
Computer Graphics

2 - Rendering Basics

Yoonsang Lee
Hanyang University

Spring 2026

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
 - Lectures will be conducted in English.
 - Please ask questions if anything is unclear. Answers will be given in both English and Korean.
- **You MUST read "1 – Course Intro – 2026-v2.pdf" CAREFULLY.**

Outline

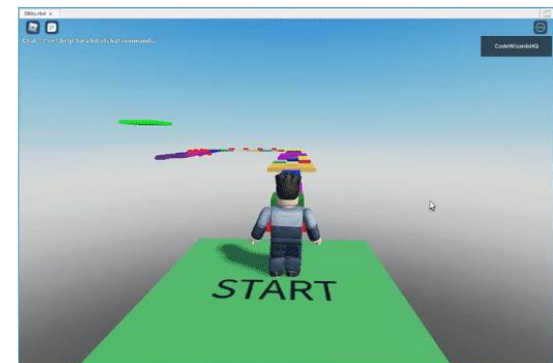
- Basic Concepts for Rendering
- Rendering Approaches
 - Rasterization
 - Ray Tracing

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.



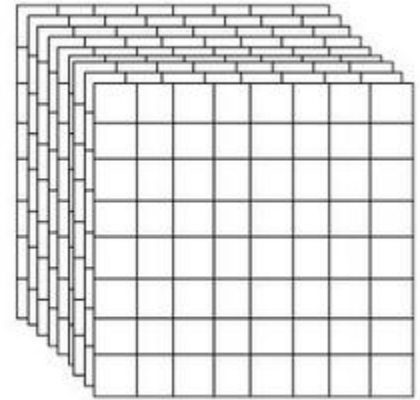
Encanto, 2021



Roblox

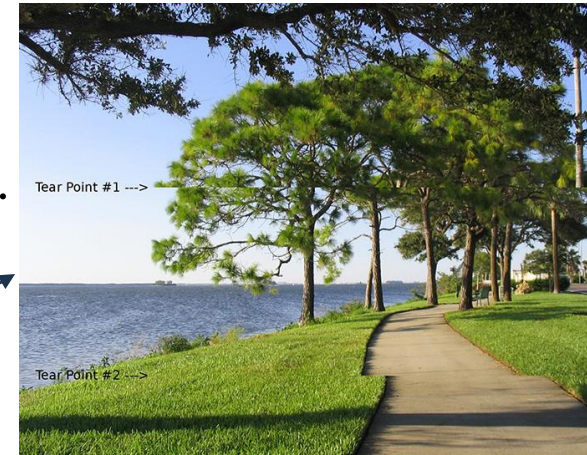
Basic Concepts: Frame Buffer

- *Frame buffer* is the portion of memory that stores the bitmapped image sent to the 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 = $3840 \times 2160 \times 32$ bits (≈ 32 MB)
- Typically stored in GPU memory (VRAM).



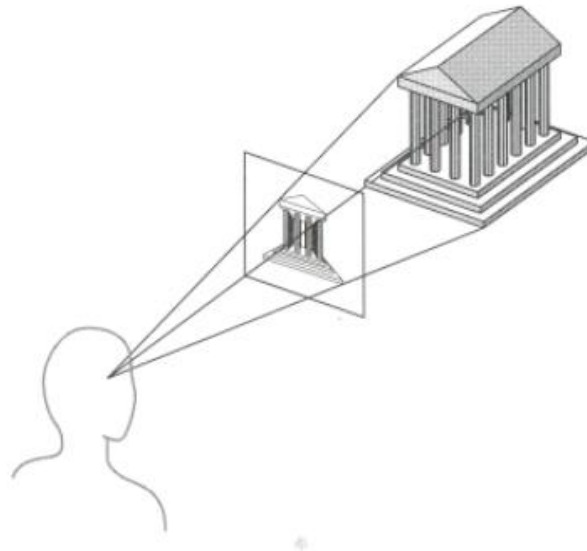
Basic Concepts: Double Buffering

- Single Buffering
 - Rendering happens directly in the frame buffer.
 - The screen may show **partially rendered images**.
 - → visual artifacts (e.g., flickering, tearing)
- Double Buffering
 - Two frame buffers are used.
 - **Front buffer**: frame currently displayed
 - **Back buffer**: frame being rendered
 - When rendering finishes → **swap buffers**
 - The display always shows a **complete frame** → Fewer visual artifacts
 - Rendering and display can happen **in parallel** → Higher frame rate
- Most graphics applications use double buffering.

