
Computer Graphics

2 - Rendering Basics

Yoonsang Lee
Hanyang University

Spring 2025

Summary of Course Intro

- Questions
 - <https://www.slido.com/> - Join #cg-ys
- Quiz
 - <https://www.slido.com/> - Join #cg-ys - Polls
 - Your quiz answers must be submitted in the correct format to count for attendance.
 - Whether a submitted answer is correct or not has nothing to do with your attendance!
- Language
 - I'll “paraphrase” the explanation in Korean for most slides.
- **You MUST read "1 – Course Intro.pdf" CAREFULLY.**

Outline

- Basic Concepts for Rendering
- Rendering Approaches
 - Rasterization
 - Ray Tracing

Basic Concepts: Rendering

- *Rendering* is the process of **generating an image from a 2D or 3D model (scene)** by means of a computer program. [Wikipedia]
- Rendering output can be ...
 - saved as an image file,
 - or saved as a video file (consisting of many images),
 - or stored in *frame buffer* for display.



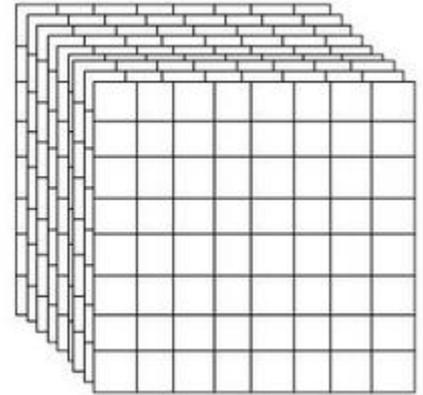
Encanto, 2021



Roblox

Basic Concepts: Frame Buffer

- *Frame buffer* is the portion of memory to hold the bitmapped image that is sent to the (raster) display device.
- A frame buffer is characterized by its width, height, and depth.
 - E.g. The frame buffer size for 4K UHD resolution with 32bit color depth = 3840 x 2160 x 32 bits
- Typically stored on the graphic card's memory.

