Shader-Based OpenGL: An Intermediate Summary

- Vertex Array Objects (VAOs) and Vertex Buffer Objects (VBOs)
    - Names generated using `glGenVertexArrays`, `glGenBuffers`
    - Memory allocated *and* data sent from CPU to GPU using `glBufferData`
    - Memory in an existing VBO can be modified using `glBufferSubData`
    - Names *and* memory deallocated using `glDeleteVertexArrays`, `glDeleteBuffers`
- VAOs and VBOs: Packaging of "per-vertex" attribute (PVA) definition
    - VBOs are used to store PVAs (including geometry) on the GPU.
        - ‡ Recall PVAs are those whose values *<u>might</u>* change from one vertex to another.
        - ‡ The PVA base type must be some floating point type and may be scalar (i.e., a 1-tuple), 2-, 3-, 4-tuple, and matrix values
    - VAOs encapsulate a collection of VBOs and related state:
        - ‡ "Enabled" status of VBOs (i.e., whether `glEnableVertexAttribArray` or `glDisableVertexAttribArray` was specified for the PVA in this VAO)
        - ‡ Attribute array storage structure specification (i.e., information specified via `glVertexAttribPointer` for enabled VBOs)
- CPU-side specification of attribute values:
    - **Per-vertex**: <u>Two choices</u>:
        - ‡ Passed in VBOs; enabled and described, respectively, via `glEnableVertexAttribArray` and `glVertexAttribPointer`.

          (In our framework, this is normally done during execution of a `ModelView` constructor.)
        - ‡ If a PVA is constant throughout a given primitive, then its VBO can be disabled via `glDisableVertexAttribArray`, and the attribute can be set during the display callback using `glVertexAttrib`*.

          (In our framework, the `glDisableVertexAttribArray` call is normally done during execution of a `ModelView` constructor, and no `glVertexAttribPointer` call will be made for that PVA; the `glVertexAttrib` call is then normally done during execution of a `ModelView::render` method.)*
    - **Per-primitive** via `glUniform`* (typically during execution of a `ModelView::render` method during a display callback)

glGenVertexArrays and ***glGenBuffers***

- Generates one (or more) previously unused VAO or VBO name(s)

glBindVertexArray(vao)

- Closes the previously "open" VAO, if any.
- Creates the VAO, if this is the first time its name has been passed to ***glBindVertexArray***.
- Opens the VAO for usage/editing:
  - Reestablishes all the settings as they were the last time this VAO was open, including reestablishing all its VBOs.
  - Makes this VAO "open", hence allowing changes to its state.

glBindBuffer(target, vbo)

- Closes the previously "open" VBO bound to the given *target*, if any.
- Creates the VBO, if this is the first time its name has been passed to ***glBindBuffer***.
- Adds this VBO to the currently open VAO.
- Opens the VBO for usage/editing:
  - Reestablishes all the settings as they were the last time this VBO was bound.
  - Makes this VBO "open" (and bound to *target*), hence allowing changes to its state, e.g.,via `glBufferSubData`.

The Model-Render-Edit Processes

Applies to *Both* 2D and 3D

- **Typical creation process** (e.g., during a `ModelView` constructor call)

  ```
  glGenVertexArrays(…)
  glBindVertexArray(…)
  ```

  Here or inside the pseudo "for loop" that follows:

  ```
     glGenBuffers(…)
  ```

  for each VBO to be associated with the currently open VAO:

  `glBindBuffer(…)` – associates this VBO with the currently bound VAO

  `glBufferData(…)` – allocate storage and (optionally) copy data from CPU to GPU

  `glVertexAttribPointer(…)` – define a "template" for the raw data in the buffer

  `glEnableVertexAttribArray(…)` – enable use of this VBO for the given PVA

- **Typical rendering process** (e.g., during a display callback; i.e., a `ModelView::render` method)

  <perform any required initial processing; establish desired per-primitive uniforms>

  ```
  glBindVertexArray(…)
  ```

  one or more calls to such routines as `glDrawArrays(…)`, `glDrawElements(…)`

- **Typical modification process** (e.g., during an event callback)

  ```
  glBindVertexArray(…)
  ```

  for each VBO associated with this VAO that needs to be modified:

  `glBindBuffer(…)`

  `glBufferSubData(…)` – overwrite all or part of the buffer without changing its size

- **Be sure you understand** (i.e., both for projects *and* exams)
  - The "times" at which we have been calling these functions: initialization, modification in response to events, rendering during display callbacks, etc.
  - All about the differences between per-primitive and per-vertex attributes.
  - The differences between `glGen`*Xxx*`s` and `glBind`*Xxx*