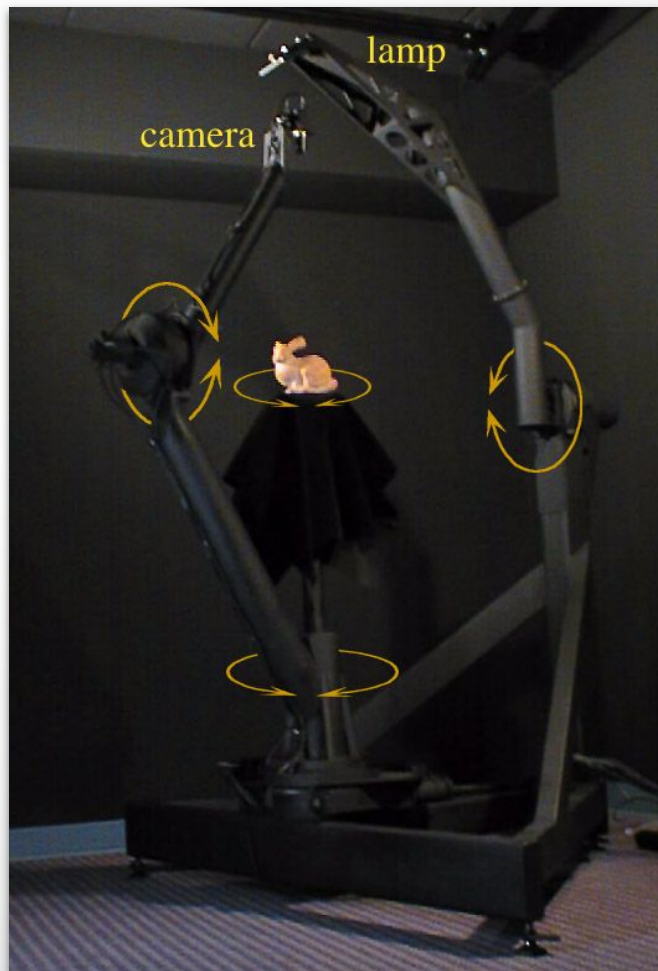
- Different **Normals**



# BRDF (Bidirectional reflectance distribution function)

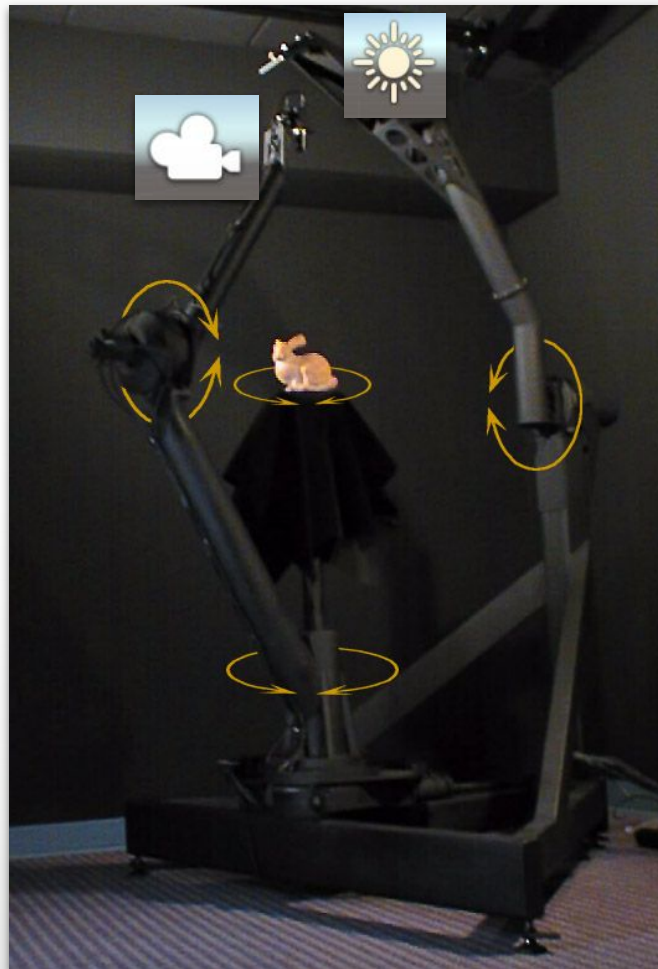


# BRDF Measurement



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

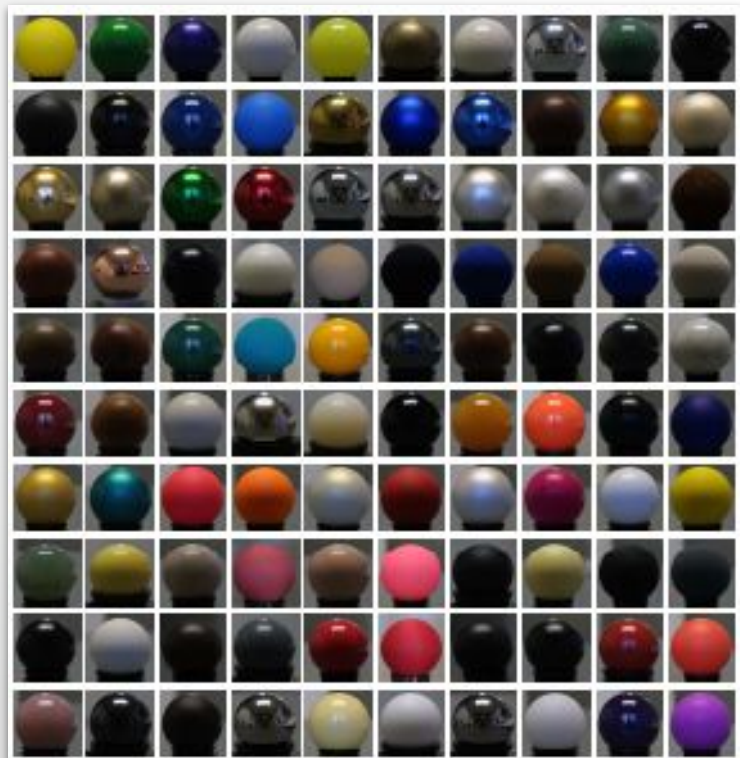
# BRDF Measurement



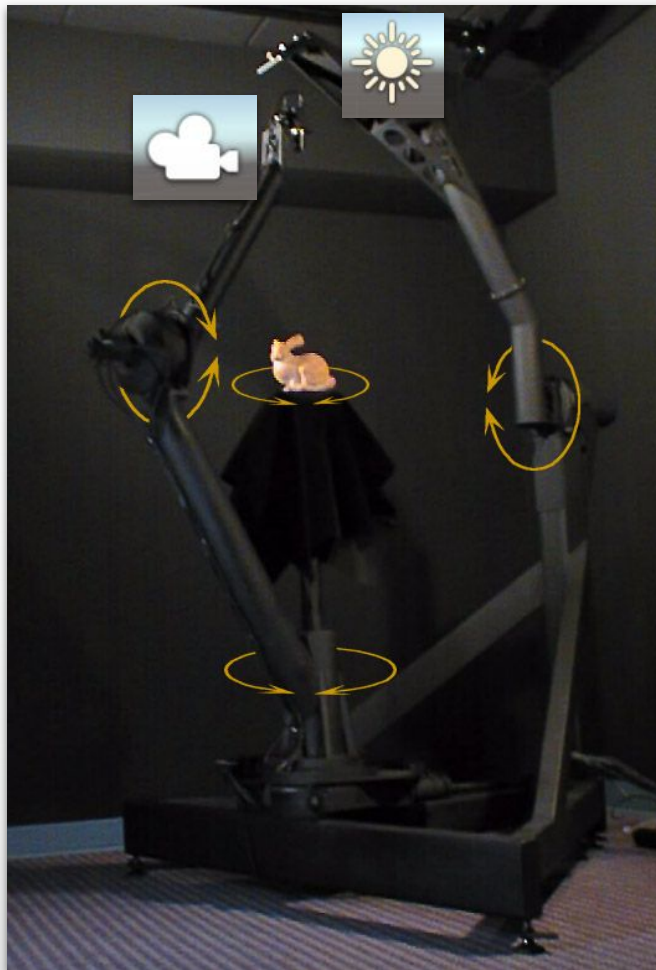
<https://graphics.stanford.edu/projects/gantry/>



# BRDF Measurement

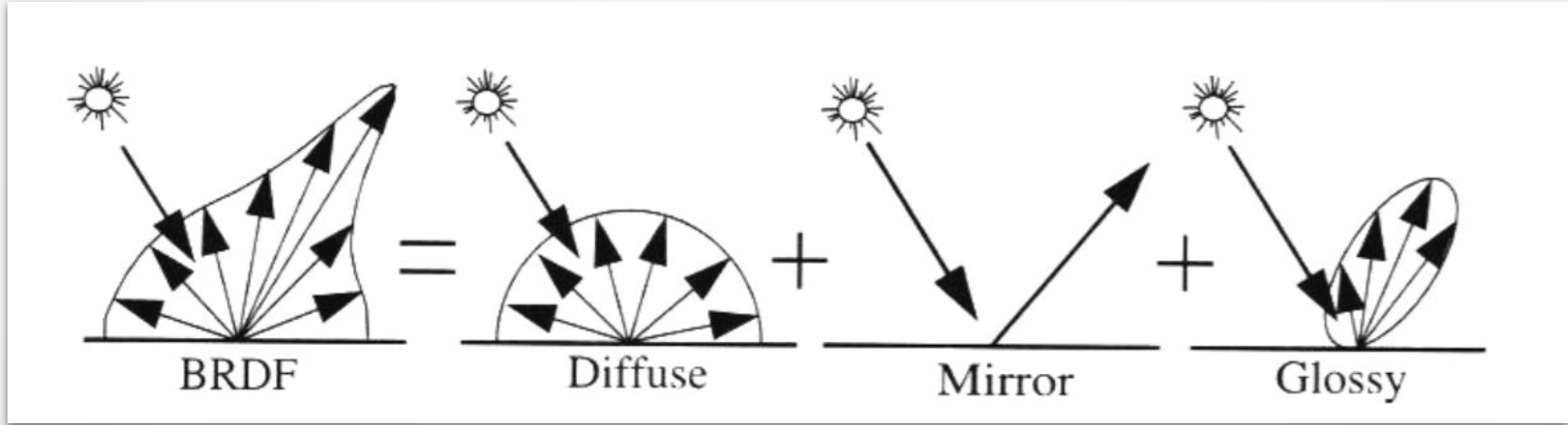


<https://www.merl.com/brdf/>



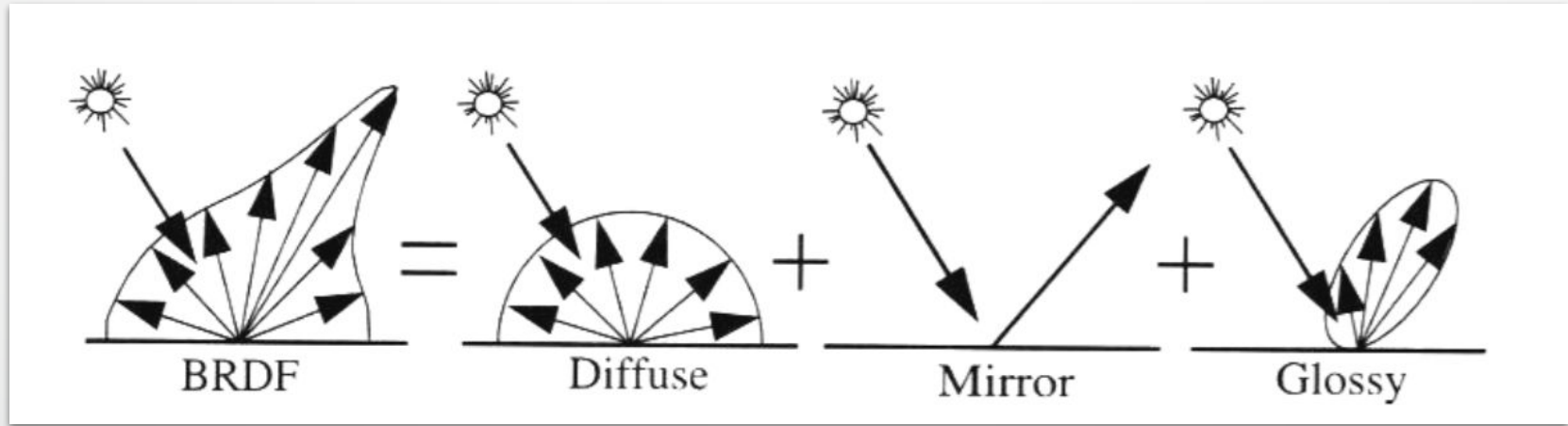
<https://graphics.stanford.edu/projects/gantry/>

# Analytic BRDF (Bidirectional reflectance distribution function)



<http://resources.mpi-inf.mpg.de/departments/d4/teaching/ws200708/cg/slides/CG07-Brdf+Texture.pdf>

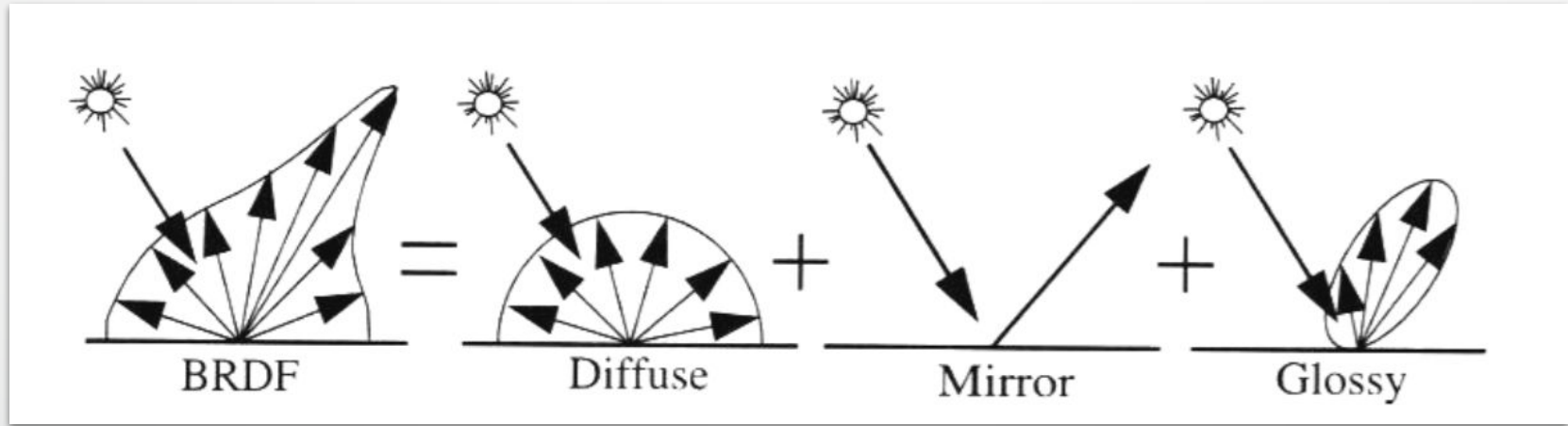
# Analytic BRDF (Bidirectional reflectance distribution function)



<http://resources.mpi-inf.mpg.de/departments/d4/teaching/ws200708/cg/slides/CG07-Brdf+Texture.pdf>

$$= f_D(\text{ } ) + f_M(\text{ } ) + f_G(\text{ } )$$

# Analytic BRDF (Bidirectional reflectance distribution function)

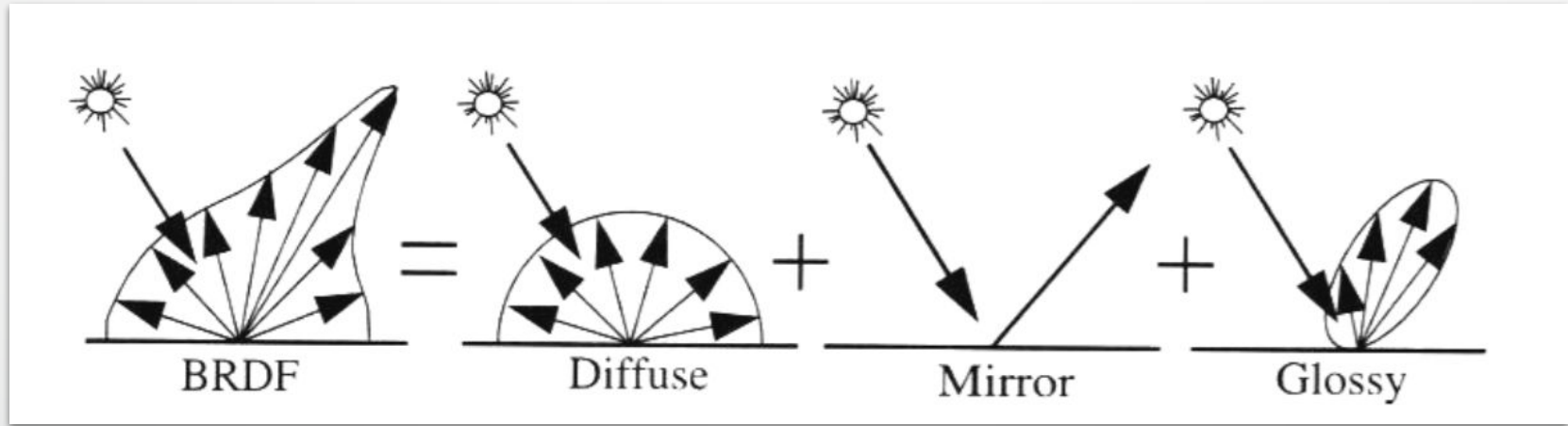


<http://resources.mpi-inf.mpg.de/departments/d4/teaching/ws200708/cg/slides/CG07-Brdf+Texture.pdf>



$$= f_D(0.8) + f_M(2) + f_G(0.1)$$

# Analytic BRDF (Bidirectional reflectance distribution function)

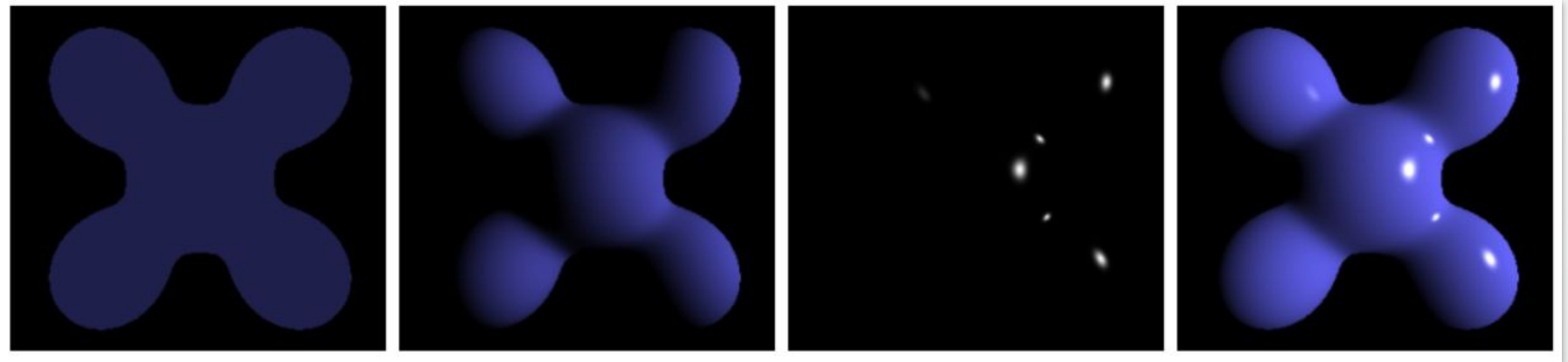


<http://resources.mpi-inf.mpg.de/departments/d4/teaching/ws200708/cg/slides/CG07-Brdf+Texture.pdf>



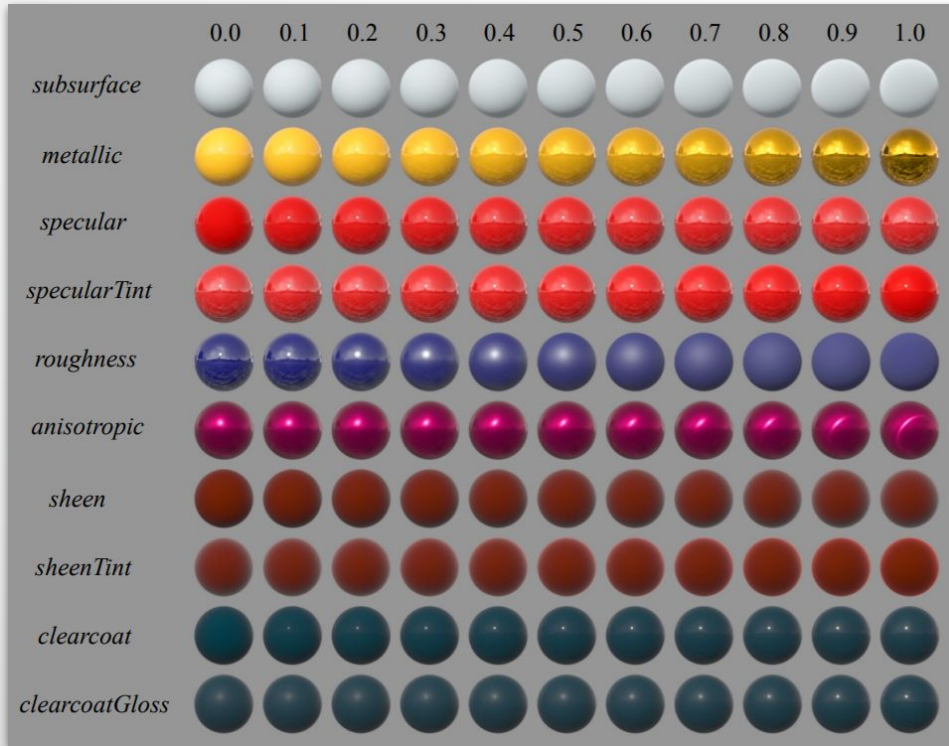
$$= f_D(0.8) + f_M(2) + f_G(0.1)$$

# Phong illumination model



$$\mathbf{I} \text{ Ambient}(k_a) + \boxed{\mathbf{I} \text{ Diffuse}(k_d) + \mathbf{I} \text{ Specular}(k_s)} = \mathbf{I}$$

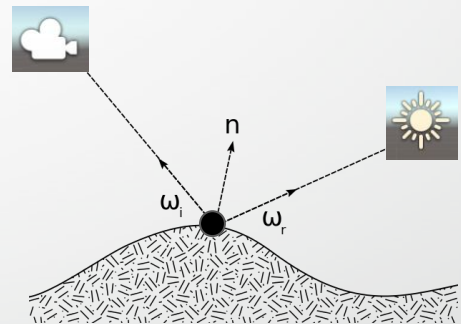
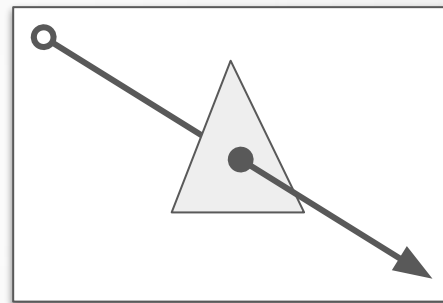
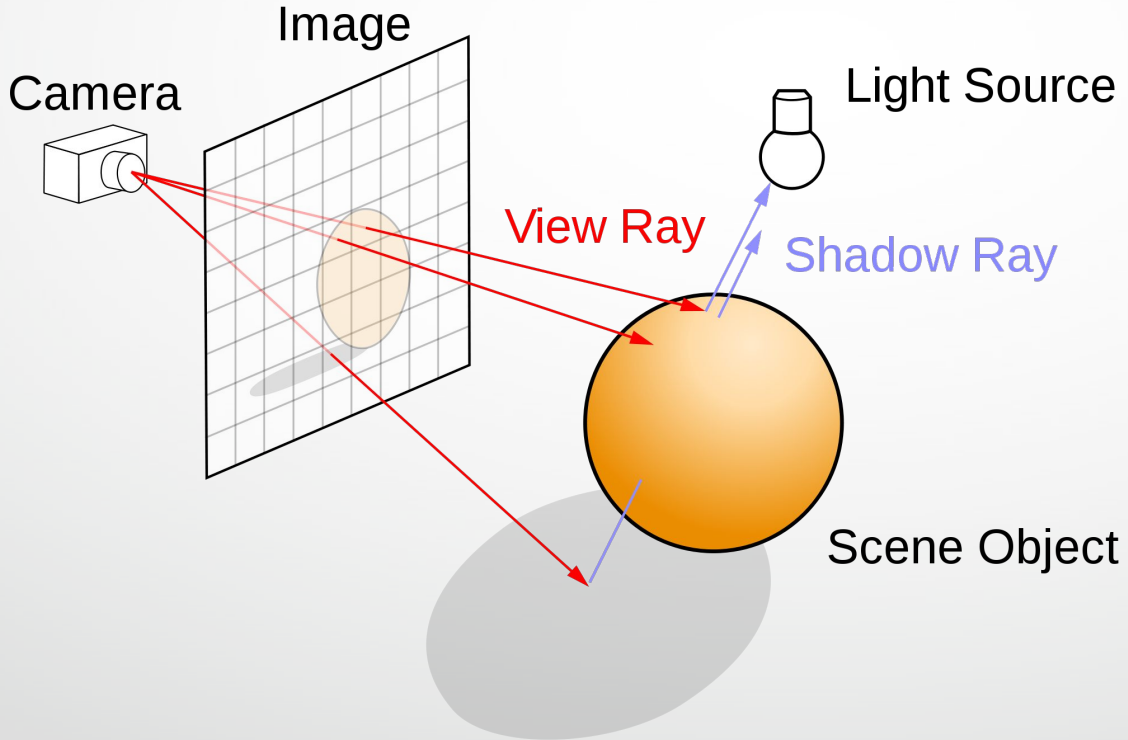
# Disney Principled BRDF



Github :

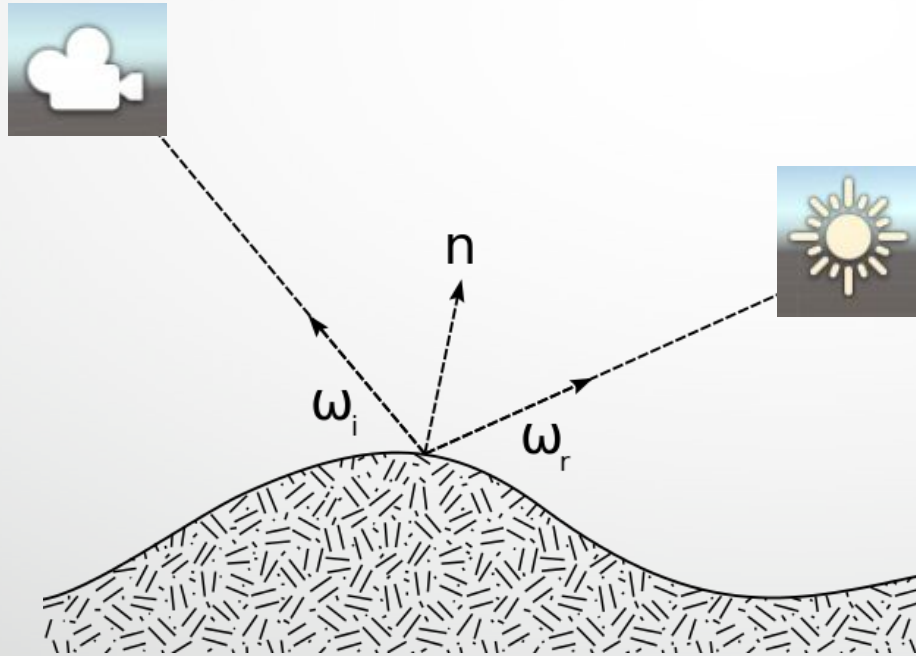
<https://github.com/wdas/brdf/blob/main/src/brdfs/disney.brd>

