3GC3 Fall 2022 - Assignment 2
Blendshapes for Face Expressions

In assignment 2, we will load meshes described in obj file format, render them in OpenGL and use blendshapes for creating various facial expressions.

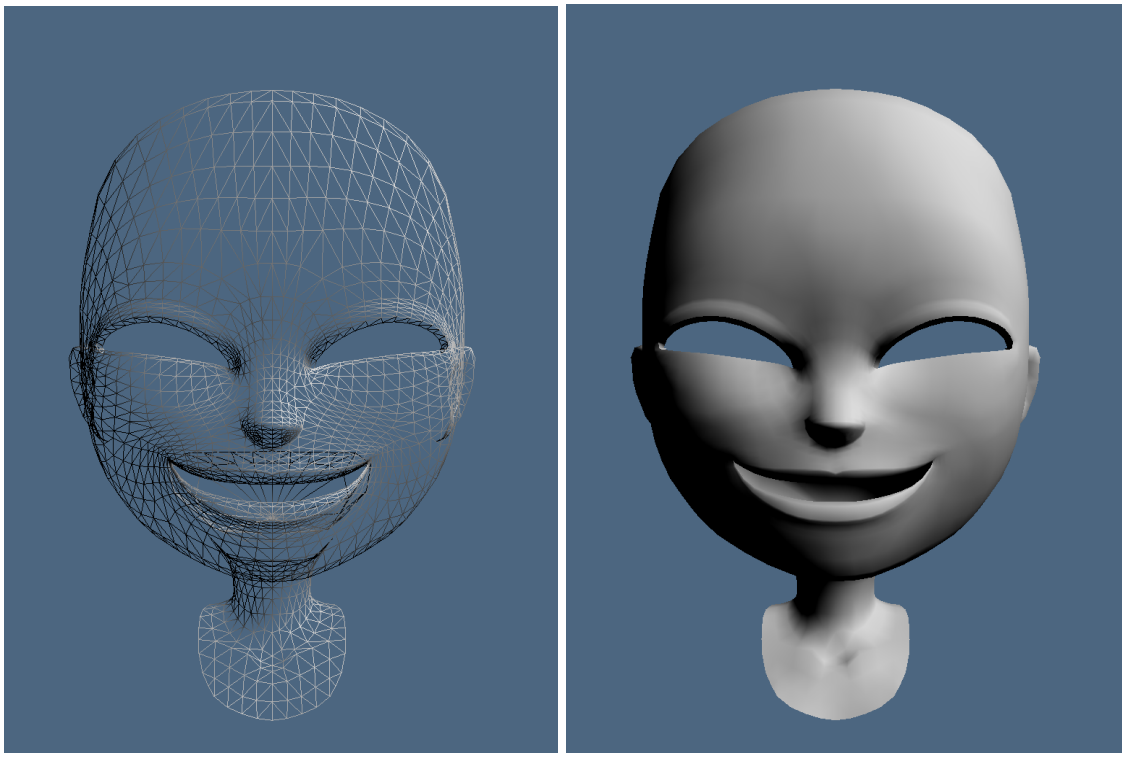

Figure 1. Facial Expression Rendered in Wireframe and Solid Mode Using OpenGL.

Task 1: Load and Render Mesh Using Tinyobj

1.1 Understanding Obj File Format

Obj is a very popular file format for describing meshes. Information stored in obj format is similar to the Indexed Triangle Mesh structure that we introduced in class. In the file, `v` indicates vertex position, `vt` indicates texture coordinate, `vn` indicates normal direction, and `f` indicates face construction with vertices, texture coord and normals, using format `f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3 ...`. For example, the cube we rendered in Assignment 1 in obj file format can be described as follows:

```
v -1.00000 -1.00000 1.00000
v 1.00000 -1.00000 1.00000
v -1.00000 1.00000 1.00000
v 1.00000 1.00000 1.00000
v -1.00000 1.00000 -1.00000
v 1.00000 1.00000 -1.00000
v -1.00000 -1.00000 -1.00000
v 1.00000 -1.00000 -1.00000
vt 0.000000 0.000000
vt 1.000000 0.000000
vt 0.000000 1.000000
vt 1.000000 1.000000
vn 0.000000 0.000000 1.000000
vn 0.000000 1.000000 0.000000
vn 0.000000 0.000000 -1.000000
vn 0.000000 -1.000000 0.000000
vn 1.000000 0.000000 0.000000
vn -1.000000 0.000000 0.000000
f 1/1/1 2/2/1 3/3/1
f 3/3/1 2/2/1 4/4/1
f 3/1/2 4/2/2 5/3/2
f 5/3/2 4/2/2 6/4/2
f 5/4/3 6/3/3 7/2/3
f 7/2/3 6/3/3 8/1/3
f 7/1/4 8/2/4 1/3/4
f 1/3/4 8/2/4 2/4/4
f 2/1/5 8/2/5 4/3/5
f 4/3/5 8/2/5 6/4/5
f 7/1/6 1/2/6 5/3/6
f 5/3/6 1/2/6 3/4/6
```

