

CS6060- Final

Max Thrun

Fall 2013

Overview

The goal of this project was to render a 3D model of my head. A screenshot of the end result is shown below.

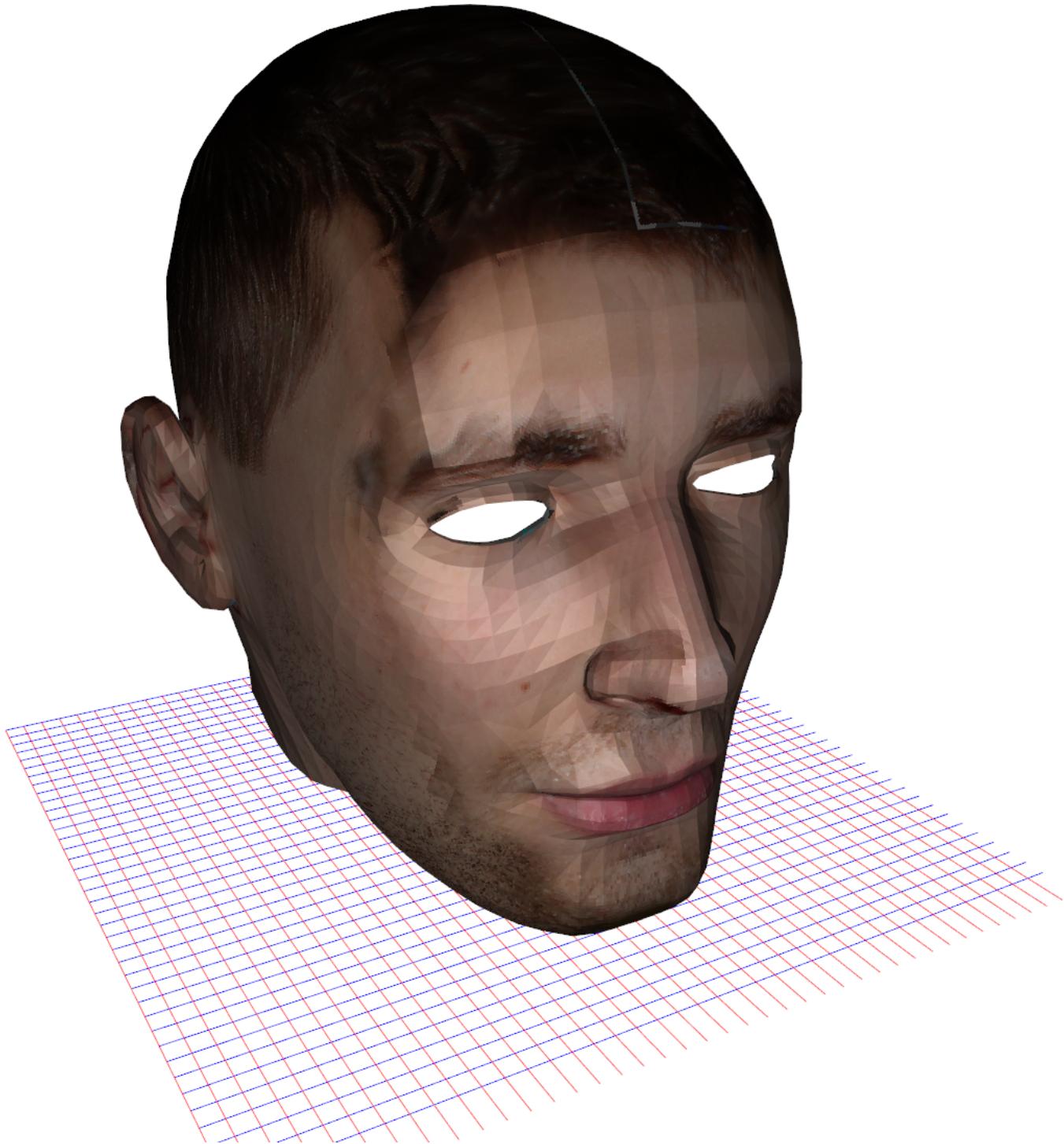


Figure 1: Program Screenshot

A video demonstration can be found here: <http://youtu.be/1Sn57Y0Xirw>

All code and resources can be found here: <https://github.com/bear24rw/CS6060/tree/master/final>

Head Model

The first step in this project was to model my head. I didn't have the time or experience to construct the model from scratch so I found a generic head online which is shown below.

