Computer Graphics

2 - Rendering Basics

Yoonsang Lee
Hanyang University

Spring 2023

Summary of Course Intro

- Questions
  - https://www.slido.com/ - Join #cg-ys

- Quiz
  - https://www.slido.com/ - Join #cg-ys - Polls
  - You must submit all quiz answers in the correct format to receive points.
  - Whether a submitted answer is correct or not has nothing to do with your quiz score!

- 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]

Encanto, 2021

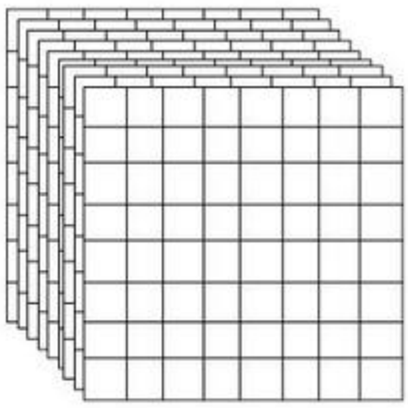
- Rendering output can be …
  – saved as an image file,
  – or saved as a video file (consisting of many images),
  – or stored in *frame buffer* for display.

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.
  - But integrated graphics (e.g. Intel HD Graphics) use the main memory to store the frame buffer.

Basic Concepts: Double Buffering

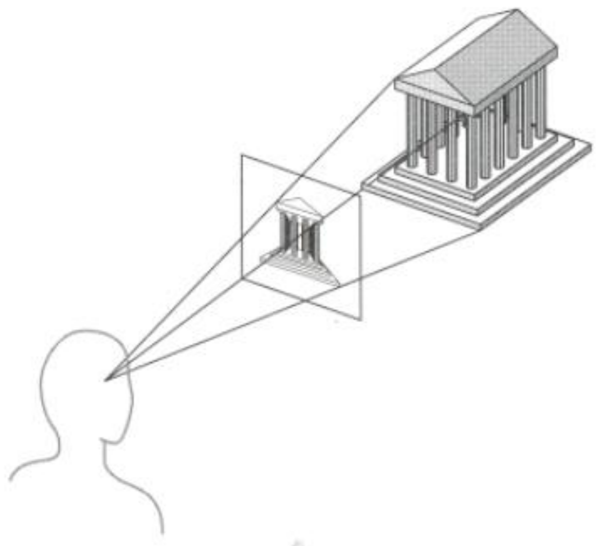
- Using two frame buffers for rendering and displaying:
  - Display image data in ***front buffer***
  - Draw new image data to ***back buffer***
  - When drawing image data for one frame is done, **swap** front and back buffer.

- Allows drawing a new image to the *back buffer* while displaying an image to the *front buffer*.
  → Higher frame rate, no (or less) artifacts such as flickering

- Most graphics applications are working with double buffering.

Basic Concepts: Image Plane

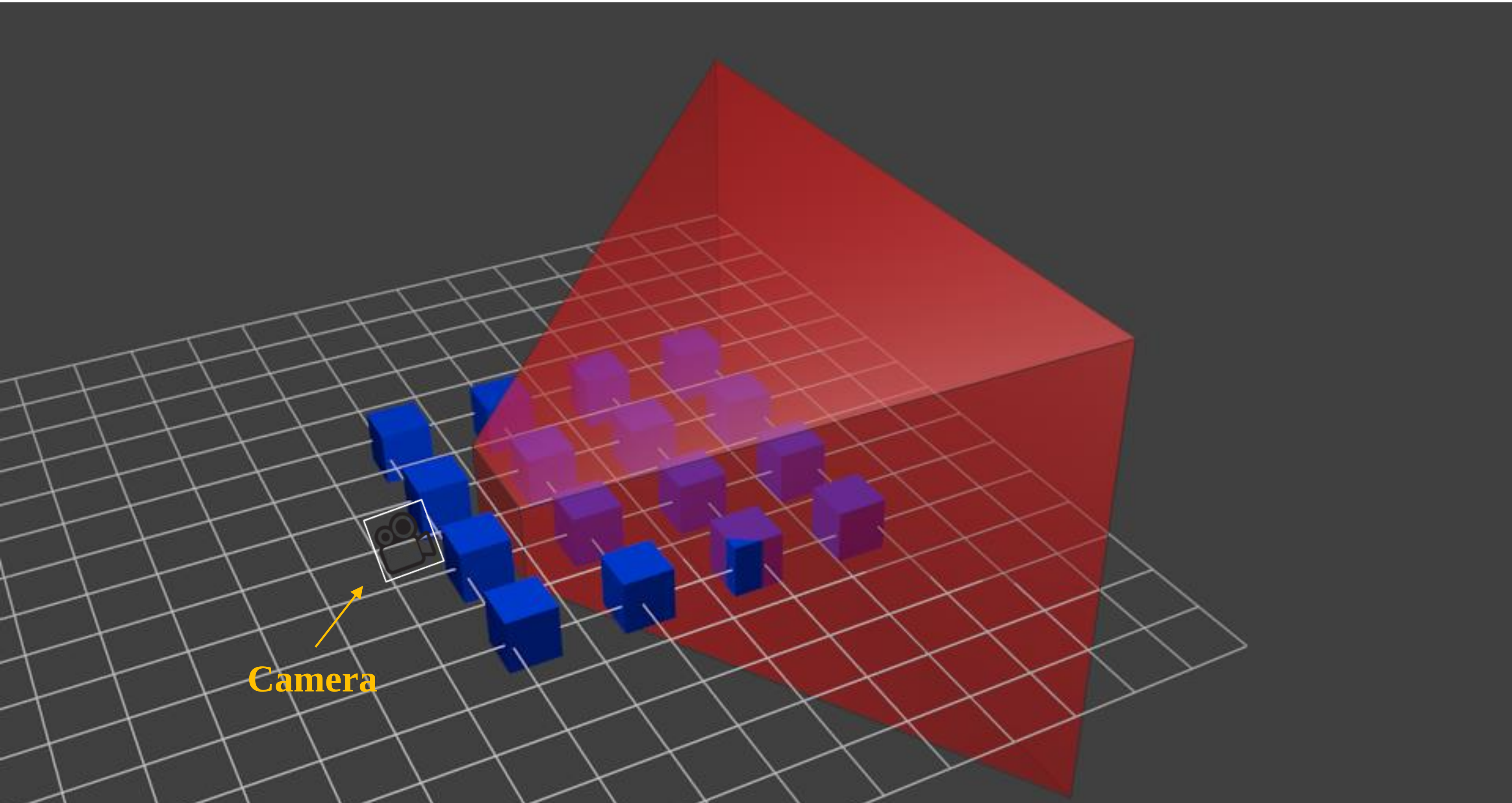
- *Image plane* is **the conceptual plane that represents the actual display screen** through which **a user views (a rendered image of) a virtual 3D scene**.