In the cube mesh, there are 8 unique vertices (defined by lines starting with `v`) and 6 unique face normals (defined by lines starting with `vn`). Ignore texture coordinate for this assignment. There are 12 triangle faces constructing the cube mesh (defined by lines starting with `f`). For example, based on `f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3`, line `f 1/1/1 2/2/1 3/3/1` means this face is a triangle face. It has three vertices with index 1, 2 and 3, and the normals associated with the vertices are with index 1, 1, 1.
For more details of obj file format you can check with https://en.wikipedia.org/wiki/Wavefront_.obj_file. The cube.obj file is provided for the assignment.

1.2 tinyobj loader

After understanding the obj file format, we will load meshes described in obj format to our OpenGL program and render them. Given the obj files, you don't need to implement

the obj file parser, but you can include the tinyobjloader library for loading obj meshes https://github.com/tinyobjloader/tinyobjloader.

Here is some example code:

```
tinyobj::attrib_t attrib;
std::vector<tinyobj::shape_t> shapes;
std::vector<tinyobj::material_t> materials;
// tinyobj load obj
std::string warn, err;
bool bTriangulate = true;
bool bSuc = tinyobj::LoadObj(&attrib, &shapes, &materials, &warn, &err,
                    obj_path.c_str(), nullptr, bTriangulate);
if(!bSuc)
{
   std::cout << "tinyobj error: " << err.c_str() << std::endl;
   return false;
}
return true;
```

The above code loads an obj file from obj_path. After loading the obj file into the program, you can access the faces, vertices and normals from shapes and attrib. The example code is listed below.

```
for(auto face : shapes[0].mesh.indices)
{
      int vid = face.vertex_index;
      int nid = face.normal_index;

      //fill in vertex buffer with vertex positions
      vbuffer.push_back(attrib.vertices[vid*3]);//vertex vid's x
      vbuffer.push_back(attrib.vertices[vid*3+1]);//vertex vid's y
      vbuffer.push_back(attrib.vertices[vid*3+2]);//vertex vid's z

      //fill in normal buffer with normal directions
      nbuffer.push_back(attrib.normals[nid*3]);
      nbuffer.push_back(attrib.normals[nid*3+1]);
      nbuffer.push_back(attrib.normals[nid*3+2]);
}
```

You need to get the vertex position and normal information and pass them to the OpenGL shaders to render them. You can choose to reuse the provided code or implement your own as you see fit.

1.3 Your Task to Render Meshes

Now you know the obj file format and how to load it to program using tinyobjloader. For all the rendering tasks, here are the common configurations:

Background color	(0.3f, 0.4f, 0.5f, 1.0f)
Window Size	Window Width = 1024; Window Height = 768;
Model Matrix	Identity
Projection Matrix	glm::mat4 proj = glm::perspective(glm::radians(60.0f), 4.0f / 3.0f, 0.1f, 1000.0f);

Fragment Shader	#version 330 core
	in vec3 Normal;
	out vec3 FragColor;
	void main()
	{
	float c = dot(Normal, vec3(0.8,0.7,0.6));
	FragColor = vec3(c,c,c);
	}

Others	Cull backface + Depth Test with GL_LESS

Table 1. Face Mesh Rendering Configuration

1.3.1 Render cube.obj

Load the provided cube.obj and render it in your OpenGL code. Set the view matrix as **glm::mat4 view = glm::lookAt(glm::vec3(3,4,5), glm::vec3(0,0,0), glm::vec3(0,1,0))**;
Press p to capture ppm and rename the ppm image to “cube_solid.ppm”.

Render the cube in wireframe mode by adding code
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
Press p to capture ppm and rename the ppm image to “cube_wire.ppm”.

1.3.2 Load and Render obj files for faces

You have a folder “faces”, containing obj files base.obj, 0~34.obj, totalling 36 obj files. They form the blendshapes for the face. Set the view matrix as
glm::mat4 view = glm::lookAt(glm::vec3(0,100,100), glm::vec3(0,100,0), glm::vec3(0,1,0));

Load and render base.obj in both solid and wireframe mode, and rename captured ppm images to “base_solid.ppm” and “base_wire.ppm” respectively.

