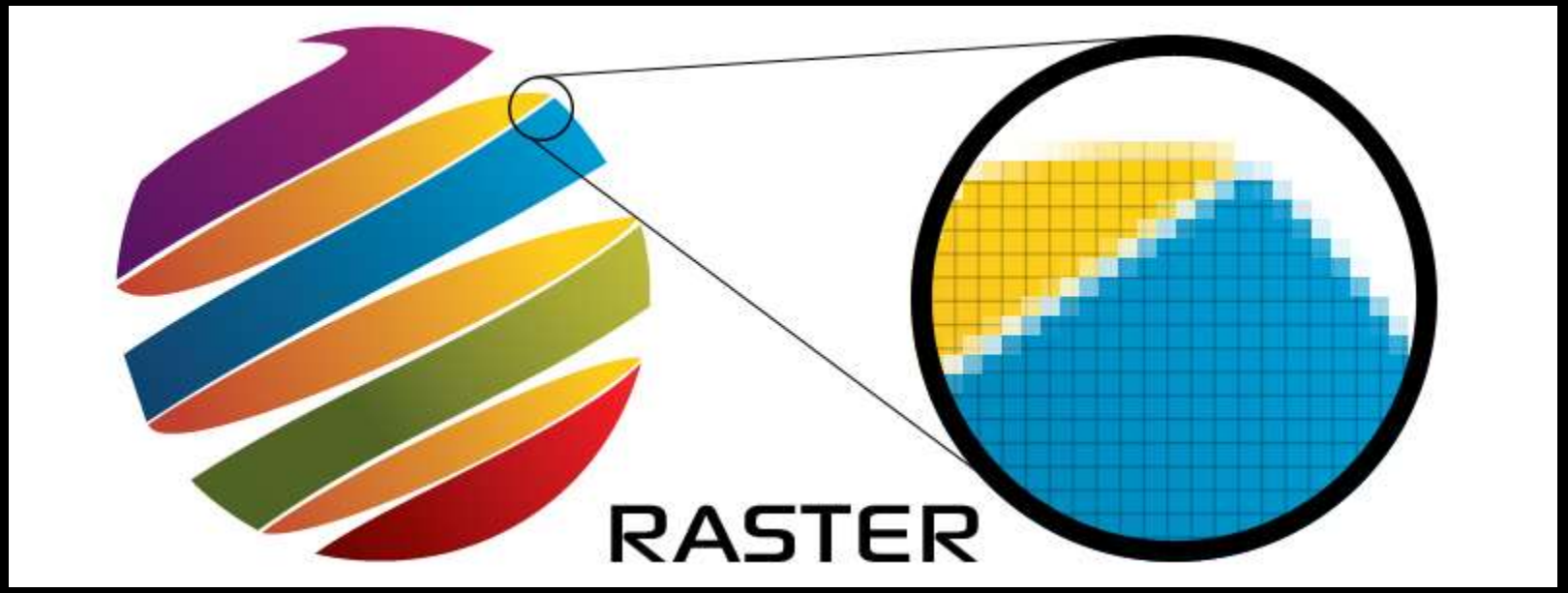
Shading and Rendering

The Graphics Pipeline

A sequence of steps used to create a 2d raster* representation of a 3d scene.

A raster graphic (bitmap image) is a dot matrix data structure representation a rectangular grid of pixels (points of color) that are viewable via a monitor


RASTER

The GPU

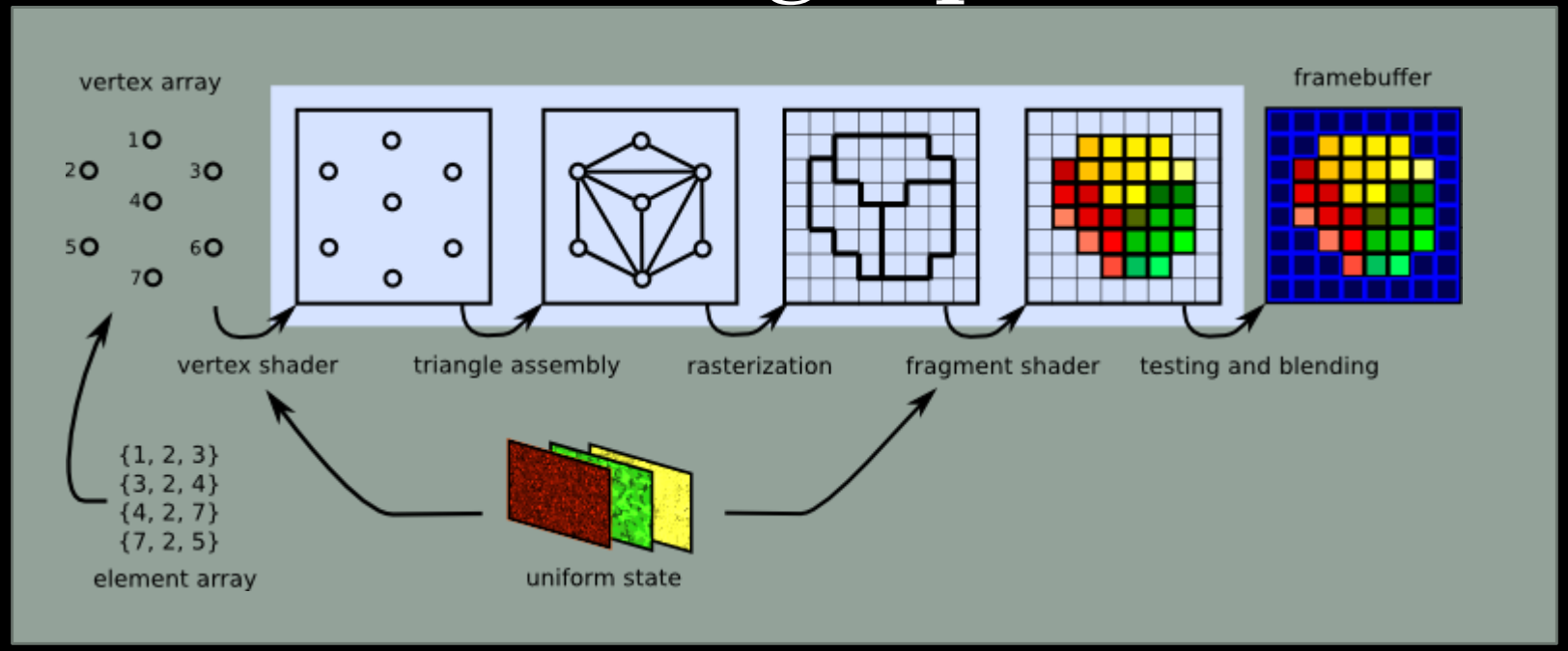
1. Triangulates
2. Interpolates
3. Multithreaded

Rasterization

Rasterization

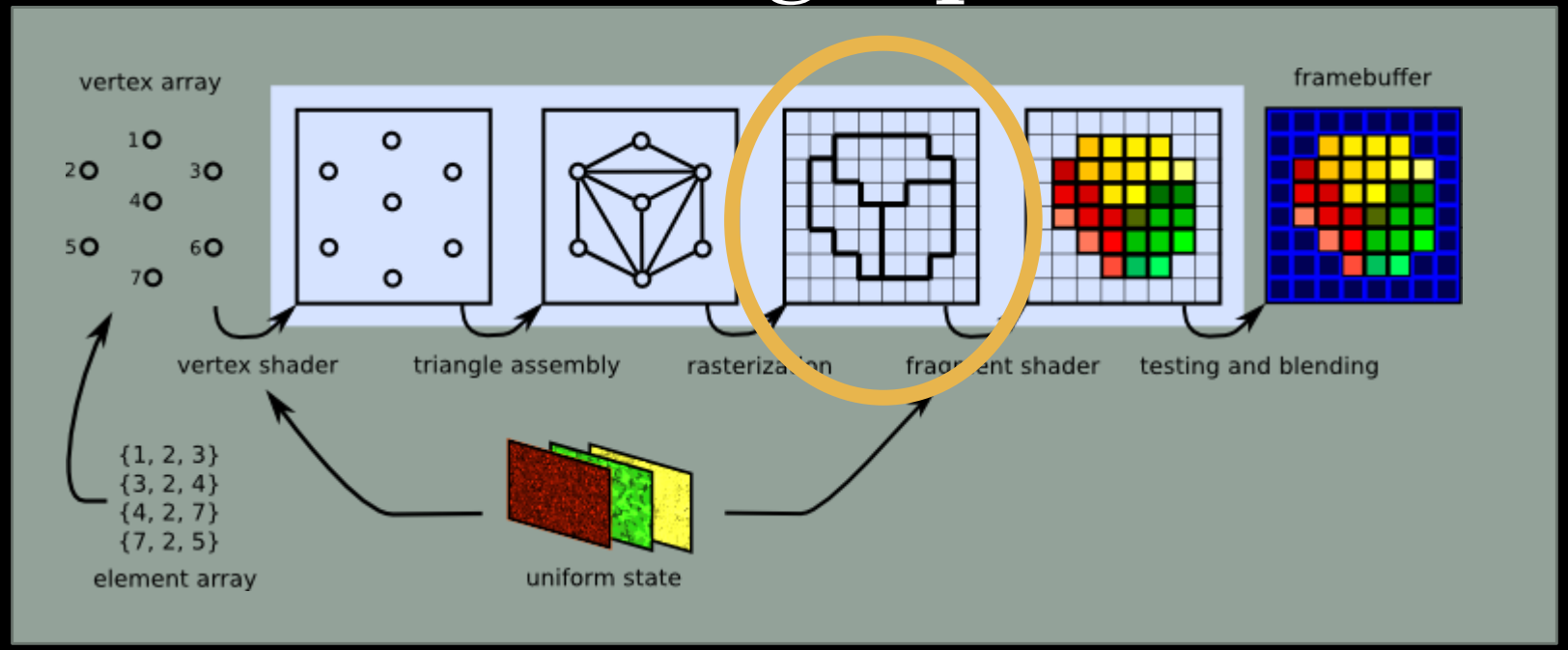
- Refers to the popular rendering algorithm use for displaying 3D models on a computer
- Rasterizers take a stream of vertices (from 3D models), transform them into corresponding 2-dimensional points on the viewer's monitor and fill in the transformed 2-dimensional triangles as appropriate
- Rasterization is currently the most popular technique for producing real-time 3D computer graphics
- Rasterization is extremely fast.