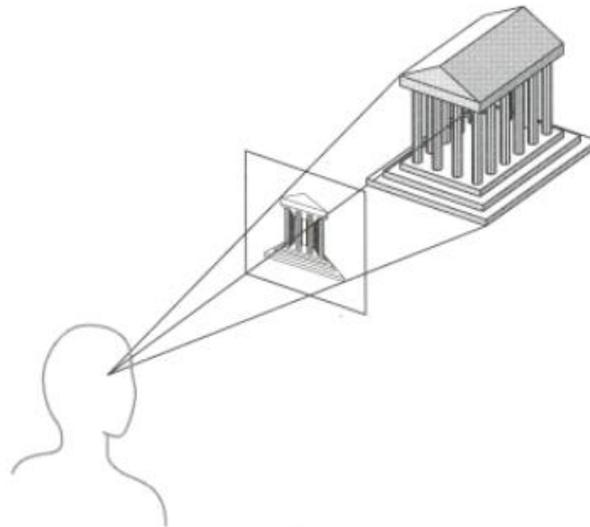


Basic Concepts: Double Buffering

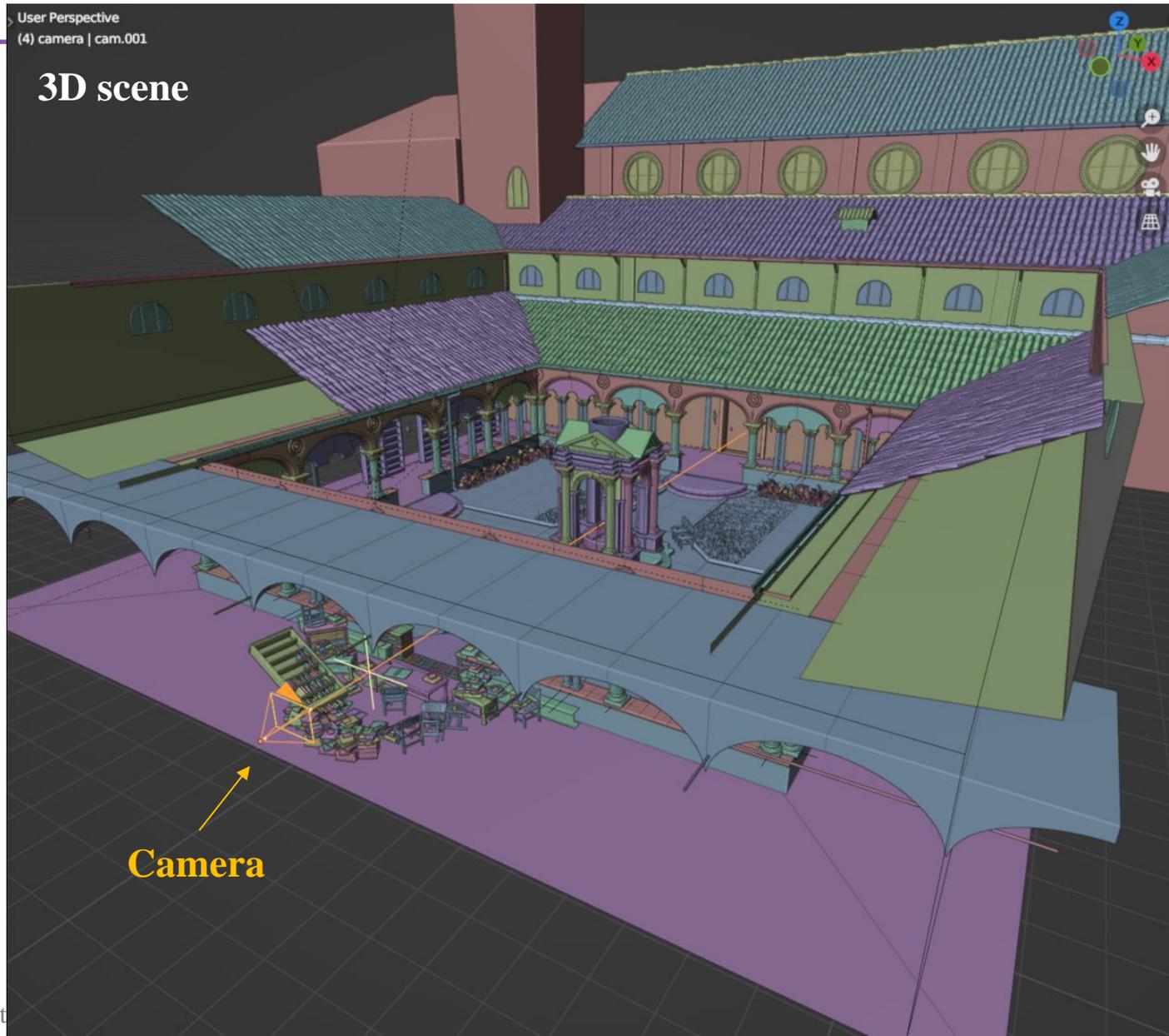
- Using two frame buffers for rendering and displaying:
 - Drawing a new image to the *back buffer* while displaying an image to the *front buffer*.
 - When drawing image data for one frame is done, **swap** front and back buffer.
- → Higher frame rate, no (or less) artifacts such as flickering
- Most graphics applications are working with double buffering.

Basic Concepts: Image Plane

- *Image plane* is the conceptual plane that represents the actual display screen through which a user views (a rendered image of) a virtual 3D scene.

