UNIVERSITÉ Concordia UNIVERSITY

COMP 371
Computer Graphics

Lab 06 – Models and EBOs

Prepared by Zachary Lapointe

This Week

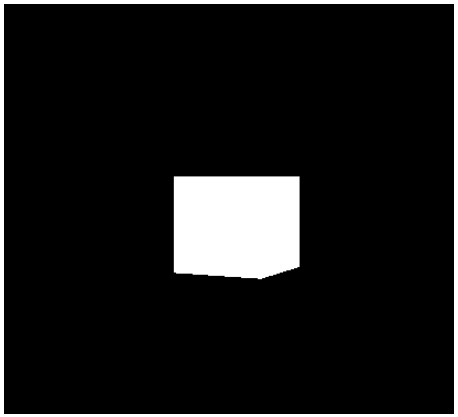
Tutorial:

Load models from OBJ files
View models using vertex normals
Buffer vertex data using an EBO

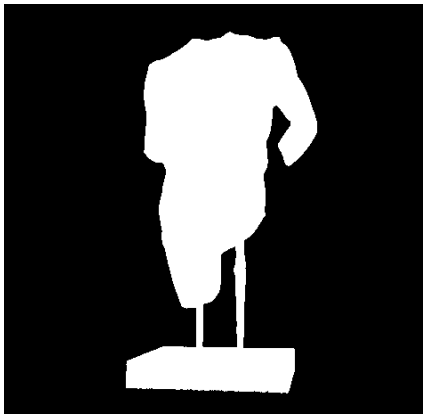
Exercises

Work on Assignment 2

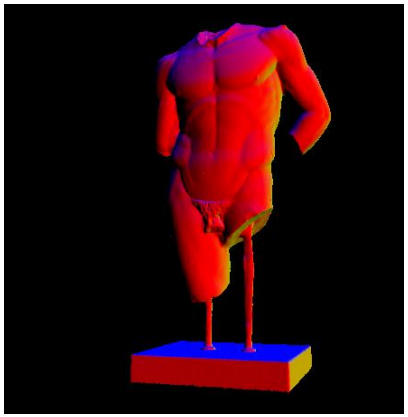
Expected results



Model Loading



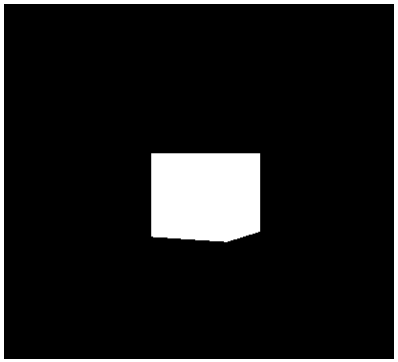
Complex Model



Visualizing Model Relief

Getting Started

- Download Lab01.zip from Moodle
- Download Lab06.zip and add lab06.cpp and the two header files to the project(Visual Studio or Xcode)
- Add copy in *Assets/Models* folder
- After compiling and running the application, you should see the image below.



- Use A S D W and mouse to move the camera

TUTORIAL
LOADING MODELS

Loading Models from Files TODO1

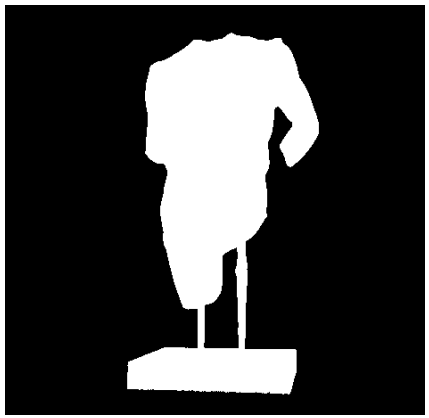
- We've been rendering simple shapes so far
  - What about models with 1000s of vertices?
  - What if models aren't known at compile time?
  - This can be fixed by loading models at runtime, from a file
  - We will be using the OBJ model file format for this
- In OpenGL, we need to load the *heracles* model,

```
//TODO 1 load the more interesting model: "heracles.obj"

int heraclesVertices;
GLuint heraclesVAO = setupModelVBO(heraclesPath, heraclesVertices);

int cubeVertices;
GLuint cubeVAO = setupModelVBO(cubePath, cubeVertices);
```

Loading Models from Files TODO1



```
//Using number keys to switch between models
if (glfwGetKey(window, GLFW_KEY_1) == GLFW_PRESS)
{
        activeVAO = cubeVAO;
        activeVAOVertices = cubeVertices;
}
if (glfwGetKey(window, GLFW_KEY_2) == GLFW_PRESS)
{
        //TODO 1 Add a key to switch between the two models
}
```

.OBJ files

- A human readable file for models
  - Stores vertex positions, UVs, normals, and more
  - Stores triangle information made from the above-mentioned data.