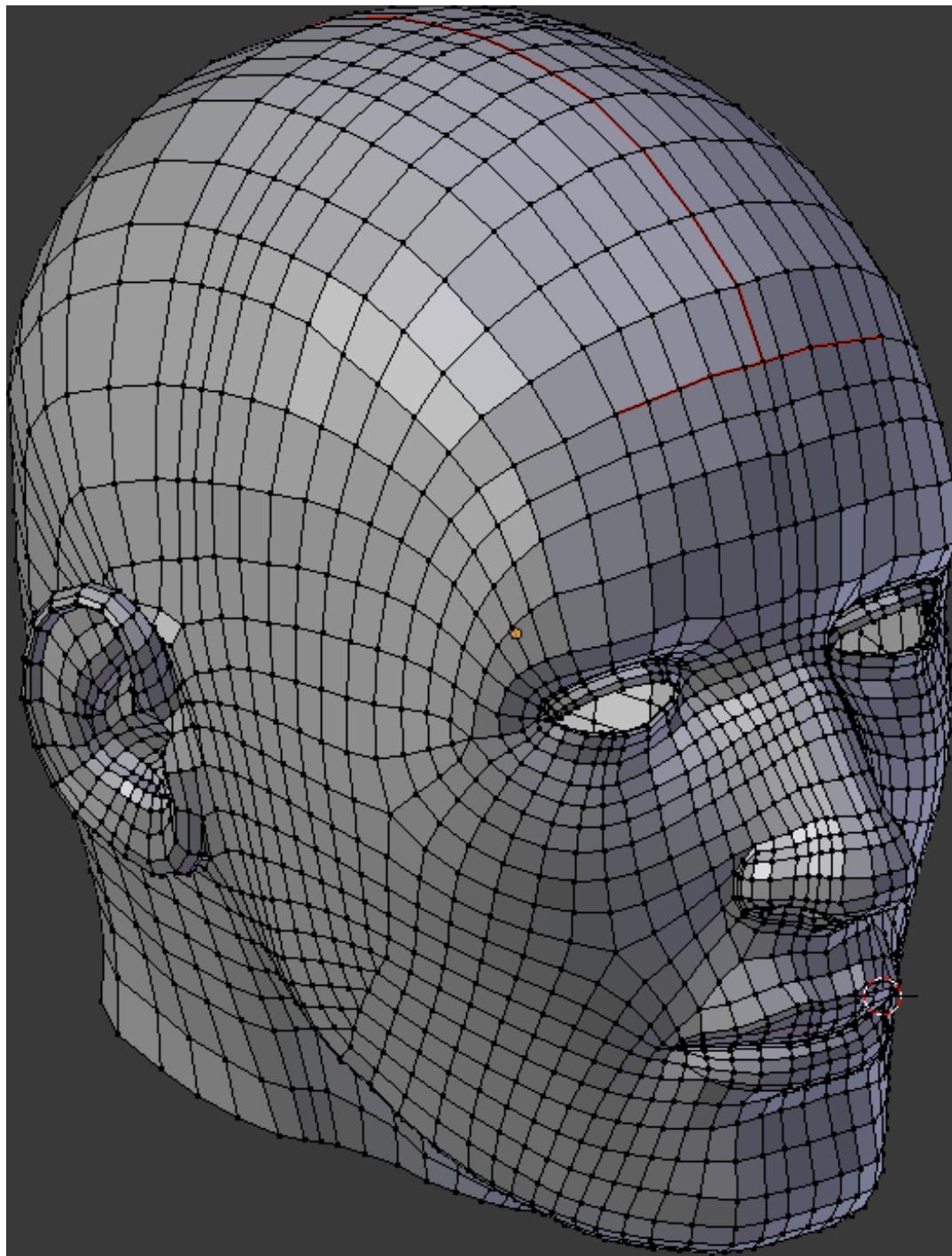
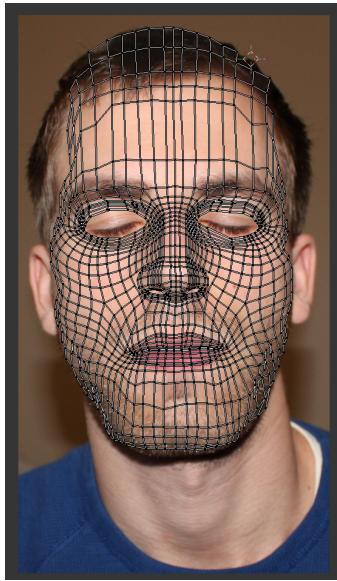
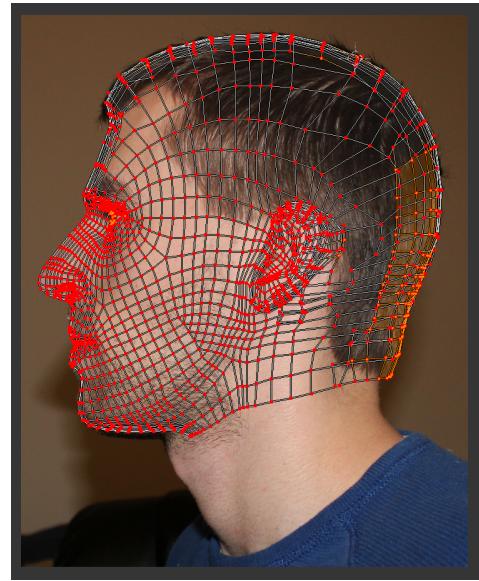