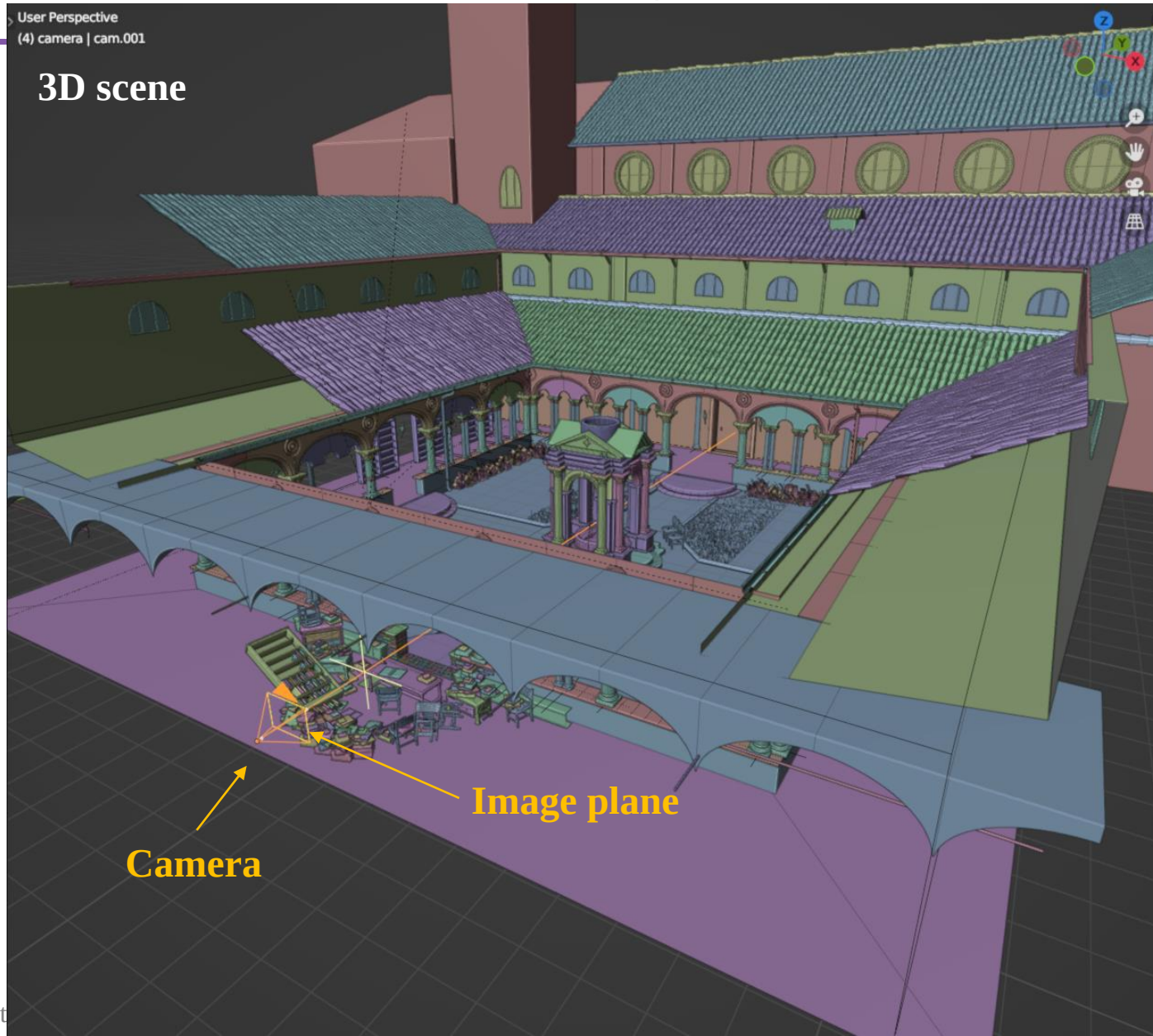


Basic Concepts: Image Plane

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