- Look at *cube.obj*

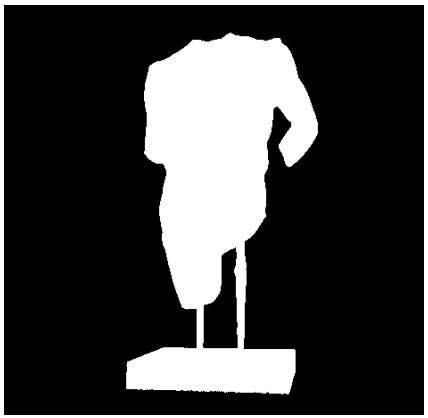
```
#vertices
# format: 'v' x y z
v 5.000000 -5.000000 -5.000000
v 5.000000 -5.000000 5.000000
v -5.000000 -5.000000 5.000000
v -5.000000 -5.000000 -5.000000
v 5.000000 5.000000 -5.000000
v 5.000000 5.000000 5.000000
v -5.000000 5.000000 5.000000
v -5.000000 5.000000 -5.000000
```

.OBJ files

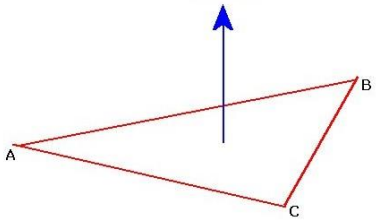
- Files are loaded using straightforward file reading, in OBJLoader.h

```
bool loadOBJ(
    const char * path,
    std::vector<glm::vec3> & out_vertices,
    std::vector<glm::vec3> & out_normals,
    std::vector<glm::vec2> & out_uvs) {
```

Visualizing Complex Models TODO2



- Models don't always have vertex colors or textures. They may rely instead on lighting to highlight relief and fine details.
- Since we haven't seen lighting yet, we'll use *vertex normals* as an approximation instead.
- *Normals* are unit vectors which stick out from vertices, perpendicular to the triangles formed by the vertex



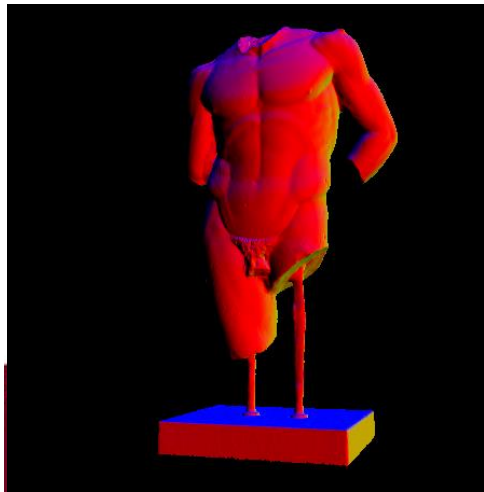
Visualizing Complex Models TODO2

- Normals are used in lighting, and can be used to approximate the effect
- **Vertex Shader:**

```
"void main()"
"{"
"    "    //TODO 2 We should pass along the normal to the fragment shader
"    vertexNormal = aNormal;"
"    mat4 modelViewProjection = projectionMatrix * viewMatrix * worldMatrix;"
"    gl_Position = modelViewProjection * vec4(aPos.x, aPos.y, aPos.z, 1.0);"
"}";
```

- **Fragment Shader:**

```
"void main()"
"{"
"    FragColor = vec4(vertexNormal, 1.0f);" //TODO 2
"}";
```

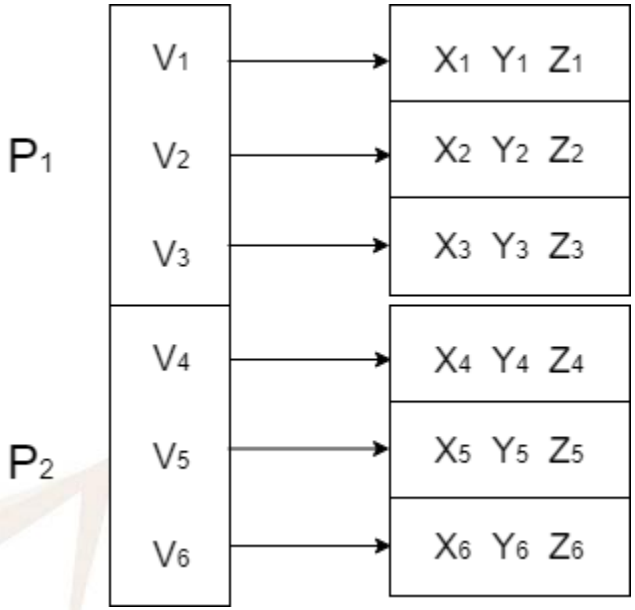


Using Element Buffer Objects (EBOs) TODO3

- Complex models like *heracles.obj* can have millions of vertices
  - This is a lot of data to buffer to the GPU
- We can reduce the amount by removing any duplicates in the data
  - Wherever triangles meet in a model, adjacent triangles share vertices
  - Currently we are sending 3 vertices per triangle to the GPU
- We can remove this redundancy using EBOs, which refer to vertices by index as opposed to position.