# Ray tracing



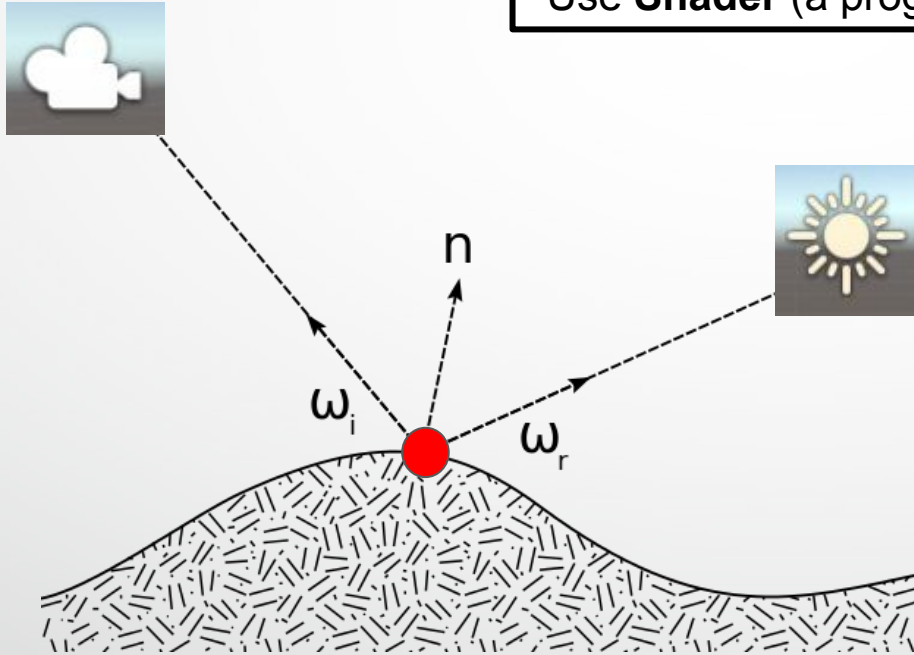


# Material



# Material

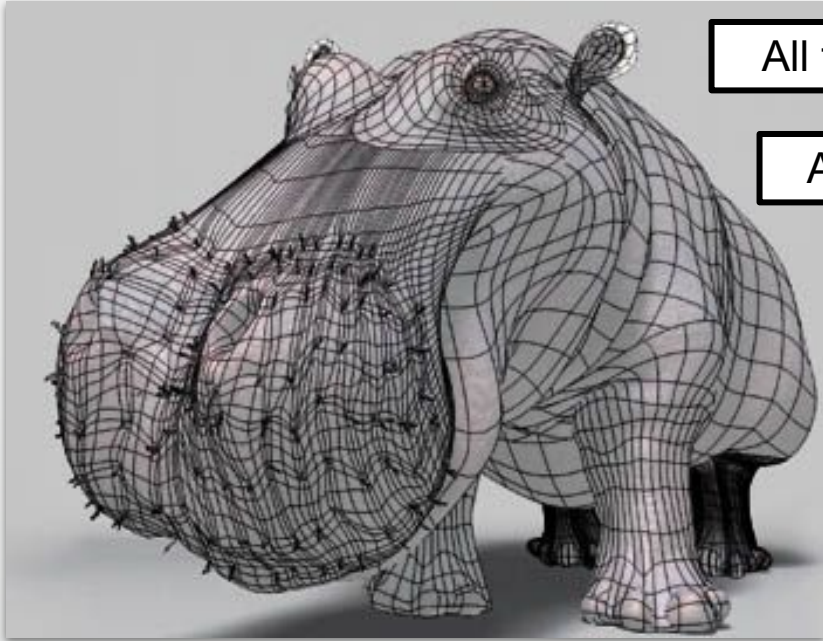
Use **Shader** (a program) to calculate the result



Given :

- Light
- View
- Normal
- BRDF parameters
- Position
- ...

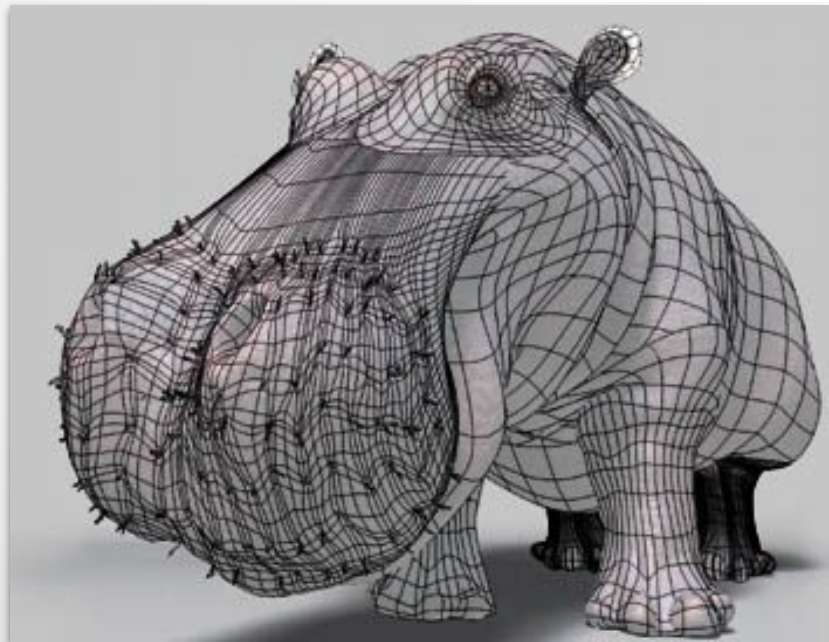
# Material (cont'd)



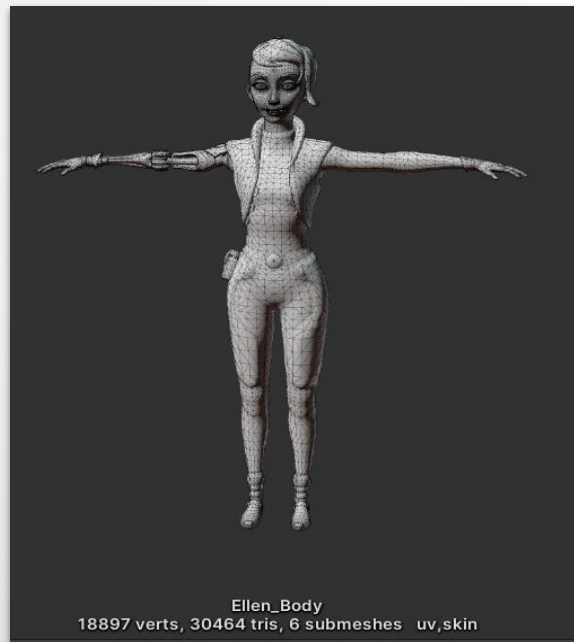
All triangles/quads have the same **Material** ?

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

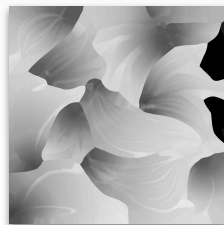
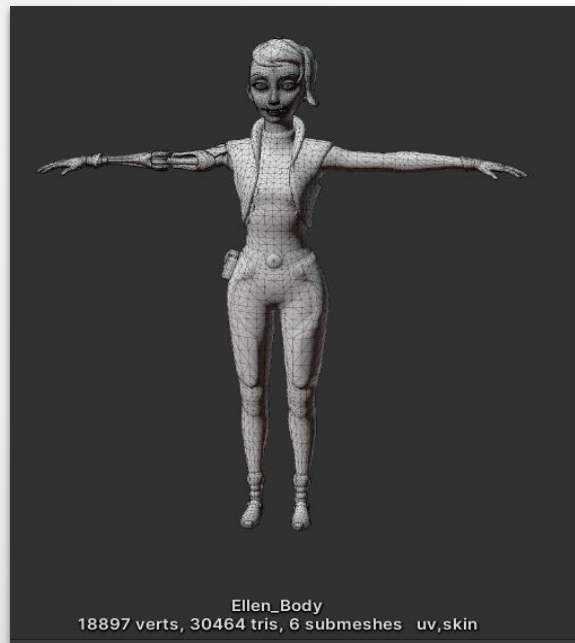
# Texture



# Texture



# Texture

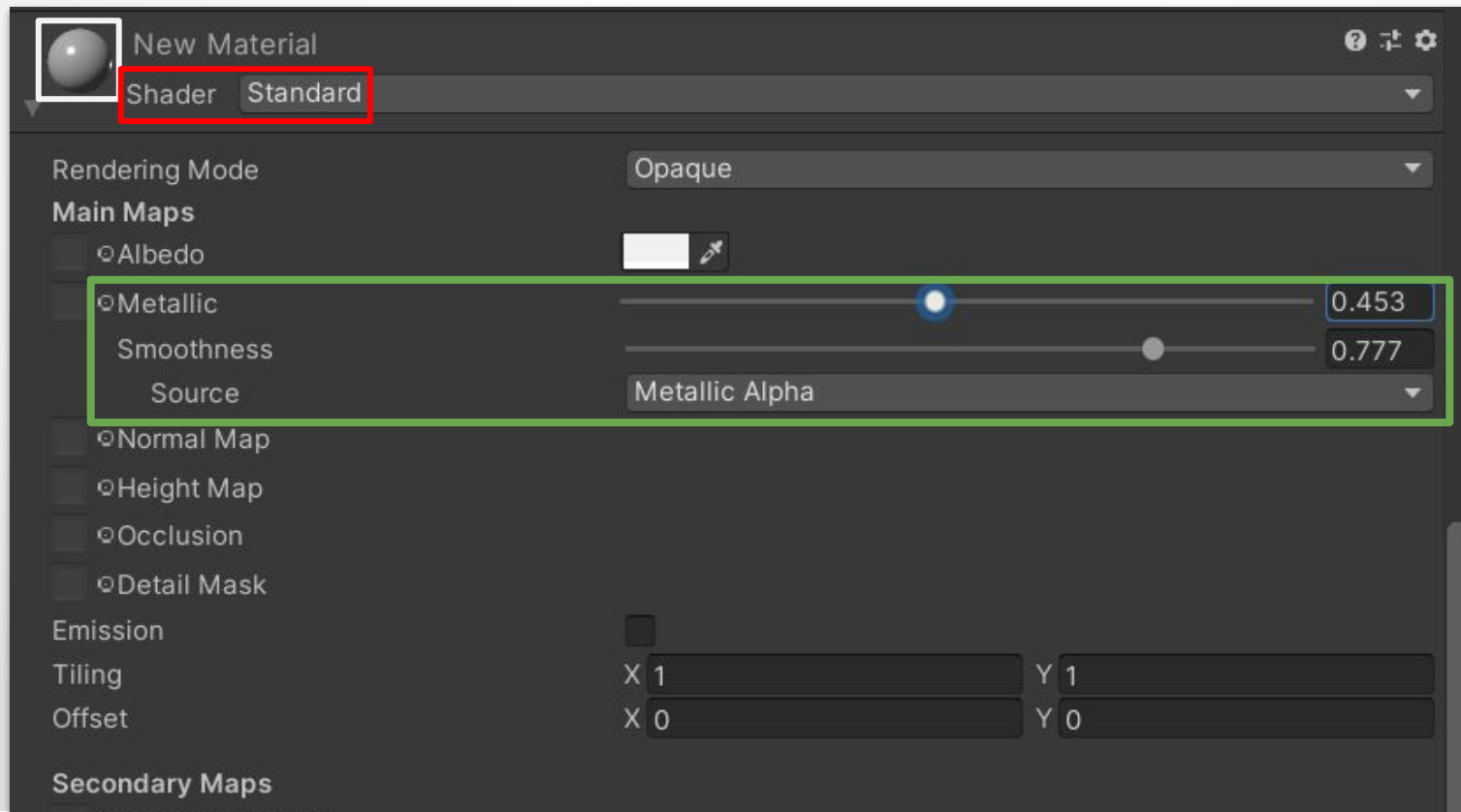


u

v



# Material



The image shows the Blender Material Properties panel. At the top, there is a 'New Material' button with a sphere icon. Below it, the 'Shader' dropdown is set to 'Standard', which is highlighted with a red box. The 'Rendering Mode' is set to 'Opaque'. Under the 'Main Maps' section, the 'Metallic' property is set to 0.453 and the 'Smoothness' property is set to 0.777. The 'Metallic Alpha' source is set to 'Metallic Alpha'. Other properties like 'Normal Map', 'Height Map', 'Occlusion', and 'Detail Mask' are currently disabled. The 'Emission' property is also disabled. The 'Tiling' and 'Offset' properties are set to X 1, Y 1 and X 0, Y 0 respectively. The 'Secondary Maps' section is visible at the bottom.

New Material

Shader Standard

Rendering Mode Opaque

Main Maps

Albedo

Metallic 0.453

Smoothness 0.777

Source Metallic Alpha

Normal Map

Height Map

Occlusion

Detail Mask

Emission

Tiling X 1 Y 1

Offset X 0 Y 0

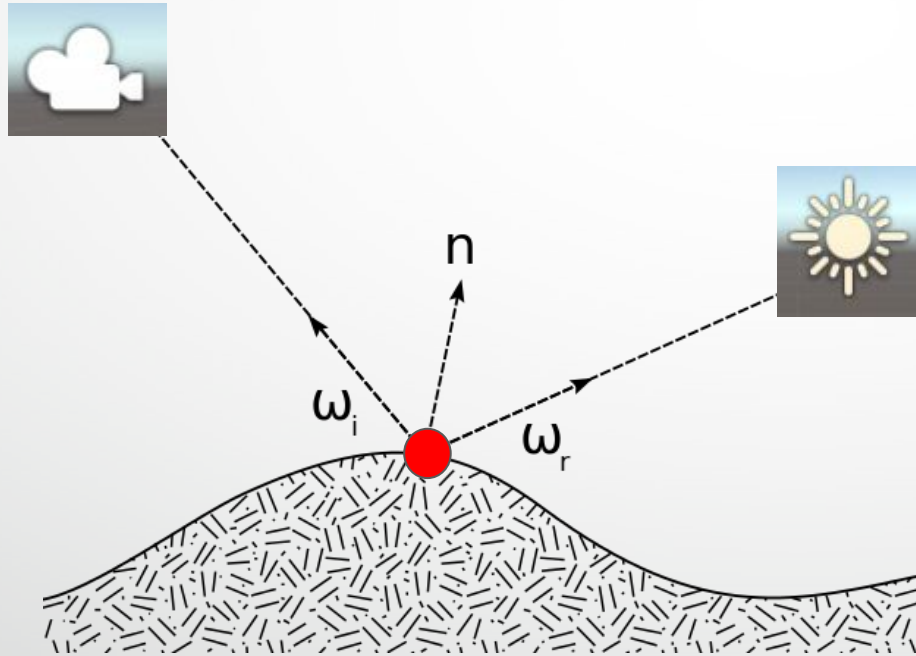
Secondary Maps

# Case Study: Concept Art





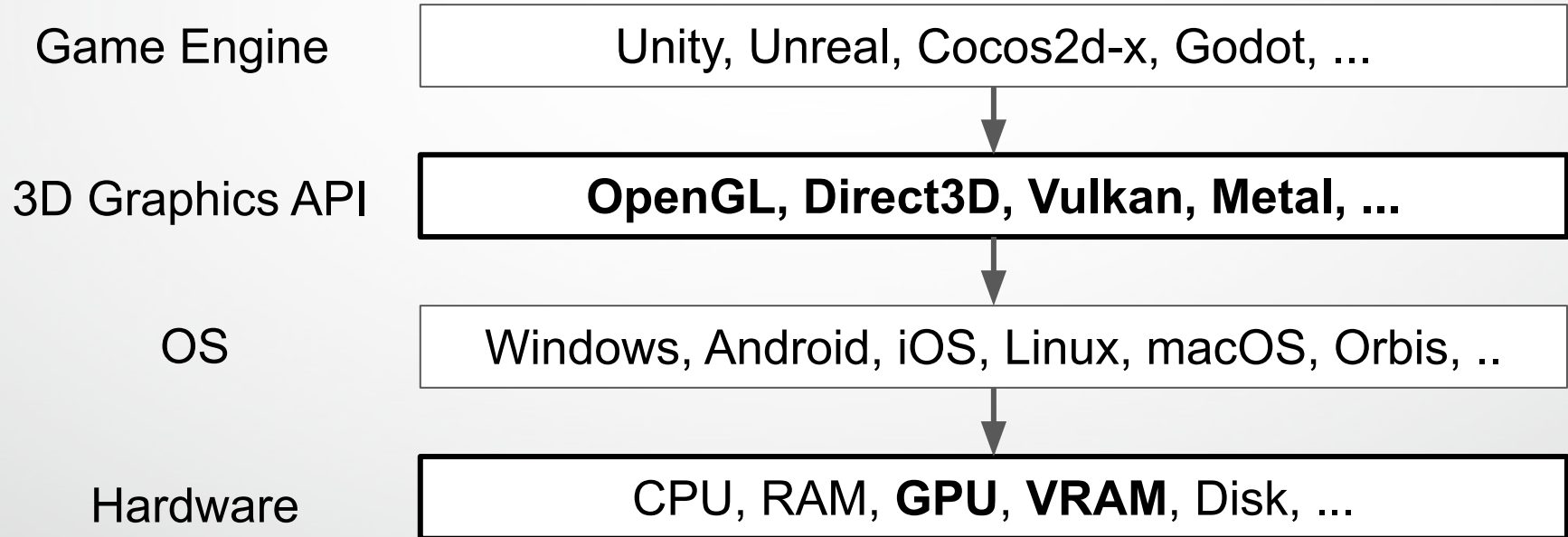
# Shader

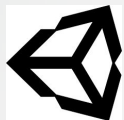


Given :

- Light
- View
- Normal
- BRDF parameters
- Position
- UVs
- Textures
- ...

# Graphics hardware





# Writing shaders



## Unity Manual

- + Unity User Manual (2019.4 LTS)
- + Packages
  - New in Unity 2019
- + Working in Unity
- + Importing
- + Input
- + 2D
- Graphics
  - + Render pipelines
  - + Cameras
    - Post-processing
  - + Lighting
  - Meshes, Materials, Shaders and Textures
    - + Mesh Components
    - + Creating and Using Materials
    - + Textures
    - **Writing Shaders**
      - + Standard Shader
        - Standard Particle Shaders

[Unity User Manual \(2019.4 LTS\)](#) / [Graphics](#) / [Meshes, Materials, Shaders and Textures](#) / [Writing Shaders](#)



## 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

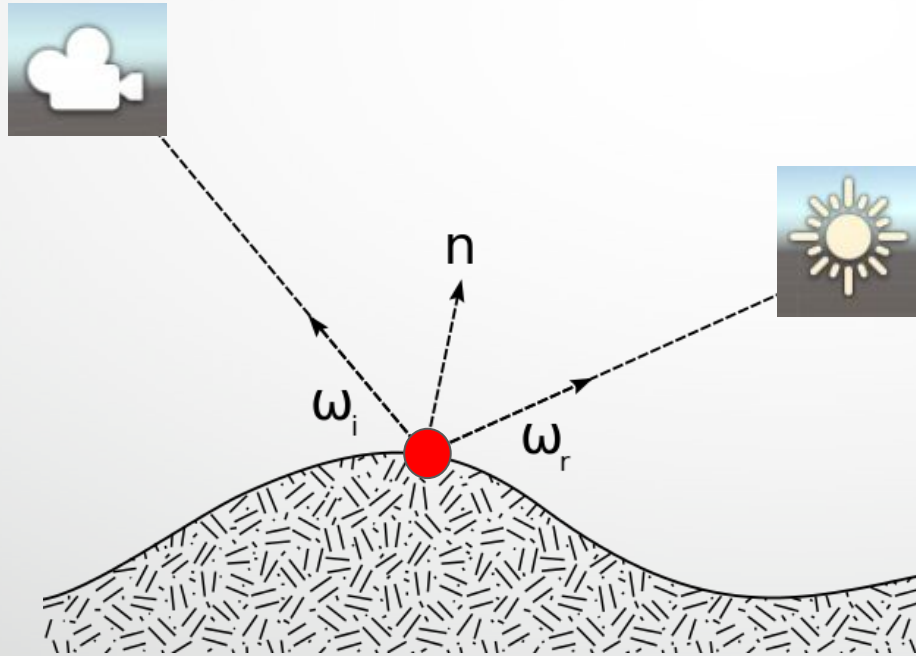
Fixed Function Shaders are legacy Shader syntax for very simple effects. It is advisable to write programmable Shaders, since that allows much more flexibility. Fixed



# Shader Graph

graphs/TextureDissolve master node preview

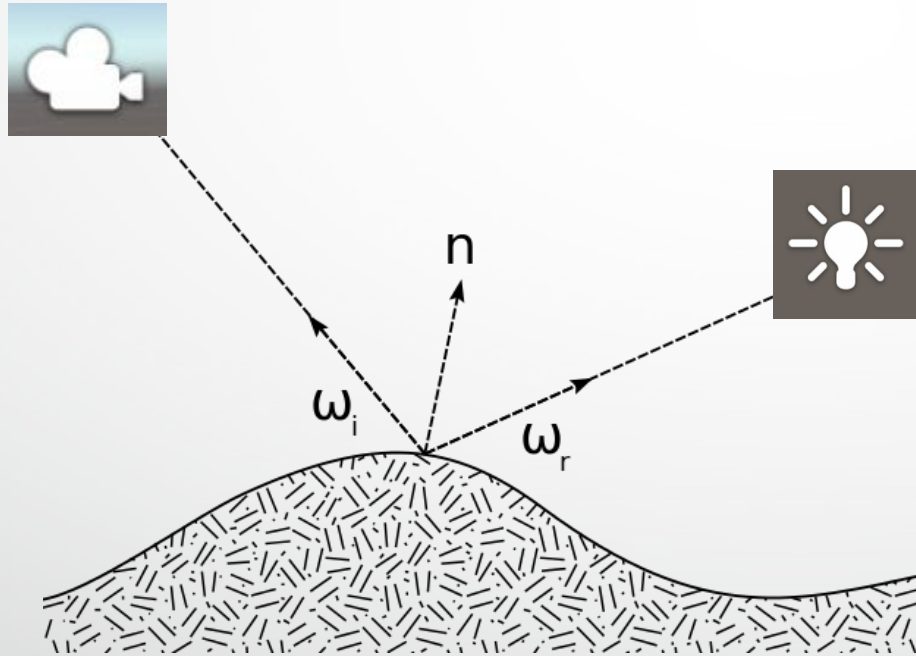
# Shader



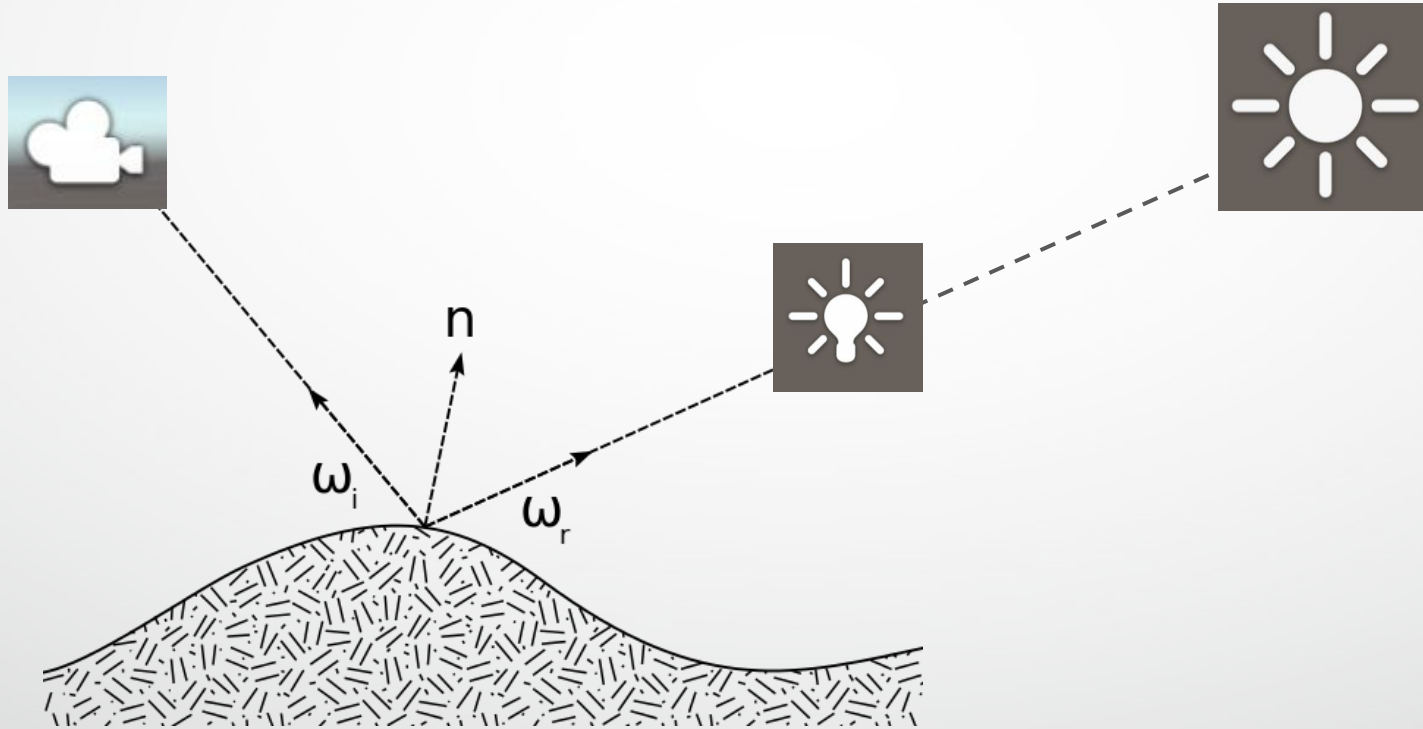
Considers :

- Light
- View
- Normal
- BRDF
- Position
- UVs
- Textures
- ...

