The OpenGL Rendering Pipeline

Michael Kreuzer

November 28, 2018

OpenGL® and the oval logo are trademarks or registered trademarks of Hewlett Packard Enterprise in the United States and/or other countries worldwide.
Overview

What is OpenGL?

Basic Concepts

Shading Pipeline

Model View Transformation

Vulkan
From the Documentation

OpenGL (for “Open Graphics Library”) is a software interface to graphics hardware. The interface consists of a set of several hundred procedures and functions that allow a programmer to specify the objects and operations involved in producing high-quality graphical images, specifically color images of three-dimensional objects.[1]
Design

- Abstract Specification for drawing 2D or 3D graphics
- Can be implemented in software or hardware (→ driver)
- Platform independent
- Language independent (Although C-ish style functions are used)
- Bindings for many languages (C, JavaScript, Java, ...)

Michael Kreuzer
OpenGL
What not?

- Windowing
- Audio
- Input
- → Frameworks like GLFW, SDL, ...
Goal

Render a 3D Object onto a 2D Plane (our Screen)

https://commons.wikimedia.org/wiki/File:Utah_teapot_simple_2.png
The OpenGL Coordinate System

OpenGL uses a **right handed** coordinate system

https://learnopengl.com/Getting-started/Coordinate-Systems
Mesh

- Represented by a set of vertices in 3D space
- Vertices form triangle faces (in our case)
- Vertex data: position, normals, texture coordinates, lighting, ...
**API Design**

- One big state machine
- Tons of functions that manipulate that state machine
Shading Pipeline
Vertex Storage

- Vertex data is stored in *Vertex Buffer Objects* in graphic card memory.
- *Vertex Array Objects* are used to index these Buffers.
- Buffers must always be bound before they can be used.
Shader

- Code executed on the graphics card
- Written in GLSL
- Different types: **VertexShader**, **FragmentShader**, **GeometryShader**, **TesselationShader**
Shading Pipeline

Figure: A simplified diagram of the rendering pipeline

1https://learnopengl.com/img/getting-started/pipeline.png
Shader

**VertexShader**
- Executed for each **vertex**
- Sets the vertex position

**FragmentShader**
- Executed for each **fragment** (pixel)
- Sets the final color of each fragment
Demo

Example 01 & 02
Uniforms

- Variables inside the shader code that can be set from outside
- Efficient method for modifying how models are displayed without the need for changing the raw vertex data
- used for nearly anything (e.g. translation, coloring, lighting)
Demo

Example 03
Element Buffers

- A vertex can be part of many triangles
- We don’t want to store vertex data more than once
- Solution: use an Element Buffer to store which indices of the vertices in the VBO correspond to which triangles
Demo

Example 04
Goal

Render a 3D Object onto a 2D Plane (our Screen)

https://commons.wikimedia.org/wiki/File:Utah_teapot_simple_2.png
Vectors

- Vertex position represented by a 4D vector
- "Homogeneous coordinates"
  \[
  \begin{bmatrix}
  x \\
  y \\
  z \\
  w
  \end{bmatrix}
  \]
  with \( w = \begin{cases} 
  1 & \text{for location vectors} \\
  0 & \text{for direction vectors}
  \end{cases} \)
- Allows us to do all kinds of transformations with 4x4 Matrices
Matrices

Translation:

\[
\begin{bmatrix}
1 & 0 & 0 & dx \\
0 & 1 & 0 & dy \\
0 & 0 & 1 & dz \\
0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix} = \begin{bmatrix}
x + dx \\
y + dy \\
z + dz \\
1
\end{bmatrix}
\]

Scale:

\[
\begin{bmatrix}
sx & 0 & 0 & 0 \\
0 & sy & 0 & 0 \\
0 & 0 & sz & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix} = \begin{bmatrix}
x \times sx \\
y \times sy \\
z \times sz \\
1
\end{bmatrix}
\]

Rotation:

\[
\begin{bmatrix}
x & 0 & 0 & 0 \\
0 & \cos \theta & -\sin \theta & 0 \\
0 & \sin \theta & -\cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix} = \begin{bmatrix}
x \\
\cos \theta y - \sin \theta z \\
\sin \theta y + \cos \theta z \\
1
\end{bmatrix}
\]
The OpenGL Coordinate System

OpenGL uses a **right handed** coordinate system

https://learnopengl.com/Getting-started/Coordinate-Systems
From a 3D model to a 2D image

https://learnopengl.com/Getting-started/Coordinate-Systems
From a 3D model to a 2D image

- Model: Object position relative to world origin
- View: Camera position
- Projection: Project 3D scene onto a 2D image
- Clipping: All vertices not within \([-1.0 \ldots 1.0]\) will be discarded
Demo

Example 0x
What is Vulkan?

- Vulkan is a "next-gen" graphics API
- Developed by the same people as OpenGL (Khronos Group)
- Will **not** replace OpenGL in the near future
- Orientated around a command buffer / command pipeline structure
Advantages

- Far more low level than OpenGL
- Thread and memory management left to application
- Sophisticated validation and diagnostic layers
- Similar API between mobile and desktop
Disadvantages

- Far more low level than OpenGL
- No thread and memory management
- A lot more boilerplate to set up
References

- **OpenGL Specification**
  The OpenGL © Graphics System: A Specification (Version 4.0 (Core Profile) - March 11, 2010)

- **Learn OpenGL**
  A good tutorial to get started with OpenGL
  https://learnopengl.com/