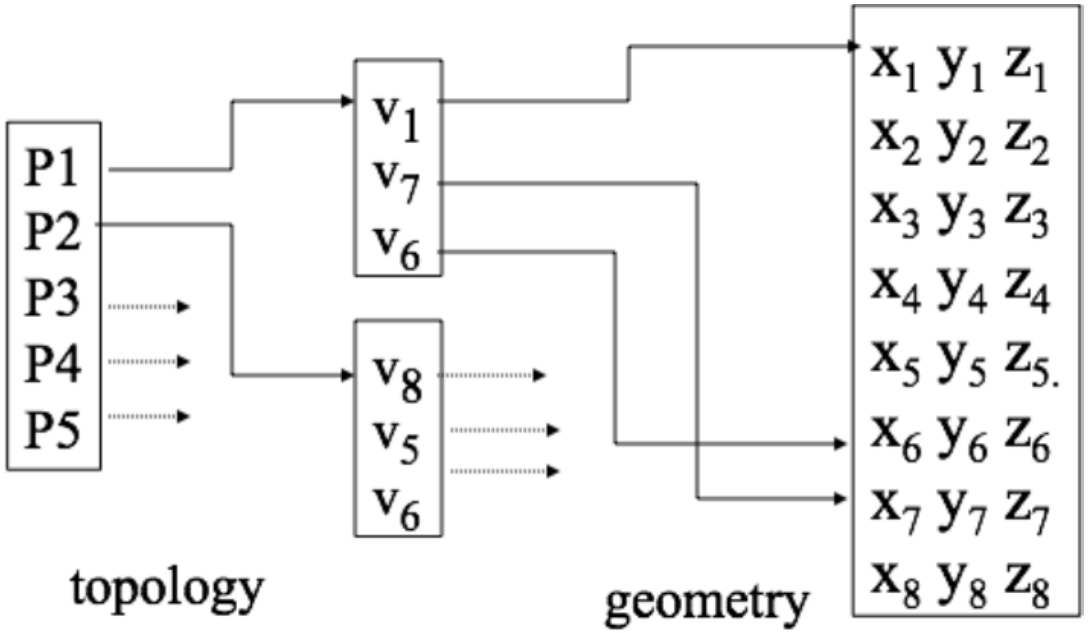
Using Element Buffer Objects (EBOs) TODO3

- VBOs are *Vertex Lists*, blending topology and geometry
  - Vertices may be redundant!



Using Element Buffer Objects (EBOs) TODO3

- EBOs are *Polygon Lists*, separating topology and geometry
  - Polygons can reuse vertices



Using Element Buffer Objects (EBOs) TODO3

- OBJ files are designed to use the concept of a polygon list:
  - Each triangle is defined as triplets of vertex data indices

```
#faces (index data)
# format: vertex/Uvcoordinate/normal
f 5/1/1 1/2/1 4/3/1
f 5/1/1 4/3/1 8/4/1
f 3/5/2 7/6/2 8/7/2
f 3/5/2 8/7/2 4/8/2
f 2/9/3 6/10/3 3/5/3
f 6/10/4 7/6/4 3/5/4
f 1/2/5 5/1/5 2/9/5
```

- From cube.obj

Using Element Buffer Objects (EBOs) TODO3

- We simply take into account these indices when loading in the model, using OBJLoaderV2.h
- **Model Loading:**

```
//TODO 3 load the models as EBOs instead of only VBOs
int cubeVertices;
GLuint cubeVAO = setupModelEBO(cubePath, cubeVertices); //Only one letter to change!
```

- **Drawing using elements:**

```
//TODO3 Draw model as elements, instead of as arrays
//glDrawElements(GL_TRIANGLES, activeVAOVertices, GL_UNSIGNED_INT, 0);
//This replace the glDrawArrays(...) code
```

Using Element Buffer Objects (EBOs) TODO3

- It will not make any visible difference to the rendering, but will significantly reduce data buffering to GPU
  - Redundant `vec3` data are replaced by `integer` data
  - This can reduce buffered data by a factor of 2 or more