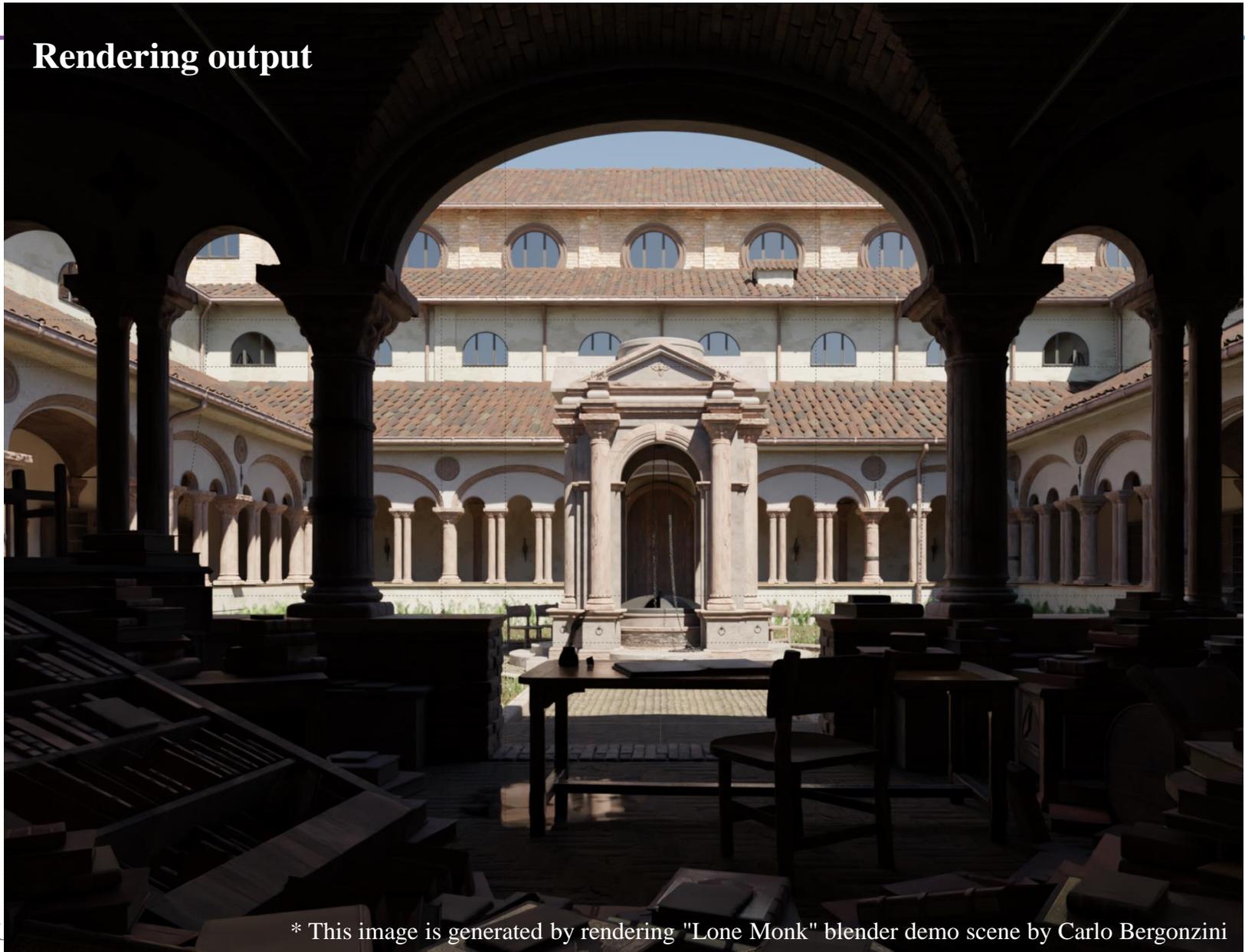


Example of Rendering a 3D Scene



Example of Rendering a 3D Scene

Rendering output



* This image is generated by rendering "Lone Monk" blender demo scene by Carlo Bergonzini

Render Output



- The result of rendering is a 2D image comprises of *picture elements* or *pixels*.
- That is, rendering is the process of **computing each pixel color** in the final image based on 3D scene information.

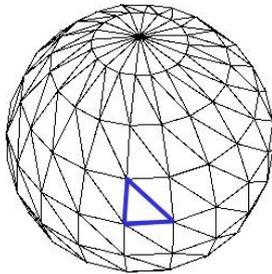
Rendering Approaches

- How to compute each pixel color?
- Rasterization
- Ray tracing
- Learnable scene representations:
 - NeRF: Learns implicit 3D scene representation.
 - Gaussian Splatting: Learns point-based 3D scene representation.

