Intro to OpenGL
Rendering Objects

- Object has internal geometry (Model)
- Object relative to other objects (World)
- Object relative to camera (View)
- Object relative to screen (Projection)

Need to transform all geometry then draw…
The Graphics Pipeline

Designed to optimize rendering process

- Raytracing is too slow
- Focus on high throughput and parallelization
Rasterization

Objects composed of vertex data
Vertex data tessellated into primitives
Rasterization

Primitives have color and position
Draw pixels to screen
Embarrassingly parallel with great hardware support!
OpenGL

Open Graphics Library

• Standardized in 1992 by Silicon Graphics
• Currently managed by Kronos Group

Microsoft equivalent is DirectX
Simplified Graphics Pipeline

Application

Vertex batching & assembly

Clipping

NDC to window space

Rasterization

Fragment shading

Depth testing

Color update

OpenGL API

NDC = Normalized Device Coordinates, this is a \([-1,+1]^3\) cube

Depth buffer

Framebuffer
A Little Expanded…

Application

Vertex batching & assembly

- Vertex transformation
- Lighting
- User defined clipping
- View frustum clipping
- Triangle assembly

Texture coordinate generation

Perspective divide

NDC to window space

Triangle rasterization

Fragment shading

- Back face culling
- Depth testing
- Color update

Depth buffer

Framebuffer
Old vs Modern OpenGL

Originally OpenGL was a “Fixed Function” Pipeline

• Exposed graphics hardware through user configurations

• Built-in math operations manipulate data accordingly
Old vs Modern OpenGL

OpenGL 3.0 is programmable allowing for greater flexibility and control

...Also changes hardware pipeline and how a programmer interacts with the GPU
The modern rendering pipeline (blue stages are fully programmable)
Vertex Specification

Specify vertices for GPU to process
  • One vertex/triangle at a time is slow

Specify how to process
  • Attributes inform vertex shader what data represents
Vertex Buffer Objects (VBOs)

- Source of data for vertex arrays
- `glBindBuffer` binds given buffer to global target
  - `GL_ARRAY_BUFFER` specifies Buffer Object is vertex attribute data
- `glVertexAttribPointer` specifies attribute data for vertices
  - What are the data components and how are they arranged?
VBO Data

Contain data for:

- Vertex position
- Vertex colors
- Texture info
- Normal info
- etc
Vertex Array Objects (VAOs)

- OpenGL Objects associated with an OpenGL context (state of the instance)
- Stores attribute data and Buffer Objects for bussing to GPU
  - Can contain multiple VBOs
- VAOs allow switches between vertex attribute configurations without performance hit
- `glGenVertexArrays` creates VAO
- `glBindVertexArray` binds that VAO to target
Using VAOs

1. Create VAO with necessary information:
   1. Create VAO
   2. Bind VAO
   3. Generate and bind VBO
   4. Disable/unbind VAO and VBO

2. Rendering using VAO:
   1. Bind VAO
   2. Draw data in VBO
   3. Unbind VAO
Vertex List

Triangle List

Vertex Shader

Geometry Shader

Primitive Assembly

Fragment Shader

Rasterization

Textures

Framebuffer

Screen

CPU

GPU
Coordinate Systems

object

model matrix

world

view matrix

camera

center

tangent

tangent

perspective matrix

normalized device
Camera Coordinates

Note: Look down negative z direction
Normalized Device Coordinates

Note:
X and Y map to screen width and height
Z used for depth (deeper points are higher)
Except…

Screen coordinates use different system!
Also...

```
glViewport(x, y, width, height)
transforms NDC to window coordinates
```

Allows for an aspect ratio in final display to screen after being normalized

Incidentally (x, y) specifies the lower left corner of the viewport
Framebuffer

Memory region containing pixel data
Controlled by GPU

Layers:
- Color buffer (RGB)
- Depth buffer (Z axis position)
- Stencil buffer (extension of depth buffer)
Displaying a Framebuffer

CRTs: beam sweeps across screen to draw pixels (one pass every 1/60 secs)

LCDs: grab framebuffer (every 1/60 secs)
Flickering and Tearing

Framebuffer changes while monitor draws

How to solve?
Double-Buffering

Use two framebuffers

Render to **back buffer** while showing **front buffer**

Then swap
When to Draw

On CRTs: wait for *vertical retrace* to swap
- “vsync”
- Occurs 1/60 sec
- Introduces lag

On LCDs: swap when not reading
OpenGL Tutorial

Work through:

http://www.opengl-tutorial.org/beginners-tutorials/tutorial-2-the-first-triangle/