Figure 2: Generic 3D Head Model

In order to texture it I first took a front and side image of my head and then projection mapped them onto the model in Blender.



(a) Front Projection Map



(b) Side Projection Map

Figure 3: Projection Texture Mapping

The resulting mapping isn't totally perfect since the head model I used doesn't exactly match my face and I also only used a front and side image of my face. The images below show the model pre and post texturing.

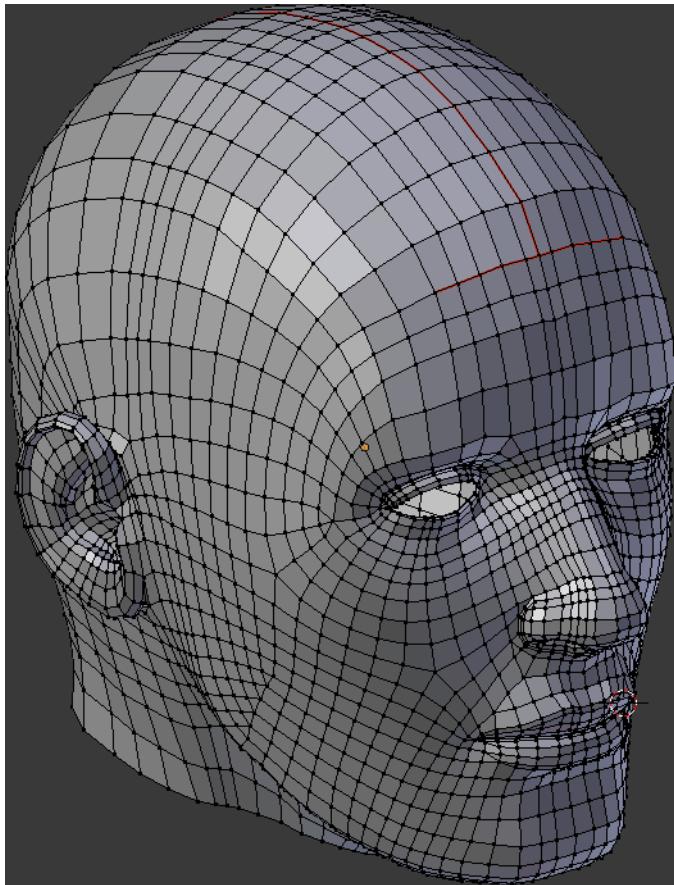


Figure 4: Untextured Model

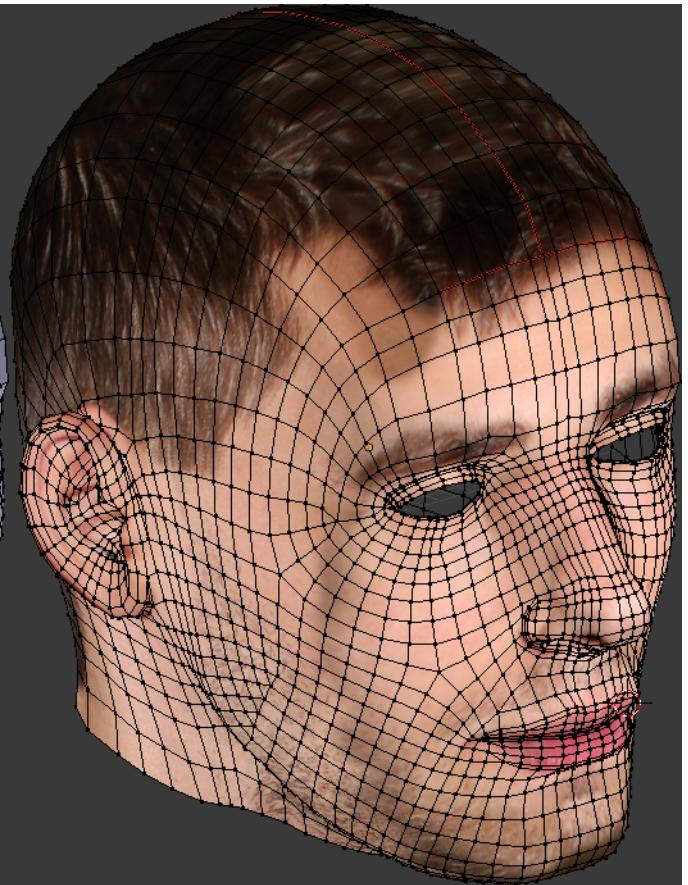


Figure 5: Textured Model

I then exported the model as an OBJ file and exported the texture as a PNG. The final texture is shown below.



Figure 6: Final Texture

Head Rendering

In order to render the head we first load the shaders and get IDs for all the uniforms that we will be manipulating.

```

21     shader.id = LoadShaders("head.vert.glsl", "head.frag.glsl");
22
23     model_id = glGetUniformLocation(shader.id, "M");
24     view_id = glGetUniformLocation(shader.id, "V");
25     proj_id = glGetUniformLocation(shader.id, "P");
26     light_id = glGetUniformLocation(shader.id, "light.pos.world");
27     texture_id = glGetUniformLocation(shader.id, "tex");

```

Listing 1: Load Shader (./code/head.cpp)

For loading the texture we take a shortcut and use the texture functionality provided by SFML. Since we are just loading a simple 2D texture there is no need to manually load and bind the texture with raw opengl calls.

```

29     texture.loadFromFile("../assets/uvmap.png");

```

Listing 2: Load Texture (./code/head.cpp)

We then create some vectors to hold the vertices, normals, and texture coordinates for our model and call a helper function to populate them.

```

32     std::vector<glm::vec3> indexed_vertices;
33     std::vector<glm::vec2> indexed_uvs;
34     std::vector<glm::vec3> indexed_normals;
35     load_obj("../assets/head.obj", indices, indexed_vertices, indexed_uvs, indexed_normals);

```

Listing 3: Read OBJ (./code/head.cpp)