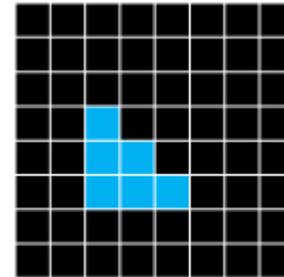
Rasterization

- *Primitive-by-primitive* approach
 - primitive: triangle, line, point, ...
- Each primitive determines which pixel in the image is affected and determines the color of that pixel.

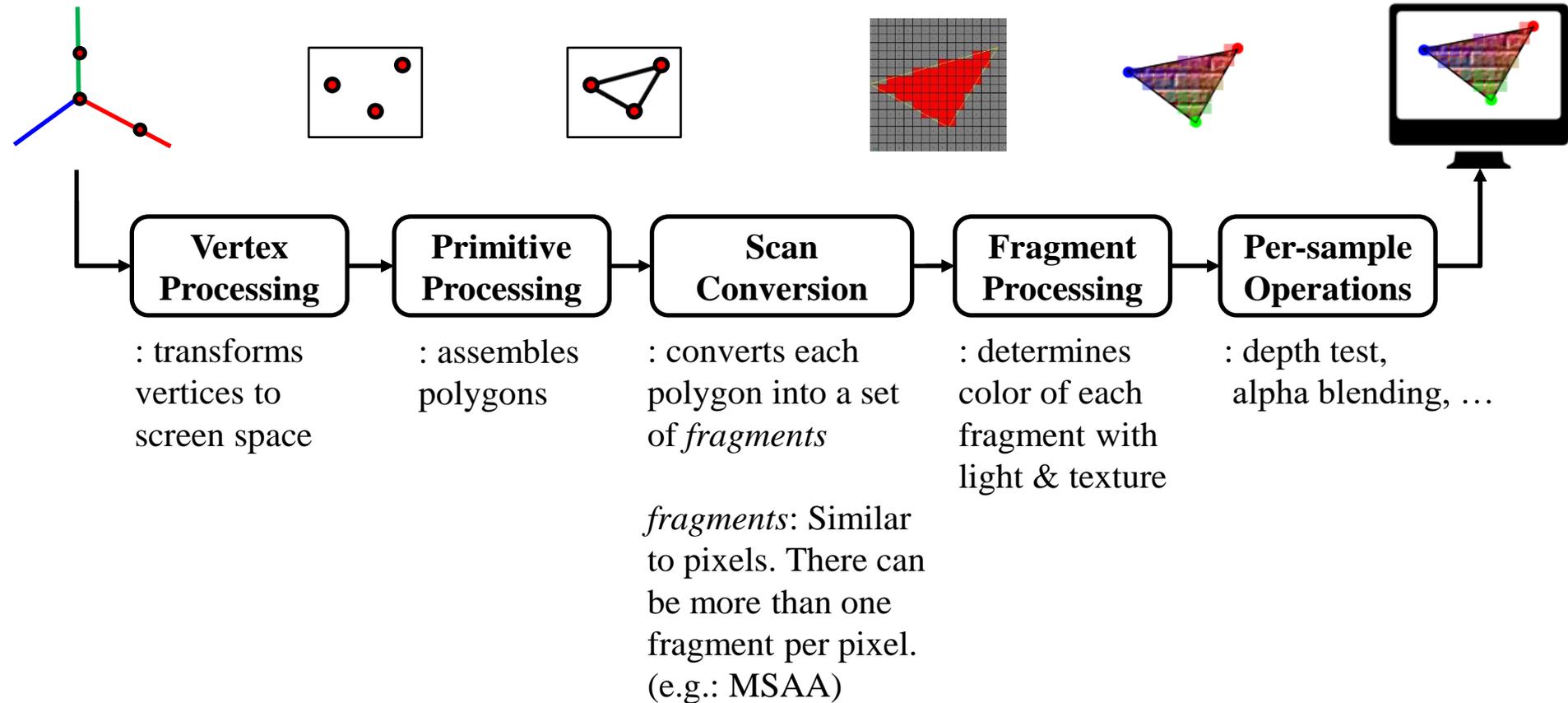
```
for each primitive in scene
  transform the primitive to viewport
  find pixels for the primitive
  set color of the pixels based on texture and lighting model
```