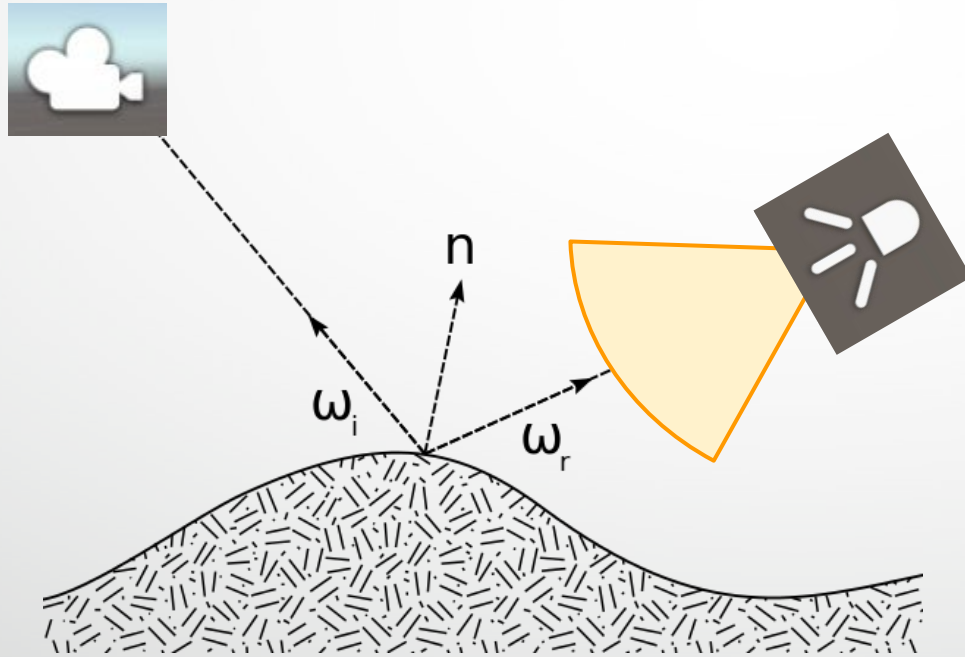
# Point light



# Point light / Directional light

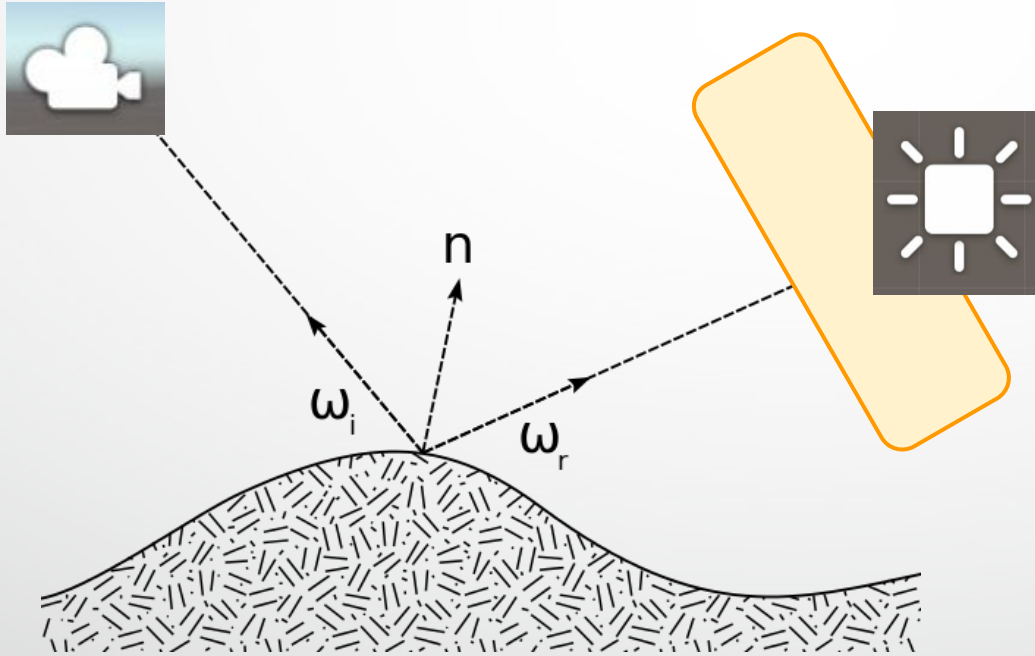


# Spot light

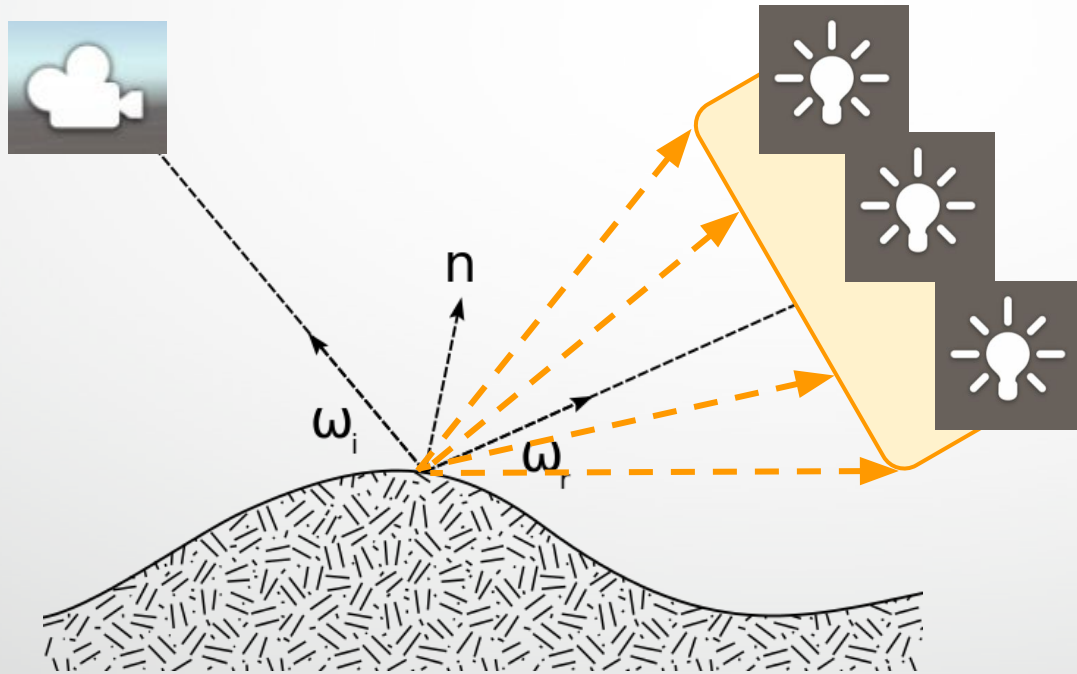




# Area light ?

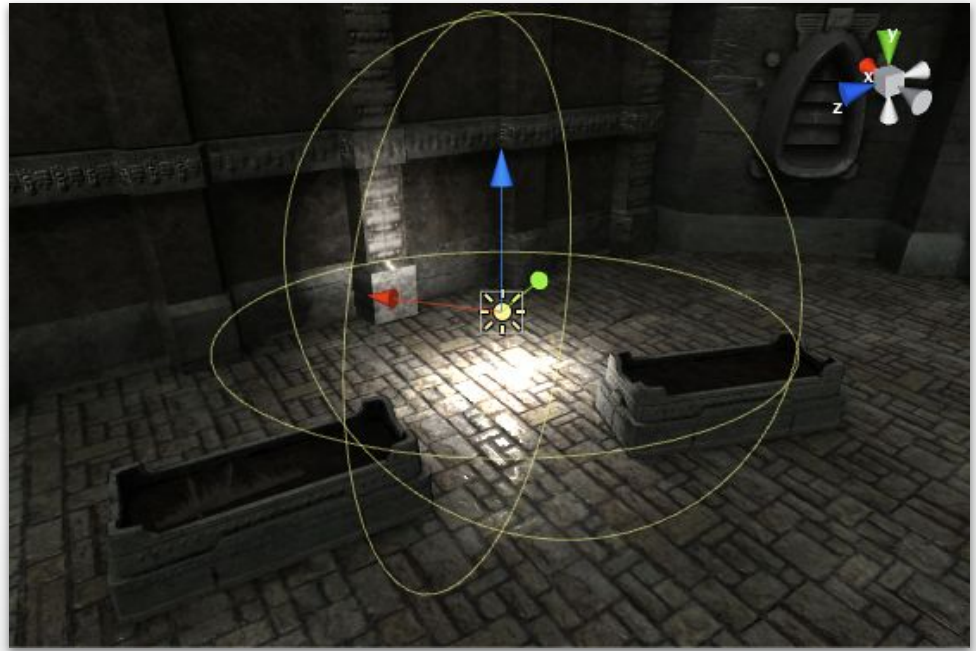
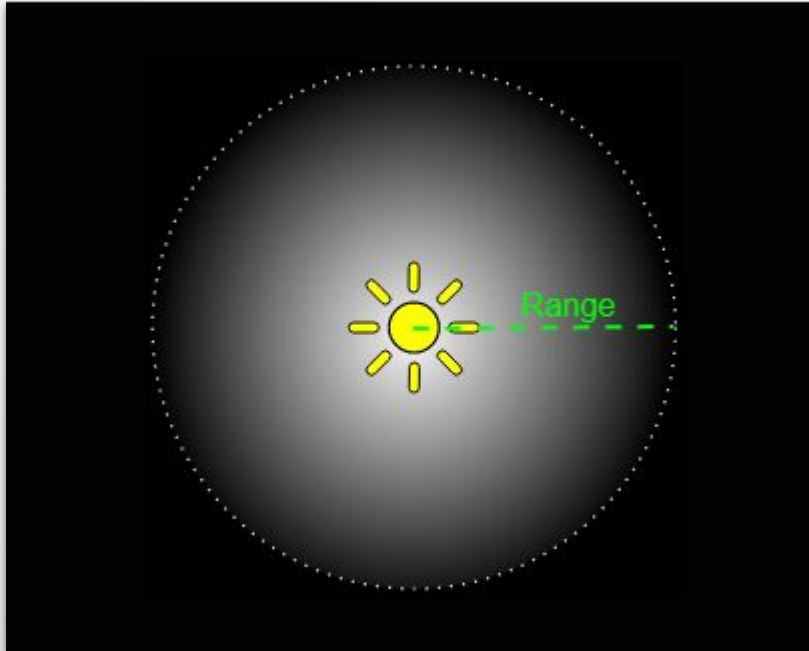


# Lots of point lights ? integral ?

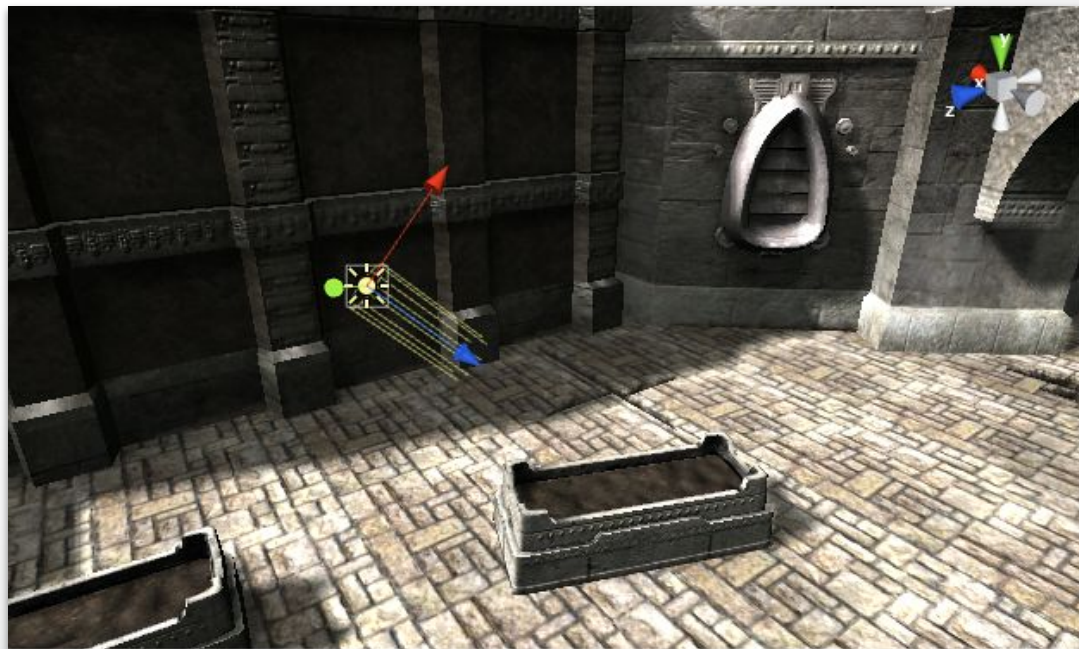
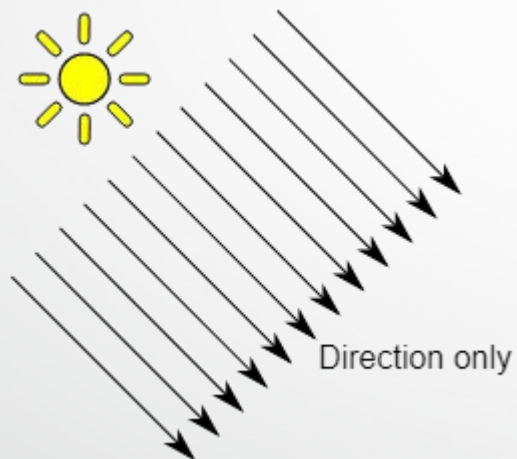




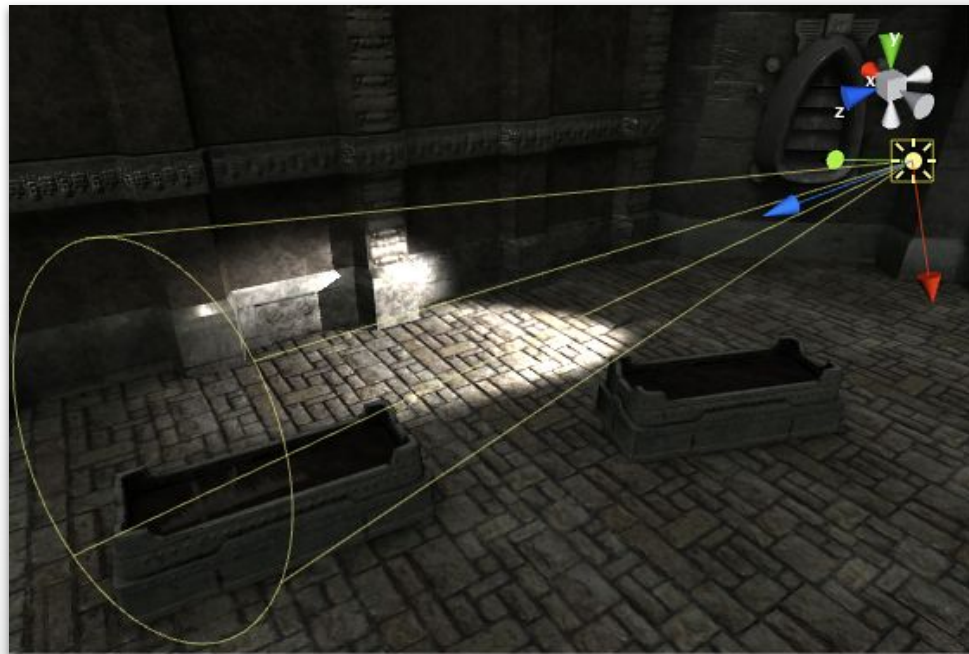
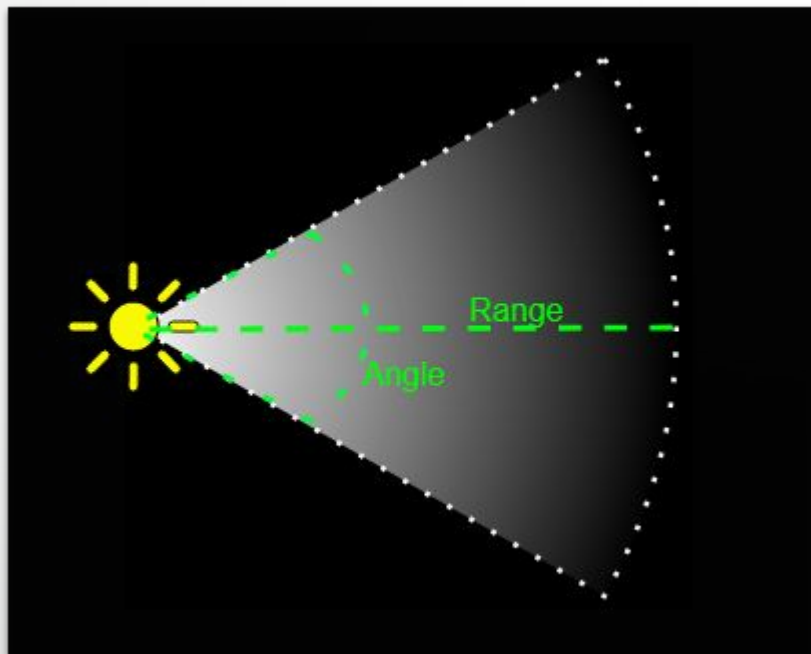
# Point light



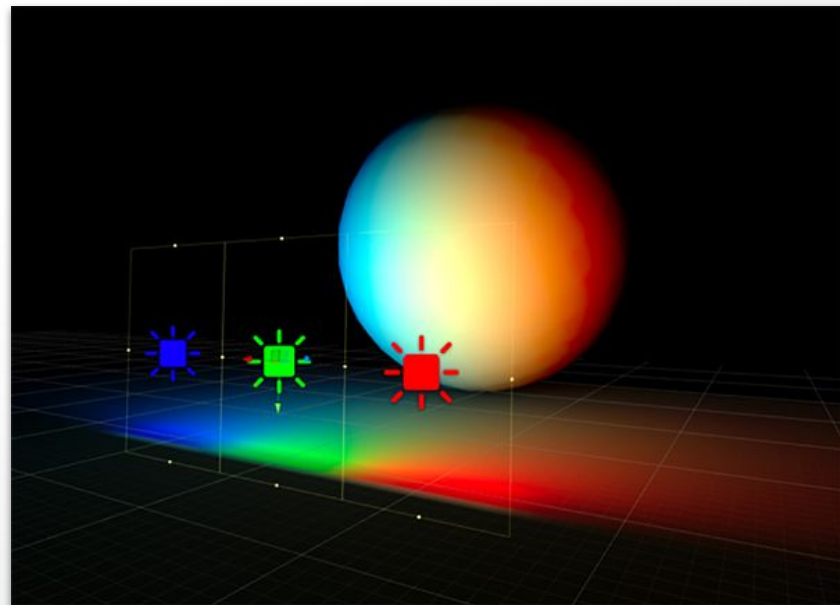
# Directional light



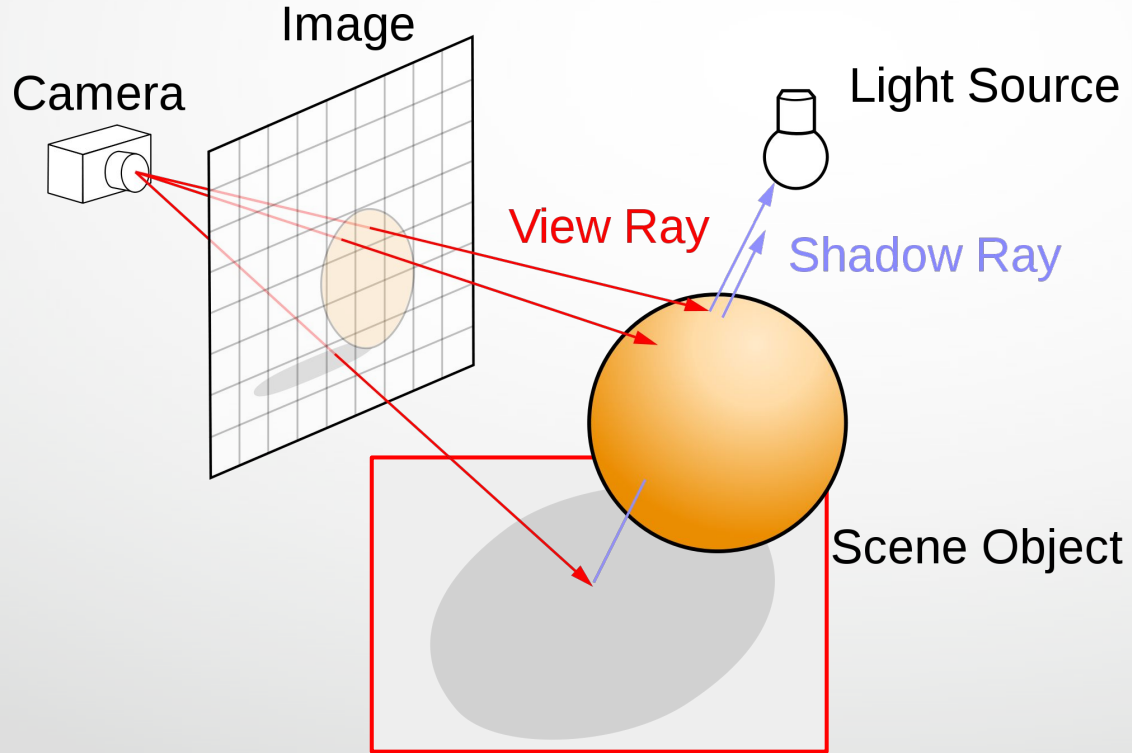
# Spot light



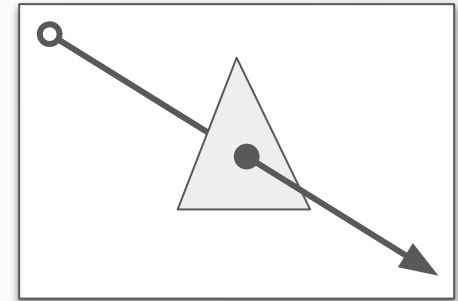
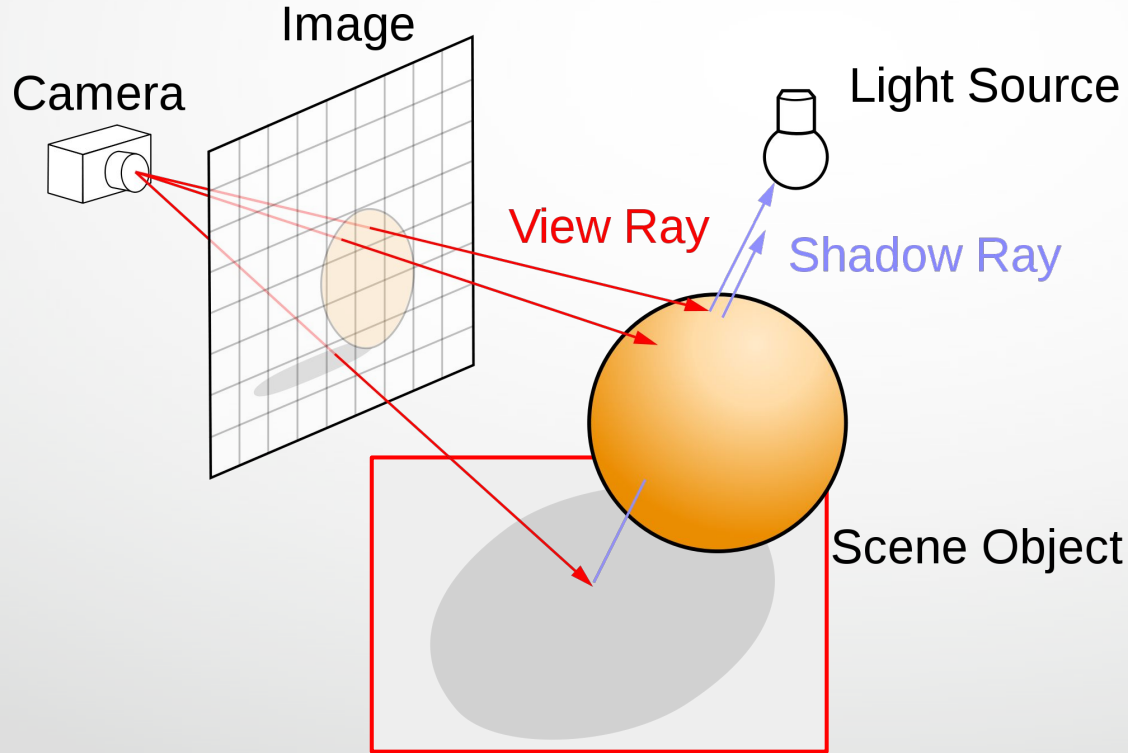
# Area light



# Shadow ?



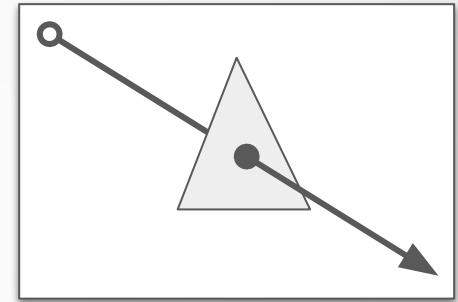
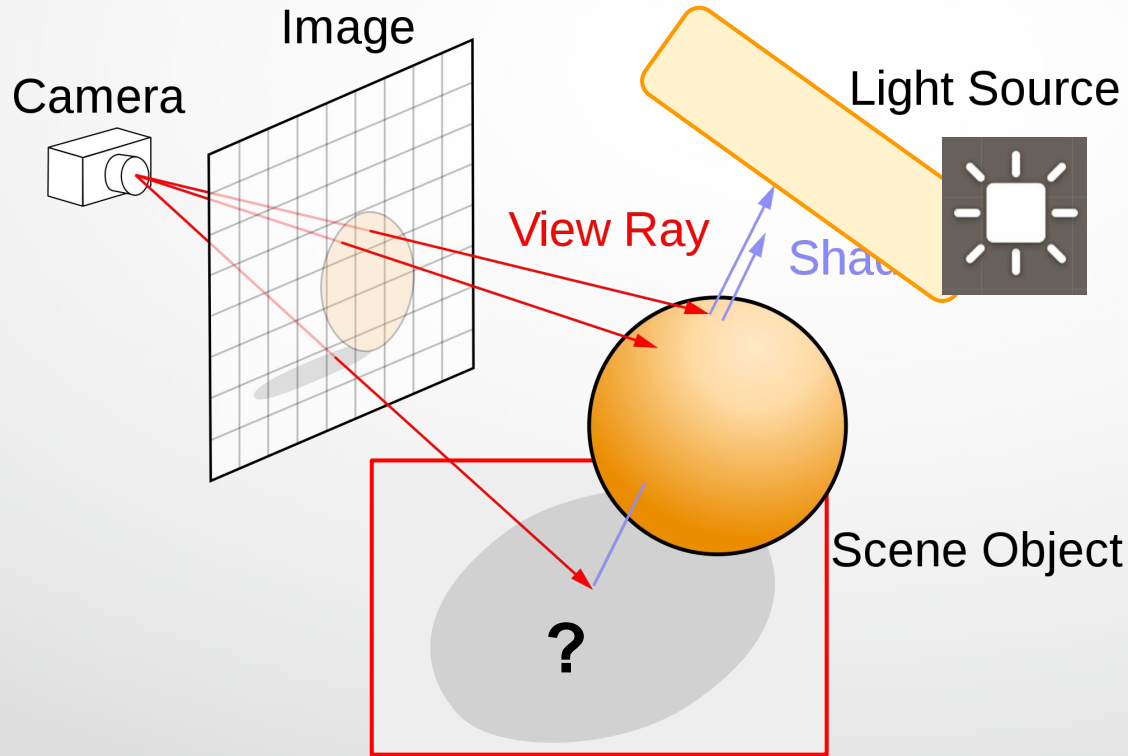
# Shadow ?



