ADVANCED DATA

Previously, we've been extensively using buffers in OpenGL to store data on the GPU.

A buffer in OpenGL is, at its core, an object that manages a certain piece of GPU memory, and nothing more.
OpenGL internally stores a reference to the buffer per target and, based on the target, processes the buffer differently.

So far, we've been using the buffer's memory by calling glBufferData, which allocates a piece of GPU memory and adds data into this memory. If we were to pass NULL as its data argument, the function would only allocate the memory and not fill it.
- This is useful if we want to reserve a specific amount of memory and later come back to this buffer.

We can also fill specific region of the buffer by calling `glBufferSubData`, which expects a buffer target, an offset, the size of the data, and the actual data as its arguments. What makes this function different than `glBufferData` is that we can now give an offset that specifies from where we want to fill the buffer. This allows us to insert/update only certain parts of the buffer's memory.
- **NOTE:** The buffer should have enough allocated memory, so you must call `glBufferData` on the buffer before calling `glBufferSubData` on that buffer.

Another method of getting data into a buffer is to ask for a pointer to the buffer's memory and directly copy data in memory yourself, using `glMapBuffer`.
- glUnmapBuffer tells OpenGL that we're done using the pointer and returns true if the data was mapped successfully to the buffer.
- glMapBuffer is useful for directly mapping data to a buffer without first storing it in temporary memory.
- Example glMapBuffer Implementation

Batching Vertex Attributes

We have been storing all of our vertex data in one, **interleaved** array. However, we could also batch all the vertex data into large chunks per attribute type instead.
- Instead of an interleaved layout (123123123123), we can take a batched approach (111122223333).
- This is much easier to use when loading vertex data from a file, since the data is typically separated by attributes rather than it all being crammed into one data structure.
- Example Batched Vertex Attibutes using glBufferSubData

While batching is easier to use in certain circumstances, it is still recommended to use interleaved vertex attributes since the vertex attributes for each vertex shader run will be closely aligned in memory.

Copying Buffers

Once your buffers are filly with data, you may want to share that data with other buffers or copy the buffer's content into another buffer. You can accomplish this with `glCopyBufferSubData`.
- `void glCopyBufferSubData(GLenum readtarget, GLenum writetarget, GLintptr readoffset, GLintptr writeoffset, GLsizeiptr size)`
  - **`readtarget`:** The buffer target (e.g. `GL_VERTEX_BUFFER`) to read from.
  - **`writetarget`:** The buffer target to write to.
  - **`readoffset`:** The offset of the data in the buffer that is read from.
  - **`writeoffset`:** The offset of the data in the buffer that is written to.
  - **`size`:** The size (in bytes) of the data that is being copied.
- You can't bind two of the same type of buffer at the same time, so if you want to read and write from, say, two vertex array buffers, you should bind the buffer objects to `GL_COPY_READ_BUFFER` and `GL_COPY_WRITE_BUFFER`, respectively, and use those buffer targets as the `readtarget` and `writetarget` arguments.
  - Example glCopyBufferSubData Implementation

Example glMapBuffer Implementation

Monday, April 25, 2022 3:54 PM

```
float data[] = {
    0.5f, 1.0f, -0.35f
    // More data
    [...]
};
glBindBuffer(GL_ARRAY_BUFFER, buffer);
// get pointer
void *ptr = glMapBuffer(GL_ARRAY_BUFFER, GL_WRITE_ONLY);
// now copy data into memory
memcpy(ptr, data, sizeof(data));
// make sure to tell OpenGL we're done with the pointer
glUnmapBuffer(GL_ARRAY_BUFFER);
```

Example Batched Vertex Attibutes using glBufferSubData

Monday, April 25, 2022 4:03 PM

```
float positions[] = { ... };
float normals[] = { ... };
float tex[] = { ... };
// fill buffer
glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(positions), &positions);
glBufferSubData(GL_ARRAY_BUFFER, sizeof(positions), sizeof(normals), &normals);
glBufferSubData(GL_ARRAY_BUFFER, sizeof(positions) + sizeof(normals), sizeof(tex), &tex);

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), 0);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)(sizeof(positions)));
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void*)(sizeof(positions) + sizeof(normals)));
```

This allows us to directly transfer the attribute arrays as a whole into the buffer without first having to interleave them.

NOTE: The stride parameter is equal to the size of the vertex attribute. This is because the data immediately after, say, a vertex' s position, is the next vertex's position, since the positions array ONLY contains position data.

Example glCopyBufferSubData Implementation

Monday, April 25, 2022 4:44 PM

This method utilizes the special read and write buffer targets to bind the two VBOs to, since you can't bind more than one of the same type of buffer at a time.

```
glBindBuffer(GL_COPY_READ_BUFFER, vbo1);
glBindBuffer(GL_COPY_WRITE_BUFFER, vbo2);
glCopyBufferSubData(GL_COPY_READ_BUFFER, GL_COPY_WRITE_BUFFER, 0, 0, 8 * sizeof(float));
```

However, you only need to utilize one of the read/write buffer targets to bypass this issue.

```
glBindBuffer(GL_ARRAY_BUFFER, vbo1);
glBindBuffer(GL_COPY_WRITE_BUFFER, vbo2);
glCopyBufferSubData(GL_ARRAY_BUFFER, GL_COPY_WRITE_BUFFER, 0, 0, 8 * sizeof(float));
```