In order to parse the OBJ file I used the Open Asset Import Library (Assimp). Assimp takes care of all the dirty work required to parse the OBJ and provides us with simple lists for the vertices, normals, and uv coordinates. The following code shows the body of the function which is responsible for opening the OBJ file with Assimp and populating vectors of GLM objects.

```

23     Assimp::Importer importer;
24
25     const aiScene* scene = importer.ReadFile(path, 0/* aiProcess_JoinIdenticalVertices | aiProcess_SortByPType */);
26     if( !scene) {
27         fprintf(stderr, importer.GetErrorString());
28         getchar();
29         return false;
30     }
31     const aiMesh* mesh = scene->mMeshes[0]; // In this simple example code we always use the 1rst mesh (in OBJ files there is often only one anyway)
32
33     // Fill vertices positions
34     printf("[assimp] filling vertice positions\n");
35     vertices.reserve(mesh->mNumVertices);
36     for(unsigned int i=0; i<mesh->mNumVertices; i++){
37         aiVector3D pos = mesh->mVertices[i];
38         vertices.push_back(glm::vec3(pos.x, pos.y, pos.z));
39     }
40
41     // Fill vertices texture coordinates
42     printf("[assimp] filling vertice tex coords\n");
43     uvs.reserve(mesh->mNumVertices);
44     for(unsigned int i=0; i<mesh->mNumVertices; i++){
45         aiVector3D UW = mesh->mTextureCoords[0][i]; // Assume only 1 set of UV coords; Assimp supports 8 UV sets.
46         uvs.push_back(glm::vec2(UW.x, UW.y));
47     }
48
49     // Fill vertices normals
50     printf("[assimp] filling vertice normals\n");
51     normals.reserve(mesh->mNumVertices);
52     for(unsigned int i=0; i<mesh->mNumVertices; i++){
53         aiVector3D n = mesh->mNormals[i];
54         normals.push_back(glm::vec3(n.x, n.y, n.z));
55     }
56
57     // Fill face indices
58     printf("[assimp] filling face indices\n");
59     indices.reserve(3*mNumFaces);
60     for (unsigned int i=0; i<mesh->mNumFaces; i++){
61         // Assume the model has only triangles.
62         indices.push_back(mesh->mFaces[i].mIndices[0]);
63         indices.push_back(mesh->mFaces[i].mIndices[1]);
64         indices.push_back(mesh->mFaces[i].mIndices[2]);
65     }