Example of Rendering a 3D Scene - 1

3D scene

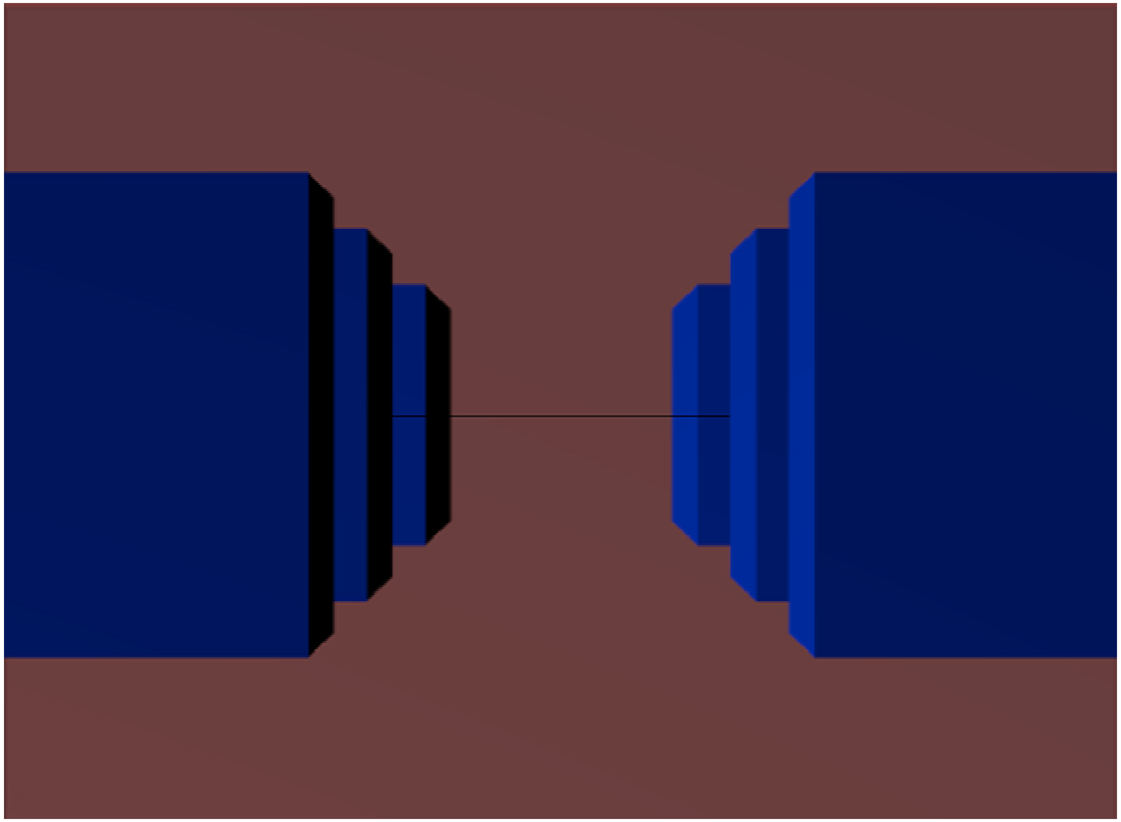
Red: view volume, Blue: objects



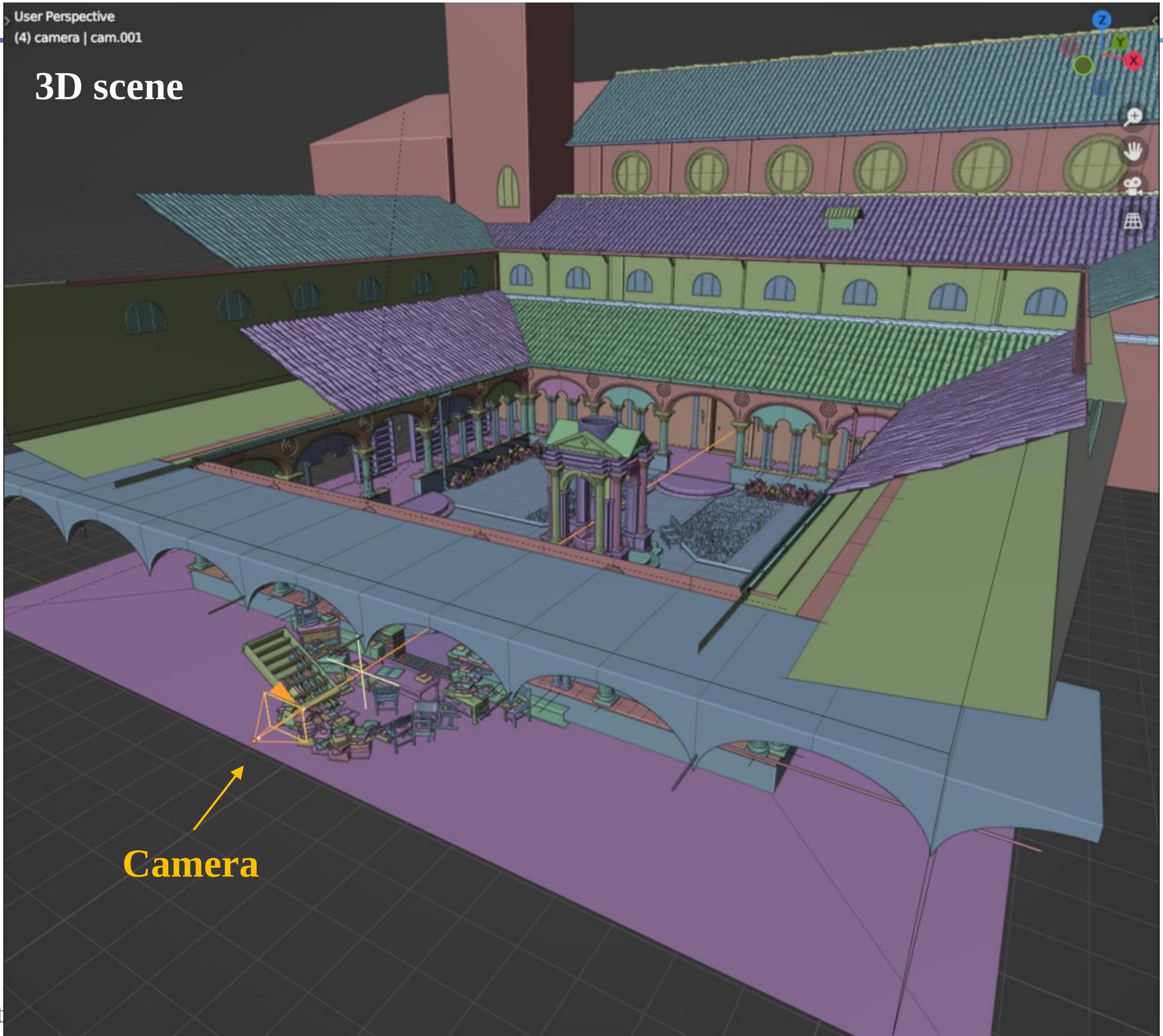
* This image is from http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/