(triangle is rendered to screen)

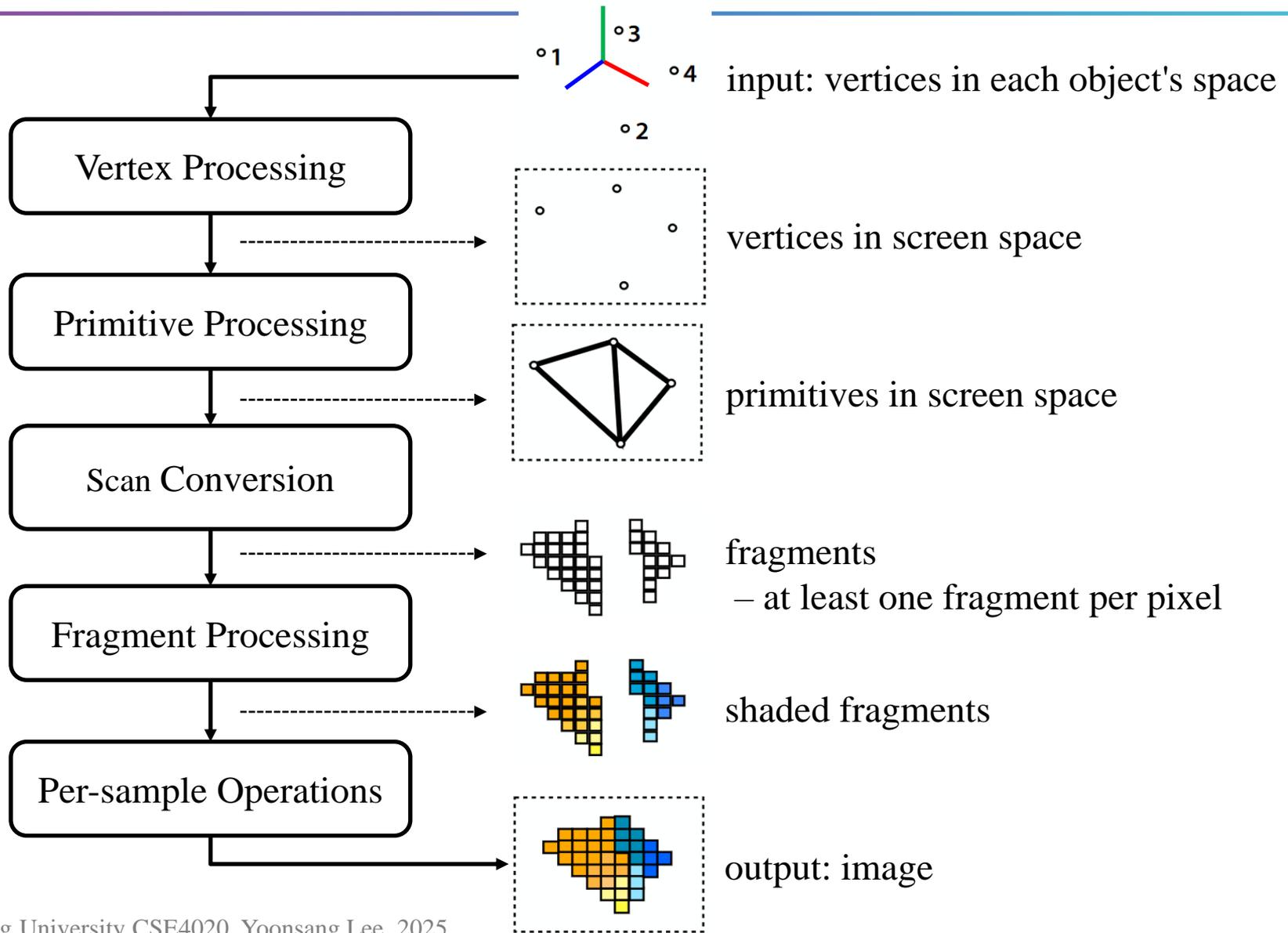


Rasterization Pipeline



- A.k.a. *rendering pipeline* or *graphics pipeline*.