The standard graphics model



Rasterize – creating an image by filling in tris with pixels for a digital screen
Framebuffer – portion of RAM which holds a completed frame of data (a bitmap) used to refresh the screen (allows for all these stages of processing to be invisible to the viewer)

The standard graphics model



Rasterize – creating an image by filling in tris with pixels for a digital screen
Framebuffer – portion of RAM which holds a completed frame of data (a bitmap) used to refresh the screen (allows for all these stages of processing to be invisible to the viewer)

Don’t forget multithreading

- The GPU is performing all of this multiple time at the same time, in parallel
- The GPU is very fast at rasterization
- The GPU is built specifically for rasterization – turning data to images built from pixels

Shading

Vertex Shader and Pixel Shader

Part of the Rasterization Pipeline

Shader

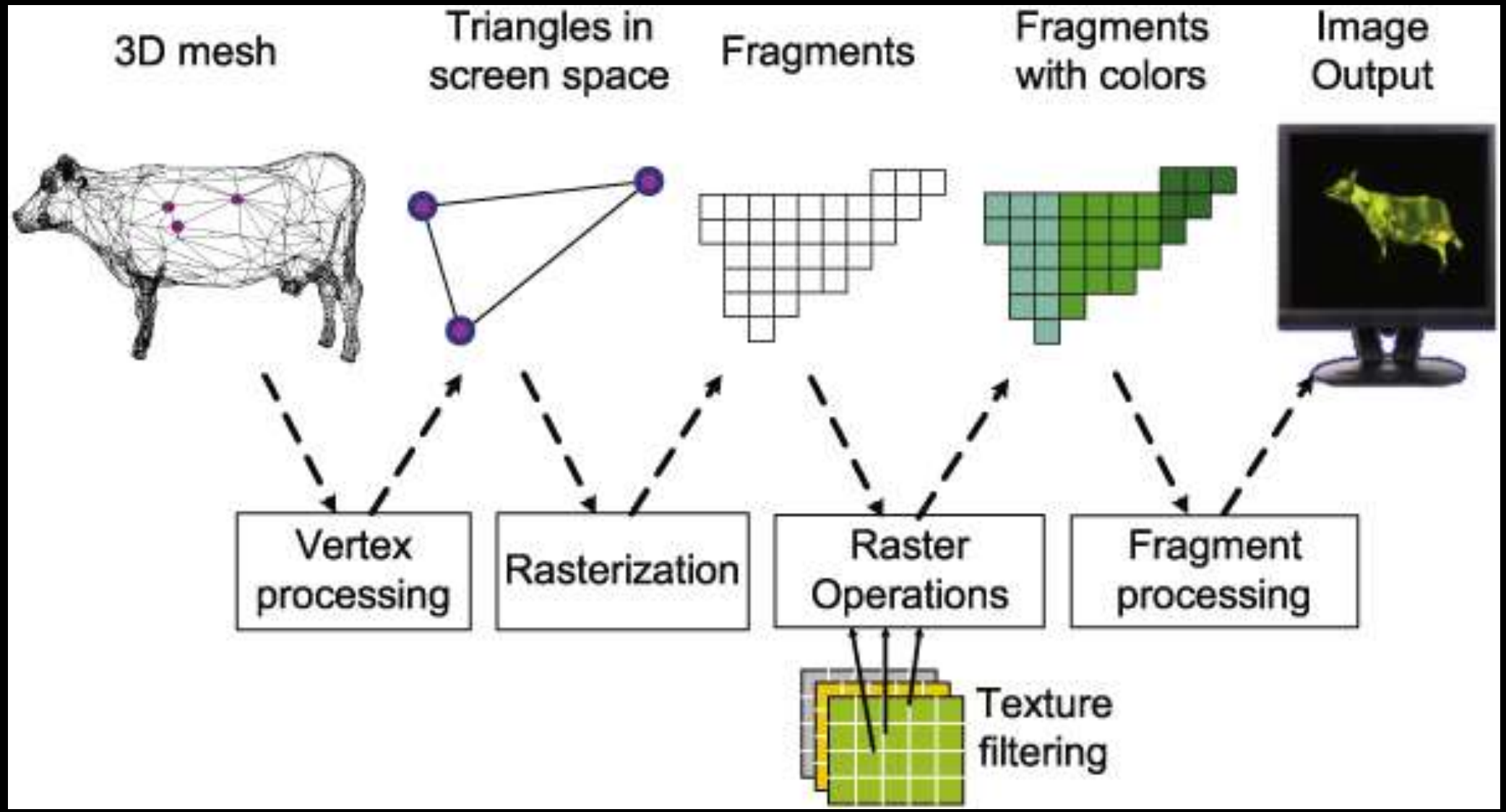
- A shader is simply a program that describes how something looks
- It describes the rendering steps for the GPU to create an accurate representation of the object it describes
- Pixel and Vertex Shaders work together to make most of the environments and characters we love in our games look the way they do

Vertex Shader

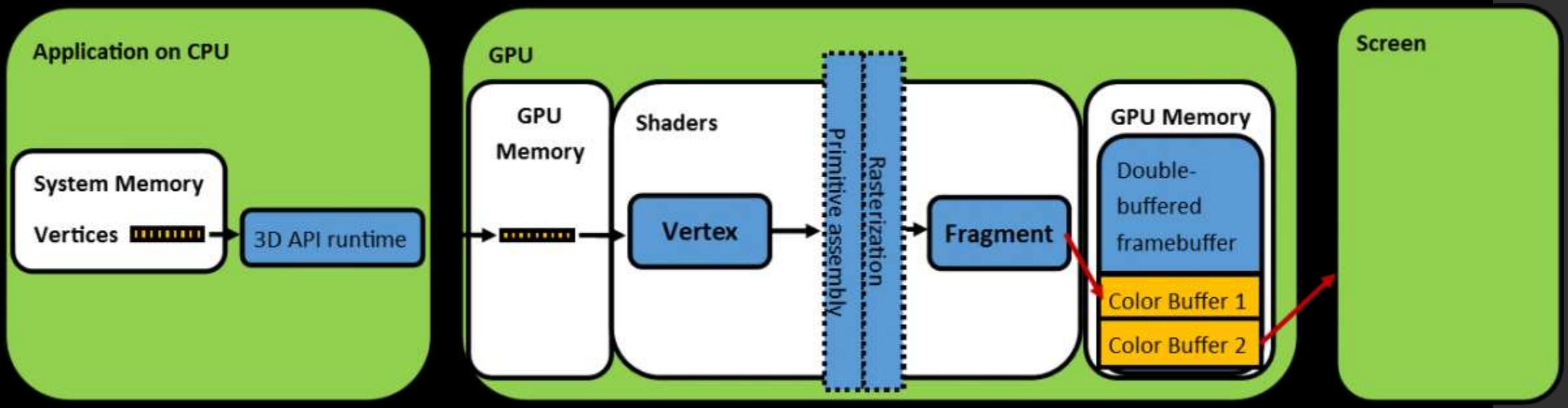
- All assets rendered on screen have a Vertex Shader associated with them to handle processing of individual vertices at render time
- The Vertex Shader processes vertex related data such as the position of the vertice, it's normal direction and texture coordinate
- Vertex Shaders are able to make changes to a model's existing vertices
- Skinning and animation data is also computed by the Vertex Shader