Example of Rendering a 3D Scene - 1

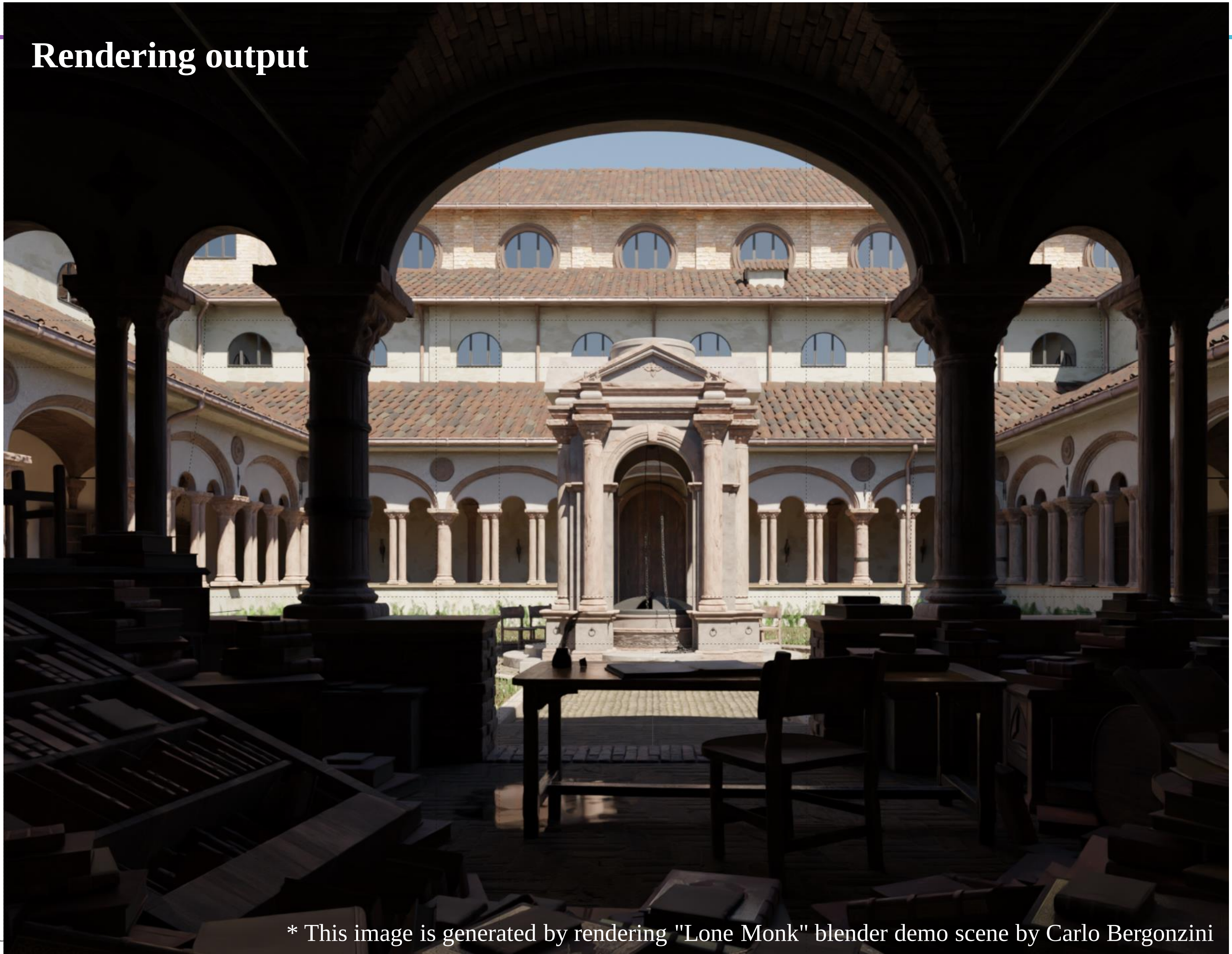
Rendering output



Example of Rendering a 3D Scene - 2



Example of Rendering a 3D Scene 2



* 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

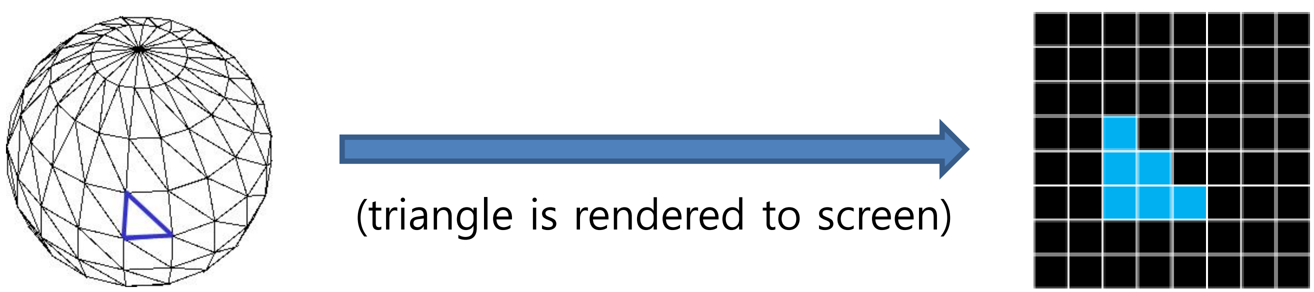
- Ray tracing

- Recent emerging approach:
  - Neural rendering: Use deep neural networks to learn representation of scenes (e.g. NeRF)