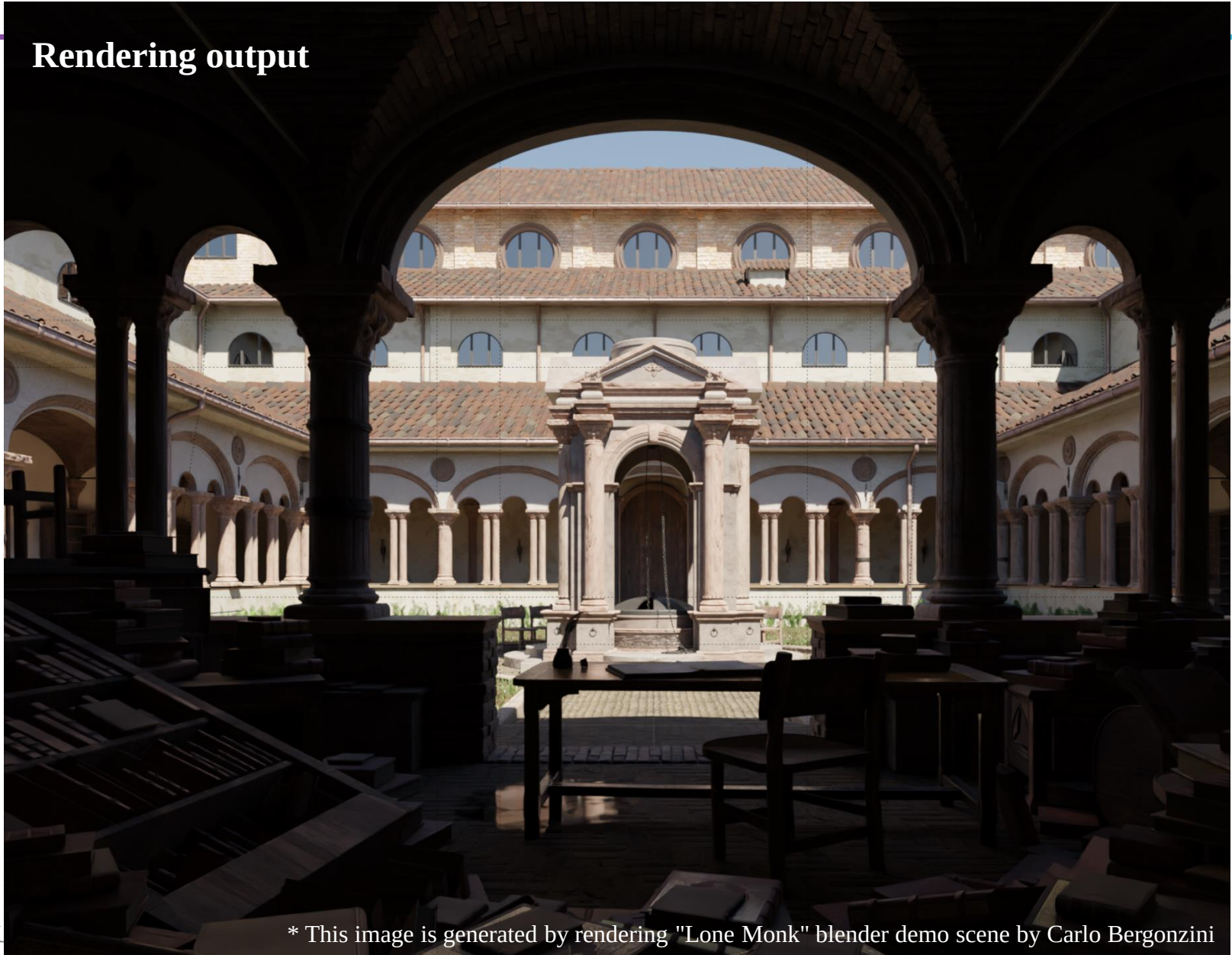


Example of Rendering a 3D Scene



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.