Pixel Shader

- Fragments = Pixels
- The Pixel Shader is a program that computes the color of a pixel based on information supplied by the Vertex Shader, textures, and other user added input
- Pixel Shaders create the details of your assets
- Handles calculations of lighting, shadowing, specularity, reflectivity as well as many other surface effects


3D mesh
Triangles in screen space
Fragments
Fragments with colors
Image Output
Vertex processing
Rasterization
Raster Operations
Fragment processing
Texture filtering

Overview


Application on CPU
System Memory
Vertices
3D API runtime
GPU
GPU Memory
Shaders
Vertex
Primitive assembly
Rasterization
Fragment
GPU Memory
Double-buffered framebuffer
Color Buffer 1
Color Buffer 2
Screen

A note on spaces

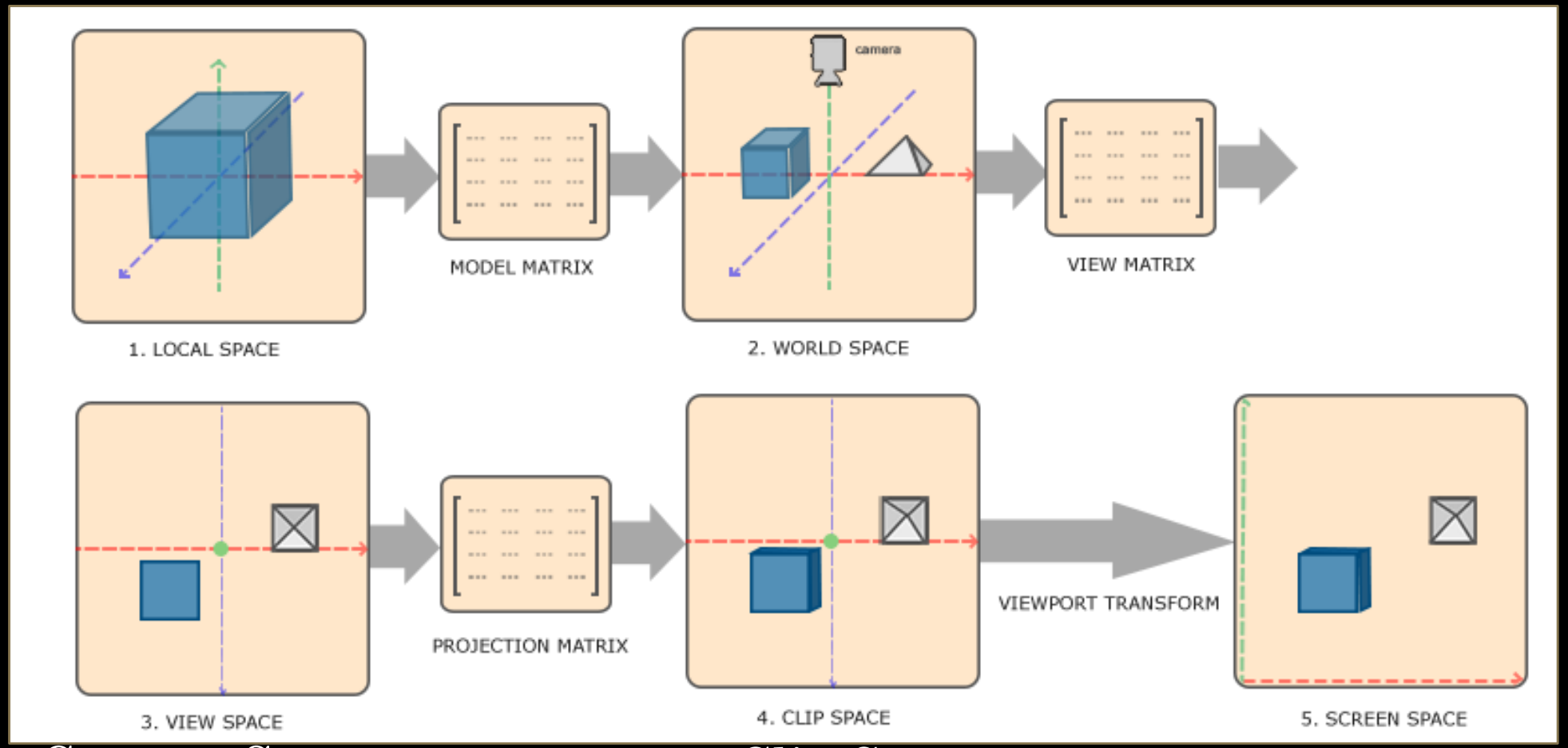
In a 3D game engine, we usually deal with several different **coordinates spaces**. Among the most commonly used spaces are Object space, World space, Inertial space, and Camera space.

Local Space

Your object relative to its local origin. The is the “space” you model your asset in

World Space

Your object is now relative to a global origin of the world.



Screen Space

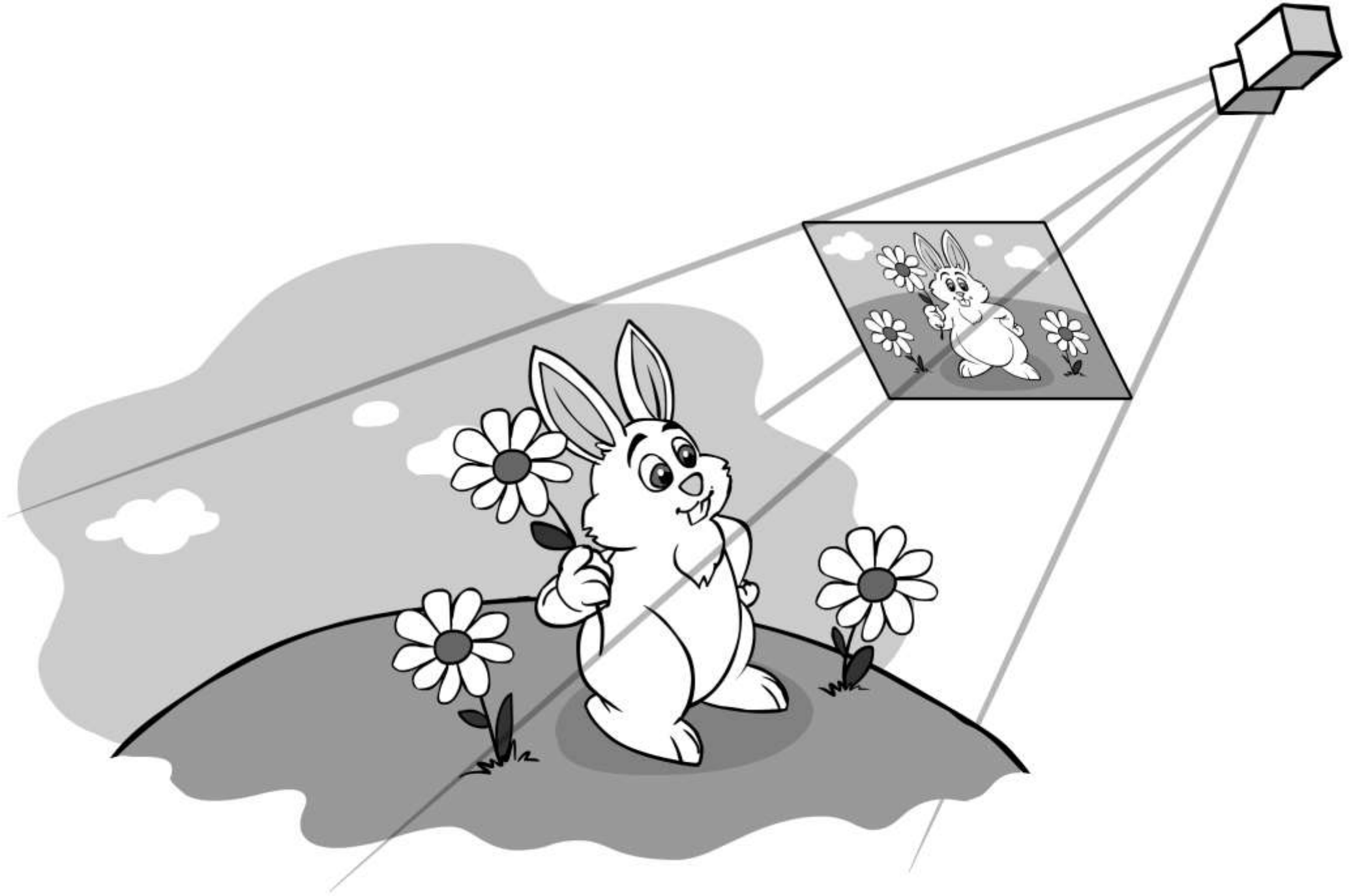
Lastly we transform the clip coordinates to screen coordinates.