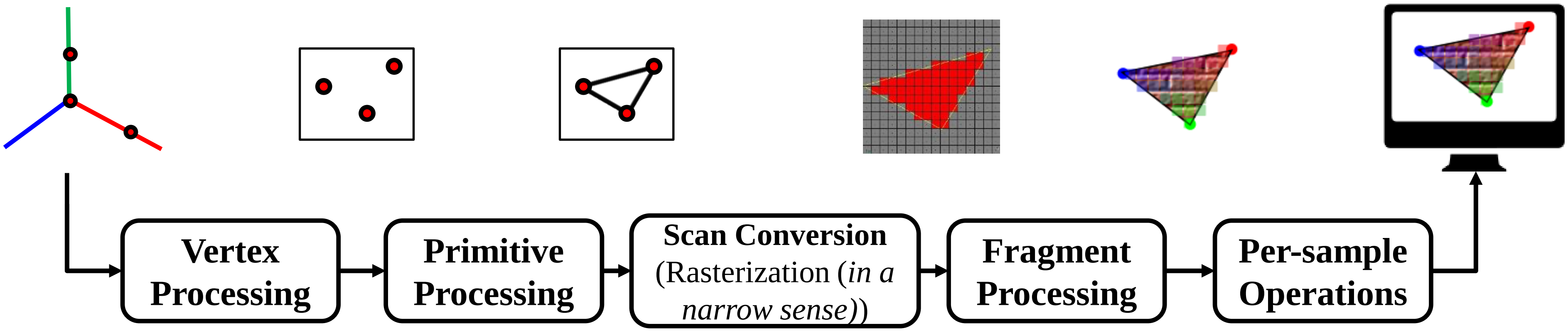
Rasterization

- *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
```



Rasterization Pipeline



Vertex Processing: transforms vertices to screen space

Primitive Processing: assembles polygons

Scan Conversion: converts each polygon into a set of *fragments*

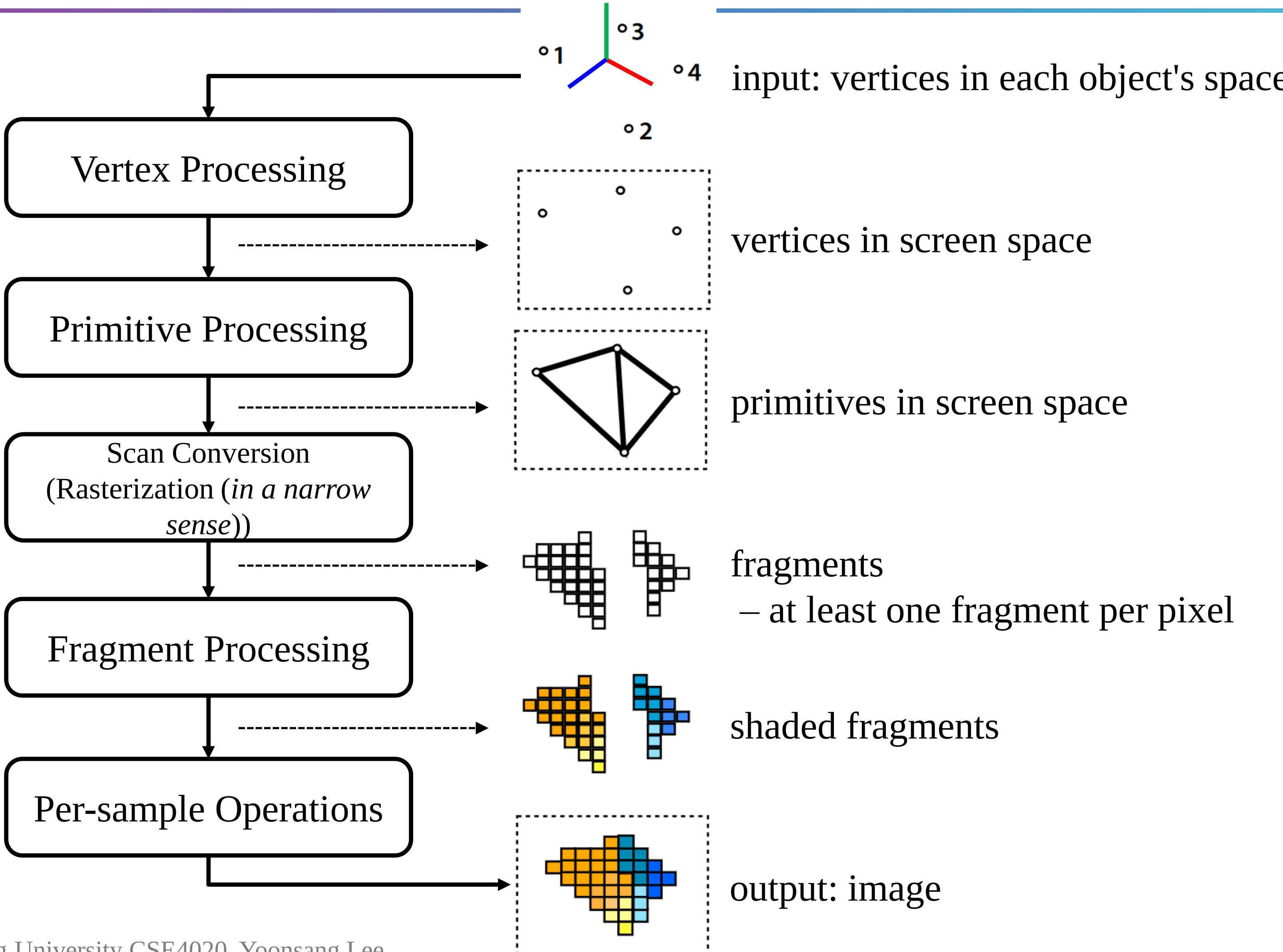
fragments: Similar to pixels. There can be more than one fragment per pixel. (e.g.: MSAA)

Fragment Processing: determines color of each fragment with light & texture

Per-sample Operations: depth test, alpha blending, …

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

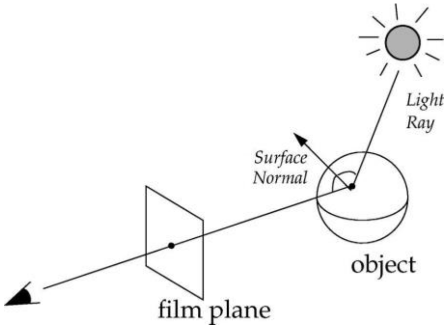
Rendering Pipeline again