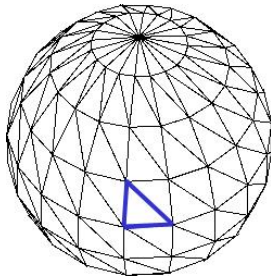
Rendering Approaches

- How do we compute the color of each pixel?
- Rasterization (1970s)
- Ray tracing (1980s)
- Learnable scene representations:
 - NeRF (2020): Learns an implicit 3D scene representation.
 - Gaussian Splatting (2023): Learns a point-based 3D scene representation.

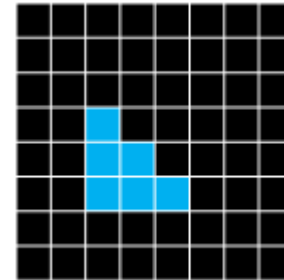
Rasterization

- *Primitive-by-primitive* approach
 - primitives: triangle, line, point, ...
- Each primitive determines which pixels it covers, and the pixel colors are computed based on object properties and lighting.

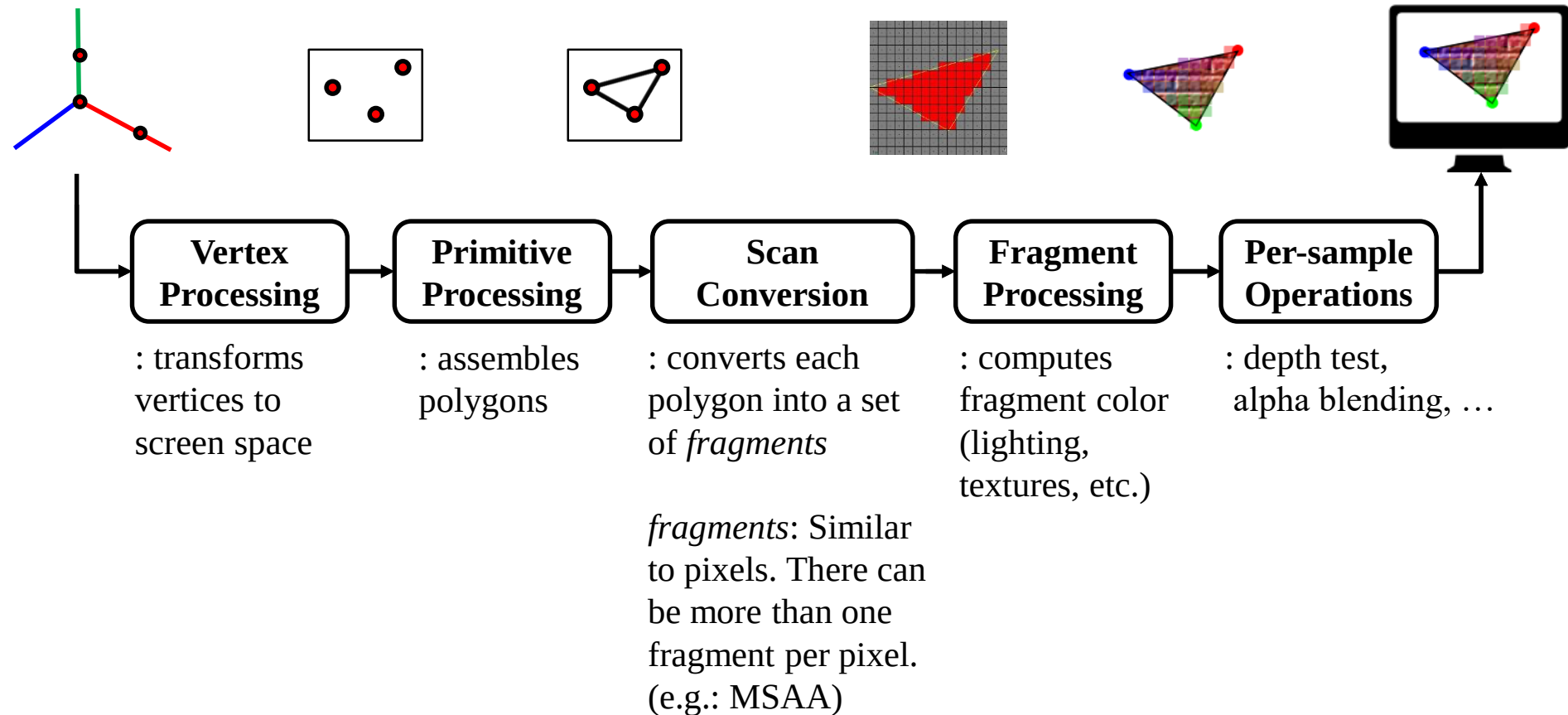
for **each primitive** in scene
transform the primitive to screen space
find pixels covered by the primitive
compute pixel colors based on material and lighting