```

Listing 4: OBJ Loading (./code/objloader.cpp)

Next we want to be able to place the bottom of the model at Y=0 and also center the camera target to the middle of the model. To do this we simply loop through each vertices of the model and keep track of the lowest one. We will use this offset later in the render function. The camera is targeted to three fourths of the way up the model which was determined experimentally to be a nice position.

```

32     std::vector<glm::vec3> indexed_vertices;
33     std::vector<glm::vec2> indexed_uvs;
34     std::vector<glm::vec3> indexed_normals;
35     load_obj("../assets/head.obj", indices, indexed_vertices, indexed_uvs, indexed_normals);
36
37     offset = 0.0f;
38     for (glm::vec3 i : indexed_vertices) {
39         if (i.y < offset) offset = i.y;
40     }
41
42     cam->target.y = -offset * 0.75f;

```

Listing 5: Calculate model offset (./code/head.cpp)

We then get the shader attribute locations for the vertices, normals, and texture. We then create a Vertex Array Object (VAO) which will keep track of all the bindings and attribute parameters we are about to set. Then, for each attribute, we generate a vertex buffer, bind it, upload the data for it with `glBufferData`, set the parameters for the data with `glVertexAttribPointer`, and finally enable the attribute. We also bind the texture after this.

```

46     GLuint posAttrib = glGetAttribLocation(shader_id, "vert.pos.model");
47     GLuint norAttrib = glGetAttribLocation(shader_id, "vert.nor.model");
48     GLuint texAttrib = glGetAttribLocation(shader_id, "vert.tex");
49
50     glGenVertexArrays(1, &vao);
51     glBindVertexArray(vao);
52
53     glGenBuffers(1, &vertexbuffer);
54     glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
55     glBufferData(GL_ARRAY_BUFFER, indexed.vertices.size() * sizeof(glm::vec3), &indexed.vertices[0], GL_STATIC_DRAW);
56     glVertexAttribPointer(posAttrib, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);
57     glEnableVertexAttribArray(posAttrib);
58
59     glGenBuffers(1, &normalbuffer);
60     glBindBuffer(GL_ARRAY_BUFFER, normalbuffer);
61     glBufferData(GL_ARRAY_BUFFER, indexed.normals.size() * sizeof(glm::vec3), &indexed.normals[0], GL_STATIC_DRAW);
62     glVertexAttribPointer(norAttrib, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);
63     glEnableVertexAttribArray(norAttrib);
64
65     glGenBuffers(1, &uvbuffer);
66     glBindBuffer(GL_ARRAY_BUFFER, uvbuffer);
67     glBufferData(GL_ARRAY_BUFFER, indexed.uvs.size() * sizeof(glm::vec2), &indexed.uvs[0], GL_STATIC_DRAW);
68     glVertexAttribPointer(texAttrib, 2, GL_FLOAT, GL_FALSE, 0, (void*)0);
69     glEnableVertexAttribArray(texAttrib);
70
71     glGenBuffers(1, &elementbuffer);
72     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, elementbuffer);
73     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(unsigned short), &indices[0], GL_STATIC_DRAW);
74
75     sf::Texture::bind(&texture);

```

Listing 6: Setup Bind and Load Buffers (./code/head.cpp)

The rendering function is extremely simple. We first activate the shader responsible for rendering the head. We then create an identity model matrix which is first rotated and then translated to achieve the animation and place the head on the 'ground'. We then send the updated matrices to the shader using `glUniformMatrix4fv`. We also update the position of the light to the position of the camera. Finally we bind our VAO and draw all the elements of our model.

```

80     void Head::render()
81     {
82         glUseProgram(shader_id);
83
84         glm::mat4 model = glm::mat4(1.0f);
85         model = glm::rotate(model, rotation, glm::vec3(0, 1, 0));
86         model = glm::translate(model, glm::vec3(0.0f, -offset, 0.0f));
87
88         glUniformMatrix4fv(proj_id, 1, GL_FALSE, glm::value_ptr(cam->proj.mat()));
89         glUniformMatrix4fv(view_id, 1, GL_FALSE, glm::value_ptr(cam->view.mat()));
90         glUniformMatrix4fv(model_id, 1, GL_FALSE, glm::value_ptr(model));
91
92         glUniform3f(light_id, cam->position.x, cam->position.y, cam->position.z);
93
94         glBindVertexArray(vao);
95         glDrawElements(GL_TRIANGLES, indices.size(), GL_UNSIGNED_SHORT, (void*)0);
96     }