Camera Space

Transforming your world-space coordinates to the space as seen from the camera's point of view

Clip Space

Determines which vertices to render to screen based on their position – inside out outside the camera frustum

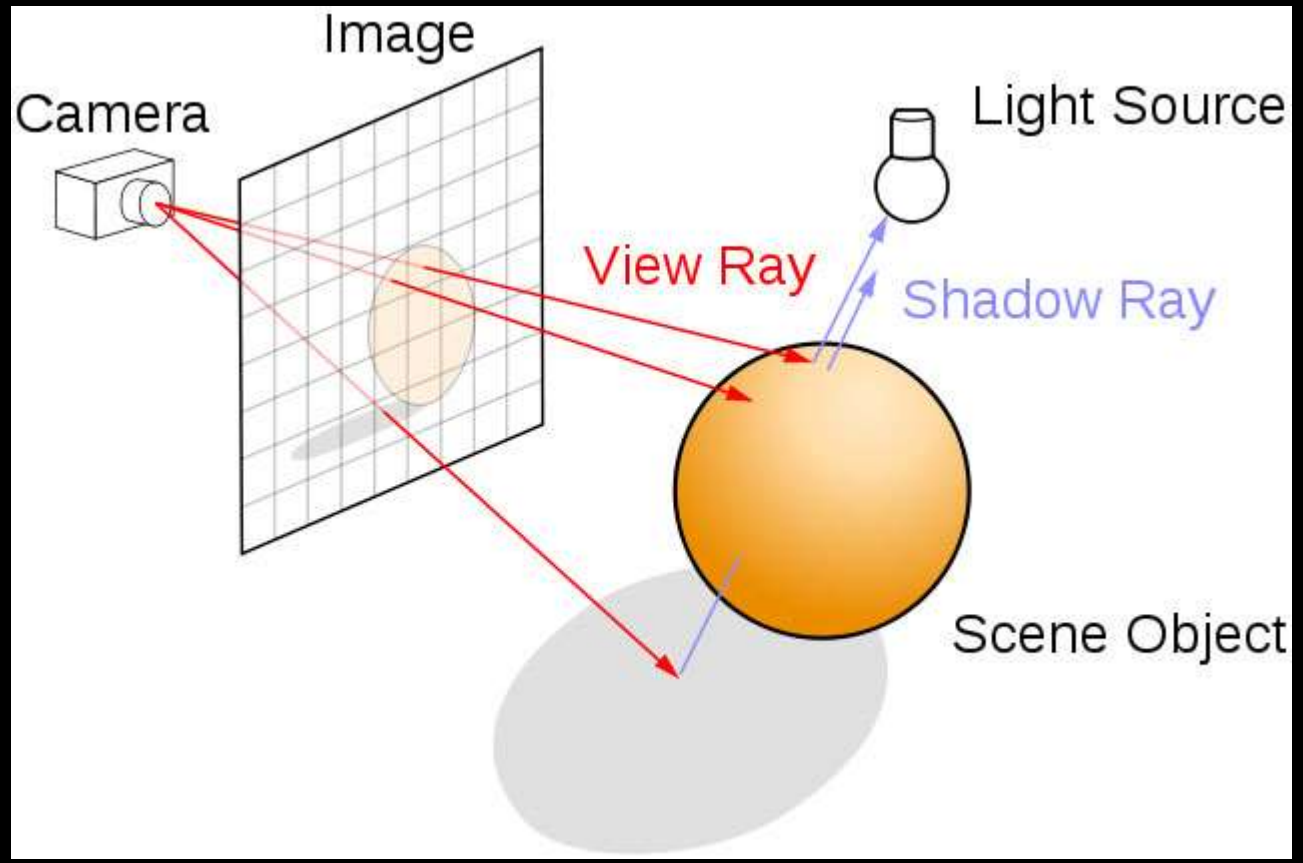


Other rendering techniques?

Ray tracing

- Ray tracing is a technique for generating an image by tracing the path of light through pixels in an image plane and simulating the effects of its encounters with virtual objects
- Each ray is tested for intersections with the objects in the scene
- Once an object is hit a ray is cast to each of the light sources in the scene to calculate illumination and surface shading for the intersection point (this determines the pixel color)
- If the material has specular reflectivity, a ray is traced in the reflection direction to test for reflected objects and return their illuminated color to the first surface
- If the surface is transparent the ray is sent out further into the scene, possibly at an angle to simulate refraction

Ray tracing



Comparing rasterization and ray tracing

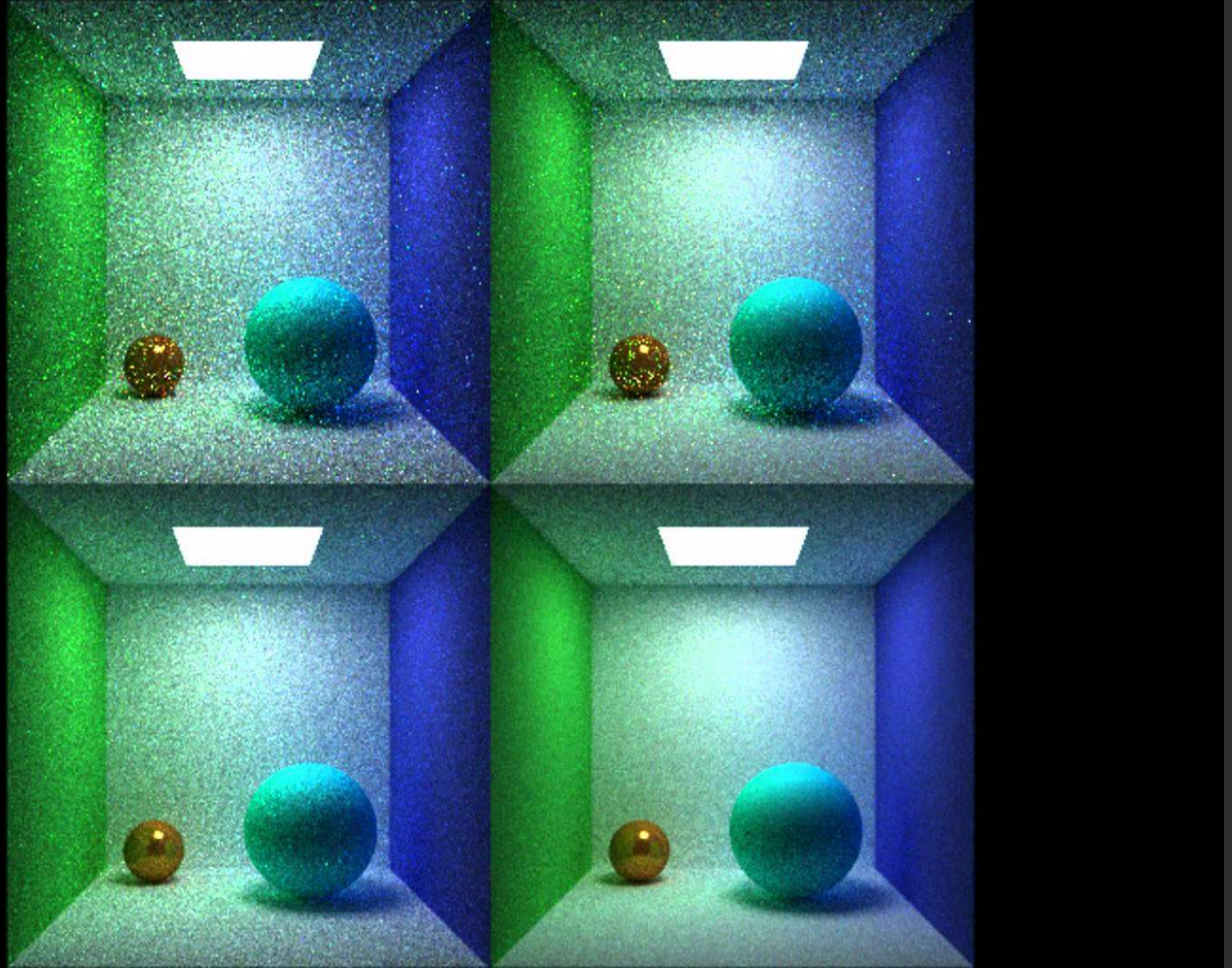
- Rasterization rendering time is linear to the number of triangles that are drawn
- High end graphics cards can display about 1 million polygons in real time, though in games we are doing a lot more then just drawing polygons. Modern graphic cards are designed for rasterization techniques
- The vast majority of computer graphics today are drawn using this technique
- Rasterization can only look at a single triangle at a time. However, most effects require access to multiple triangles: e.g. casting a shadow from one triangle to the other, computing the reflection of one triangle off of another, or simulating the indirect illumination due to light bouncing between all triangles in the scene
- Rasterization must do various tricks to approximate these effects, e.g. using reflections maps instead of real reflection