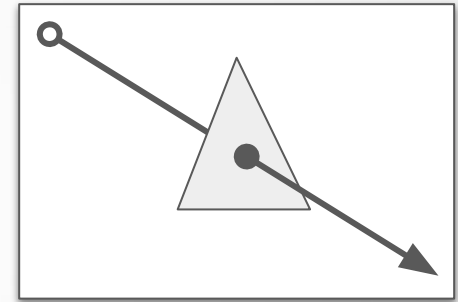
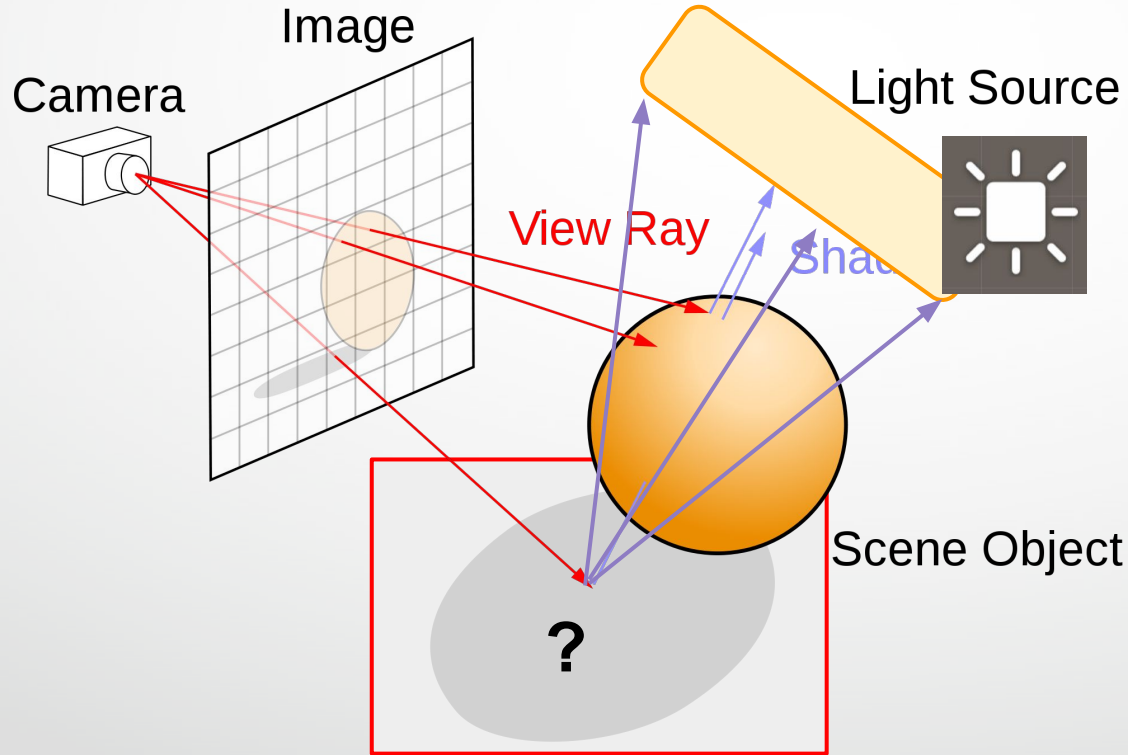
**shadow rays**



# Shadow ?

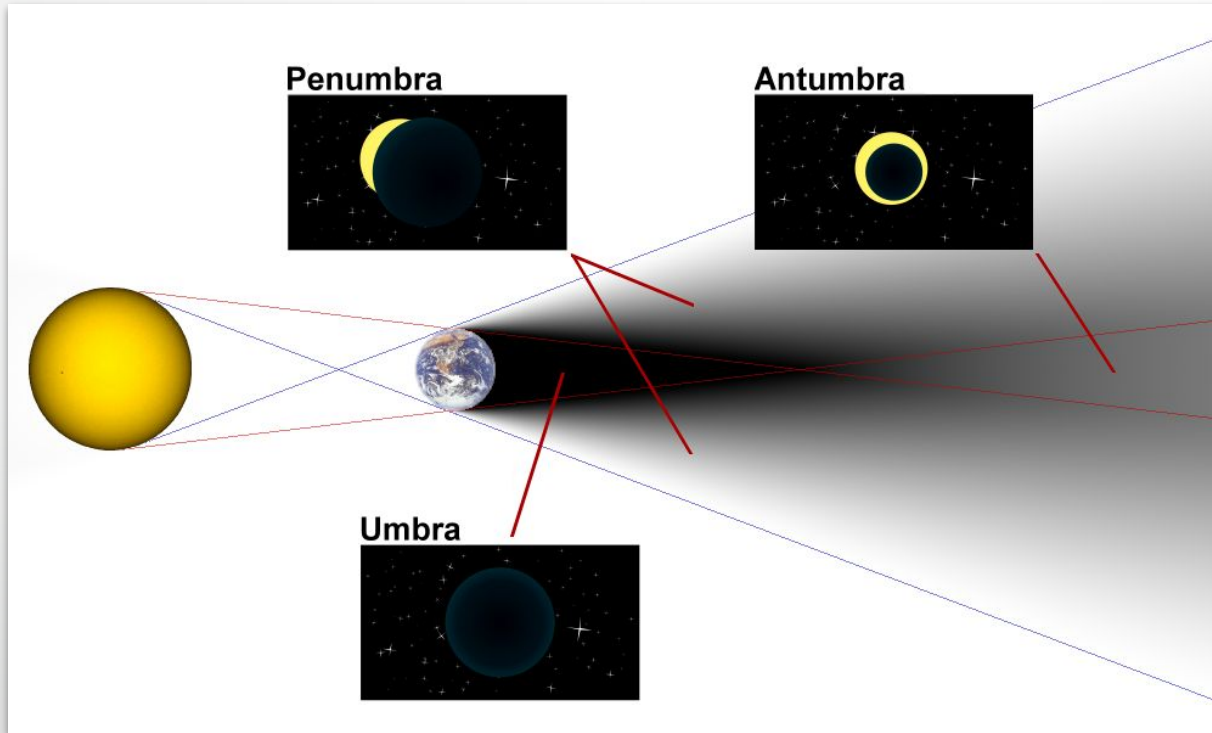


# Shadow ?



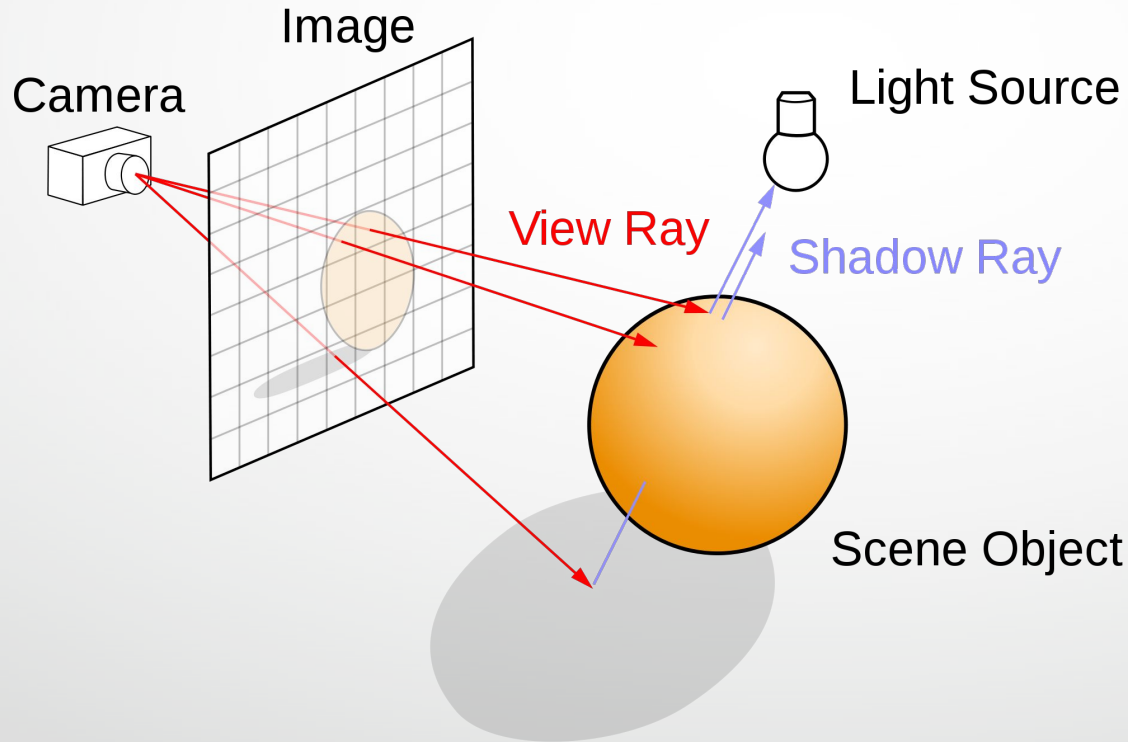
shadow rays ?

# Soft shadows



<https://en.wikipedia.org/wiki/Shadow>

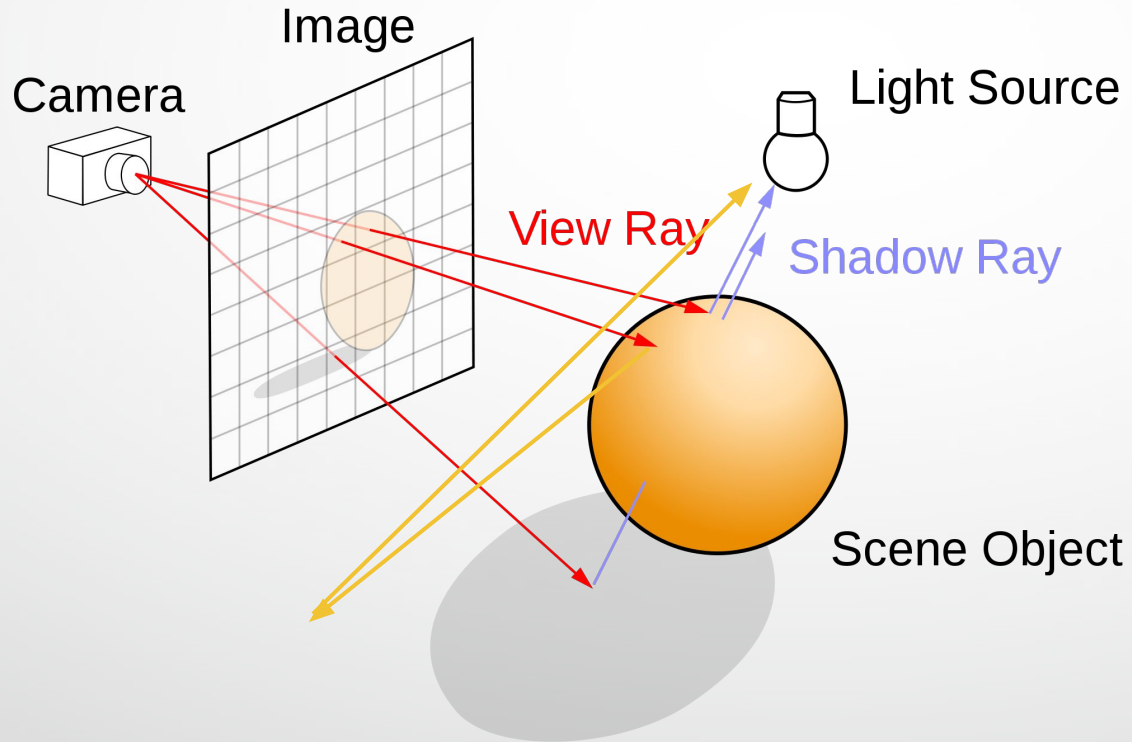
# Direct illumination



**Number of lights**

**Hard / Soft shadows  
(Light type)**

# Indirect illumination →



# Direct vs. indirect illumination



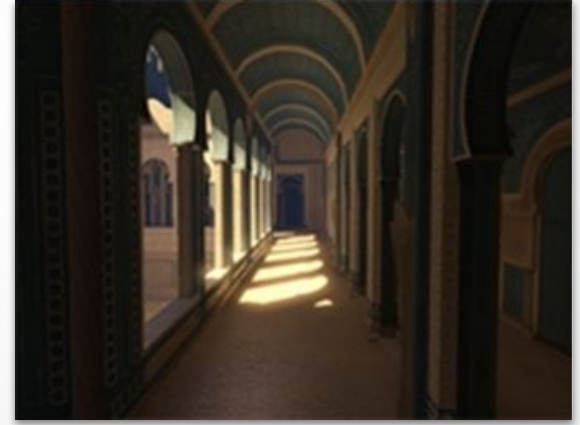
Direct



Indirect

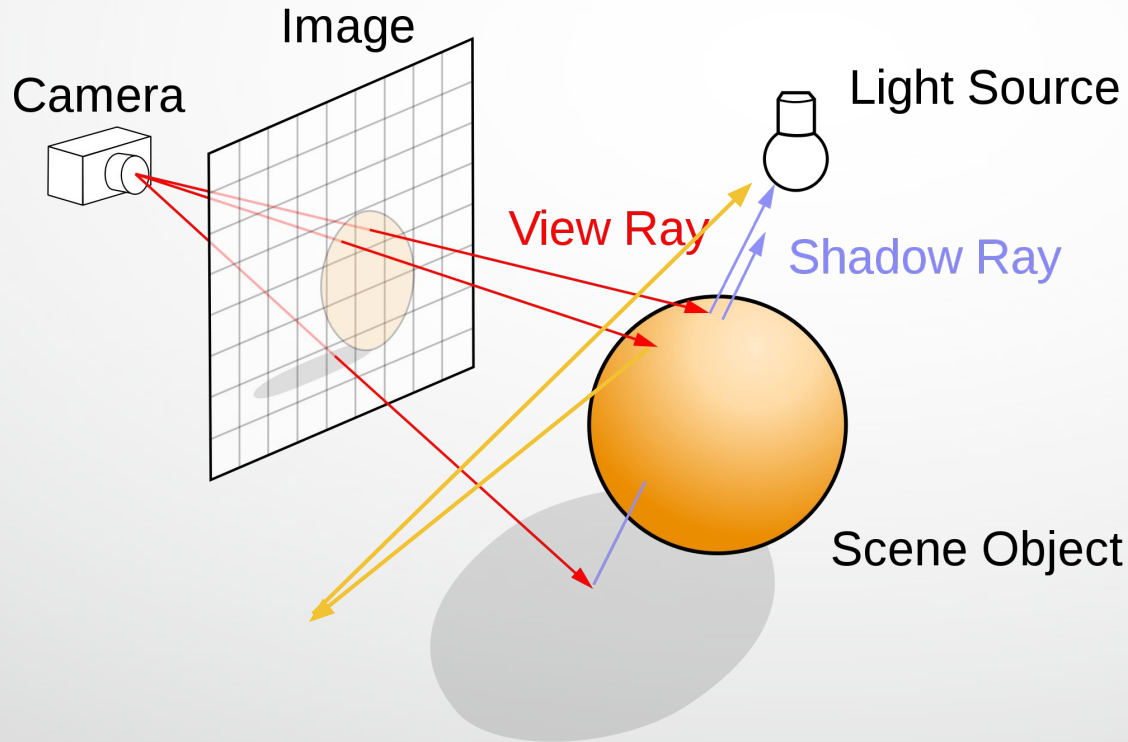
+

=

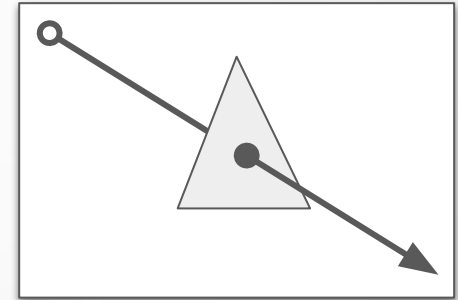
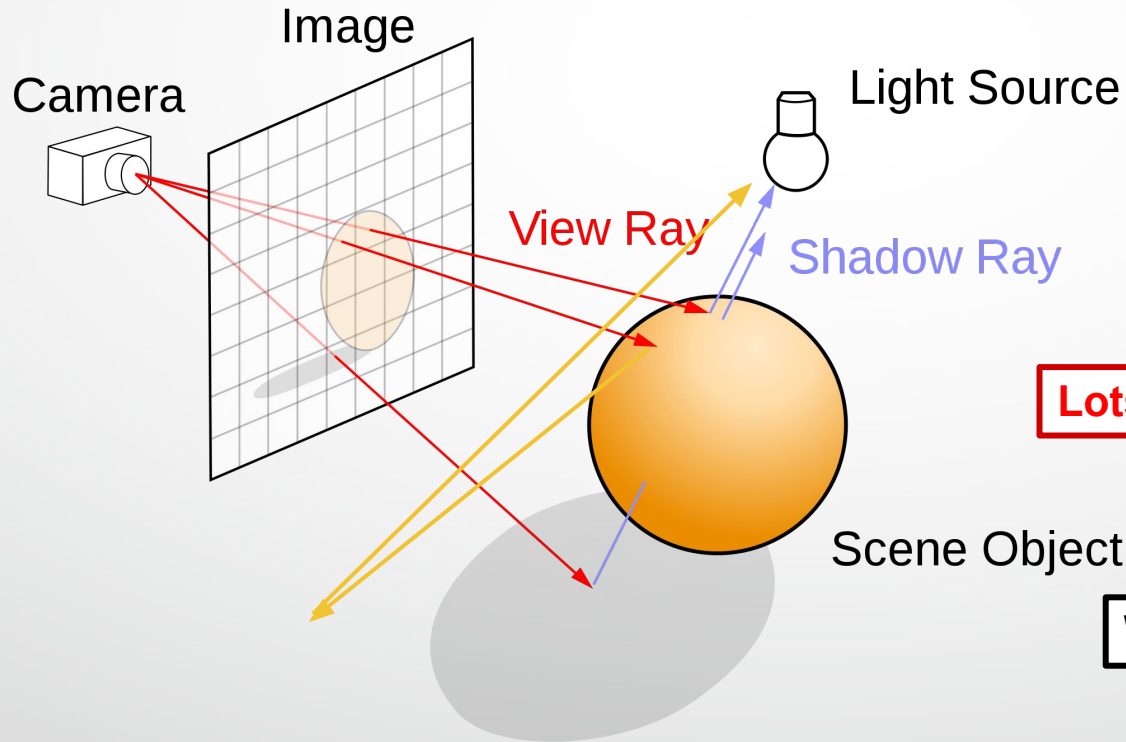


Final

# Global illumination = Indirect illumination



# Global illumination = Indirect illumination

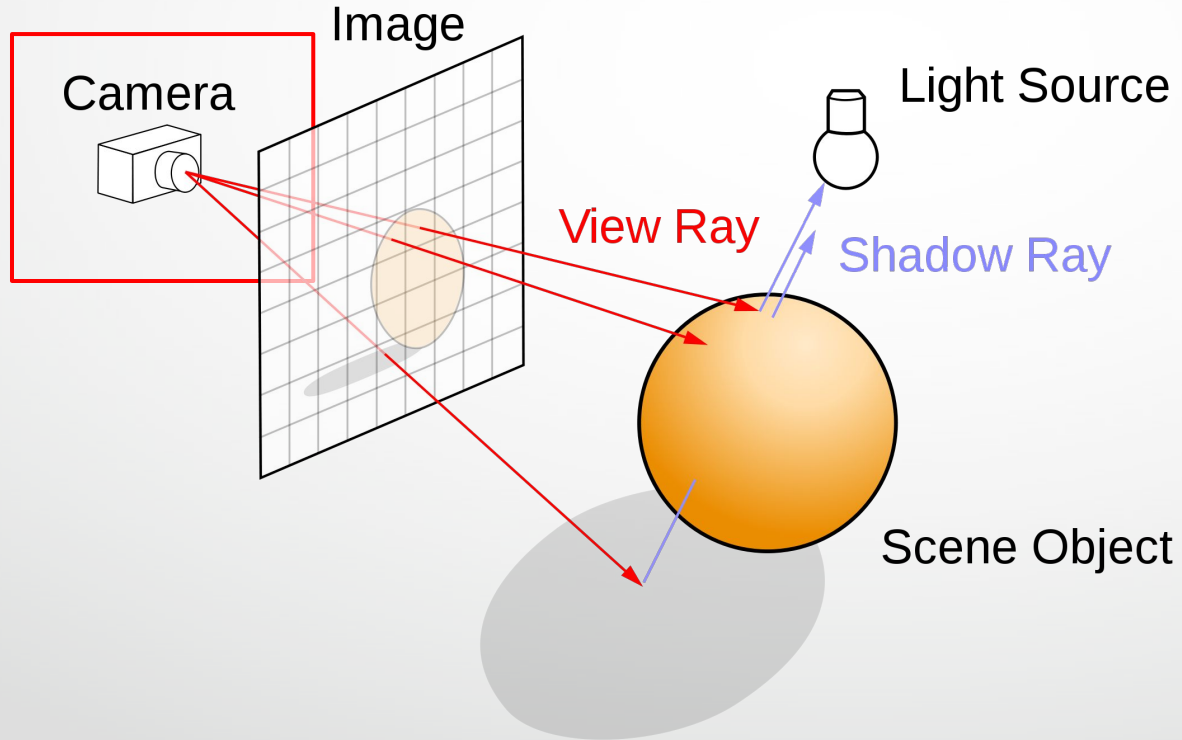


**Lots of ray intersection tests**

**We'll talk about this later.**



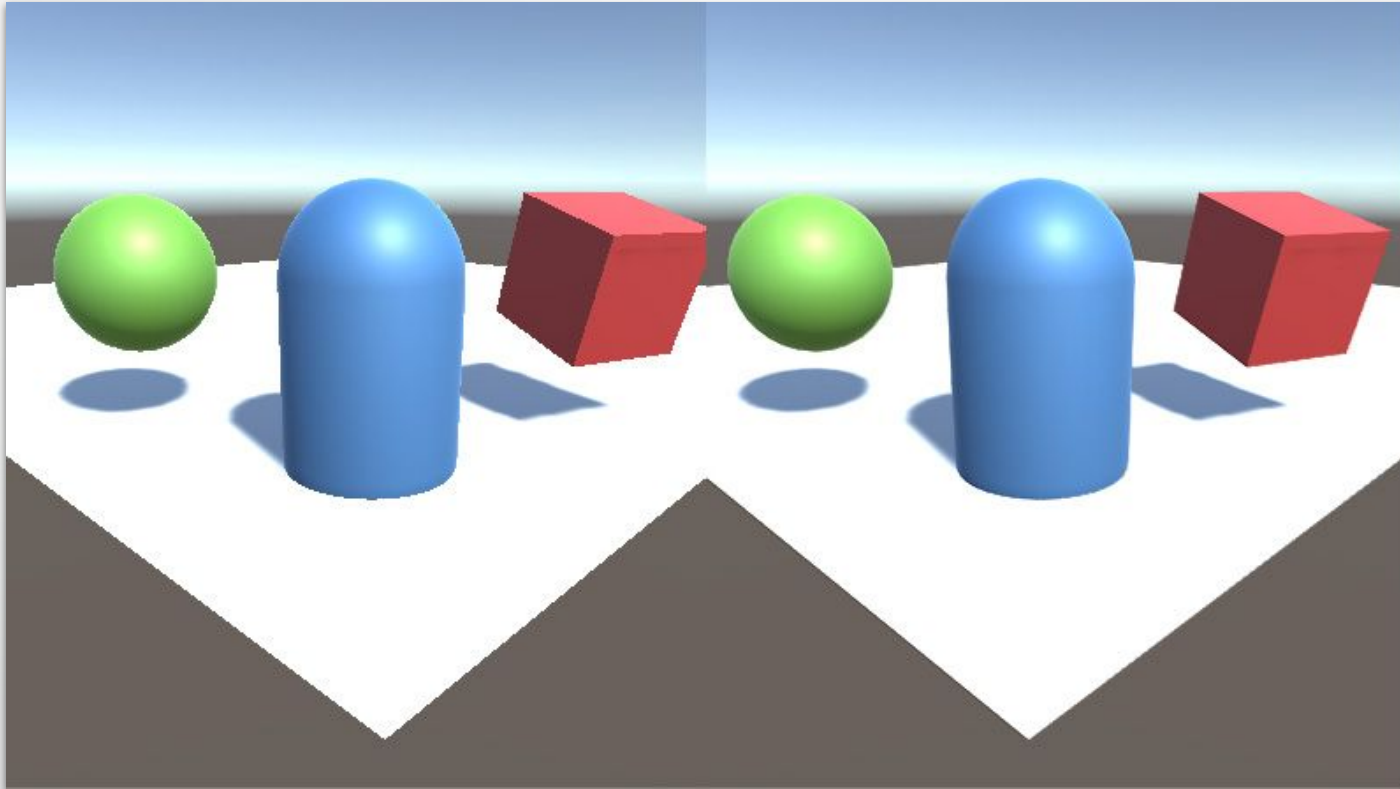
# Camera ?



# Camera

- Anti-aliasing
- Depth of field
- Auto exposure
- White balance
- Color adjustments
- Lens distortion
- Grain
- Motion blur
- HDR (High-dynamic-range)
  - Bloom
  - Tonemapping
- ...