```

Listing 7: Render Function (./code/head.cpp)

Animation

A simple rotation animation is achieved by keeping track of total rotation angle. In the main render loop the `rotation` instance variable is incremented at a rate of 10 degrees per second. `delta` is the lapsed time since the last loop.

```
113     if (rotate_mode)
114         head.rotation += 10.0f * delta.asSeconds();
```

Listing 8: Rotation (./code/main.cpp)

The head rendering function, discussed in the previous section, then simply rotates the model matrix by the number of degrees accumulated in the `rotation` variable.

```
84     glm::mat4 model = glm::mat4(1.0f);
85     model = glm::rotate(model, rotation, glm::vec3(0,1,0));
```

Listing 9: Model Matrix Rotation (./code/head.cpp)

Lighting

Lighting is achieved through the combination of diffuse, ambient, and specular components.

The ambient component simply bumps up the brightness of each fragment to simulate a global indirect light source.

The diffuse component takes into account the distance of the fragment to the light source and angle of the income light and the fragments normal.

The specular component accounts for the fact that there will be more light reflecting in the direction that the light intersected the surface.

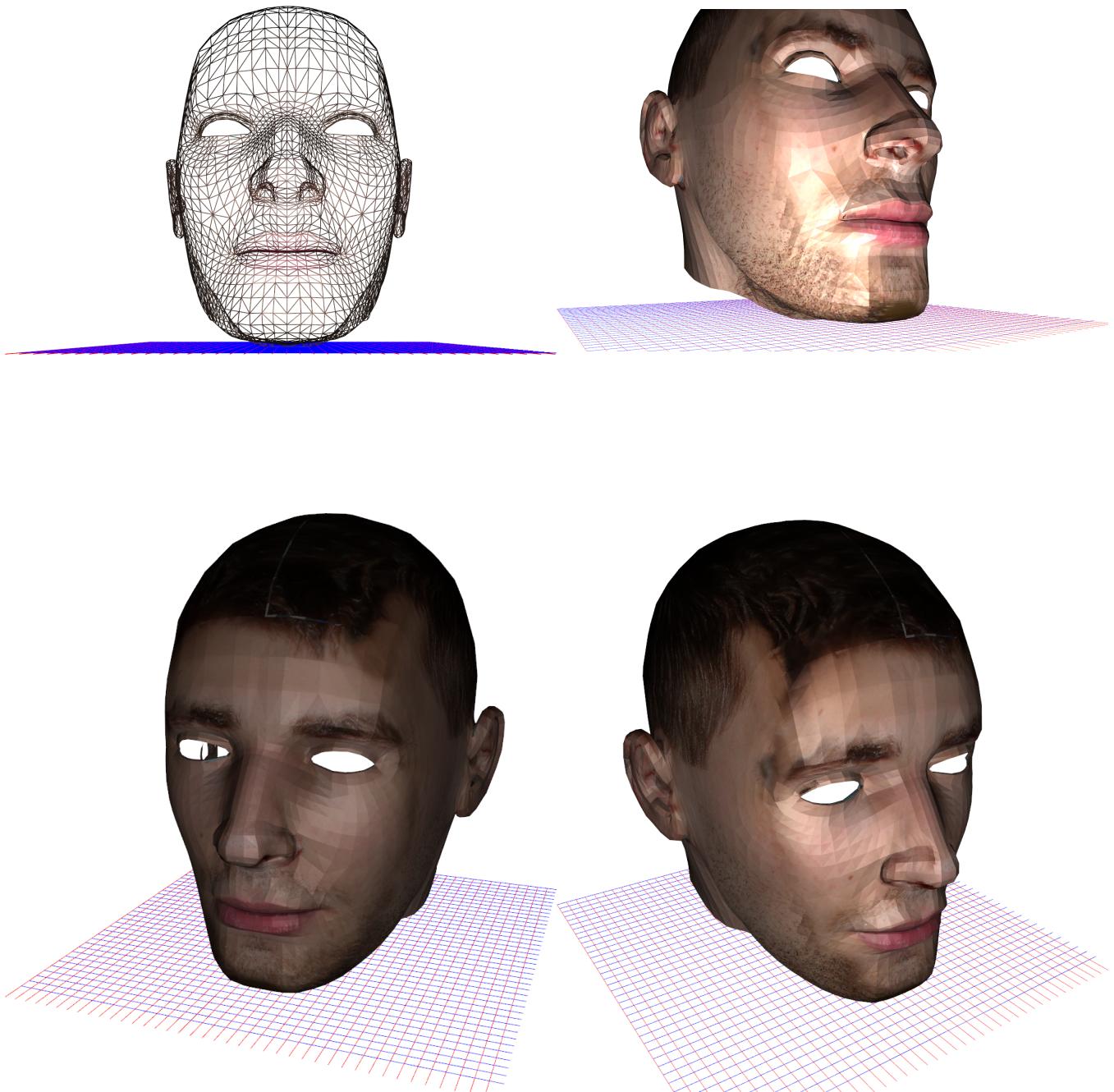
```
18 void main() {
19
20     // Light emission properties
21     vec3 light.color = vec3(1,1,1);
22     float light.power = 1500.0f;
23
24     // Material properties
25     vec3 diffuse_color = texture2D(tex_UV).rgb;
26     vec3 ambient_color = vec3(0.1,0.1,0.1) * diffuse_color;
27     vec3 specular_color = vec3(0.3,0.3,0.3);
28
29     // Distance to the light
30     float distance = length(light.pos_world - vert.pos_world );
31
32     vec3 n = normalize(vert.nor_cam);
33     vec3 l = normalize(light.dir_cam);
34     float cosTheta = clamp(dot(n,l), 0,1);
35
36     vec3 E = normalize(eye.dir_cam);
37     vec3 R = reflect(-l,n);
38     float cosAlpha = clamp(dot(E,R), 0,1);
39
40     color = ambient_color +
41             diffuse_color * light.color * light.power * cosTheta / (distance*distance) +
42             specular_color * light.color * light.power * pow(cosAlpha,5) / (distance*distance);
43
44 }
```