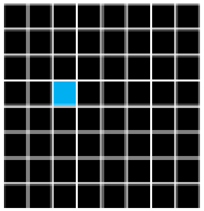
Ray Tracing

- *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.

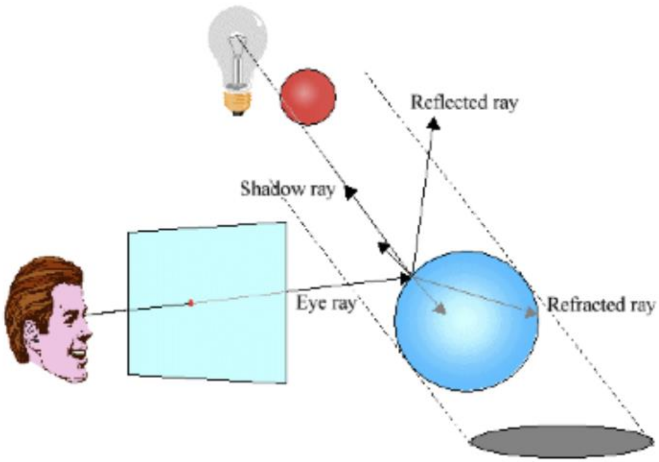
```
for each pixel in image(plane)
    determine which object should be shown at the pixel
    set color of the pixel based on texture and lighting model
```



(ray intersection is rendered to screen)



Types of Rays

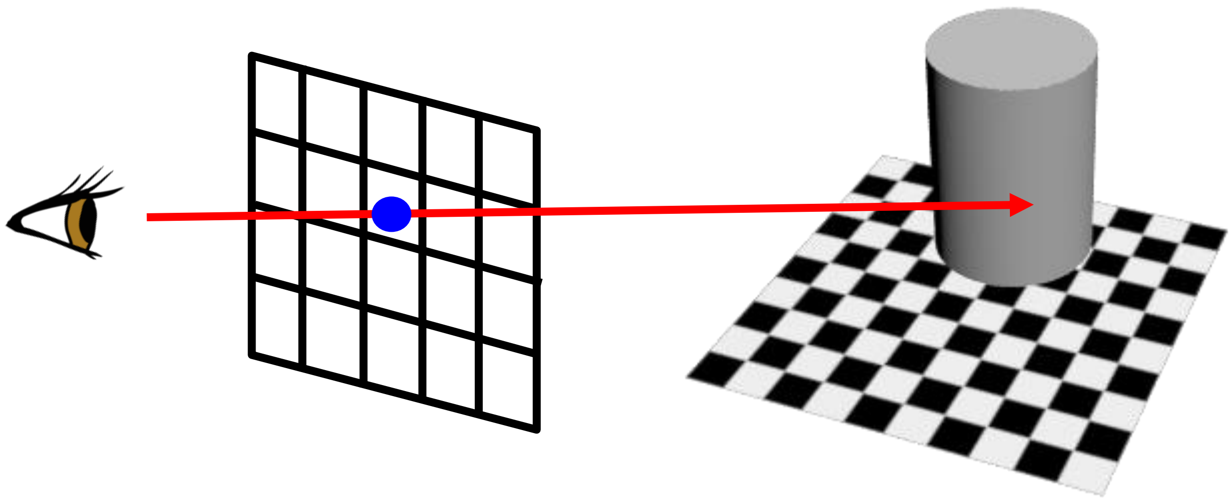


- Eye rays
  - from eye to surface, passing through each pixel
- Shadow (Illumination) rays
  - from surface point to light source
- Reflection rays
  - from surface point in mirror direction
- Refraction rays
  - from surface point in refracted direction

Eye Rays

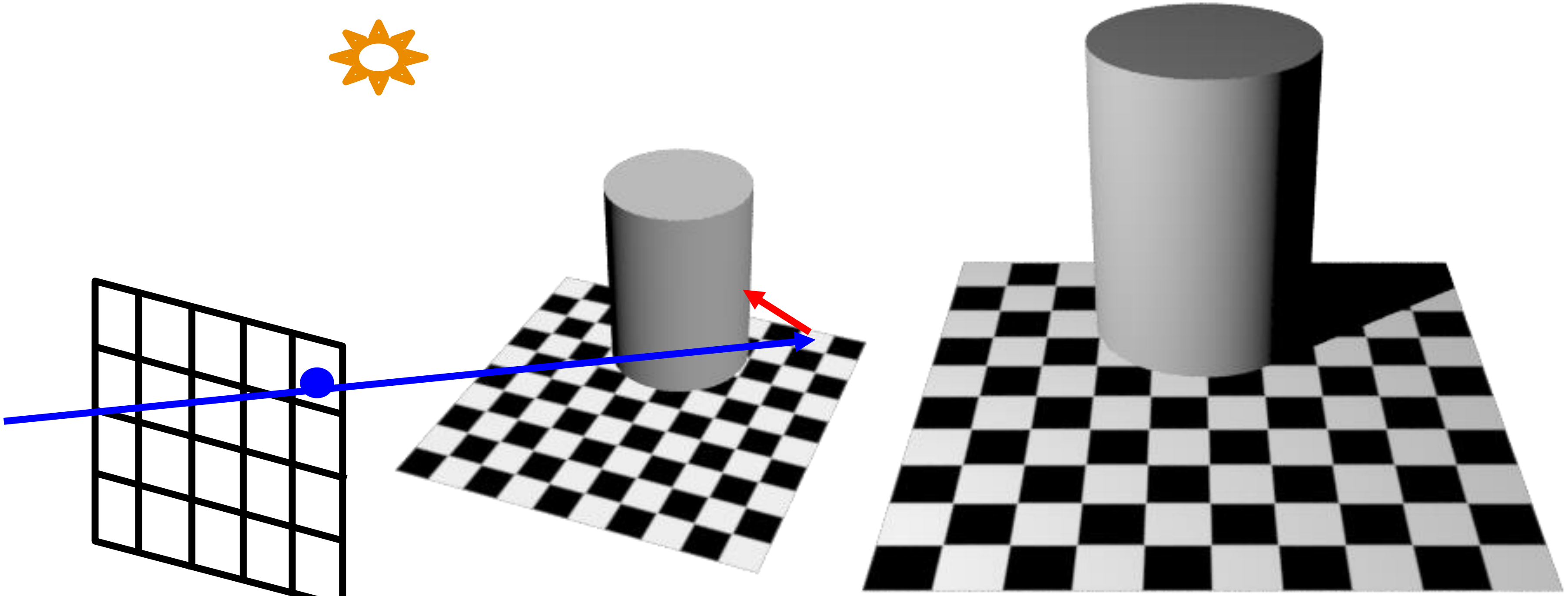