# Anti-aliasing

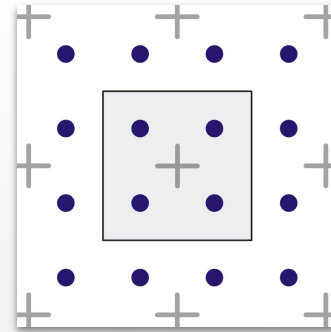


# Anti-aliasing (cont'd)

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



[https://en.wikipedia.org/wiki/Fast\\_approximate\\_anti-aliasing](https://en.wikipedia.org/wiki/Fast_approximate_anti-aliasing)



<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

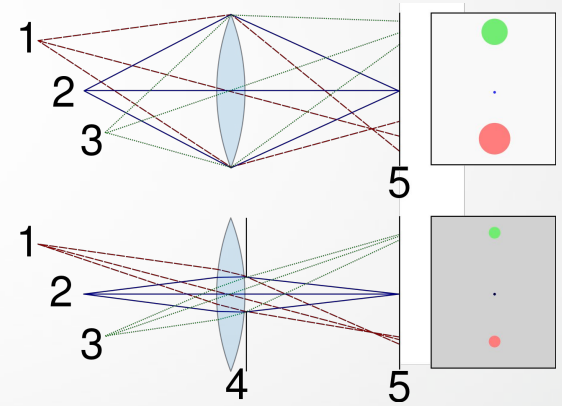


[https://en.wikipedia.org/wiki/Fast\\_approximate\\_anti-aliasing](https://en.wikipedia.org/wiki/Fast_approximate_anti-aliasing)

# Depth of Field



[https://en.wikipedia.org/wiki/File:Dof\\_blocks\\_f4\\_0.jpg](https://en.wikipedia.org/wiki/File:Dof_blocks_f4_0.jpg)



- 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\\_\(shader\\_effect\)](https://en.wikipedia.org/wiki/Bloom_(shader_effect))





# Post Processing (before)



<https://docs.unity3d.com/Manual/PostProcessingOverview.html>



# Post Processing (after)



<https://docs.unity3d.com/Manual/PostProcessingOverview.html>



# Post Processing (package for Built-in RP)

[Manual](#)[Scripting API](#)[Changelog](#)[License](#)[docs.unity3d.com](https://docs.unity3d.com) →

Post Processing 2.3.0 v

[Home](#)[Installation](#)[Quick-start](#)

## - Effects

- Ambient Occlusion
- Anti-aliasing
- Auto Exposure
- Bloom
- Chromatic Aberration
- Color Grading
- Deferred Fog
- Depth of Field
- Grain
- Lens Distortion
- Motion Blur
- Screen Space Reflections
- Vignette

## + Scripting

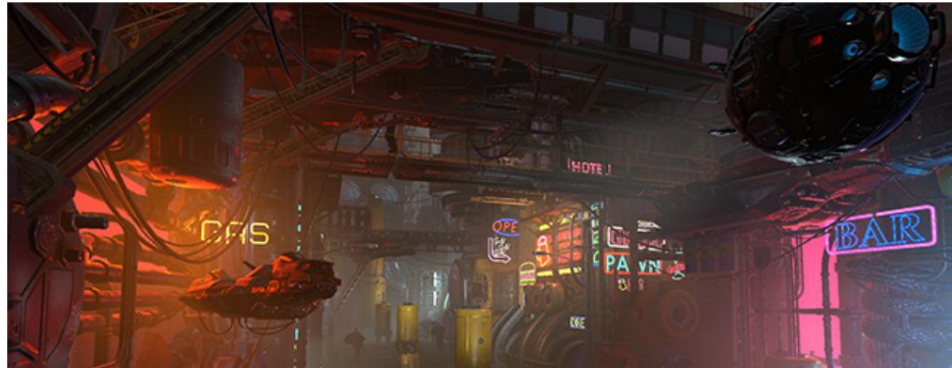
[Manual](#) / [Home](#)

## Home

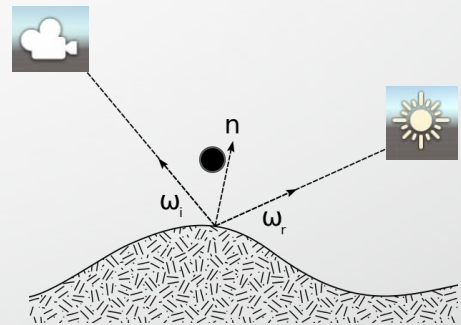
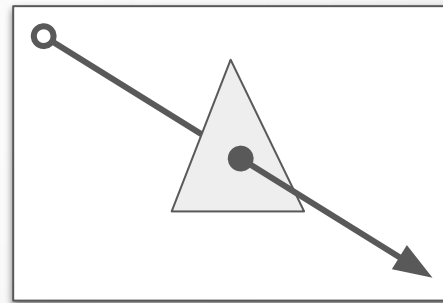
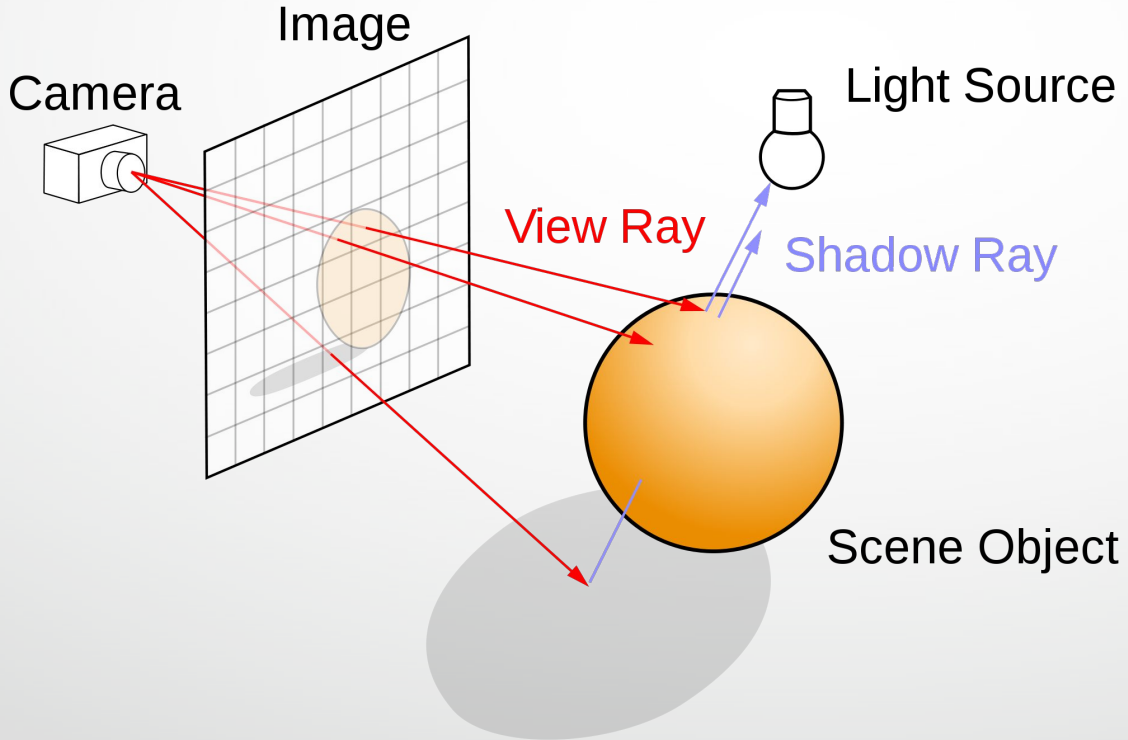
**Post-processing** is the process of applying full-screen filters and effects to a camera's image buffer before it is displayed to screen. It can drastically improve the visuals of your product with little setup time.

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

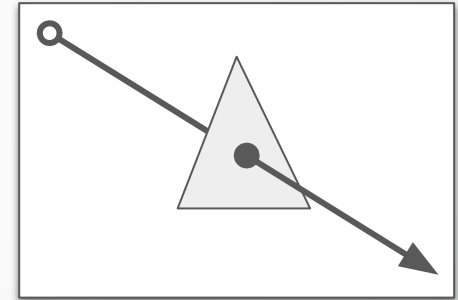
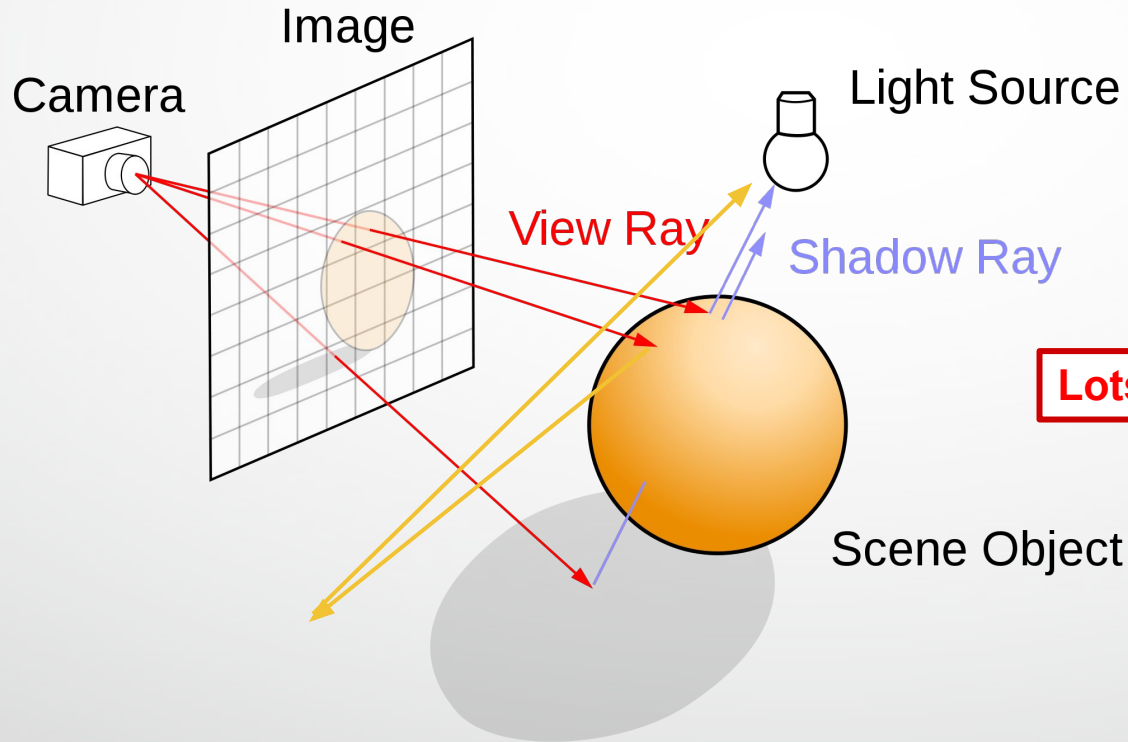
The images below demonstrate a scene with and without post-processing.



# Ray tracing

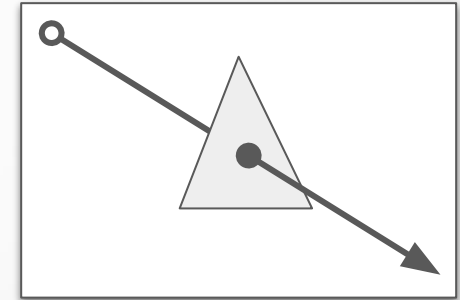
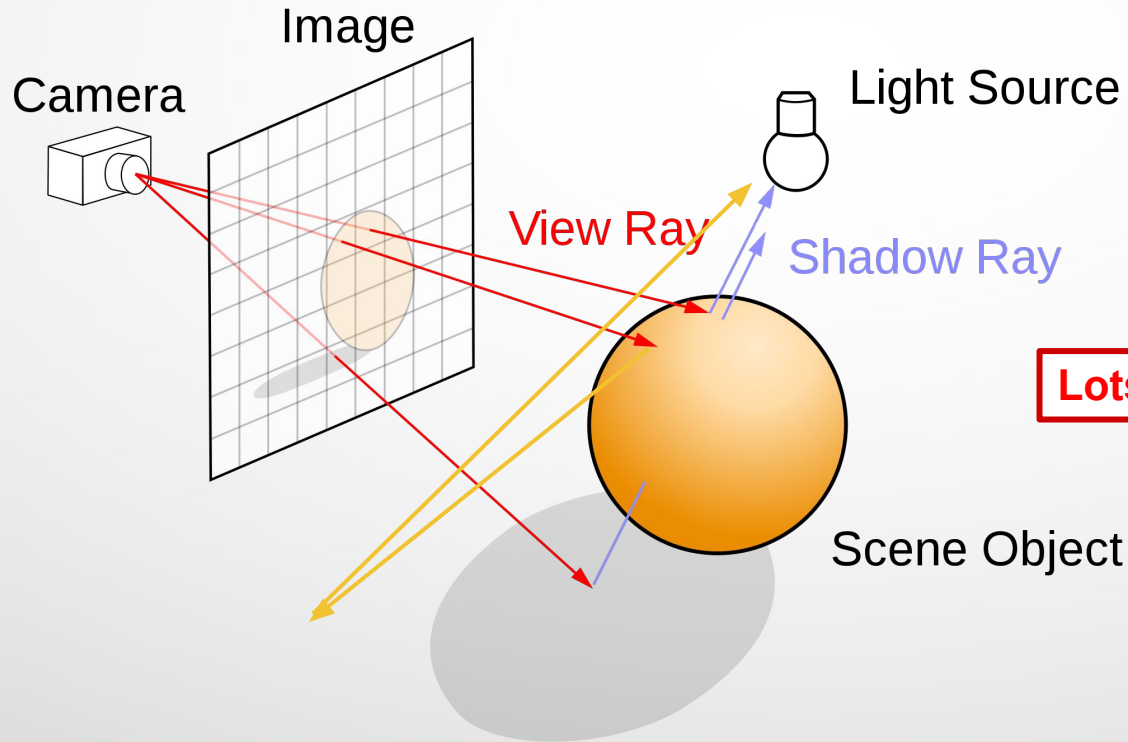


# Global illumination = Indirect illumination



**Lots of ray intersection tests**

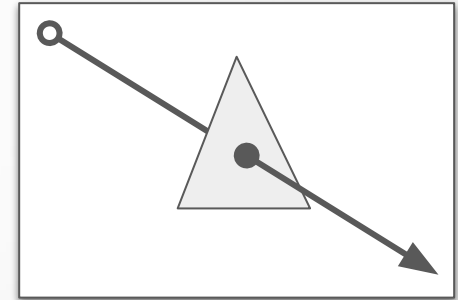
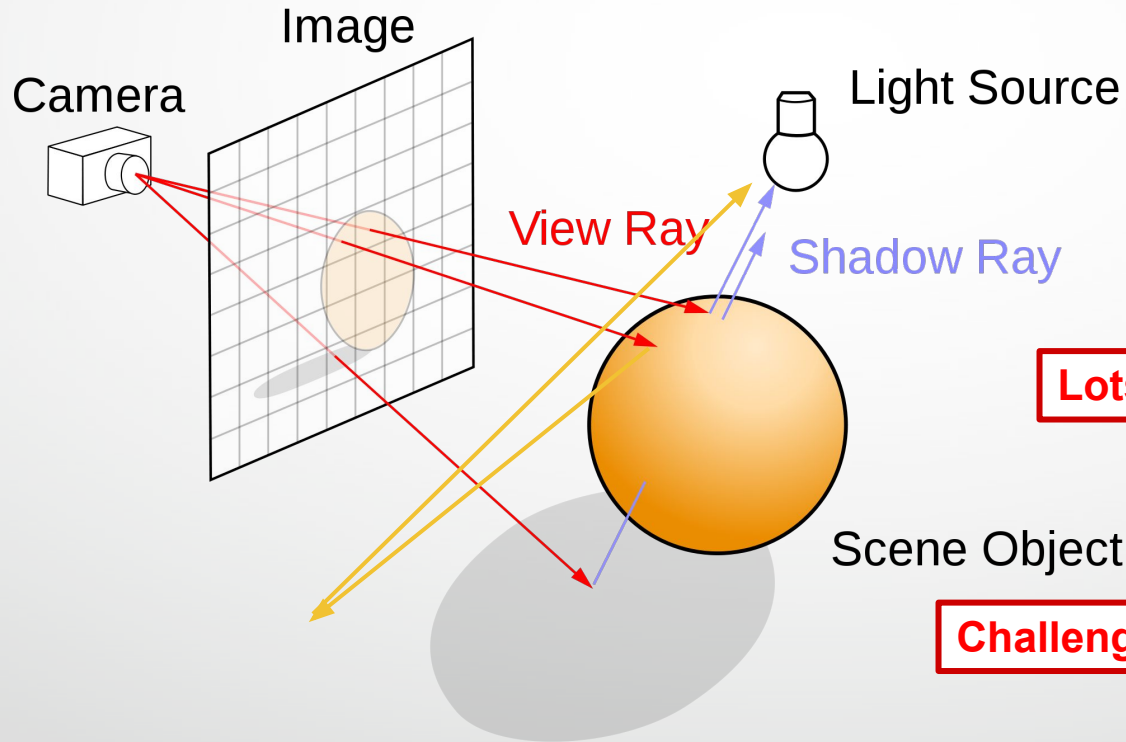
# Global illumination = Indirect illumination



**Lots of ray intersection tests**

**hardware acceleration ?**

# Global illumination = Indirect illumination

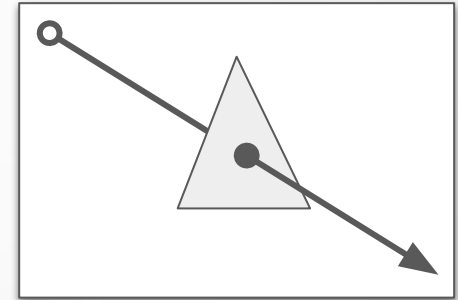
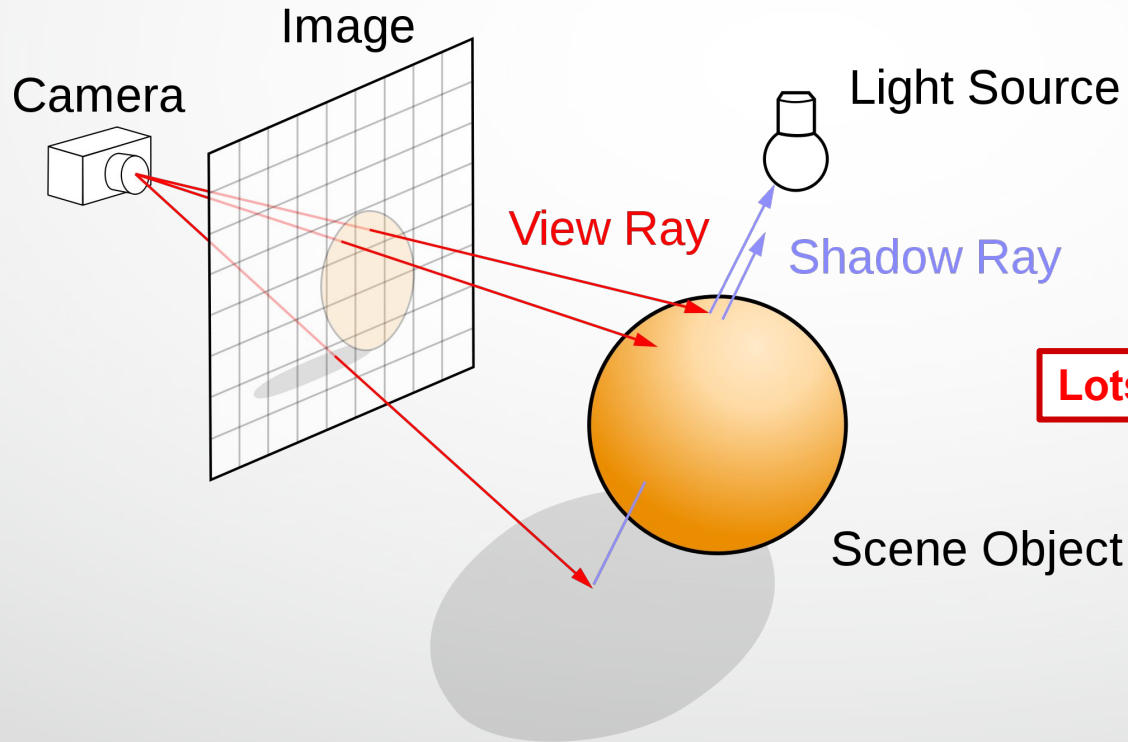


**Lots of ray intersection tests**

**hardware acceleration ?**

**Challenge: divergence of ray paths**

# Direct illumination

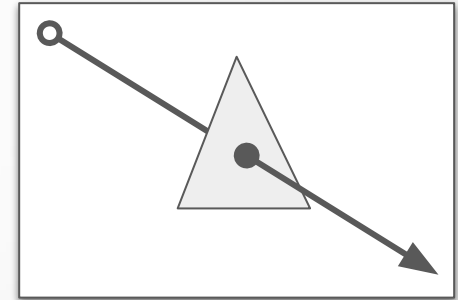
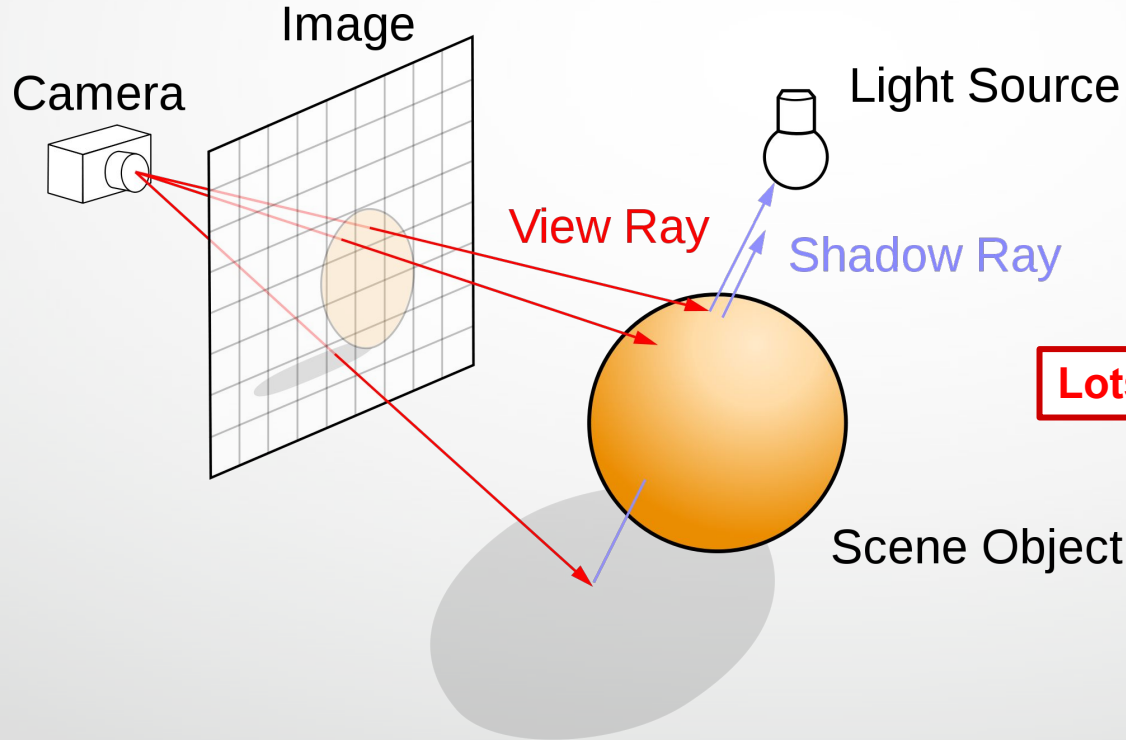


**Lots of ray intersection tests**

**hardware acceleration ?**



# Direct illumination

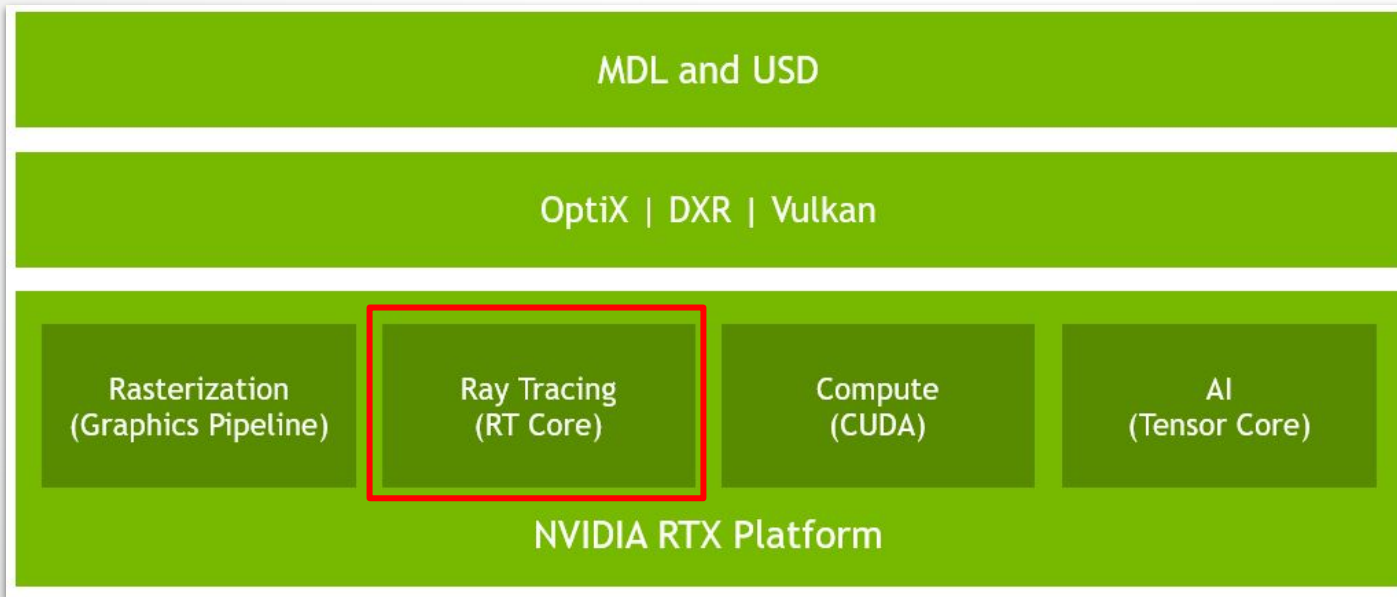


**Lots of ray intersection tests**

**hardware acceleration ?**

**GPU & VRAM**

# Case study: NVIDIA RTX™ Platform



<https://developer.nvidia.com/rtx>

# Case study: NVIDIA RTX™ Platform

MDL and USD

OptiX | DXR | Vulkan

Rasterization  
(Graphics Pipeline)

Ray Tracing  
(RT Core)

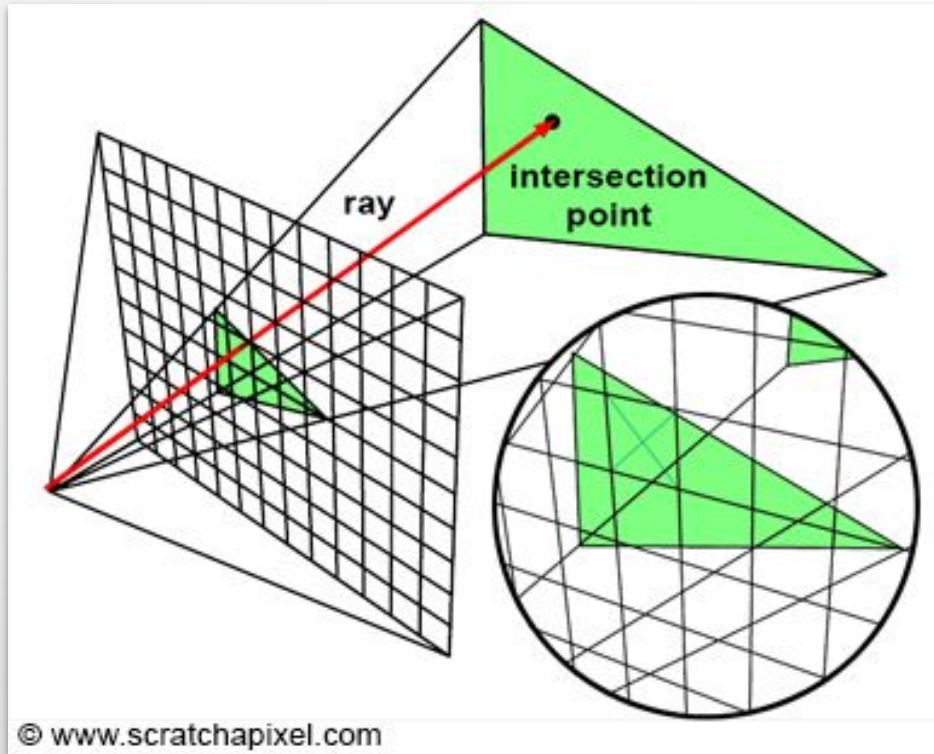
Compute  
(CUDA)

