Intro to Computer Graphics: Parts of OpenGL

Updated: September 17, 2019
Slides by: Philmo Gu
Primitives

Note: Vertices are not primitives! Vertices are used to build primitives.

Image Source: https://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_Introduction.html
Rendering Pipeline

• Sequence of steps that OpenGL takes when rendering objects

1. **Vertex Specification**
   - Set up an ordered list of vertices to send to pipeline

2. **Vertex Shader**
   - Perform basic processing to each individual vertex

3. **Tessellation**
   - (Optional) Patches of vertex data are subdivided into smaller primitives

4. **Geometry Shader**
   - (Optional) Governs the processing of primitives
   1. Records data from vertex processing steps into buffer object
   2. Primitives are clipped to the view volume

5. **Primitive Assembly**
   - Collects a run of vertex data output and composes them into a sequence of primitives

6. **Rasterization**
   - Primitives are converted into a sequence of fragments

7. **Fragment Shader**
   - (Optional) Fragments are output with color, depth, and stencil value

8. **Per-Sample Operations**
   - Fragments are processed, and their data are written to various buffers

Source: [https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview](https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview)
Vertex Array Objects (VAO)

- OpenGL Object that stores all of the state needed to supply vertex data
  - E.g. format of vertex data (e.g. float, short, vec3), reference to buffer objects

Source: https://www.khronos.org/opengl/wiki/Vertex_Specification
**Vertex Buffer Object (VBO)**

- **Buffer Objects**: OpenGL objects that store an array of unformatted memory allocated by the OpenGL context (AKA the GPU)
  - **Vertex Buffer Object**: buffer object used as a source for vertex array data

Source: [https://www.khronos.org/opengl/wiki/Buffer_Object](https://www.khronos.org/opengl/wiki/Buffer_Object)
Shader Objects

• Object in the OpenGL API that encapsulates the linked shader
  • *Shader*: user-defined program designed to execute one of the programmable stages of the rendering pipeline.

Source: [https://www.khronos.org/opengl/wiki/Shader](https://www.khronos.org/opengl/wiki/Shader)
Source: [https://www.khronos.org/opengl/wiki/GLSL_Object](https://www.khronos.org/opengl/wiki/GLSL_Object)
Image Source: [https://learnopengl.com/Getting-started/Hello-Triangle](https://learnopengl.com/Getting-started/Hello-Triangle)
Vertex Shader

• Handles processing of individual vertices
  • Input: vertex attribute data (e.g. position, colour, normal)
  • Output: vertex

Source: https://www.khronos.org/opengl/wiki/Vertex_Shader
Source: Image Source: https://learnopengl.com/Getting-started/Hello-Triangle
Fragment Shader

• Process a fragment into a set of colours and depth value
  • *Fragment*: data needed to draw a single pixel
    • E.g. window-space position (X-Y-Z), interpolated value across surface (e.g. colour, texture coordinates), colour values, depth value, stencil value
  • Input: Fragment
  Output: colour values, depth value

Source: https://www.khronos.org/opengl/wiki/Fragment_Shader
Source: https://www.khronos.org/opengl/wiki/Fragment
Source: https://gamedev.stackexchange.com/questions/8977/what-is-a-fragment-in-3d-graphics-programming
Image Source: Image Source: https://learnopengl.com/Getting-started/Hello-Triangle
Recommended Resources

(These are quite good!)

• Wiki: https://www.khronos.org/opengl/wiki/Main_Page
• Easy-to-read explanation: https://learnopengl.com/
• Tutorial for simple implementation: http://www.opengl-tutorial.org/
Exercise

• Import the code for Sierpinski triangle to core-profile OpenGL
• Draw a green circle with core-profile OpenGL