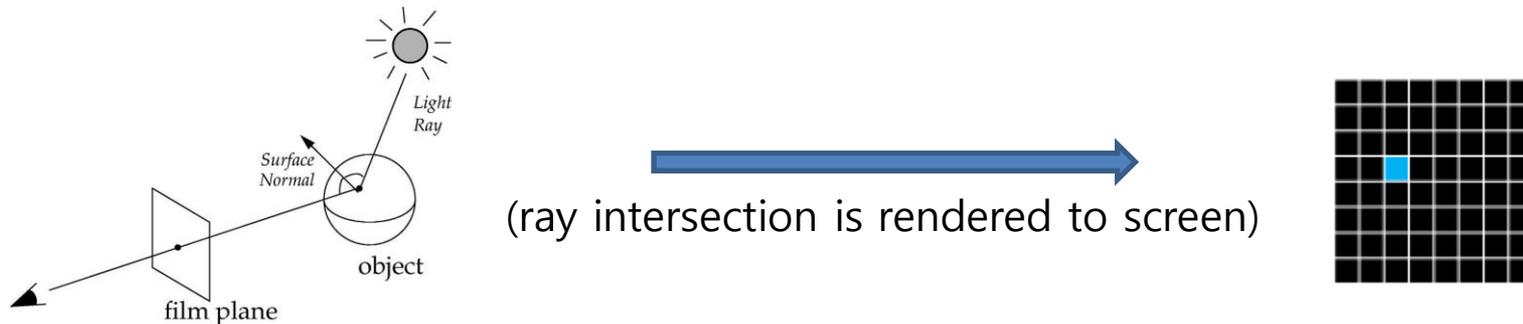
Rendering Pipeline again



Ray Tracing

- *Pixel-by-pixel* approach
- Each ray goes through each pixel in image plane from eye position.
- Color of each pixel is determined based on which object the ray intersects with.

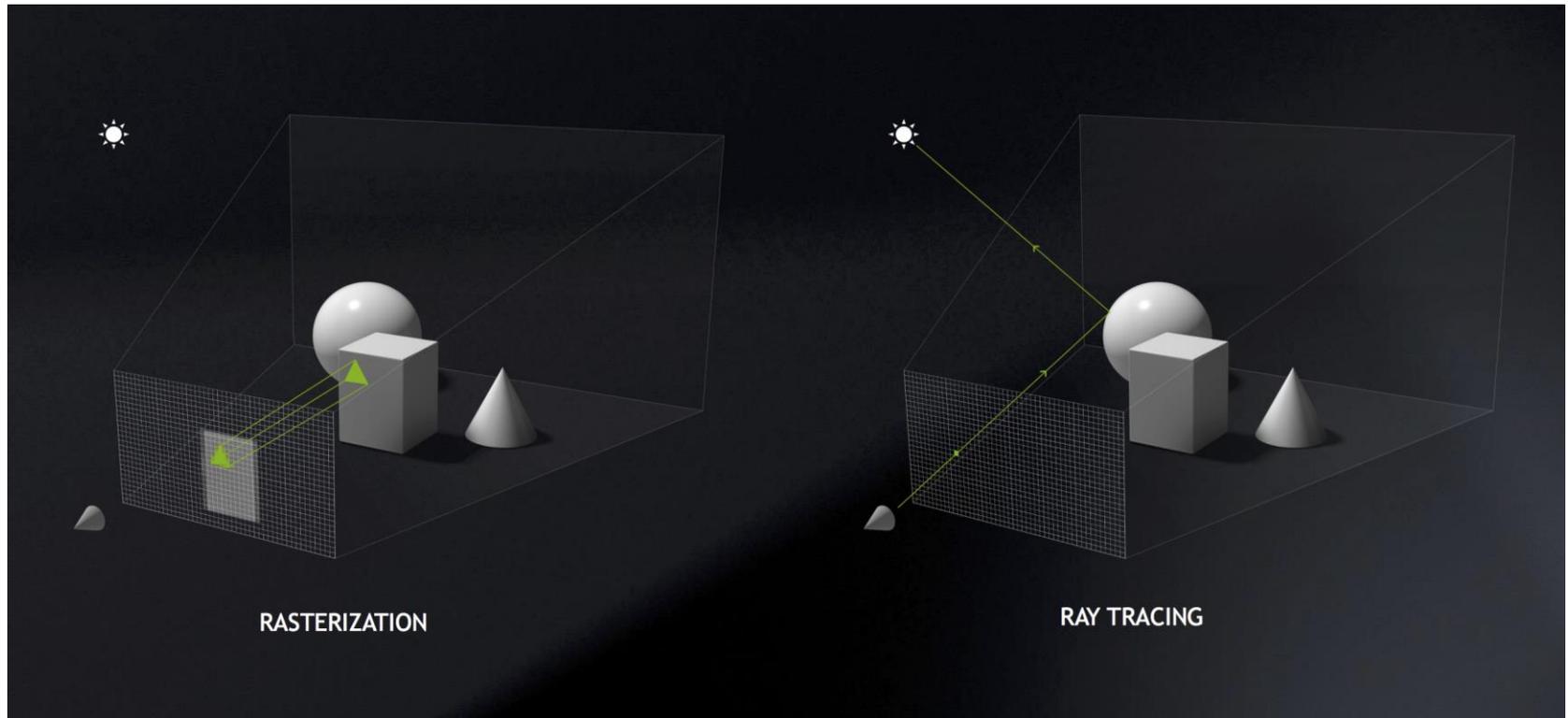
for **each pixel** in image(plane)
determine which object should be shown at the pixel
set color of the pixel based on texture and lighting model