AI  
(Tensor Core)

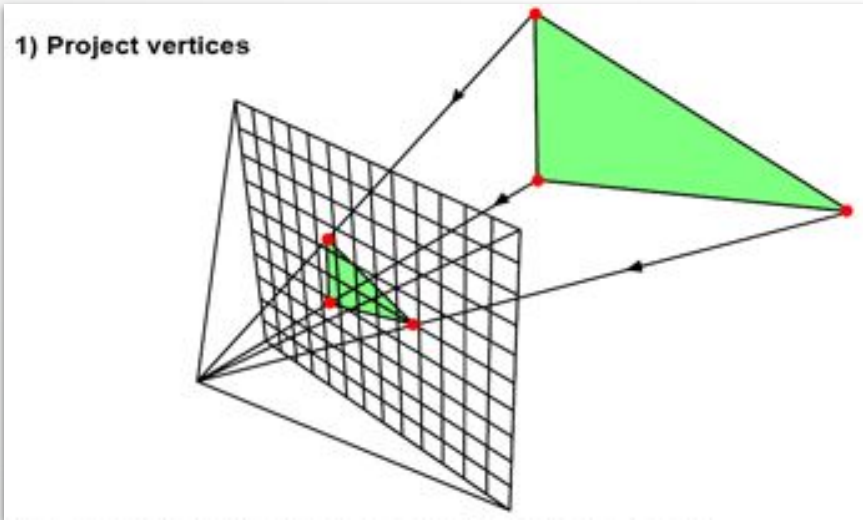
NVIDIA RTX Platform

<https://developer.nvidia.com/rtx>

# Rasterization

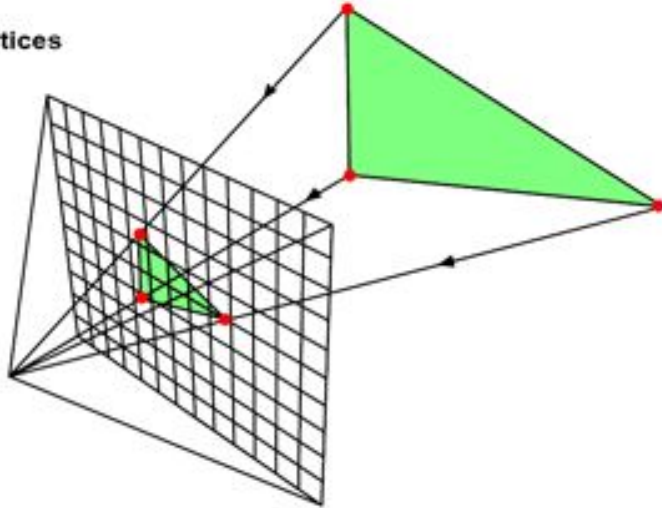


# Rasterization (cont'd)

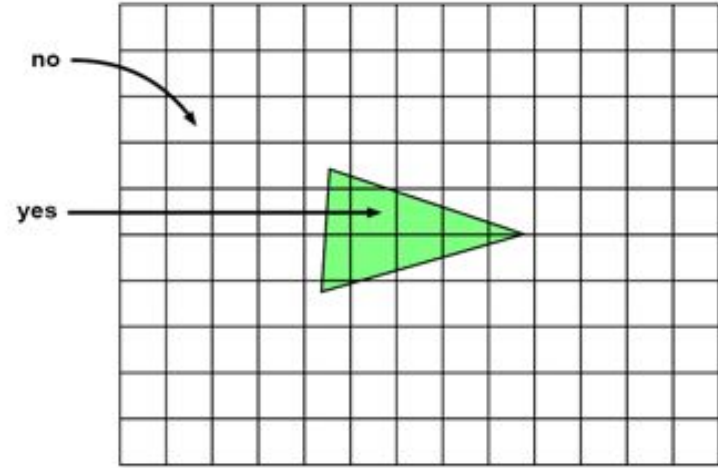


# Rasterization (cont'd)

1) Project vertices

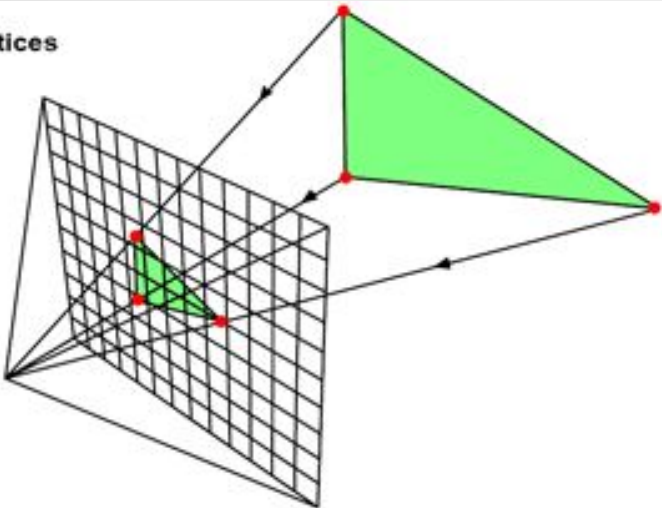


2) Loop over pixels. Does the pixel lie in the triangle?

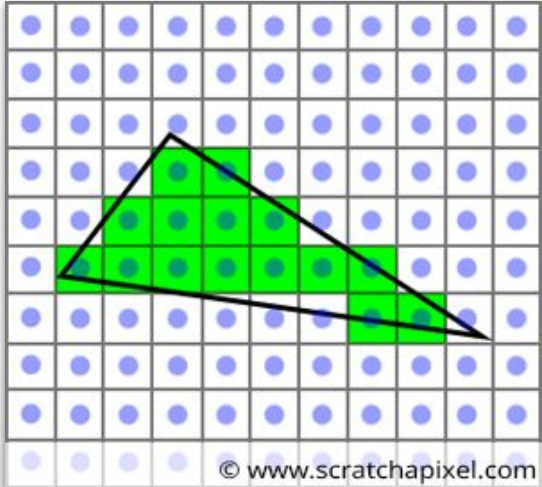
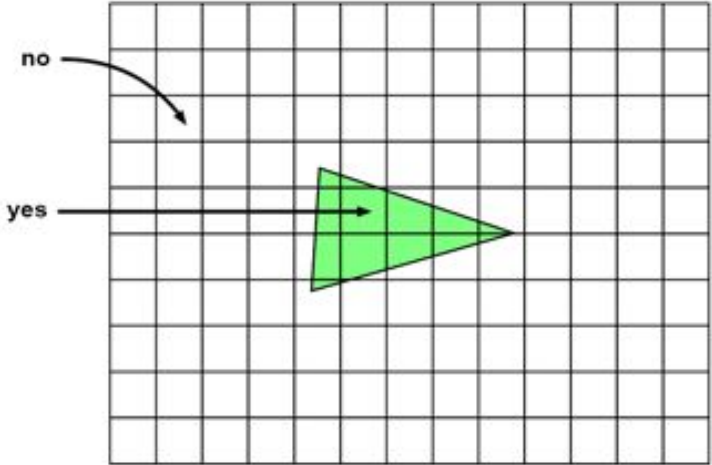


# Rasterization (cont'd)

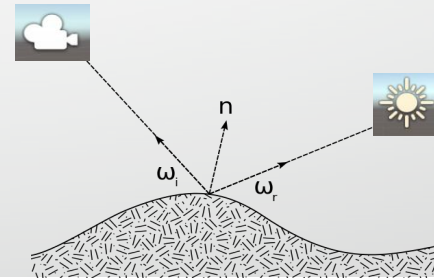
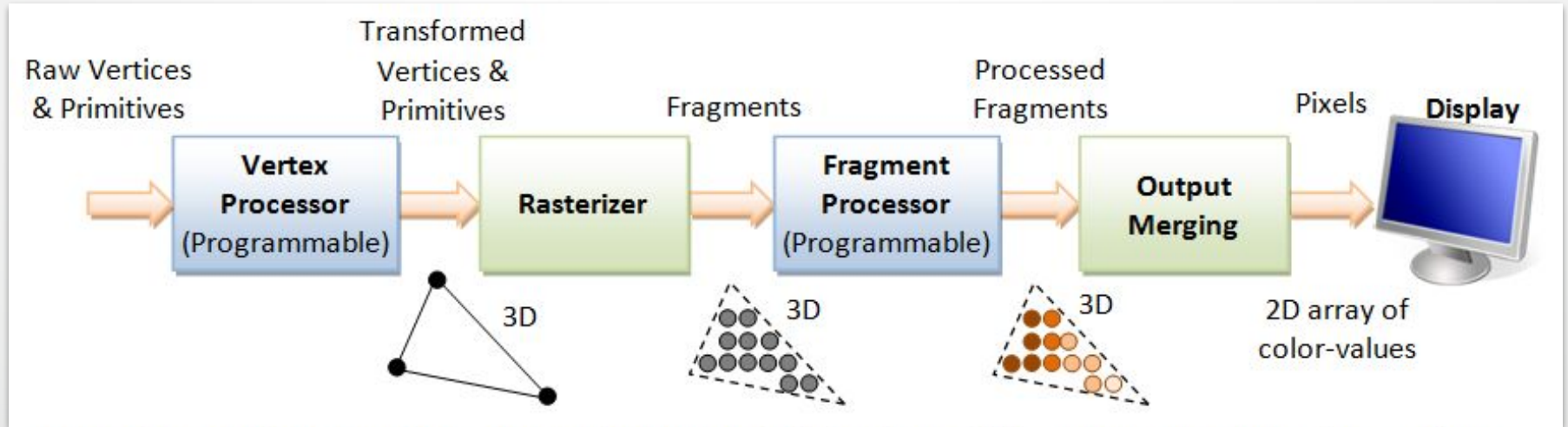
1) Project vertices



2) Loop over pixels. Does the pixel lie in the triangle?

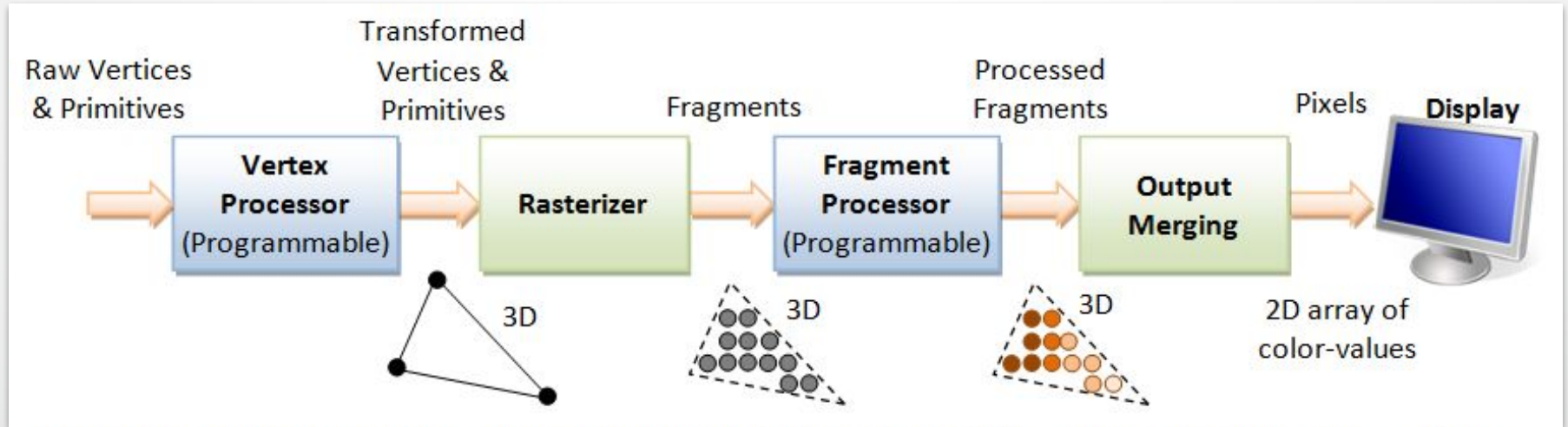


# 3D Graphics Rendering Pipeline

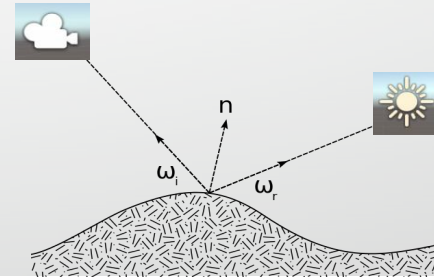




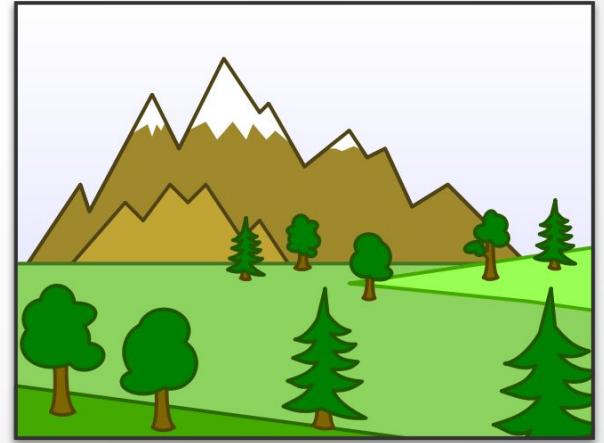
# 3D Graphics Rendering Pipeline



**Which triangle/mesh needs to be drawn first ?**

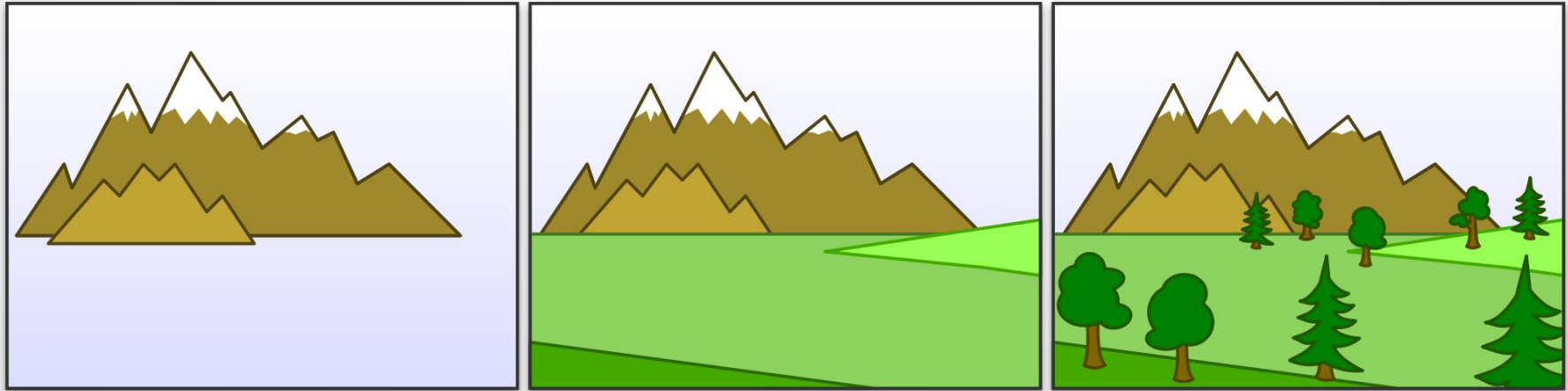


# Painter's algorithm



[https://en.wikipedia.org/wiki/Painter%27s\\_algorithm](https://en.wikipedia.org/wiki/Painter%27s_algorithm)

# Painter's algorithm



[https://en.wikipedia.org/wiki/Painter%27s\\_algorithm](https://en.wikipedia.org/wiki/Painter%27s_algorithm)

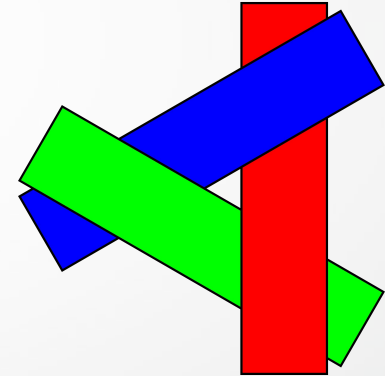
Far

Near

# Painter's algorithm (cont'd)

?

?

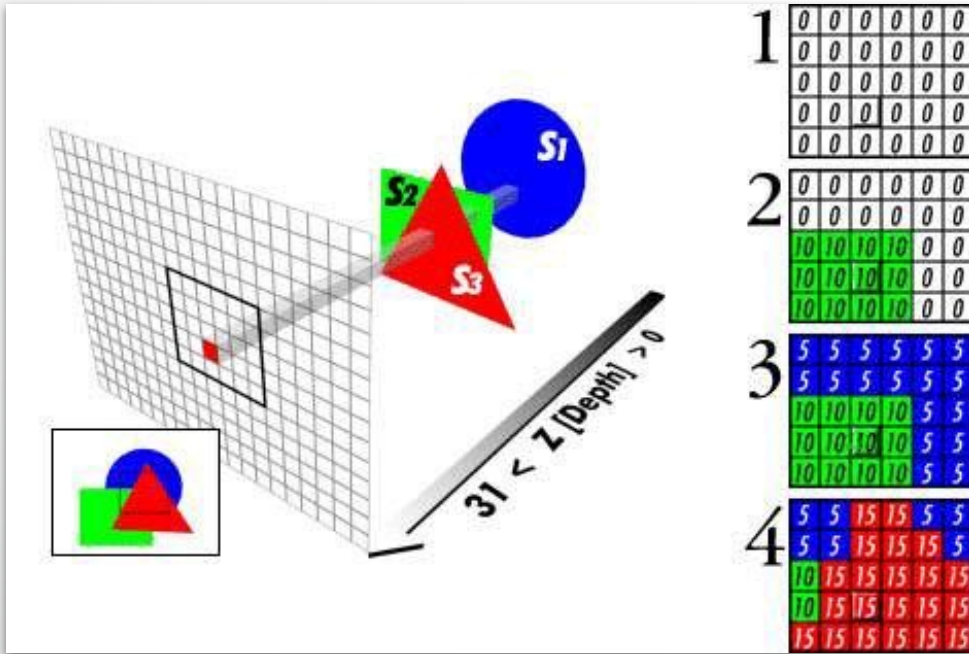


[https://en.wikipedia.org/wiki/Painter%27s\\_algorithm](https://en.wikipedia.org/wiki/Painter%27s_algorithm)

Far

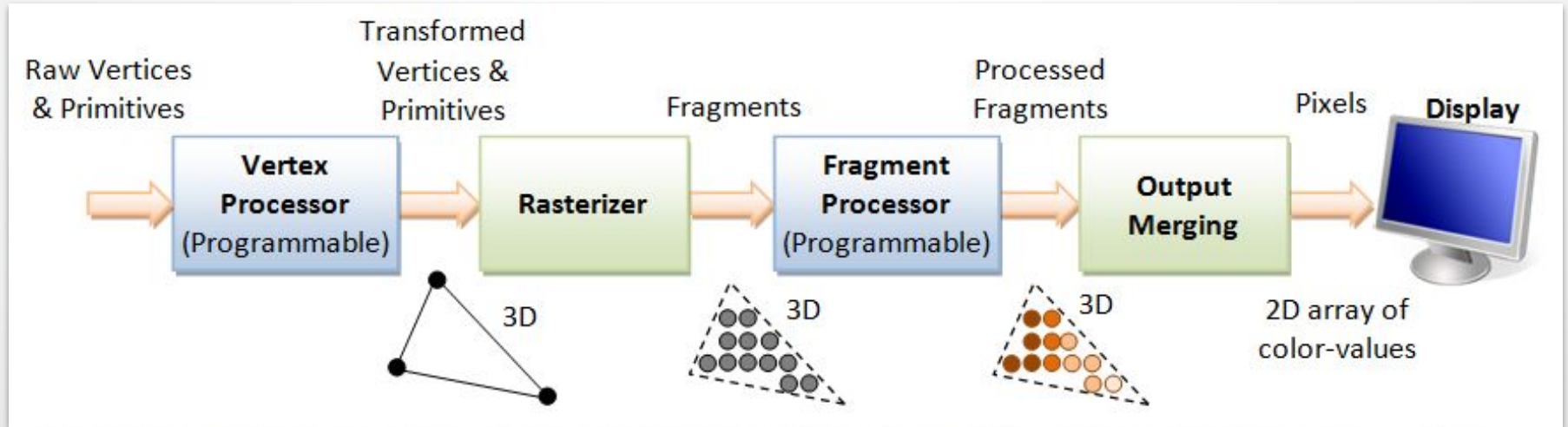
Near

# Z (Depth)-buffering

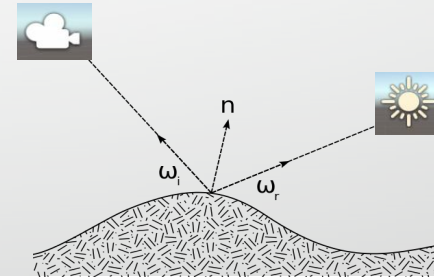
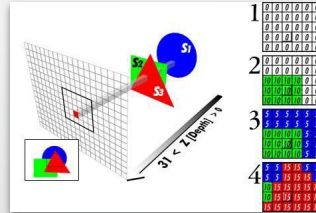


<https://www.racketboy.com/retro/about-video-games-rasterization-and-z-buffer>

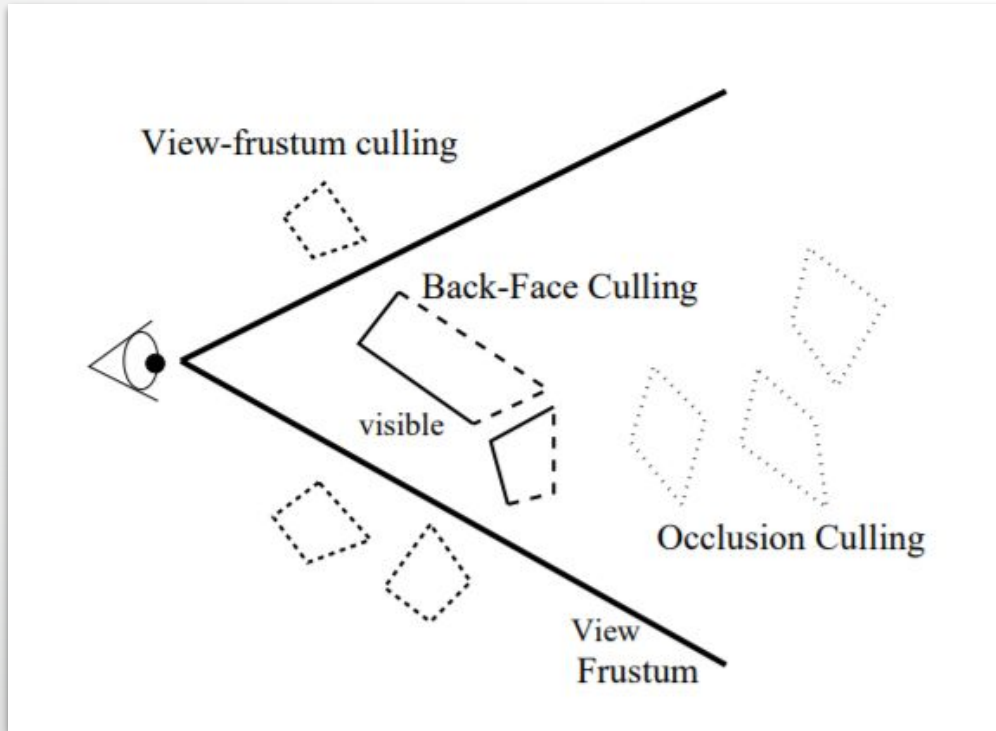
# 3D Graphics Rendering Pipeline



**Are all triangles visible ?**

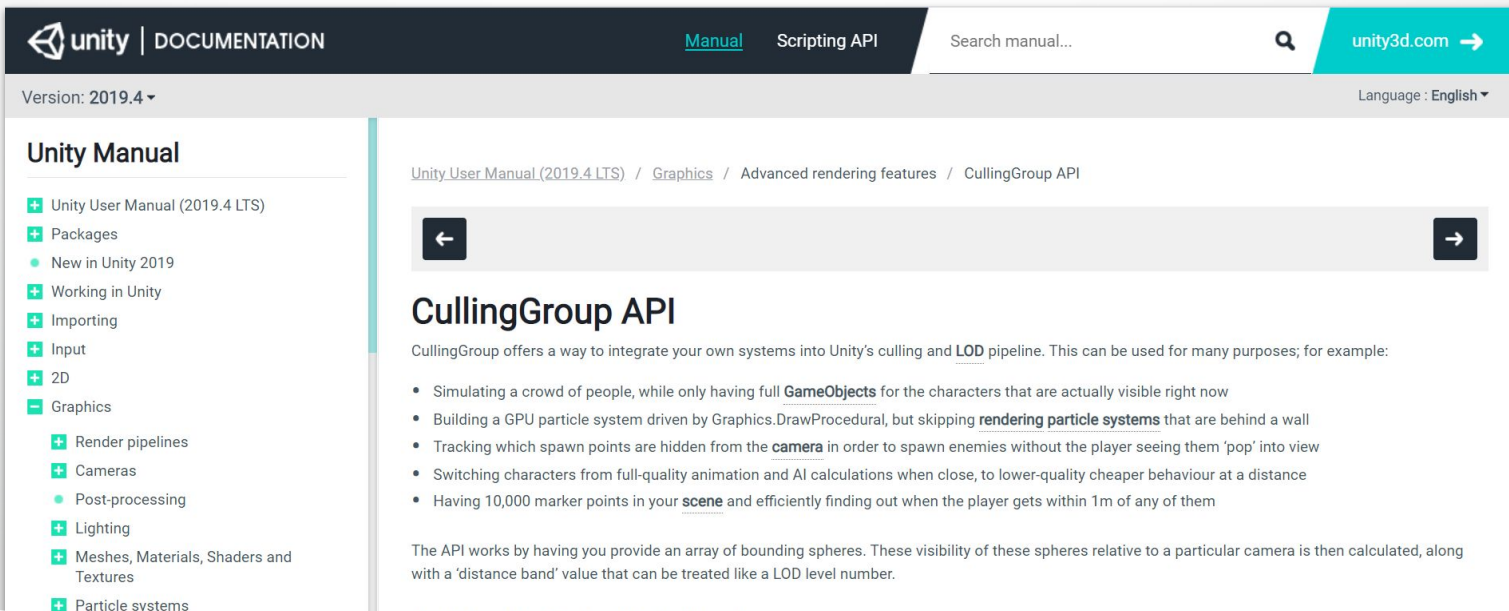


# Culling



# View-frustum culling

- Yes, Unity applies frustum culling to every renderer.
  - But, you can also use the CullingGroup API



The screenshot shows the Unity documentation website for the CullingGroup API. The page title is "CullingGroup API" and it is part of the "Advanced rendering features" section. The page content includes a list of use cases for the API and a brief description of how it works.