- Casted from eye (or camera) to surface, passing through a pixel.

- Find closest surface point hit by the ray.



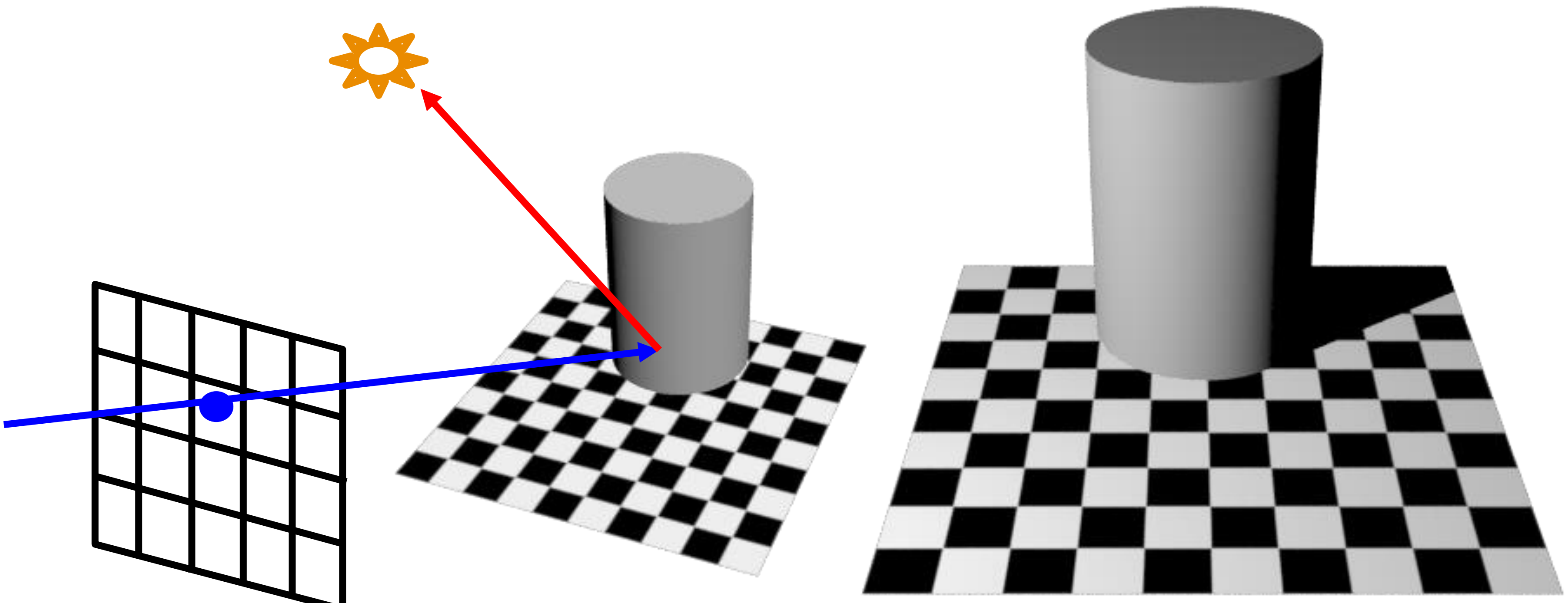
Shadow (Illumination) Rays

- Casted from surface point to *each* light source.
  - If the ray is **blocked** by an opaque object, no contribution of the light for the pixel color (shadow).



Shadow (or Illumination) Rays

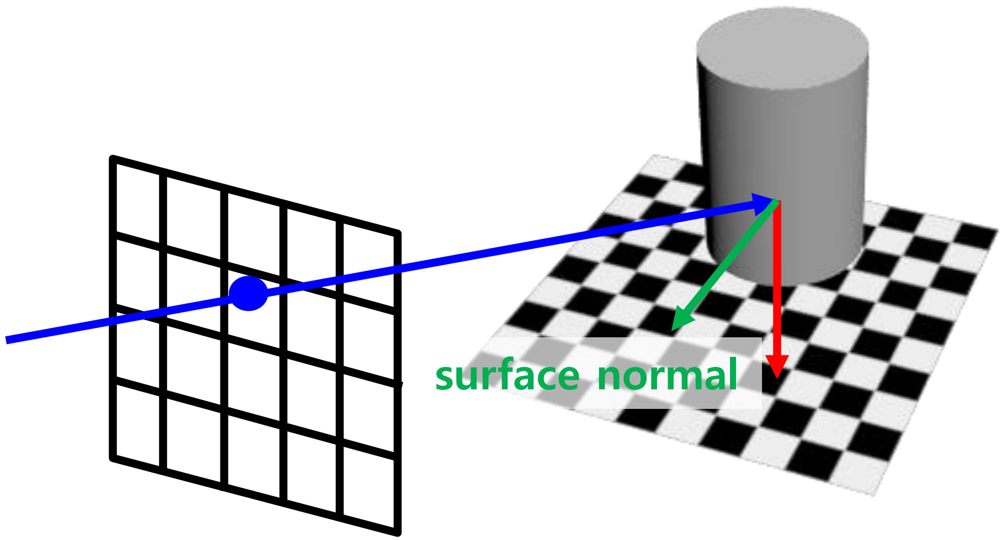
- Casted from surface point to *each* light source.
  - If the ray **reaches the light**, compute the contribution of the light for the pixel color using local illumination model.

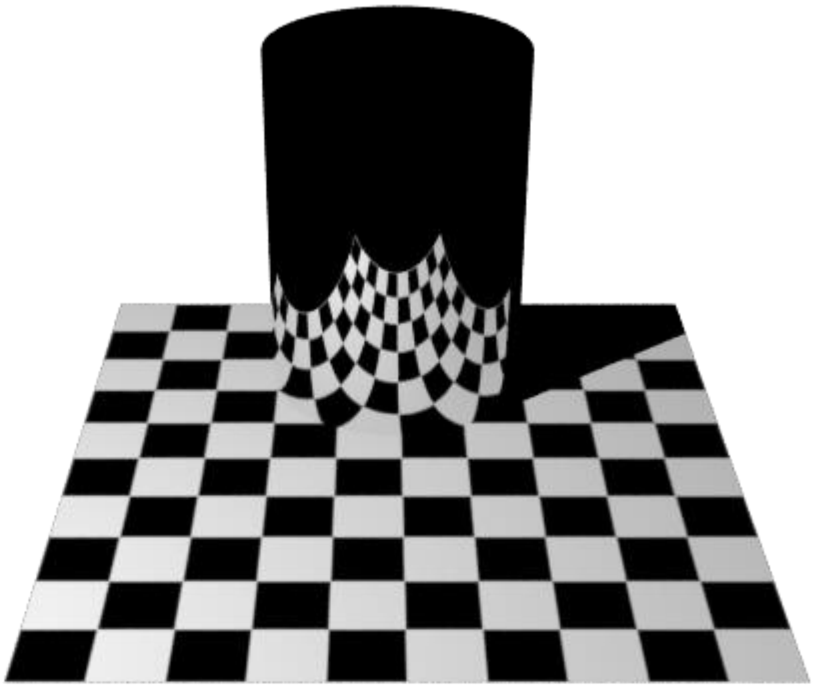


Reflection Rays

