Vertex Buffer Objects (VBOs)

References

Notes based on
Shreiner, D., Woo, M., Neider, J. and Davis, T., (2006), OpenGL Programming Guide, 5th Edition, Addison Wesley

Vertex Buffer Objects (VBOs)

- Vertex arrays (VAs) are stored in client space, i.e. in system memory.
- Vertex arrays in client space allow improved performance by operating on larger chunks of data, and moving away from the vertex at a time approach of immediate mode.
- However, still need to transfer vertex data into server space, i.e. graphics memory, usually repeatedly.
- *Vertex buffer objects* or VBOs allow storing of vertex arrays in server space, i.e. in graphics memory.

OpenGL Objects

- OpenGL *objects* are entities which have data or *state*, can be created, read, written, manipulated etc using functions.
- There are many kinds of OpenGL objects: texture objects, buffer objects, framebuffer objects, render objects, etc.
- OpenGL is not an object oriented programming API, and its objects are not quite objects in that sense, but are related.
- OpenGL can be viewed as a state machine, (bound) objects are part of it.
- An OpenGL program runs using a OpenGL *context*, essentially an instance of an OpenGL state machine.
- Different OpenGL programs have different *contexts*, and do not affect each other, e.g. setting colour in one does not affect the other.
- Objects *bound* to a context become part of and affect it, *unbound* objects don't, although they may still occupy space.

OpenGL Objects (cont.)

- OpenGL objects are given integer identifiers/names, and are *references* not *pointers*.
- The identifier 0 is a special case, usually (but not always) akin to NULL.
- OpenGL objects are created using `glGen*(GLsizei` *`n`*`, GLuint *`objects`*`)` functions.
- OpenGL objects are bound using `glBind*(GLenum` *`target`*`, GLuint` *`object`*`)` functions.
- There are different ways to change the objects' state or data, but primarily using appropriate GL functions.

OpenGL Buffer Objects and Vertex Buffer Objects

- Buffer objects are a kind of OpenGL object
- Buffer objects store an array of data server-side i.e. in graphics memory.
- Vertex buffer objects are a kind of buffer object, in which vertex or index data is stored.

Vertex Buffer Objects: Steps

Six steps to use vertex buffer objects

1. Create vertex buffer objects
2. Bind a buffer object, specifying *target* as vertex or index data
3. Request storage, optionally initialise
4. Specify data including offsets into buffer object
5. Bind buffer object to be used in rendering
6. Render using vertex array techniques, e.g. `glDrawElements`

Step 1: Create Buffer Objects

- Similar to creating identifiers for display lists using `glGenLists`
- To generate one or more buffer objects use `glGenBuffers(GLsizei` *`n`*`, GLuint *`buffers`*`)`
- Names/identifers returned in *buffers*.
- `glIsBuffer(GLuint` *`buffer`*`)` to test if an integer is in use
- Zero is reserved identifier
- To delete use `glDeleteBuffers`

Step 2: Bind Buffer Object

- Binding makes a buffer object active
- Once bound a buffer object is used for operations to initialise it with data and/or for vertex array rendering operations
- Use
  `void glBindBuffer(GLenum` *`target`*`, GLuint` *`*buffer`*`)`
- *target* can be `GL_ARRAY_BUFFER` for vertex data e.g. coordinates, normals etc. or `GL_ELEMENT_ARRAY_BUFFER` for index data.

Step 3: Allocate and Initialise

- Need to reserve space for the buffer object in the OpenGL server
- Once bound a buffer object is used for operations to initialise it with data and/or for vertex array rendering operations
- Use
  `void glBufferData( GLenum` *target*`, GLsizeiptr` *size*`, const GLvoid *`*data*`, GLenum` *usage*`)`
- *target* again `GL_ARRAY_BUFFER` or `GL_ELEMENT_ARRAY_BUFFER`
- *size* is number of bytes
- *data* is pointer to client memory or `NULL`
- *usage* is a hint for performance
- Memory is finite, may get `GL_OUT_OF_MEMORY`
- Store vertex, color, normal etc data in one or more VBOs.

Step 4: Specify Data

- For vertex arrays we saw `glVertexPointer`, `glNormalPointer`, `glColorPointer` etc for specifying pointers to the client side data.
- The same functions are used for vertex buffer objects
- However the data is in the vertex buffer object(s), initialised with `glBufferData` as above
- The *pointer* argument becomes an *offset* in the VBO.
- A single VBO can be used to store all the vertex data if it is stored in a single array, e.g. normals then coordinates.
- Data can be interleaved or non-interleaved.
- If data is stored in an interleaved array *stride* is non-zero

Step 5: Bind

- Use `glBindBuffer` as above
- No need to rebind if buffer already bound from step 2, but no harm either (other than small performance cost for unecessary operation).

Step 6: Render

- Use `glDrawArrays`, `glDrawElements` etc as for vertex arrays

Modifying VBO data

- To change/edit the values in a VBO can supply new values using `glBufferData`. This means all data is updated, not just changed values.
- Can use `glBufferSubData` to update just some values.
- Another approach is to use `glMapBuffer` to get a (special) pointer to the data in the video/graphics memory and use that to update specific values. Must use `glUnmapBuffer` when finished editing and before rendering.