Comparing rasterization and ray tracing

- Due to its mathematical correctness, ray tracing makes for more realistic graphics, e.g. indirect lighting, shadowing, refractions, reflections, and volume rendering then rasterization
- However a ray tracer still has to cheat to get soft shadows, caustics, and true global illumination
- This is difficult to do this in real time

Path tracing

- Similar to ray tracing but sends out tens, hundreds or even thousands of rays for each pixel to be rendered (instead of just one).
- When the ray hits a surface it bounces off and keeps bouncing until it hits a light source of exhausts a bounce limit.
- Then it calculates the amount of light transferred all the way to the pixel, including any color information gathered from surfaces it hit along the way
- Averaging all the values calculated from all the paths that were traced into the scene determines the final color of the pixel
- Requires a ton of computing power
- Closest solution to physically correct rendering, but still doesn't work for everything.
- For example fails to accurately represent partially translucent surfaces (such as skin, wax, milk, etc.)



Voxel Cone Tracing

- A form of ray tracing that replaces rays, which have no thickness, with thick rays, cones
- An algorithm that can be used to compute indirect lighting with fast estimations of the visibility and incoming energy
- Currently performance is scene-independent (25-70FPS) and can handle dynamic content
- Voxel cone tracing can be used to efficiently estimate Ambient Occlusion

https://youtu.be/dD9CPqSKjTU

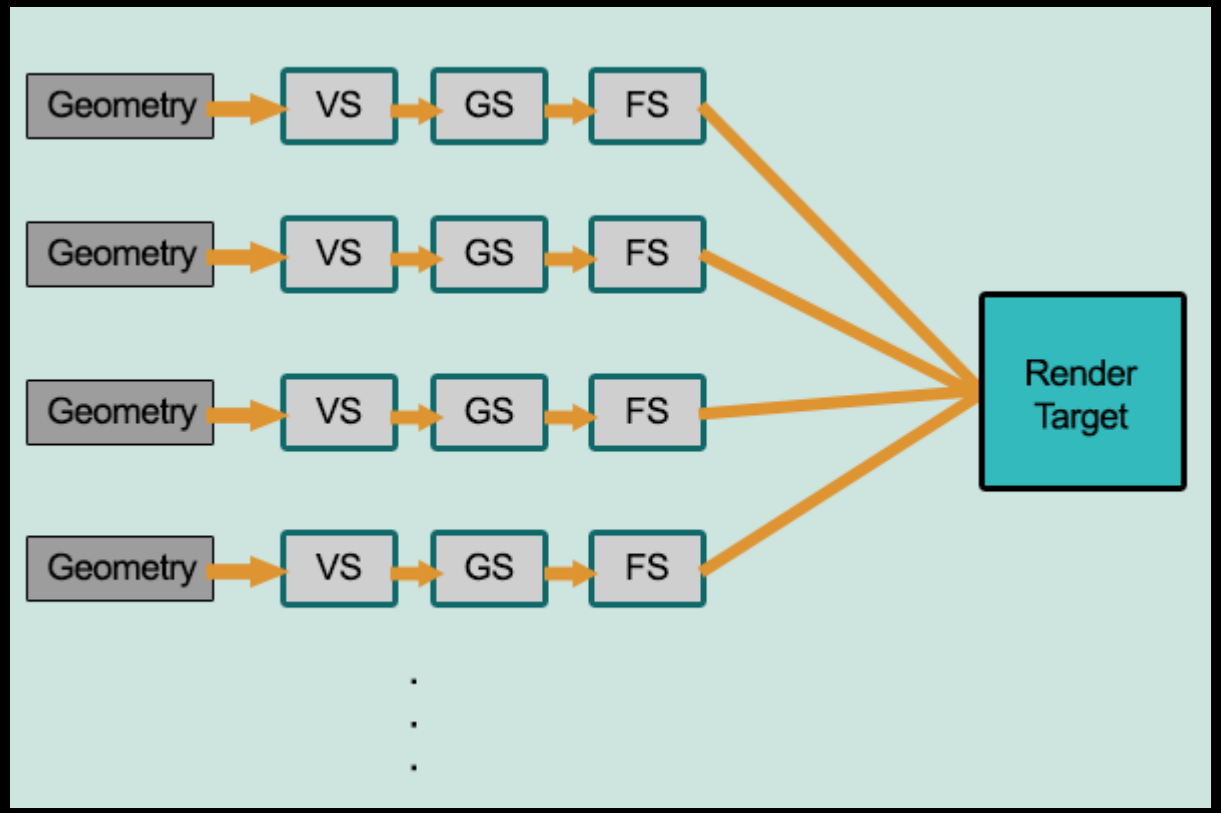
Forward and Deferred Rendering

Both rasterization techniques

Forward Rendering

- Takes the geometry, breaks it down into vertices, and then converts those to pixels that get the final rendering treatment and is then passed onto the screen
- Rasterizes each geometric object in the scene
- Every geometric object has to consider every light in the scene
- Meaning the Pixel Shader per geometric object is completely recalculated for *each* light in the scene
- The resulting output of these calculations are blended together
- Overlapping lights become extremely expensive
- Too many lights cause frame rate issues

