What is rendering?

Generating image from a (3D) scene
What is rendering?
Generating image from a (3D) scene

Let’s think HOW.

**SCENE**
- A coordinate frame
- Objects
- Their materials
- (Lights)
- (Camera)
Image Rendered to Frame Buffer

- Frame Buffer: Portion of RAM on videocard (GPU)
- What we see on the screen
- Rendering destination
Screen

*Displays what's in frame buffer*

**Terminology:**

- **Pixel:** basic element on device
- **Resolution:** number of rows & columns in device

---

Rendering

Scene

- Coordinate Frame
- Objects
- Materials
- (Lights)
- (Camera)

?  

Framebuffer

final image
**SINGLE OBJECT**

**How to describe a single piece of geometry?**

**2D**

- Triangulated polygon
- Smooth geometry => *discretized/triangulated at render time*
  - Closed curve (implicit)
  - Boolean combination of simple shapes

**SCENE**

**How to describe a scene?**
SCENE

How to describe a scene?

Local Coordinate Systems joined via Transformations

Scene
- Coordinate Frame
- Objects
- Materials
  (Lights)
  (Cameras)

Framebuffer
final image
Sketch of a rendering pipeline

**Scene**
- Coordinate frame
- Models
  - Coordinates (in local system)
  - Local to global transforms
  - properties (color, texture, material)
- Camera
- (Lights)

**Camera View**
- 2D positions in camera coordinate frame
- Depth/depth order of shapes
- (Normals)

**Image**
- Shape pixels
- Their color
- Which pixel is visible
openGL

- Open Graphics Library
- One of the most popular libraries for 2D/3D rendering
- A software interface to communicate with graphics hardware
- Cross-language API

OpenGL RENDERING PIPELINE

Vertices and attributes → Vertex Shader → Vertex Post-Processing → Rasterization → Fragment Shader → Per-Sample Operations → Framebuffer
OpenGL RENDERING PIPELINE

Scene
Vertices and attributes → Vertex Shader → Vertex Post-Processing → Rasterization

Camera Coords

Device Coords

Vertices and attributes → Fragment Shader → Per-Sample Operations → Framebuffer

Image

VERTEX SHADER

Vertices and attributes → Vertex Shader → Vertex Post-Processing → Rasterization

Fragment Shader → Per-Sample Operations → Framebuffer
**VERTEX SHADER**

- Vertices are stored in vertex buffer
- Each one is processed by vertex shader
- Converts vertex into camera coordinates (View Coordinates)
- May compute per-vertex variables (color, texture, etc.)

**RASTERIZATION**

- Vertices and attributes
- Vertex Shader
- Vertex Post-Processing
- Rasterization
- Fragment Shader
- Per-Sample Operations
- Framebuffer
RASTERIZATION

- Places three 2D vertices on a virtual screen
- Fills up the space between them
- Interpolates per-vertex variables to get per-fragment variables

FRAGMENT Shader

- Vertices and attributes
- Vertex Shader
- Vertex Post-Processing
- Rasterization
- Fragment Shader
- Per-Sample Operations
- Framebuffer
**FRAGMENT Shader**

- Each fragment is passed through Fragment Shader
- Here it computes fragment (per pixel) color

---

**FRAGMENT SHADER**

*Can simulate different materials and lights*
OpenGL RENDERING PIPELINE

Vertices and attributes → Vertex Shader → Vertex Post-Processing → Rasterization

→ Fragment Shader → Per-Sample Operations → Framebuffer

PIPELINE: More details

Vertices and attributes → Vertex Shader → Vertex Post-Processing

- Modelview transform
- Per-vertex attributes
- Viewport transform
- Clipping

→ Rasterization → Fragment Shader

- Scan conversion
- Interpolation
- Texturing/...
- Lighting/shading

→ Per-Sample Operations

- Depth test
- Blending

→ Framebuffer
PIPELINE: More details

Vertices and attributes
  - Vertex Shader
    - Modelview transform
    - Per-vertex attributes
  - Rasterization
    - Scan conversion
    - Interpolation
  - Per-Sample Operations
    - Depth test
    - Blending

Vertex Post-Processing
  - Viewport transform
  - Clipping

Fragment Shader
  - Texturing/...
  - Lighting/shading

→ Framebuffer