Unity User Manual (2019.4 LTS) / Graphics / Advanced rendering features / CullingGroup API

## CullingGroup API

CullingGroup offers a way to integrate your own systems into Unity's culling and LOD pipeline. This can be used for many purposes; for example:

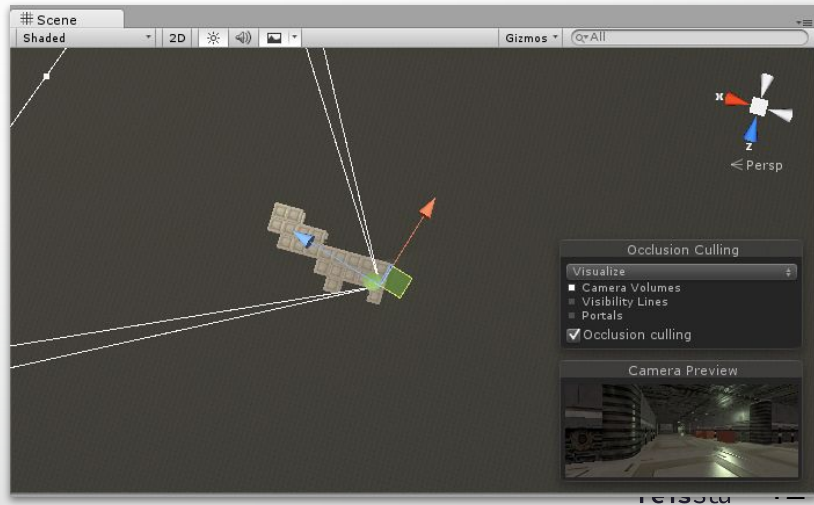
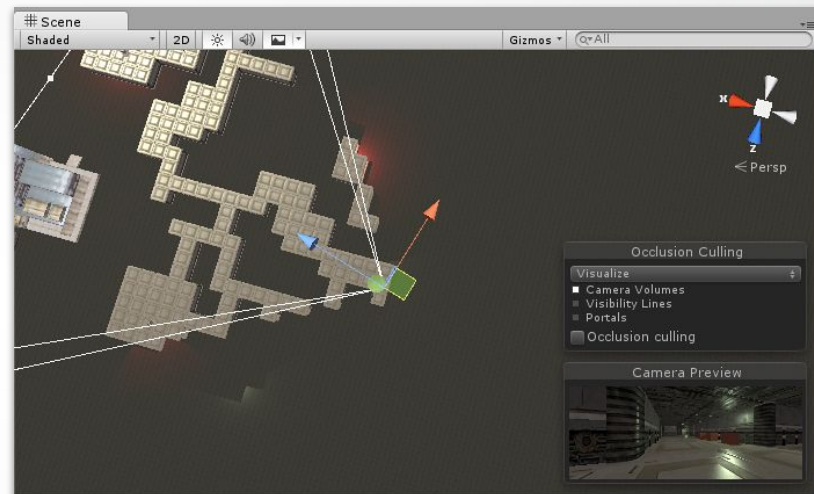
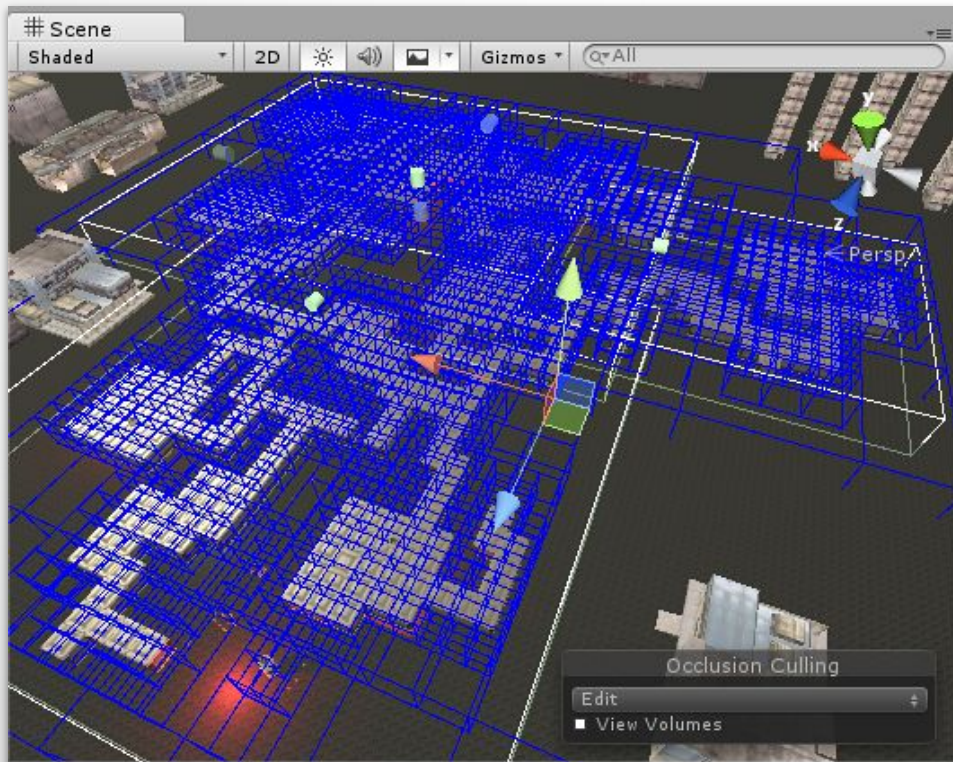
- Simulating a crowd of people, while only having full **GameObjects** for the characters that are actually visible right now
- Building a GPU particle system driven by `Graphics.DrawProcedural`, but skipping **rendering particle systems** that are behind a wall
- Tracking which spawn points are hidden from the camera in order to spawn enemies without the player seeing them 'pop' into view
- Switching characters from full-quality animation and AI calculations when close, to lower-quality cheaper behaviour at a distance
- Having 10,000 marker points in your scene and efficiently finding out when the player gets within 1m of any of them

The API works by having you provide an array of bounding spheres. These visibility of these spheres relative to a particular camera is then calculated, along with a 'distance band' value that can be treated like a LOD level number.



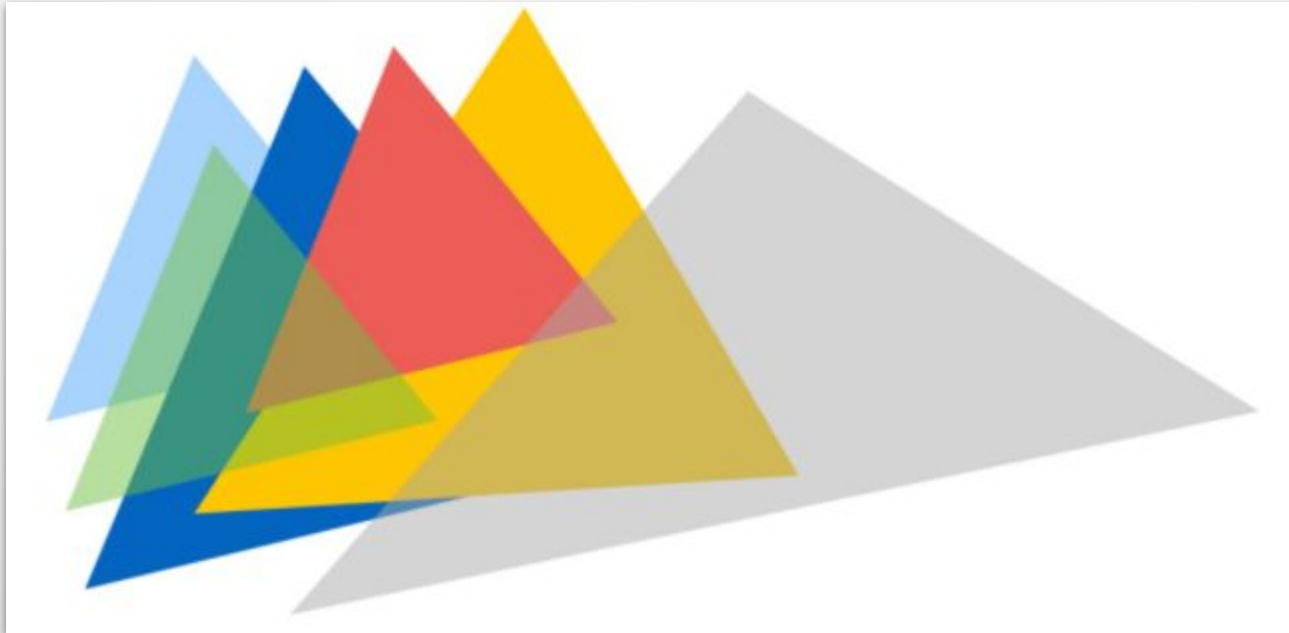


# Occlusion culling





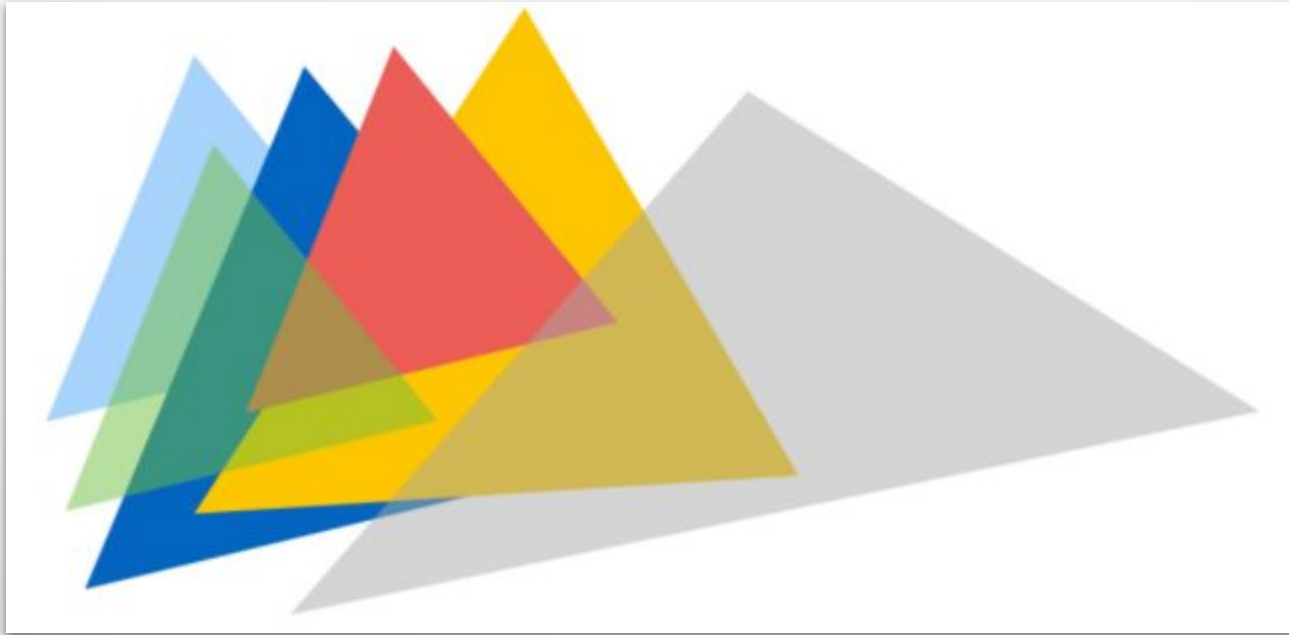
# Transparency ?



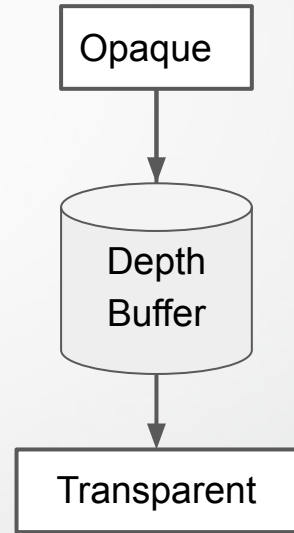
- Fog
- Particles
- Visual effects
- ...

[http://cs248.stanford.edu/winter19/lecture/pipeline/slide\\_055](http://cs248.stanford.edu/winter19/lecture/pipeline/slide_055)

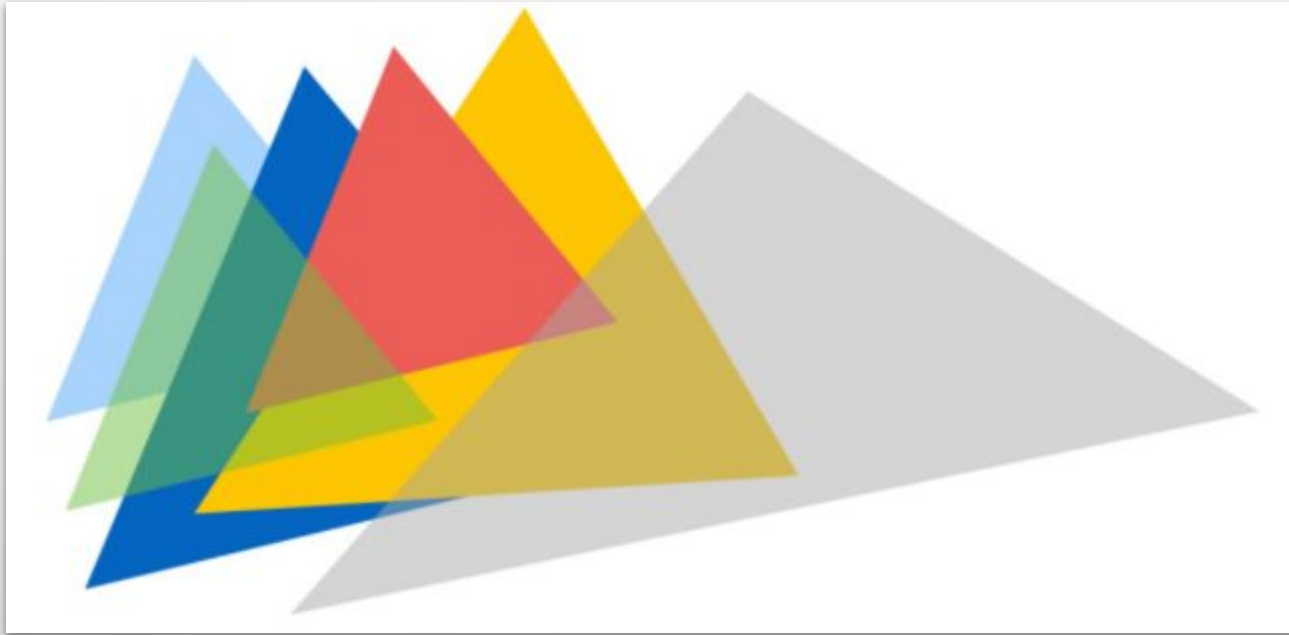
# Transparency ?



[http://cs248.stanford.edu/winter19/lecture/pipeline/slide\\_055](http://cs248.stanford.edu/winter19/lecture/pipeline/slide_055)

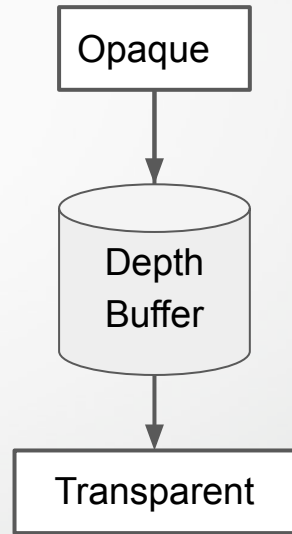


# Transparency ?



[http://cs248.stanford.edu/winter19/lecture/pipeline/slide\\_055](http://cs248.stanford.edu/winter19/lecture/pipeline/slide_055)

**Does the order of transparent objects matter ?**





# Render Queue

Ellen\_Hair\_Mat  
Shader Custom/SculptedHair

Main Color  
Highlight Color  
Highlight Color  
Diffuse (RGB) Alpha (A)

Tiling X 1 Y 1  
Offset X 0 Y 0

Metallic (RGB) \_Smooth (A)

Tiling X 1 Y 1  
Offset X 0 Y 0

Normal (Normal)

Tiling X 1 Y 1  
Offset X 0 Y 0

Anisotropic Direction (Normal)

Tiling X 1 Y 1  
Offset X 0 Y 0

Anisotropic Highlight Offset 0.04  
Anisotropic Highlight Offset -0.19  
Gloss 0.75  
Gloss2 0.5  
Specularity 0.36  
Specularity2 0.13  
Reflection 0.094  
value 0

Offset X 0 Y 0

Normal (Normal)

Tiling X 1 Y 1  
Offset X 0 Y 0

Anisotropic Direction (Normal)

Tiling X 1 Y 1  
Offset X 0 Y 0

Anisotropic Highlight Offset 0.04  
Anisotropic Highlight Offset -0.19  
Gloss 0.75  
Gloss2 0.5  
Specularity 0.36  
Specularity2 0.13  
Reflection 0.094  
value 0

Render Queue  
From Shader 2000

- From Shader
- Geometry
- AlphaTest
- Transparent

Ellen\_Body\_Mat

Auto Generate Lighting Off



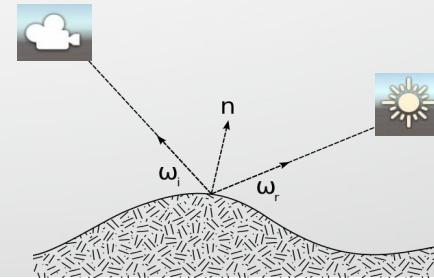
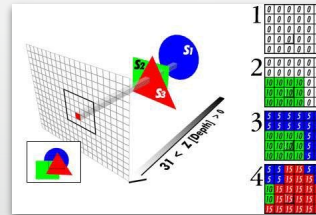
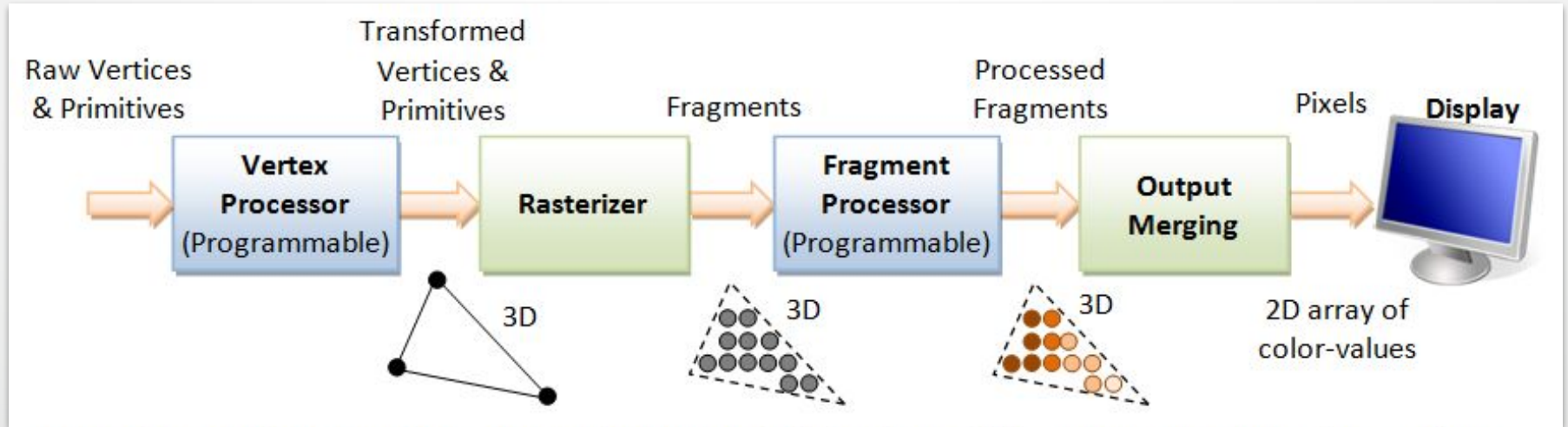
**How many draw calls are required ?**

**DEVOTION**

<https://shop.redcandlegames.com/#game>

# Draw call

Needs to have The same rendering order and material

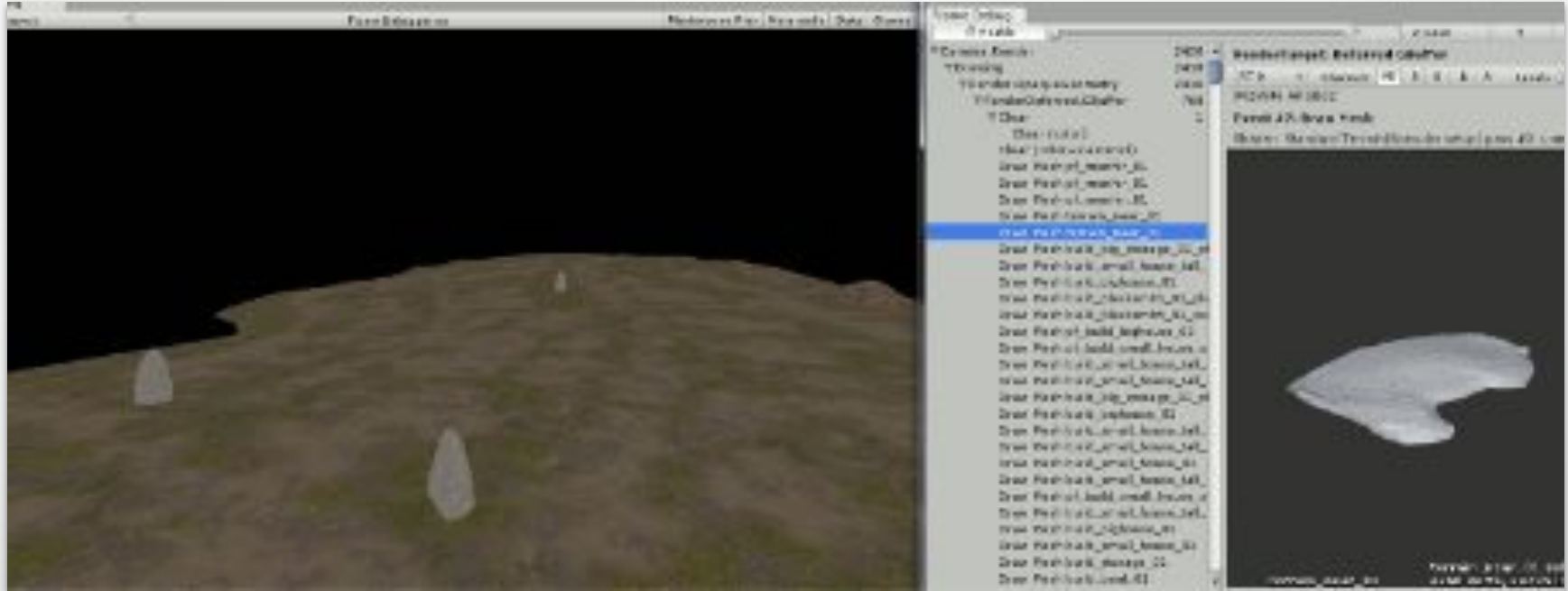








# Frame debugger



<https://docs.unity3d.com/Manual/FrameDebugger.html>

# Case study: RenderDoc

The screenshot displays the RenderDoc application interface, which is used for capturing and analyzing graphics driver events. The main window shows a 3D scene from the game Dota 2, featuring a large blue and purple dragon and a red and black wolf-like creature. The interface is divided into several panels:

- Timeline:** Shows a sequence of events with a timeline from 7000 to 10800. The current event is at approximately 7698.
- Event Browser:** A table listing events with columns for EID, Name, and Duration (µs). The selected event is `vkCmdDrawIndexed(17847, 1)` at EID 7698.
- API Inspector:** A detailed view of the selected event, showing the `vkCmdDrawIndexed` command buffer and its parameters, such as `Command Buffer 12422`, `Descriptor Set 5740440`, and `Descriptor Set 2726082`.
- Texture Viewer:** Shows the texture being used for the selected draw call, with a zoom of 1:1 and a fit of 93%.
- Inputs/Outputs:** Displays the inputs and outputs of the selected draw call, including `res:3927 = tb_colormap` and `res:5892 = swirl_effect`.
- Pixel Context:** A grid showing the pixel context for the selected draw call, with a color map of the scene.

The bottom status bar indicates the replay context is local and no problems were detected.

