Lab 02 - OpenGL Primitives (Chapter 4)

I. OpenGL Immediate (Deprecated OpenGL feature) - Most inefficient
   A. Essentially commands OpenGL to send a small data set through pipeline immediately.
      Requires collection of data on CPU and single vertex copies over to GPU. No bulk copy.
   B. Form:  
      ```
      glBegin(GL_*);
      glVertex(*) / vertex attribute e.g., normals or colors;
      glEnd();
      ```

C. Primitives:
   1. GL_POINTS specify separate points to be drawn
   2. GL_LINES vertices + and +1 define lines
   3. GL_LINE_STRIP vertices unwind +1 define lines
   4. GL_LINE_LOOP closed line +1 define lines
   5. GL_TRIANGLES Each 3 vertices define triangle
   6. GL_TRIANGLE_STRIP Each new vertex adds another triangle
   7. GL_TRIANGLES FAN Each new vertex defines another triangle
   8. GL_TRIANGLES FAN Each new vertex defines another triangle
   9. GL_QUADS GL_QUAD_STRIP, GL_QUADS

D. Each Begin + End sequence is expensive. Do not do one per point/polygon. Wrap as many
   as you can into fewer Begin/End blocks.

D1. Display Lists
   A. Alternating CPU calls to GPU with OpenGL Display Lists, similar to mini-program.

D2. Display Lists
   B. Form:  
      ```
      GLuint list = glGenLists(1);
      glNewList(list, GL_COMPILE);
      // Series of GL commands that all are used in lists*
      glEndList();
      ```

C. Form:  
   ```
   glCallList(list); // CPU command to GPU compiled program.
   ```
   in use

D. Identifier 0 or GL objects, e.g., DisplayList, is invalid. Can invoke glIs*() to check validity of objects, e.g., glIsList(list);

E. Cleanup - you must deAllocate GPU memory explicitly (similar to delete keyword)  
   ```
   glDeleteLists(list, 1);
   ```

F. Support exists for multiple lists at same time when generating, displaying, and deleting.
III. Vector Array Objects

A. Reduces copying of data (or bulk copies data) to GPU. Asynchronous, extremely efficient.
Calls to GL tell GPU how to interpret raw arrays of memory/vertices.

B. Approximate code:

i. Vertex Array Object (VAO): All of the state needed to supply vertex data/attributes to GPU.
ii. Vertex Buffer Object (VBO): An array mapping (buffer) to vertex data.
iii. Index Buffer Object (IBO): An array mapping (buffer) to indices e.g., faces/triangles.

IV. Preprocessing:

```cpp
GLuint vao;
GLuint vertexattribs(1, &vao);  // create a VAO
BindVertexArray(vao);  // make active

// Vertex buffer object
GLuint vbo;
GLenumBuffers(1, &vbo);  // create Buffer on GPU
BindBuffers(GL_ARRAY_BUFFER, VBO);  // make active
BufferData(GL_ARRAY_BUFFER, numBytes, dataPtr, GL_STATIC_DRAW);
// define mapping

// Index buffer object
GLuint ibo;
/* Some buffer process w/ GL_ELEMENT_ARRAY_BUFFER */
// Attributes can also be defined w/ glEnableVertexAttribArray(x) or
// glEnableClientState(GL*)
// and glVertexAttribPointer

V. To unbind: glBind* (GL_ARRAY_BUFFER, 0);  // for example
VI. To draw: Bind VAO w/ glBindVertexArray (vao);
DrawElements(GL..., num, type, startVertex, &index, &vertex);
DrawElements(GL..., GL..., startVertex, &index, &vertex);
VII. To delete: glDelete* (*)
VIII. Altogether complex. Lots of options. Bonus on first assignment.
```