Forward Rendering

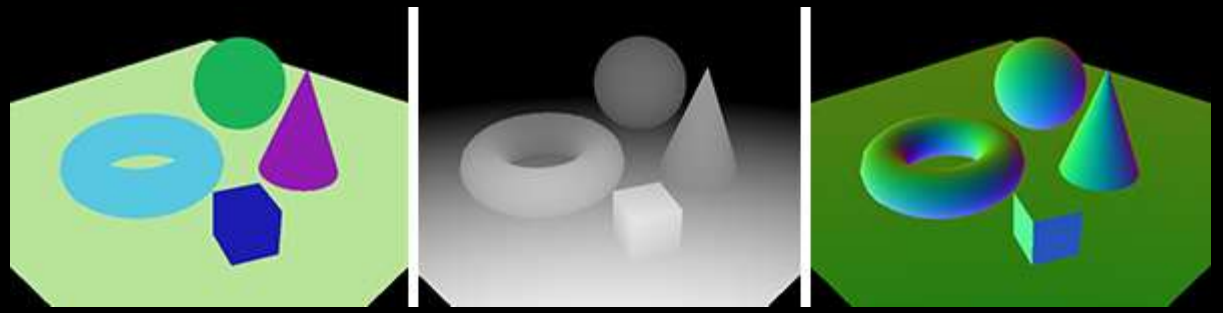


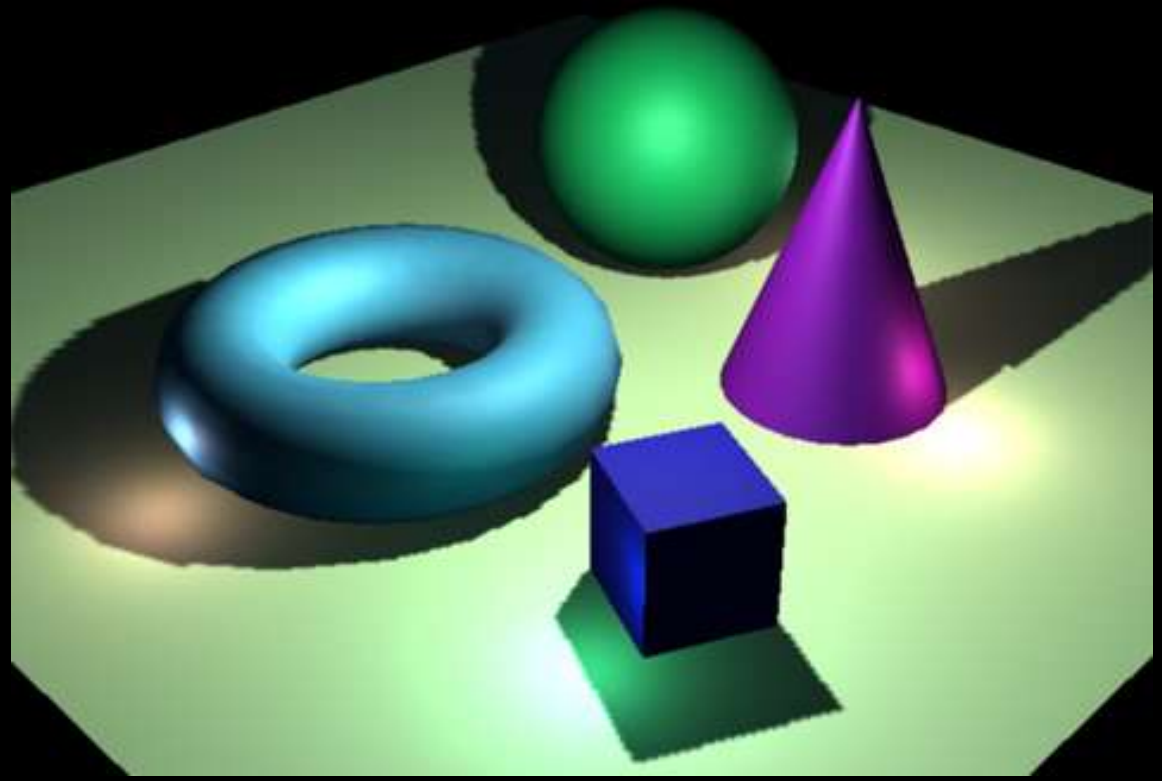
Forward Rendering

- Careful culling of lights improves performance
- Light limitation improve performance
- Many times per-pixel lighting only takes place with the closest two or three lights and per-vertex lighting on next three or four closes lights

Deferred Rendering

- Rasterizes all of the scene objects without lighting
- Stores important information into 2D “image buffers”
- This important info includes screen space depth, surface normals, diffuse color, specular color and specular power
- The combination of these images is referred to as the G-Buffer





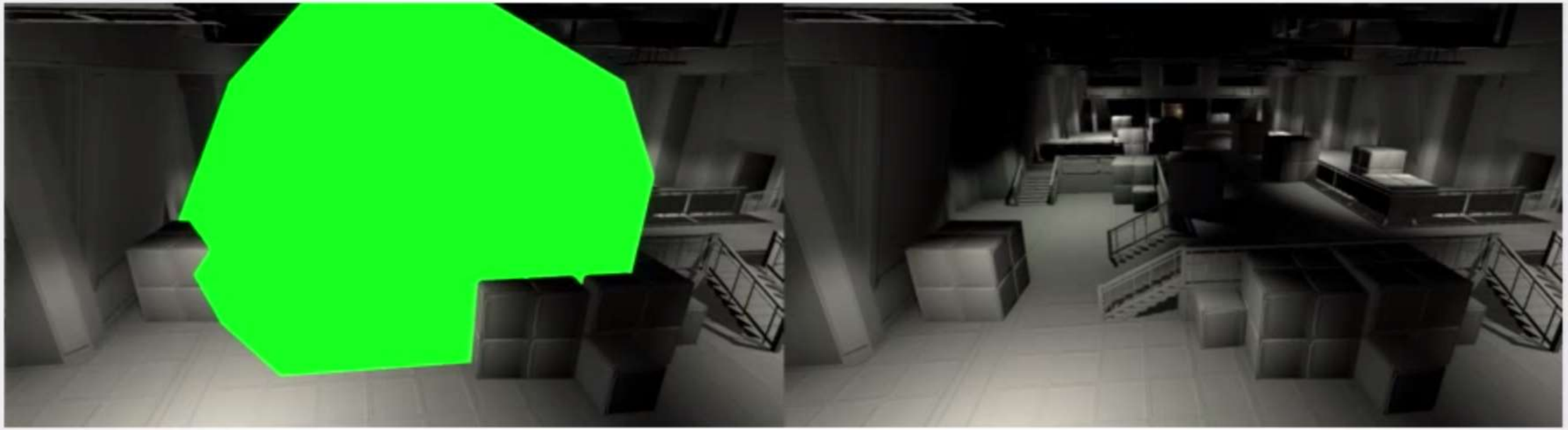
Deferred Rendering

- The lighting pass is performed by rendering each light source as a geometric object in the scene
- Each pixel that is touched by the light's geometric representation is shaded using the desired lighting equation
- Expensive lighting calculations are only computed once per light per covered pixel which allows for *many* more lights per scene