(triangle is rendered to screen)

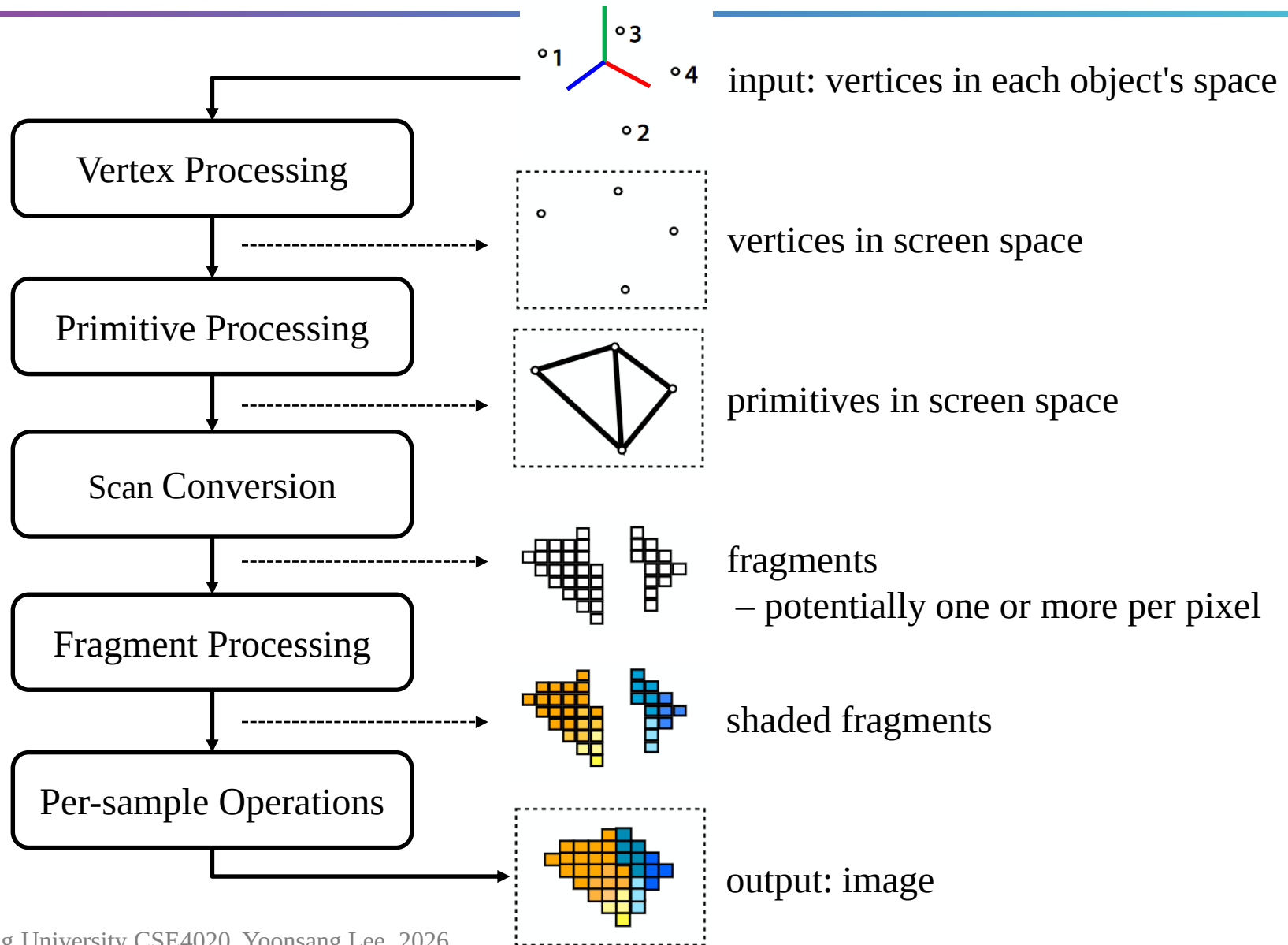


Rasterization Pipeline



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

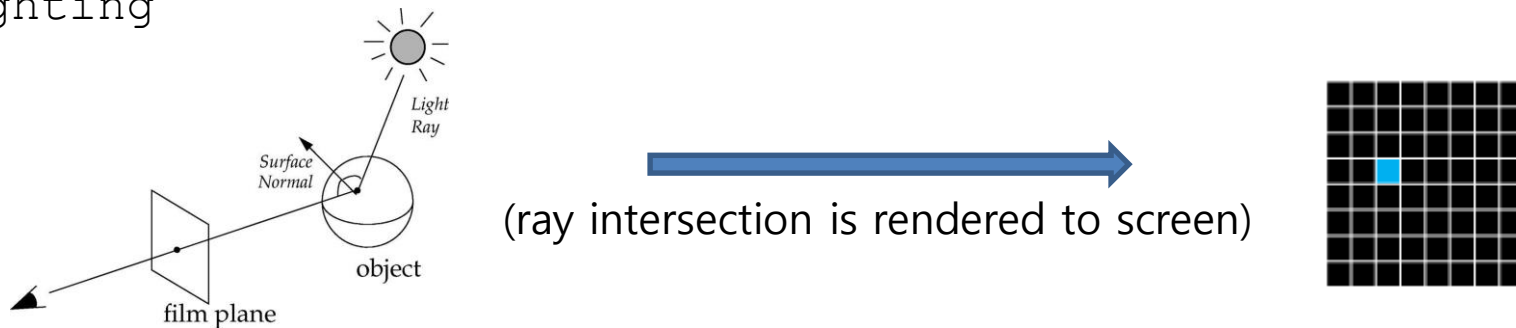
Rendering Pipeline again



Ray Tracing

- *Pixel-by-pixel* approach
- A ray is cast from the eye through each pixel on the image plane.
- The color of each pixel is determined by the first object the ray intersects and the lighting at that point.