Listing 10: Head Fragment Shader (./code/head_frag.glsL)

The shader code to achieve these effects was heavily lifted from this tutorial:

<http://www.opengl-tutorial.org/beginners-tutorials/tutorial-8-basic-shading/>

Screenshots



Code

```

1 #define GLEW_STATIC
2 #include <vector>
3 #include <GL/glew.h>
4 #include <SFML/Graphics.hpp>
5 #include <SFML/OpenGL.hpp>
6 #include <iostream>
7 #include <ctime>
8 #include <glm/glm.hpp>
9 #include <glm/gtc/matrix_transform.hpp>
10 #include <glm/gtx/rotate_vector.hpp>
11 #include <glm/gtc/type_ptr.hpp>
12 #include "camera.h"
13 #include "grid.h"
14 #include "head.h"
15
16 int main()
17 {
18     sf::ContextSettings contextSettings;
19     contextSettings.depthBits = 32;
20     contextSettings.antialiasingLevel = 8;
21
22     sf::RenderWindow window(sf::VideoMode(800, 800), "Test 1", sf::Style::Default, contextSettings);
23     window.setVerticalSyncEnabled(true);
24
25     window.SetActive();
26
27     glewExperimental = GL_TRUE;
28     glewInit();
29
30     Camera cam;
31     Grid grid(&cam);
32     Head head(&cam);
33
34     int mouse_down = 0;
35     int last_mouse_x = 0;
36     int last_mouse_y = 0;
37
38     int poly_mode = 0;
39     int rotate_mode = 0;
40
41     sf::Clock clock;
42
43     printf("Starting render loop\n");
44     while (window.isOpen())
45     {
46         sf::Time delta = clock.restart();
47
48         sf::Event event;
49         while (window.pollEvent(event))
50         {
51             if (event.type == sf::Event::Closed)
52                 window.close();
53
54             if (event.type == sf::Event::KeyPressed) {
55                 switch(event.key.code) {
56                     case sf::Keyboard::Escape:
57                         window.close();
58                         break;
59                     case sf::Keyboard::P:
60                         poly_mode = !poly_mode;
61                         break;
62                     case sf::Keyboard::Space:
63                         rotate_mode = !rotate_mode;
64                         break;
65                 }
66             }
67             if (event.type == sf::Event::MouseButtonPressed) {
68                 if (event.mouseButton.button == sf::Mouse::Left)
69                     mouse_down = 1;
70             }
71             if (event.type == sf::Event::MouseButtonReleased) {
72                 if (event.mouseButton.button == sf::Mouse::Left)
73                     mouse_down = 0;
74             }
75             if (event.type == sf::Event::MouseMove) {
76                 int dx = event.mouseMove.x - last_mouse_x;
77                 int dy = event.mouseMove.y - last_mouse_y;
78
79                 if (mouse_down) {
80                     cam.increase_angle_v((float)dy * delta.asSeconds() * 0.2f);
81                     cam.increase_angle_h((float)dx * delta.asSeconds() * 0.2f * -1.0f);
82                 }
83
84                 last_mouse_x = event.mouseMove.x;
85                 last_mouse_y = event.mouseMove.y;
86             }
87             if (event.type == sf::Event::MouseWheelMoved) {
88                 cam.increase_distance(-event.mouseWheel.delta);
89             }
90             if (event.type == sf::Event::Resized) {
91                 glViewport(0, 0, event.size.width, event.size.height);
92                 cam.set_size(event.size.width, event.size.height);
93             }
94         }
95     }
96
97     glEnable(GL_DEPTH_TEST);
98     glDepthFunc(GL_LESS);
99     glEnable(GL_CULL_FACE);
100
101     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
102     glClearColor(1.0, 1.0, 1.0, 0.0);