Load and render the following obj files, all in solid mode:

0.obj, rename captured ppm image to “0.ppm”
1.obj, rename captured ppm image to “1.ppm”
7.obj, rename captured ppm image to “7.ppm”
8.obj, rename captured ppm image to “8.ppm”
13.obj, rename captured ppm image to “13.ppm””
14.obj, rename captured ppm image to “14.ppm”
23.obj, rename captured ppm image to “23.ppm”
24.obj, rename captured ppm image to “24.ppm”
27.obj, rename captured ppm image to “27.ppm”
28.obj, rename captured ppm image to “28.ppm”

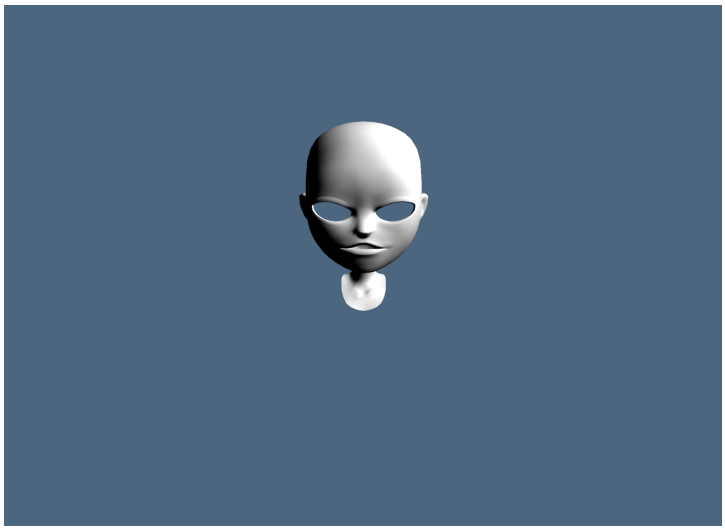

Figure 2. Example result of 20.obj rendered and captured with a different view matrix.

Task 2: Render Blendshapes for Facial Expressions

For Task 2, first you need to load all the obj files under the “faces” folder, i.e. base.obj, 0.obj ~ 34.obj, store them in the program without rendering them. Then in the code we will apply blendshape operations on the provided meshes to generate new facial expression shapes and render the generated shapes.

2.1 Blendshape Operations

New facial expression shape can be computed by the following formula:

$$\mathbf{Shape}_{\text{new expression}} = \mathbf{Shape}_{\text{base}} + \Sigma\, \mathbf{w}_i * (\mathbf{Shape}_i - \mathbf{Shape}_{\text{base}})$$

Note, you load shapes from base.obj and 0.obj ~ 34.obj, under the “faces” folder, and they all have the same number of vertices, the same order in the vertex list, and the same way how faces are constructed by vertices. The shape subtraction, summation and multiplication involved in the above formula is vertex-wise. For example, the following code is subtracting two shapes. (Let’s simplify the computation, and ignore normals and texture coordinates. During rendering, you can reuse base.obj’s normal for the generated facial expression)

```
assert(obj1.m_attrib.vertices.size() == obj2.m_attrib.vertices.size());
for(unsigned i = 0; i < obj1.m_attrib.vertices.size(); ++ i)
{
   result_vertices[i] = obj1.m_attrib.vertices[i] - obj2.m_attrib.vertices[i];
}
```

2.2 Blendshape Weights

w_i in the above formula are blending weights. Specifically, in this assignment, we need to know the weights $w_0 \sim w_{34}$ for the blending shapes. Weights are provided under folder “weights”, where each file includes 35 floating numbers, in the order for $w_0 \sim w_{34}$. According to the weight values described in one weight file, each file generates one facial expression shape.

2.3 Your Task to Compute Facial Expressions and Render Them

Read each weight file in, and compute the new facial expression shape according to the blending formula and the weight values, and render the resulting facial expression in OpenGL. Render configuration is the same as listed in Table 1, set view matrix as **glm::mat4 view = glm::lookAt(glm::vec3(0,100,100), glm::vec3(0,100,0), glm::vec3(0,1,0));** and only render facial expression shapes in solid mode.

Specifically, follow the order, load “0.weights” in, compute the blended shape given the weight value in “0.weights”, render the result shape, press p to capture ppm image, and rename it to “blended0.ppm”. Then load “1.weights” in, compute the blended shape, render the result shape, capture ppm and rename it to “blended1.ppm”. Do this for all the provided weight files, and capture ppm from “blended0.ppm” to “blended11.ppm”. Facial expression in Figure 1 illustrates one of the results.

Submission Checklist

Submit your **code** together with all the captured ppm images.

1 points - “cube_solid.ppm”
1 points - “cube_wire.ppm”
3 points - “base_solid.ppm”
1 points - “base_wire.ppm”
3 points - “0.ppm”
3 points - “1.ppm”
3 points - “7.ppm”
3 points - “8.ppm”
3 points - “13.ppm”
3 points - “14.ppm”
3 points - “23.ppm”
3 points - “24.ppm”
3 points - “27.ppm”
3 points - “28.ppm”
4 points - “blended0.ppm”
4 points - “blended1.ppm”
4 points - “blended2.ppm”
4 points - “blended3.ppm”
4 points - “blended4.ppm”
4 points - “blended5.ppm”
4 points - “blended6.ppm”
4 points - “blended7.ppm”
4 points - “blended8.ppm”
4 points - “blended9.ppm”
4 points - “blended10.ppm”
4 points - “blended11.ppm”

17 points - source code

Submission deadline is Nov. 5th, 2022. Past due submission will be penalized as in the course outline.