Deferred lighting

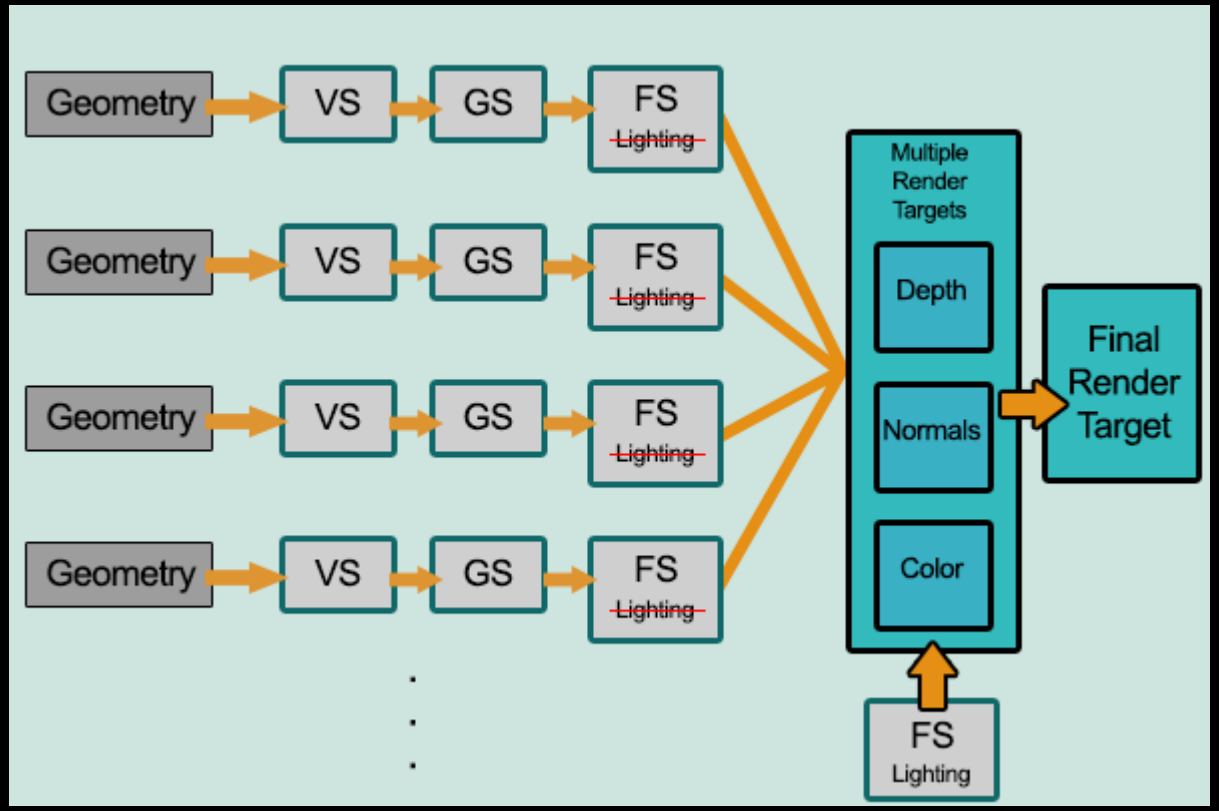
- Partially filled light buffer

Deferred lighting



- Point light

Deferred Rendering



The Disadvantage of Deferred

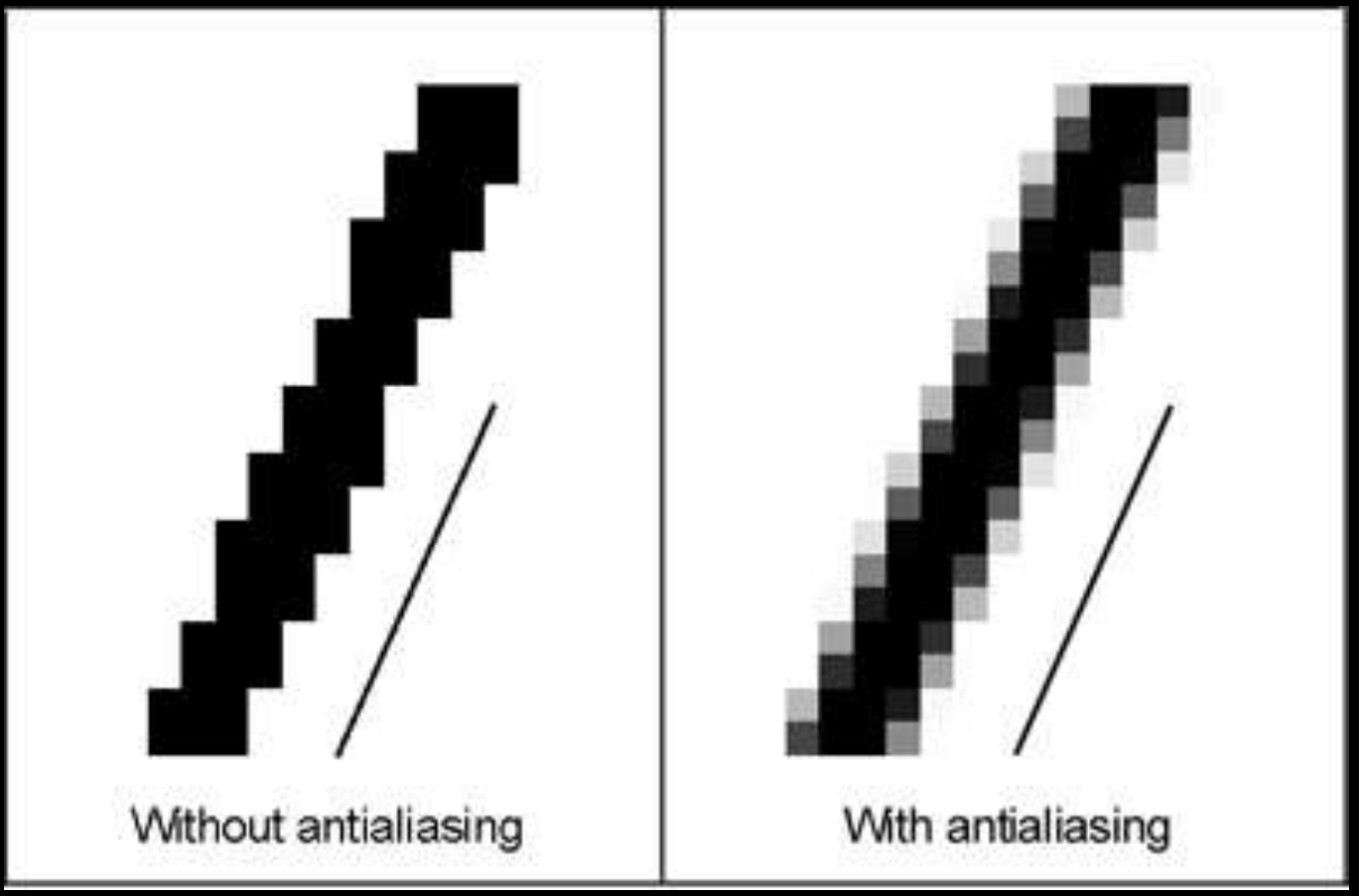
- The G-Buffer can only store data of opaque objects (only one value per position on screen is saved), meaning information about transparent and translucent objects can not be lit appropriately
- Transparent geometry must be rendered using the standard forward rendering technique
- Making lighting of transparencies extremely expensive!
- Many Anti aliasing solutions don’t work (more on this later)

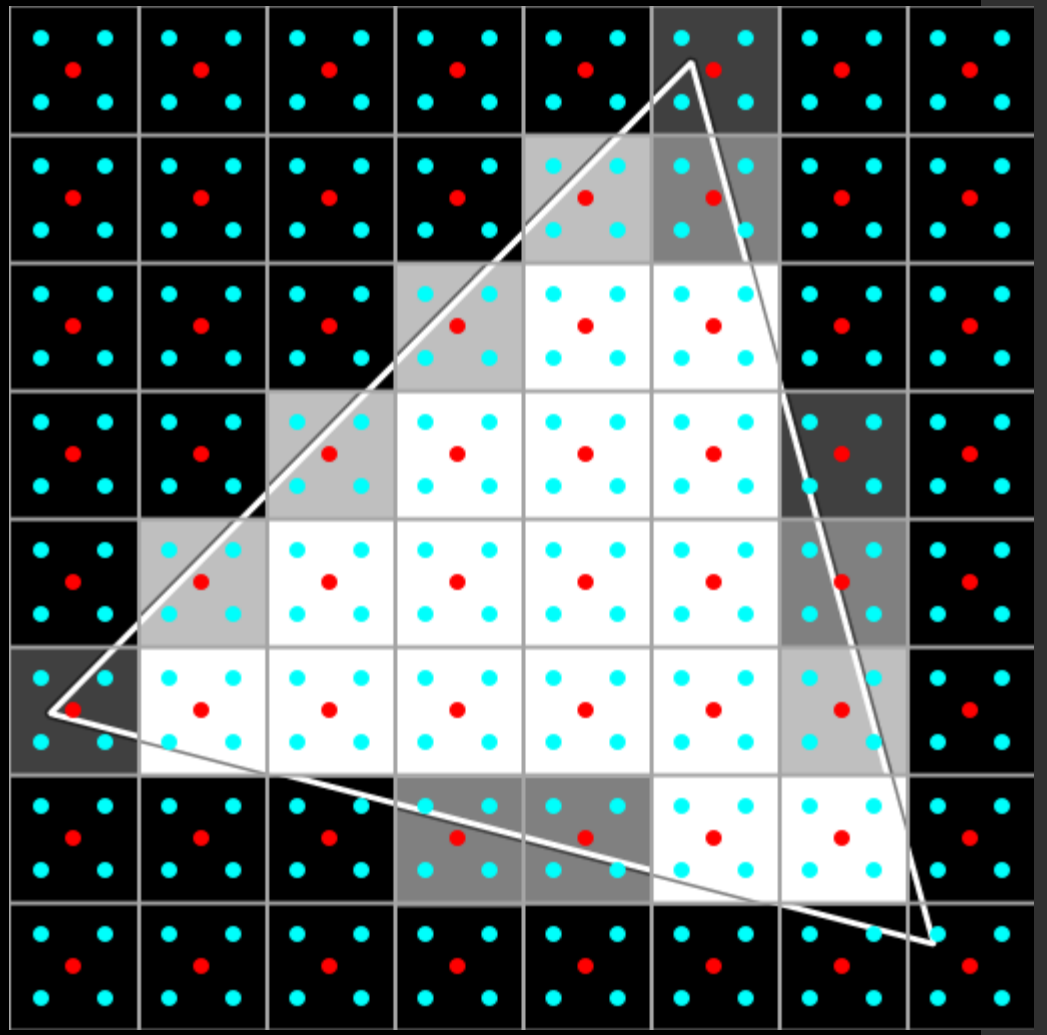
	Deferred	Forward
Performance		
Cost of a per-pixel Light	Number of pixels it illuminates	Number of pixels * Number of objects it illuminates
Number of times objects are normally rendered	1	Number of per-pixel lights
Overhead for simple scenes	High	None
Platform Support		
PC (Windows/Mac)	Shader Model 3.0+ & MRT	All
Mobile (iOS/Android)	OpenGL ES 3.0 & MRT	All
Consoles	XB1, PS4	All

Other Rendering Terms

Anti Aliasing

Technique used to smooth jagged edges on curved lines and diagonals


Without antialiasing
With antialiasing



SSAA or FSAA

- Super sampling anti-aliasing was the first type of anti-aliasing available
- Color samples are taken at several instances inside the pixel and an average color value is calculated
- To achieve this the image is rendered at a much higher resolution than the final one displayed
- The image is shrunk to the desired size, the extra pixels are used for the final color calculation
- The down sampled image has smoother transition
- The number of samples used to calculate the single final pixel color determines the quality of the output
- Uses lots of processing power