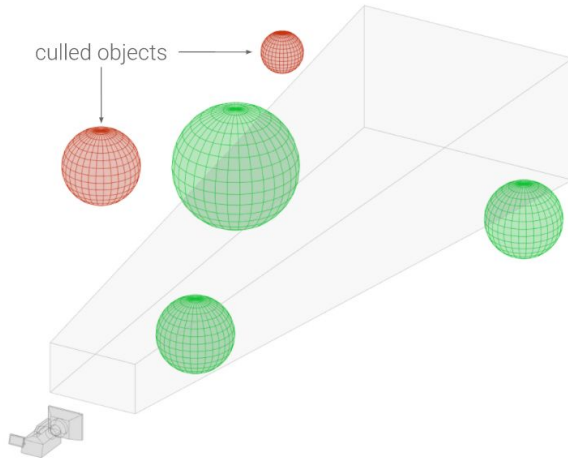


# Unity render pipeline stages

## Spotlight On Render pipeline stages

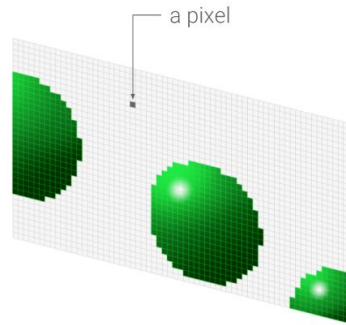
### 1. Culling

List objects to render



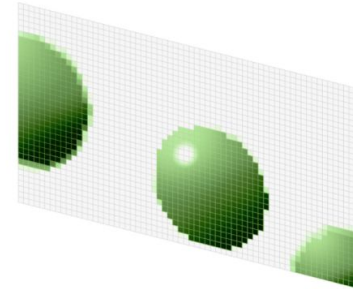
### 2. Rendering

Draw objects



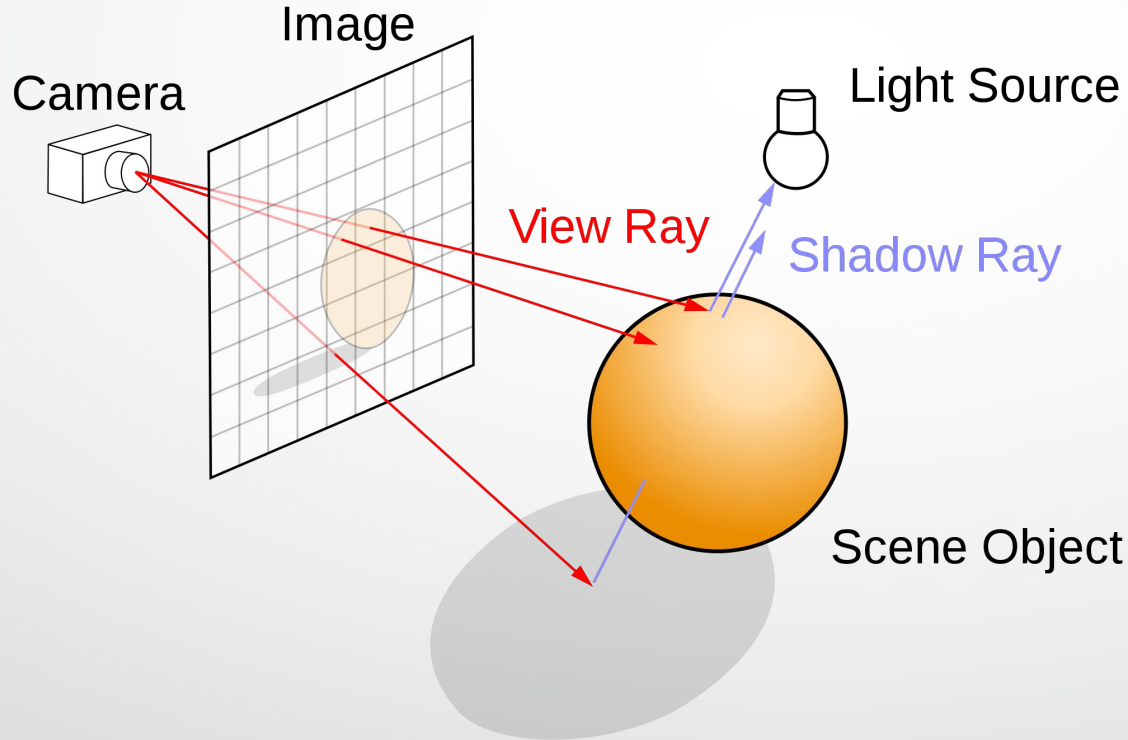
### 3. Post-processing

Apply additional image effects

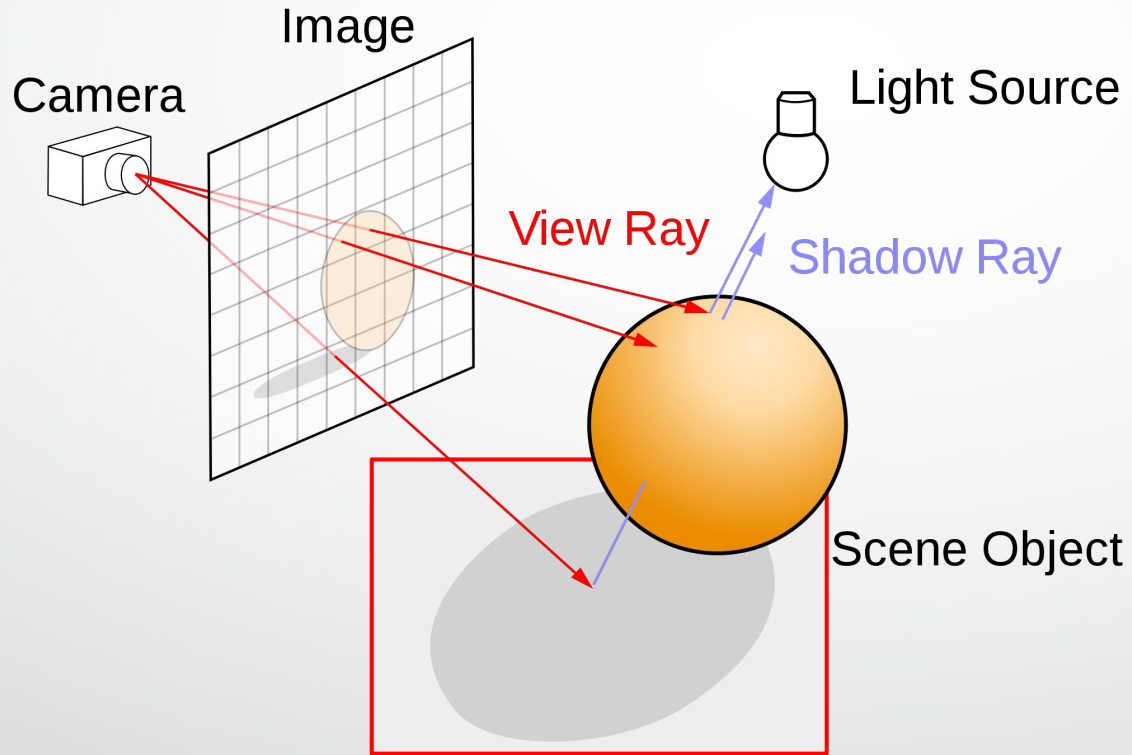


\* These images are simplified representations. The actual number of pixels is much higher.

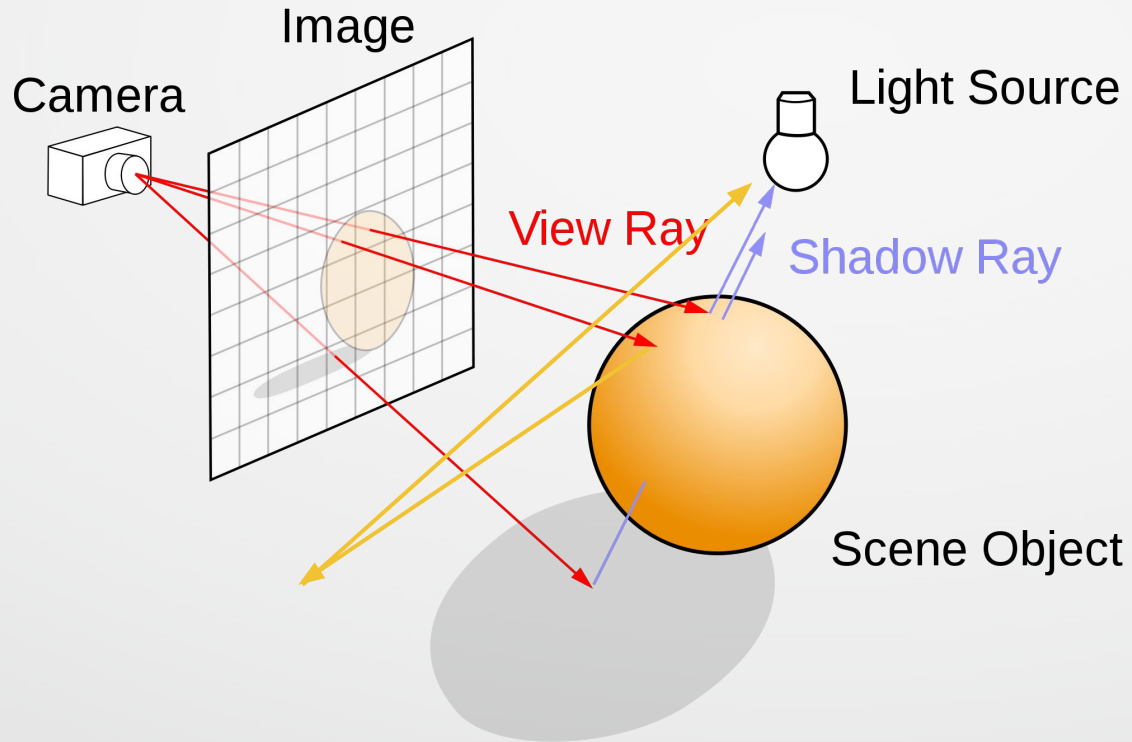
# Rasterization vs. Ray tracing



# Hard / soft shadows ?



# Indirect illumination ?



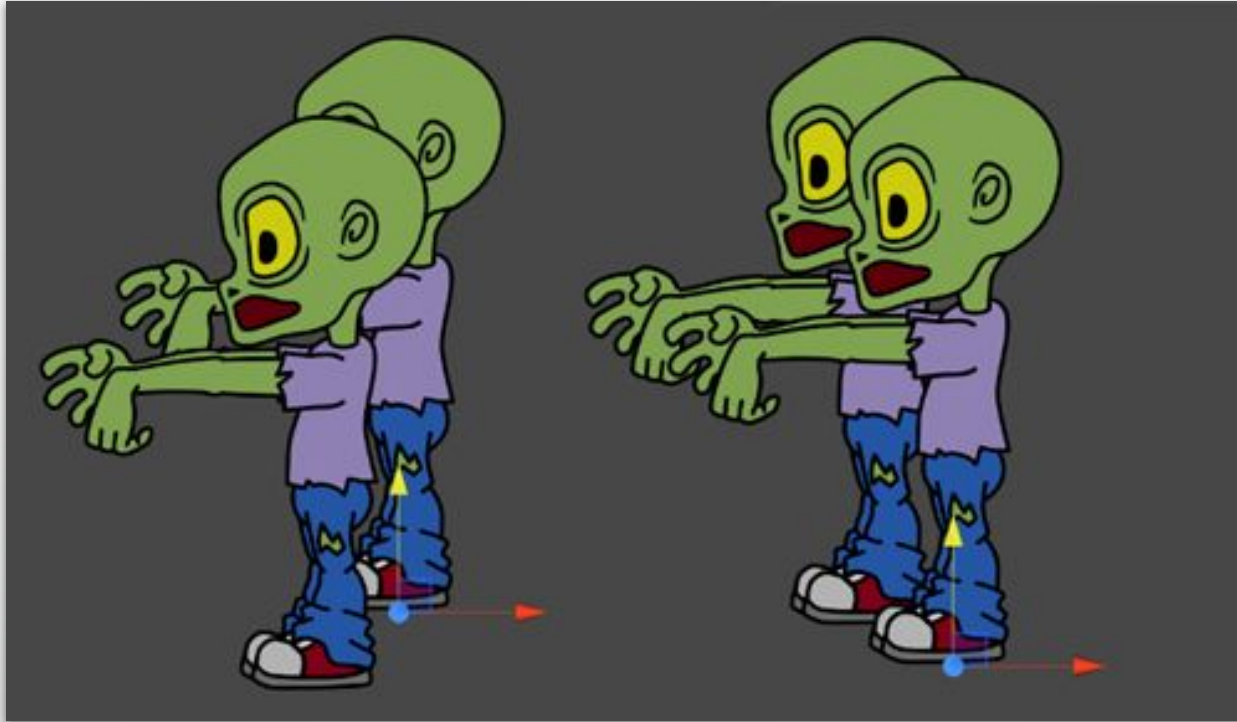


# Shadow and global illumination

- 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

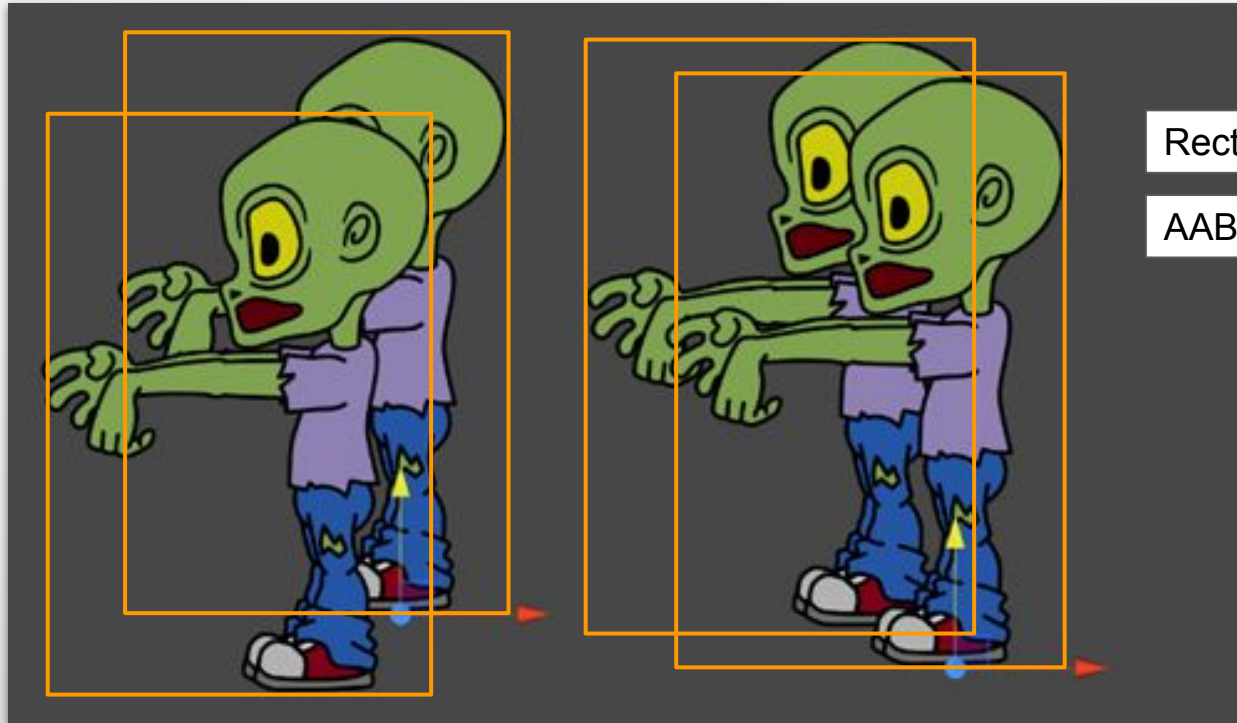


# How about 2D / GUI ?



<https://docs.unity3d.com/Manual/Sprites.html>

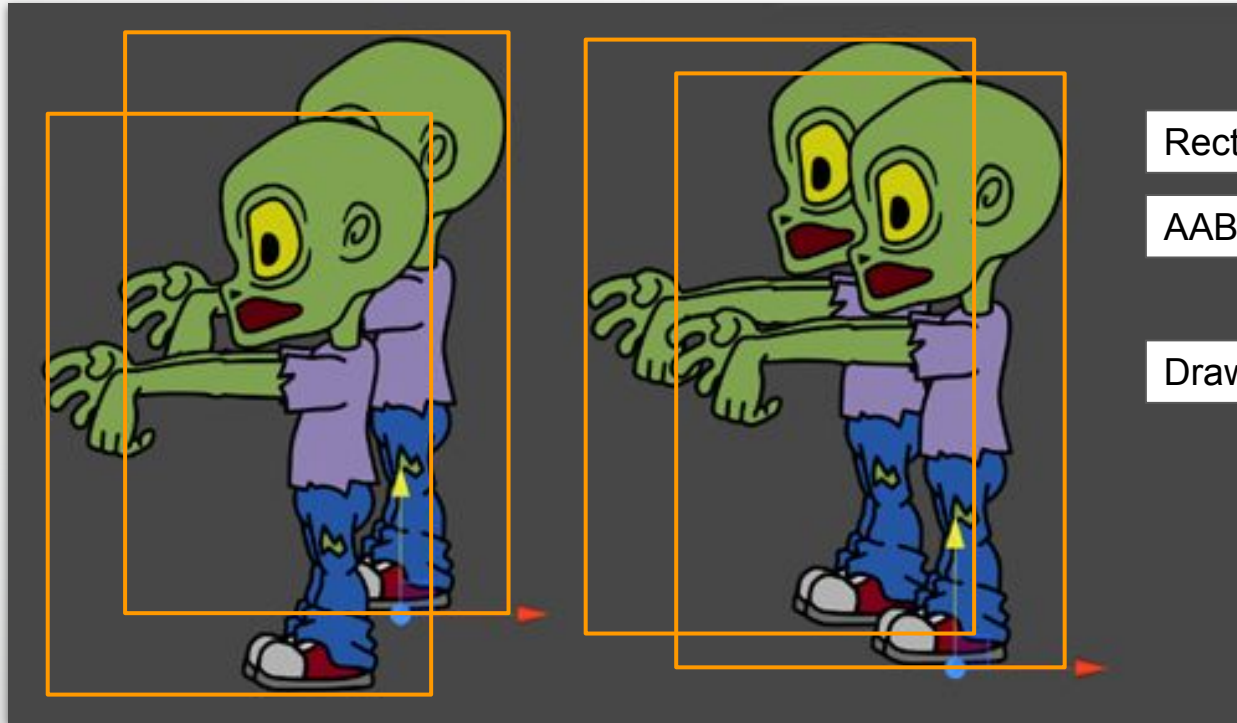
# How about 2D / GUI ?



Rectangle ?

AABB (Axis-Aligned Bounding Box)

# How about 2D / GUI ?

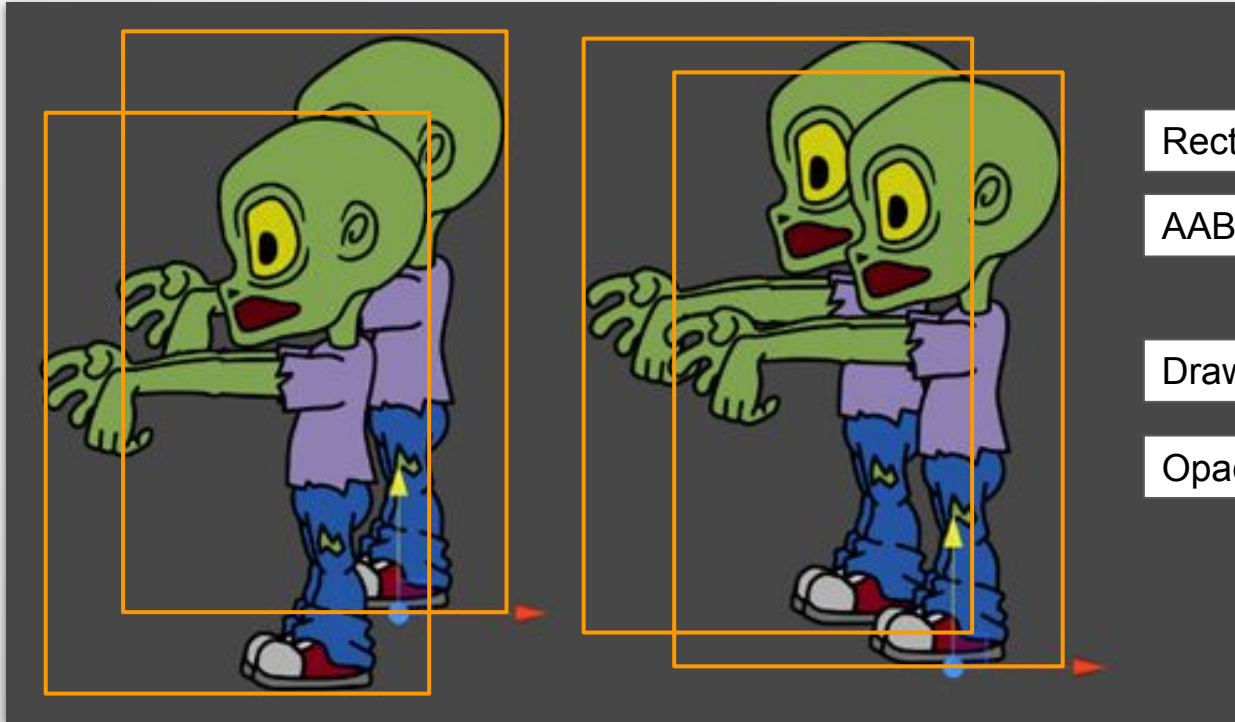


Rectangle ?

AABB (Axis-Aligned Bounding Box)

Drawing order: painter's algorithm ?

# How about 2D / GUI ?



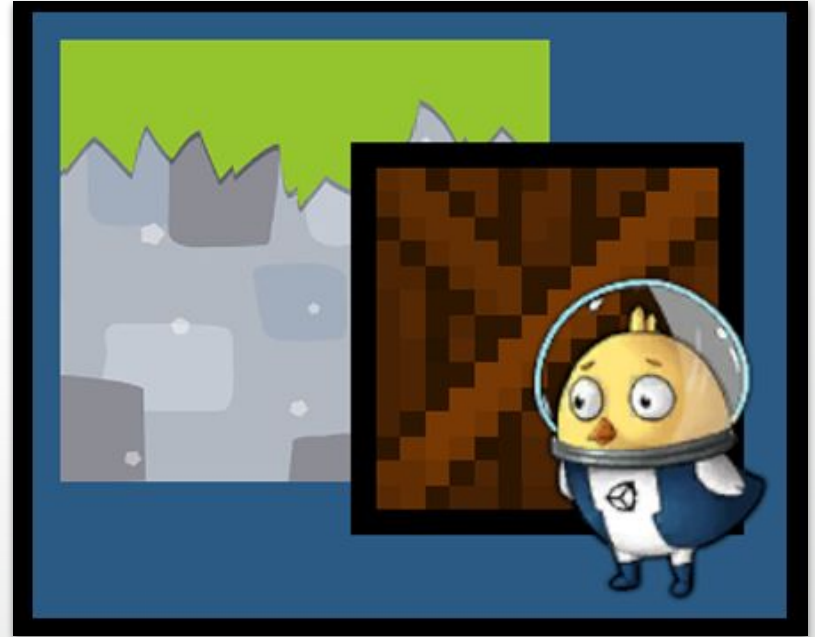
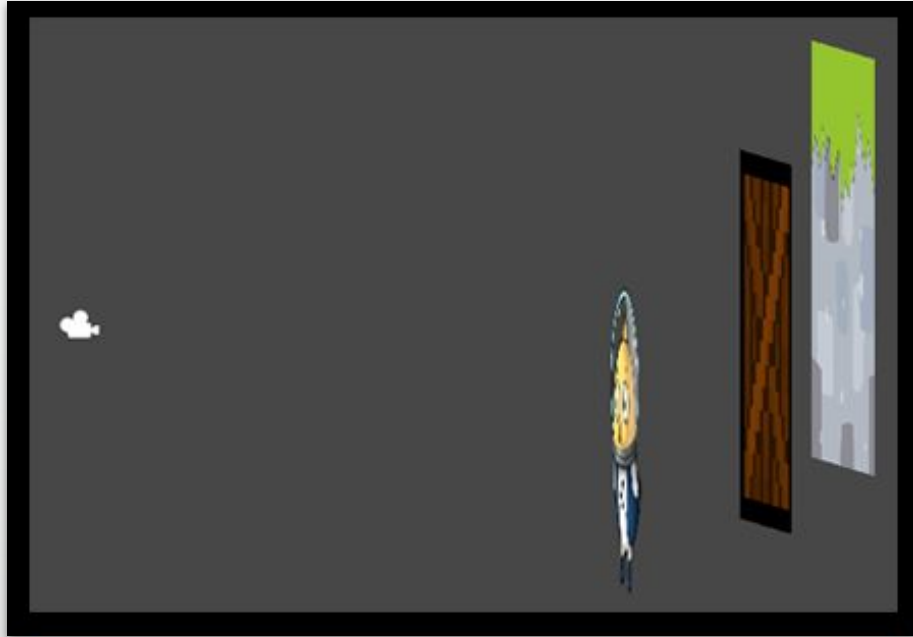
Rectangle ?

AABB (Axis-Aligned Bounding Box)

Drawing order: painter's algorithm ?

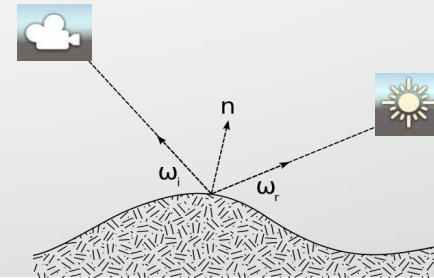
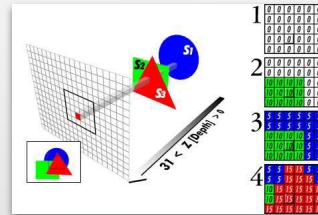
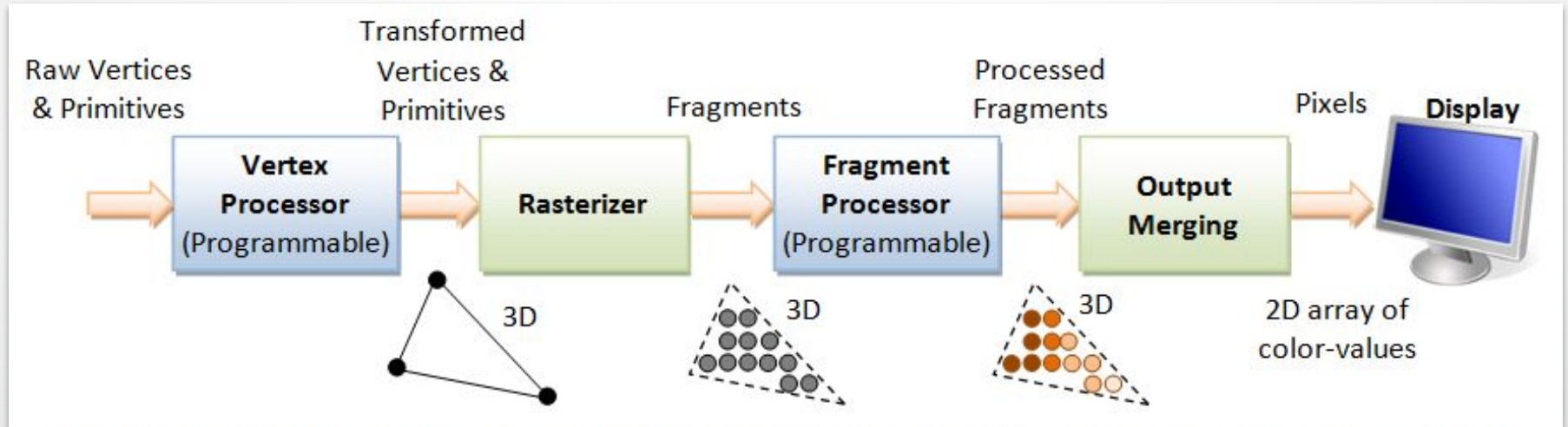
Opaque vs. Transparent

# How about 2D / GUI ? (cont'd)

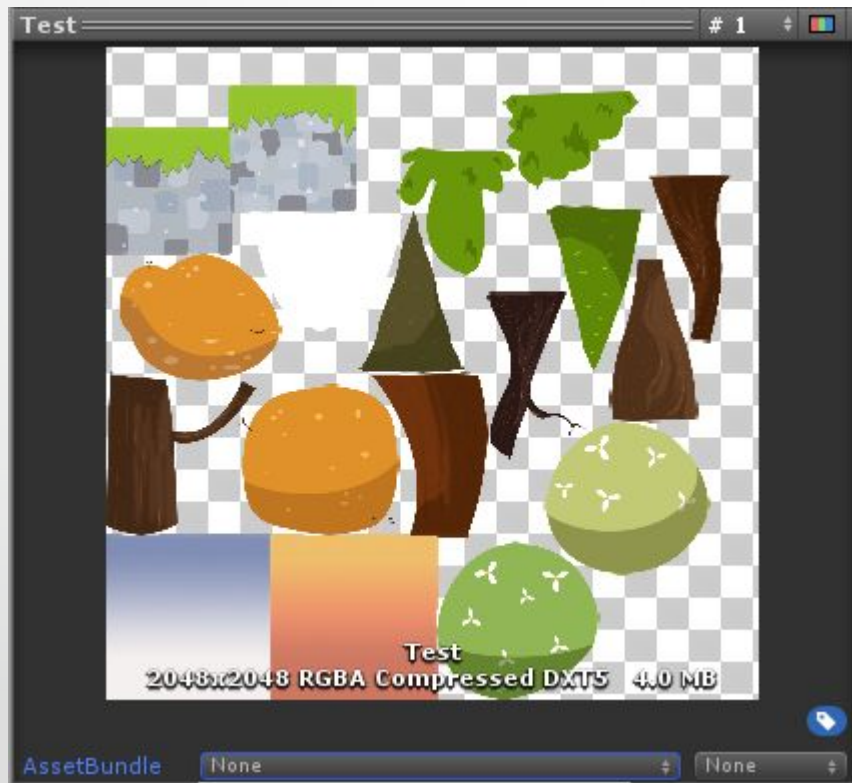


# Draw call

Needs to have The same rendering order and material



# Sprite Atlas



Pack as many textures as possible ?

# Q & A



# Deferred shading vs. Forward shading