- Casted from surface point in mirror direction if the surface is specular (following the laws of reflection).

- If this ray reaches other surfaces, cast shadow / reflection / refraction rays from that surface point again (recursive).

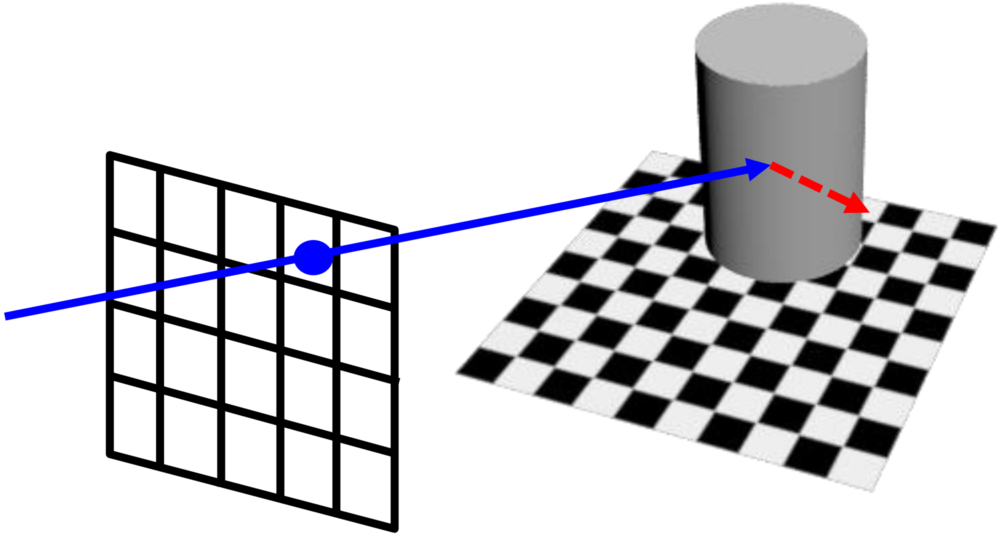


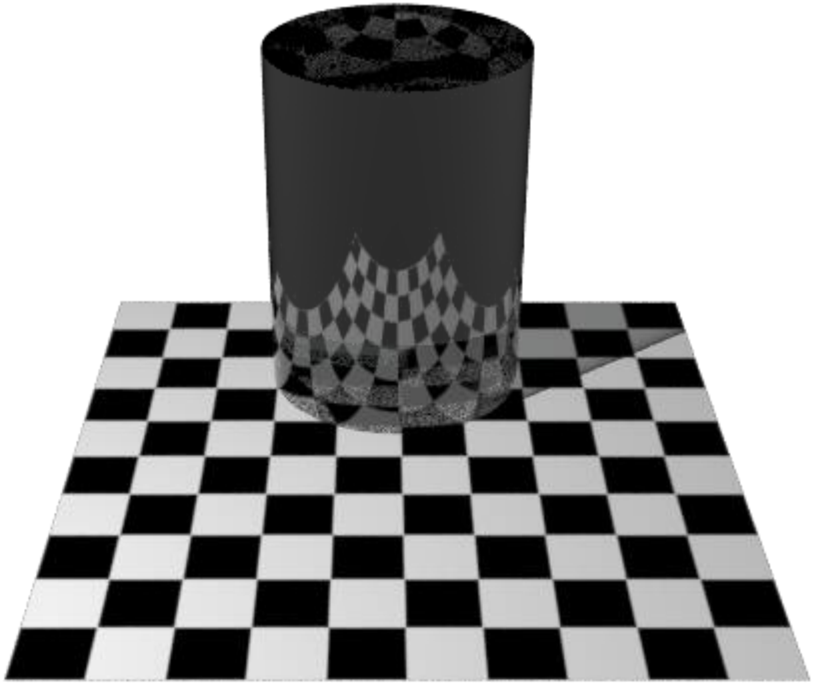


Refraction Rays

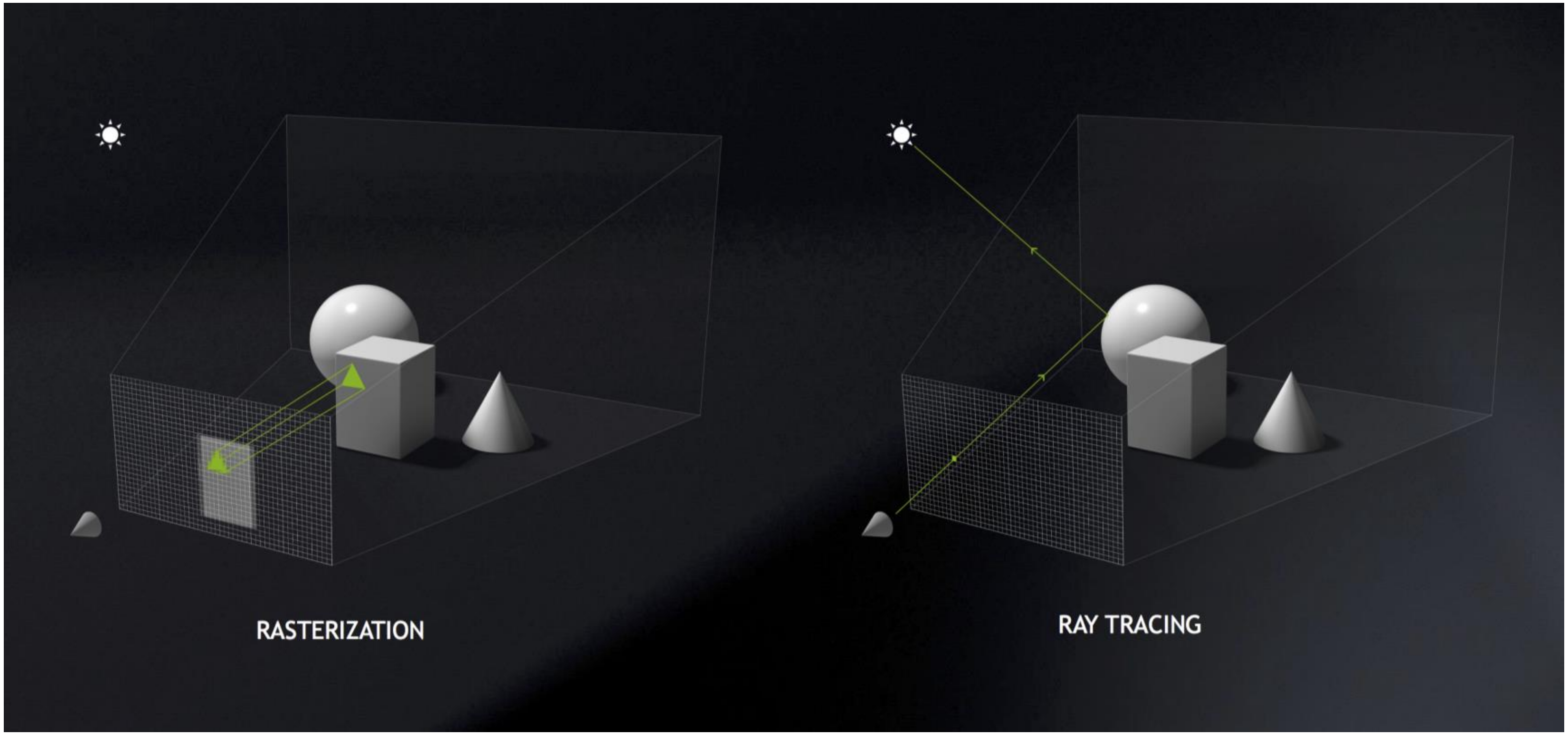
- Casted from surface point in refracted direction if the surface is transparent (following Snell's law).

- If this ray reaches other surfaces, cast shadow / reflection / refraction rays from that surface point again (recursive).





Rasterization vs. Ray Tracing



* This image is from Nvidia

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

- Cons (of the traditional view): Ray tracing was considered to be ...
  - Too slow for real-time applications
  - Hard to implement in hardware
- 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 not impossible to implement in hardware.
- Reason: the advancement of technology
  - 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/lMSuGoYcT3s

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.

Why Rasterization?

- This course does not cover ray tracing or neural rendering.

- Because...
  - Rasterization is still crucial in real-time rendering.
  - Still widely used in real-time rendering.
  - Not enough time to cover all.

Lab Session

- Now, let's start the lab today.