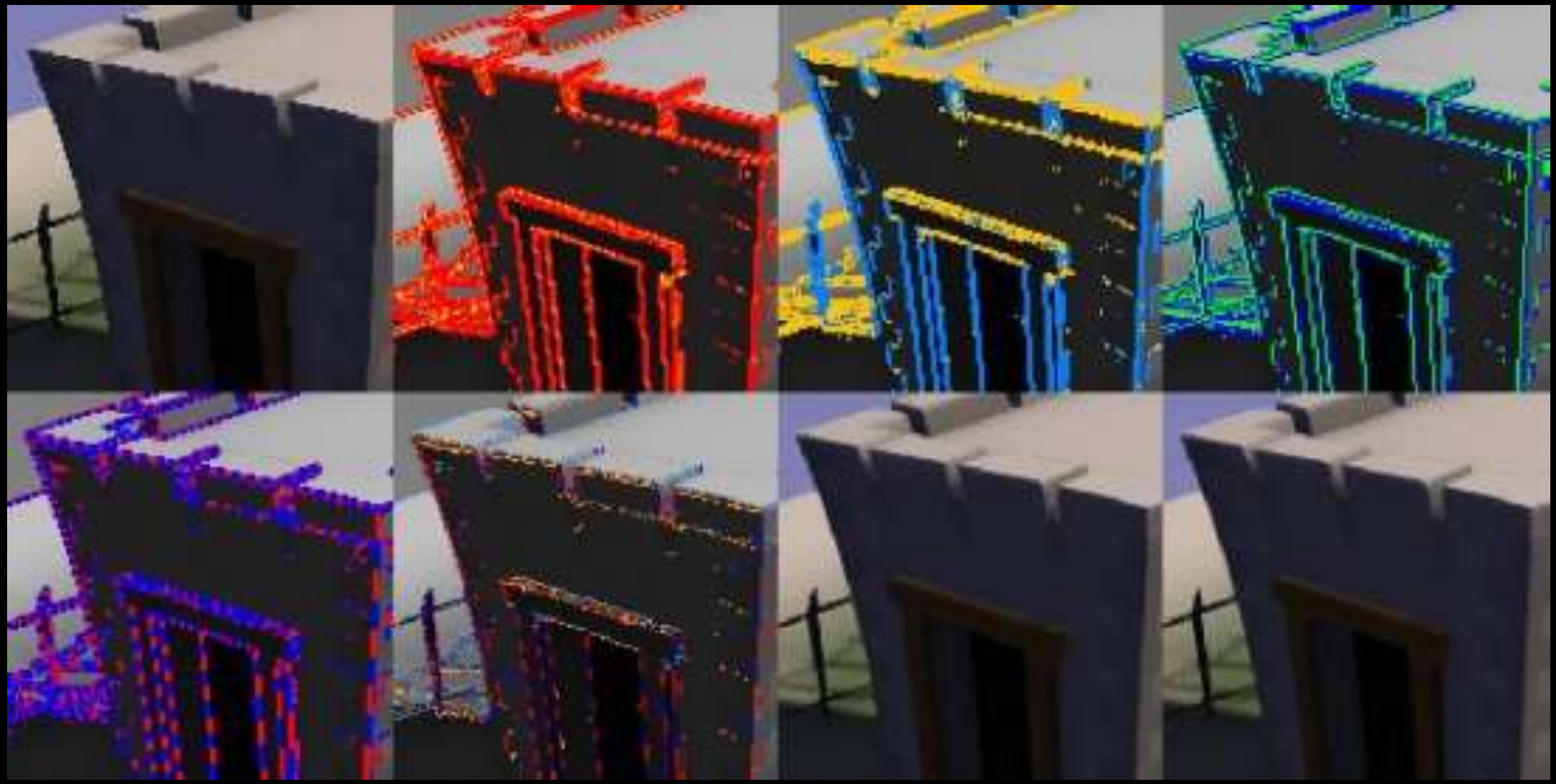
MSAA

- Multi-sample anti-aliasing is one of the more common types of anti-aliasing available in modern games
- Still a super sampling technique
- But only smooths out the edges of polygons, doesn't solve pixelated textures
- Uses less processing power then SSAA but still quite a lot
- Most common way to run a forward-rendered game
- MSAA does not work for a deferred renderer because lighting decisions are made after the MSAA calculations, resulting in lighting and shading that stills appear jagged

FXAA

- Fast approximate anti-aliasing
- Smooths out edges in all parts of the image
- Makes the image look blurry, which means it isn't ideal if you want crisp graphics
- Ignores polygons and line edges, and simply **analyzes the pixels on the screen**
- Where it sees pixels that create an artificial edge, it smooths them
- Smooths edges in all pixels on the screen, including those inside alpha-blended textures and those resulting from pixel shader effects, which other techniques ignore
- Very small performance cost

FXAA



No AA

4x MSAA

FXAA

Ambient Occlusion

Method to approximate how bright light should be shining on any specific part of a surface, based on the light and its environment

SSAO

- Screen space ambient occlusion
- The algorithm is implemented as a pixel shader that analyzes the screen depth buffer (stored as a texture in the G-buffer)
- For every pixel on the screen, the pixel shader samples the depth values around the current pixel and computes the amount of occlusion based on the depth difference between the sampled pixels and current pixels
- For runtime sampling of these pixels is decided randomly per frame
- This creates a noisy image that is then blurred

High-Dynamic-Range

Allows for the preservation of details that may be lost due to limiting contrast ratios

HDRR

- The motivation for High Dynamic Range Rendering allows bright things to be really bright, dark things to be really dark, and details that can be seen in both
- HDR in game engines means there is a larger value range for the rendering output (framebuffer)
- Typically this means 16 bit floating points that can go beyond the typically color range (vs 8 bit integers that only give 256 possible color values (per channel))
- The final output (what is displayed on the screen) uses a tone map to match colors (after light computation) in the HDR to the suitable counterpart in the LDR

Bidirectional Reflectance Distribution Function

A function of four real variables that defines how light is reflected at an opaque surface

BRDF

- A BRDF describes how much light is reflected when light makes contact with a certain material
- The appearance of light reflecting off a surface changes as your position viewing the surface changes
- It also changes when the position of the light changes
- BRDF is a function of incoming (light) direction and outgoing (view) direction relative to a local orientation at the light interaction point
- Also when light interacts with a surface, different wavelengths (colors) of light may be absorbed, reflected, and transmitted to varying degrees depending upon the physical properties of the material itself
- This means that a BRDF is also a function of wavelength

BRDF

- Light interacts differently with different regions of a surface
- Most real world materials are heterogeneous, diverse in the composition properties that the material is comprised of
- This property, known as positional variance is what allows light reflecting off of a surface to produces details in that surface

- Wood grain is a good example of this

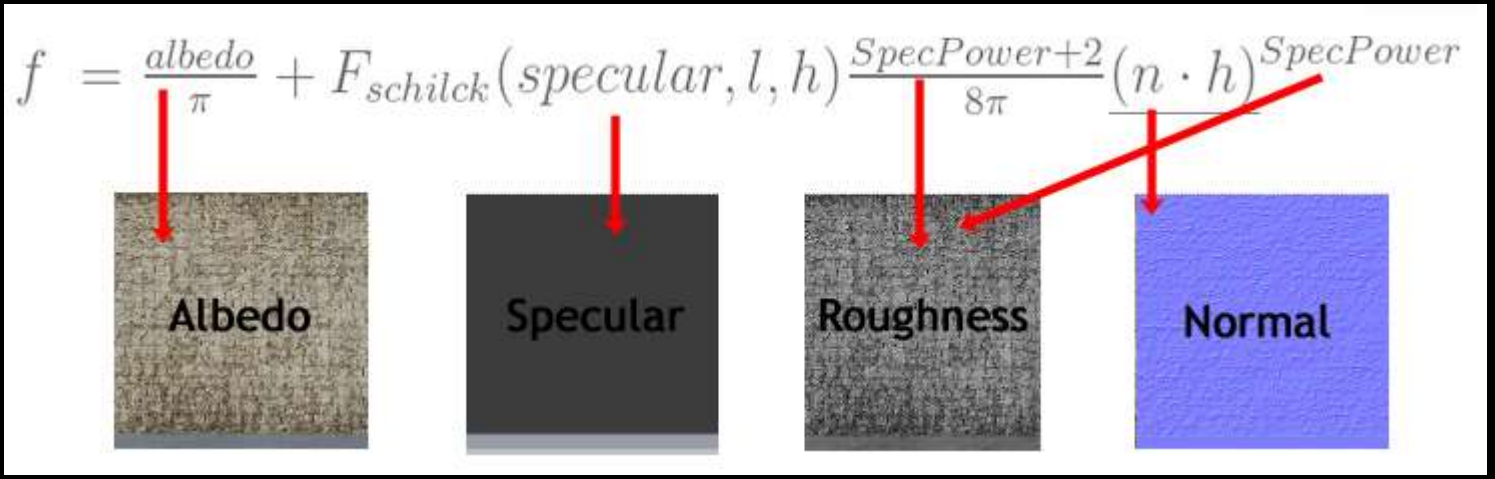
BRDF

- So what is BRDF a function of?
  - Incoming light direction
  - View direction
  - How much certain wavelengths are absorbs or reflected from a specific surface
  - The positional variance of the surface

BRDF

- So what is BRDF a function of?
  - Incoming light direction
  - View direction
  - How much certain wavelengths are absorbs or reflected from a specific surface
    - The “metal-ness”
  - The positional variance of the surface
    - The “roughness”

BRDF is the basis for PBR



Physically Based Rendering

Physically-based rendering (PBR) is a method of shading & rendering, used in order to provide a more accurate representation of the real (physics-based) world around us