OpenGL pipeline (simplified) – Draft 4/20/218 – Sam Buss – Math 155B

1. **Vertex specification (usually created on the CPU, uploaded to GPU)**
   Specify a list of vertices, with attributes (such as position, normal, color)
   Store into a VBO.
2. **Drawing calls:** CPU uses glDraw… commands to start rendering.
3. **Vertex Shader – GPU invokes once per vertex.**
   a. If a vertex appears multiple times in an EBO, vertex shader may or may not be called multiple times (recent vertex shader calls are stored in a small hash table).
   b. Inputs are: vertex attributes and uniform variables.
   c. Also a few built-in inputs including: gl_VertexID and gl_InstanceID
   d. Main outputs:
      gl_Position (x,y,z,w) giving position in 2x2x2 cube; and
      Variables declared as "out".
4. **Tessellation Shader –** split primitives such as patches, triangles and lines into subprimitives. Example application: refining a Bezier curve or surface into small pieces. – Not discussed in this outline.
5. **Geometry Shader** – Takes as input a primitive (usually a triangle or a line segment). Outputs zero or more replacement primitives.
   For an example, see the geometry shader "geomShaderNormals" in Project 4 of the past quarter (Winter 2018, Math 155A) which automatically rendered normals.
6. **Clipping to the view volume:**
   When portions or all of a primitive (triangle or line or point) lies outside the 2x2x2 view volume, it is either culled (completely removed) or clipped. Clipping means that only the portion of the primitive that is inside the view volume is retained: this may require splitting a triangle into multiple triangles.
7. **Perspective division.**
   Conversion from homogeneous coordinates to Euclidean coordinates (dividing by w).
8. **Primitive assembly** is complete by this stage.
   This means the vertices are assembled back into primitives (triangles, lines, points).
   If there are tessellation or geometry shaders, some primitive assembly has to happen both before and after those shaders are run.
9. **Face culling.** Front/back face determination is made based on winding order, from the point of view of the viewer. If enabled, back or front faces may be culled.
10. **Rasterization.** Each primitive (triangle, line, or point) is broken up into fragments, usually one per pixel.
11. **Fragment Shader.**
    Inputs from earlier shaders:
    Uniform variables and variables declared as "in" which were output by earlier shaders.
    "in" variables are generally linearly interpolated across triangles (based on barycentric coordinates) or linearly interpolated along lines (the default is to use hyperbolic

interpolation). It is also possible to use “flat” shading or “nonperspective” shading instead.

Other inputs: There are a number of built-in inputs, including:

gl_FrontFacing – indicates whether front facing.

gl_FragCoord – position in screen space coordinates.

Output: a vec4 giving the fragment color.

12. **Multisampling anti-aliasing.** Pixel values are distributed selectively into subpixels.
13. **Depth test.** For hidden surface removal.

    Fragments that fail the depth test are discarded.
14. **Blending.** Painter’s algorithm should be used to order rendered pixels.