Ray Tracing

- If you're interested in the details of ray tracing, refer to ‘Ray Tracing in One Weekend’:
 - <https://raytracing.github.io/books/RayTracingInOneWeekend.html>

Rasterization vs. Ray Tracing



Rasterization – Pros & Cons

- Pros
 - Just render stream of triangles – no need to keep entire scene data
 - Good for parallelism → Fast!
- Cons
 - No unified processing of shadows, reflection, transparency
 - May produce lower-quality results
- Traditionally used for real-time applications
 - e.g. Games using OpenGL or DirectX

Ray Tracing – Pros & Cons

- Pros
 - Generalized way of handling shadows, reflections, transparency – just intersection with a ray
 - Often produce higher-quality results
- Cons (of the traditional view)
 - Too slow for real-time applications
 - Hard to implement in hardware
- Traditionally used for offline rendering for films
 - e.g. Animation films produced using 3D authoring tools such as Maya, Blender, etc

Recent Ray Tracing Technology

- However, they are not as true anymore as they used to be.
 - Slower than rasterization, but not too slow for real-time.
 - Harder than rasterization, but possible to implement in hardware.
- Now you can see...
 - Hardware such as Nvidia RTX series
 - API such as DirectX Raytracing, Vulkan RT, ...
- This is a change not too long ago.
 - The first real-time raytracing demo "*Reflections*" was released in March 2018.
 - <https://youtu.be/IMSuGoYcT3s>



In This Course,

- The lectures focus primarily on the fundamental concepts of computer graphics that are *common to all rendering methods*.
 - **Movement & placement:** Transformations, Hierarchical Modeling, Orientation & Rotation, Kinematics & Animation, Curves
 - **Shape & appearance:** Mesh, Lighting, Texture Mapping, Curves

In This Course,

- Some lectures cover the fundamental concepts that are specific to *rasterization*.
 - **Mapping to 2D screen in rasterization:** Viewing / Projection / Viewport transformations
 - **Appearance in rasterization:** Polygon Shading
 - **Rasterization process:** Rasterization Pipeline, Scan Conversion & Visibility
- The labs cover modern OpenGL, which is still one of the most popular *rasterization* APIs.
 - Modern OpenGL is used as a tool to review the concepts learned in lectures.

In This Course,

- This course does not cover ray tracing or learnable scene representations, because
 - Rasterization is still crucial in real-time rendering.
 - Not enough time to cover all.

Lab Session

- Now, let's start the lab today.