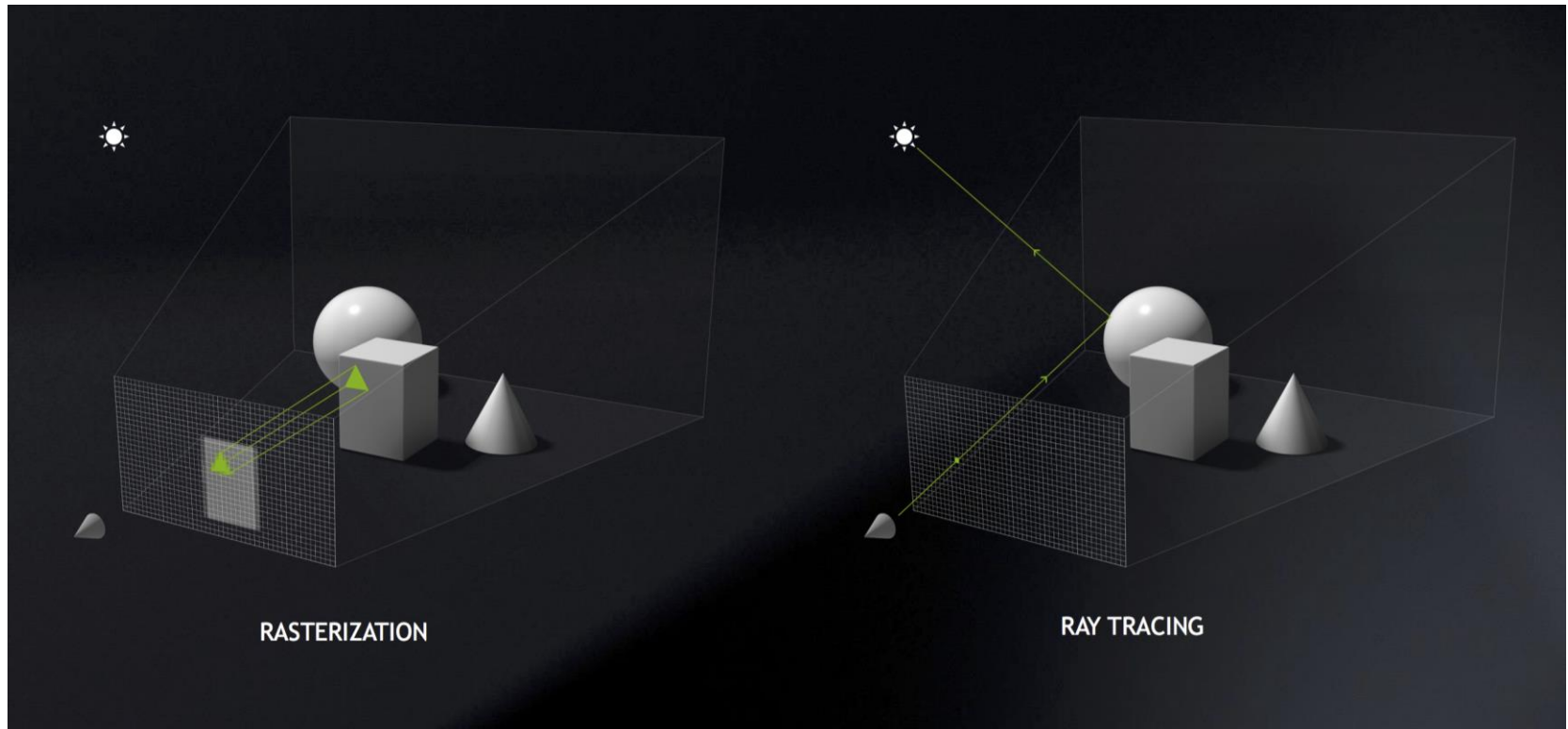
for **each pixel** in image plane
cast a ray into the scene
find the closest intersection
compute pixel color at the intersection using material and lighting



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
 - Works on a stream of primitives (triangles) – no need to store or traverse the entire scene structure
 - Good for parallelism → Fast!
- Cons
 - Shadows, reflections, transparency require special techniques
 - May produce lower-quality results
- Traditionally used for real-time applications
 - e.g. Games using OpenGL or DirectX

Ray Tracing – Pros & Cons

- Pros
 - Generalized handling of shadows, reflections, and transparency – based on ray–scene intersections
 - Often produce higher-quality results
- Cons (traditionally)
 - Too slow for real-time applications
 - Difficult to implement efficiently 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, this view is changing.
 - Still slower than rasterization, but increasingly feasible for real-time.
 - Harder to implement efficiently in hardware than rasterization, but now becoming practical.
- Examples include:
 - Hardware such as NVIDIA RTX GPUs
 - API such as DirectX Raytracing (DXR) and Vulkan RT
- This change happened not long ago.
 - The first real-time raytracing demo "*Reflections*" was released in March 2018.
 - <https://youtu.be/lMSuGoYcT3s>



In This Course,

- The lectures focus on the fundamental concepts that are *common to all rendering methods*.
 - **Movement & placement:** Transformations, Hierarchical Modeling, Kinematics & Animation
 - **Shape & appearance:** Meshes, Lighting, Texture Mapping, Curves
- These concepts apply to both rasterization and ray tracing.

In This Course,

- Some lectures focus on *rasterization-specific* concepts.
 - **2D screen mapping in rasterization:** Viewing / Projection / Viewport transformations
 - **Appearance in rasterization:** Polygon Shading
 - **Rasterization process:** Rasterization Pipeline, Scan Conversion & Visibility
- Labs cover modern OpenGL, which is one of the most widely used *rasterization* APIs.
 - Modern OpenGL is used to reinforce the concepts learned in lectures.

In This Course,

- This course focuses on rasterization rather than ray tracing or learnable scene representations:
 - Rasterization remains the core technique in real-time rendering.
 - Limited time to cover all topics in computer graphics.
- Ray tracing and learnable scene representations are important topics,
but they are beyond the scope of this course.

Lab Session

- Now, let's start the lab today.