```

```

105     if (poly_mode) {
106         glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
107     } else {
108         glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
109         glLineWidth(1);
110     }
111 }
112
113 if (rotate_mode)
114     head.rotation += 10.0f * delta.asSeconds();
115
116 grid.render();
117 head.render();
118
119 window.display();
120 }
121
122 return 0;
123 }
```

Listing 11: main (./code/main.cpp)

```

1 #include <GL/glew.h>
2 #include <SFML/Graphics.hpp>
3 #include <SFML/OpenGL.hpp>
4 #include <glm/glm.hpp>
5 #include <glm/gtc/noise.hpp>
6 #include <glm/gtc/matrix_transform.hpp>
7 #include <glm/gtc/type_ptr.hpp>
8 #include "camera.h"
9 #include "shader.h"
10 #include "objloader.h"
11 #include "head.h"
12
13 Head::Head() {}
14
15 Head::Head(Camera *_cam)
16 {
17     cam = _cam;
18
19     rotation = 0.0f;
20
21     shader.id = LoadShaders("head.vert.glsl", "head.frag.glsl");
22
23     model_id = glGetUniformLocation(shader.id, "M");
24     view_id = glGetUniformLocation(shader.id, "V");
25     proj_id = glGetUniformLocation(shader.id, "P");
26     light_id = glGetUniformLocation(shader.id, "light_pos_world");
27     texture_id = glGetUniformLocation(shader.id, "tex");
28
29     texture.loadFromFile("../assets/uvmap.png");
30
31     // read the obj file
32     std::vector<glm::vec3> indexed_vertices;
33     std::vector<glm::vec2> indexed_uvs;
34     std::vector<glm::vec3> indexed_normals;
35     load_obj("../assets/head.obj", indices, indexed_vertices, indexed_uvs, indexed_normals);
36
37     offset = 0.0f;
38     for (glm::vec3 i : indexed_vertices)
39         if (i.y < offset) offset = i.y;
40     }
41
42     cam->target.y = -offset * 0.75f;
43
44     printf("[head] offset: %f\n", offset);
45
46     GLuint posAttrib = glGetAttribLocation(shader.id, "vert.pos_model");
47     GLuint norAttrib = glGetAttribLocation(shader.id, "vert.nor_model");
48     GLuint texAttrib = glGetAttribLocation(shader.id, "vert_tex");
49
50     glGenVertexArrays(1, &vao);
51     glBindVertexArray(vao);
52
53     glGenBuffers(1, &vertexbuffer);
54     glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
55     glBufferData(GL_ARRAY_BUFFER, indexed_vertices.size() * sizeof(glm::vec3), &indexed_vertices[0], GL_STATIC_DRAW);
56     glVertexAttribPointer(posAttrib, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);
57     glEnableVertexAttribArray(posAttrib);
58
59     glGenBuffers(1, &normalbuffer);
60     glBindBuffer(GL_ARRAY_BUFFER, normalbuffer);
61     glBufferData(GL_ARRAY_BUFFER, indexed_normals.size() * sizeof(glm::vec3), &indexed_normals[0], GL_STATIC_DRAW);
62     glVertexAttribPointer(norAttrib, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);
63     glEnableVertexAttribArray(norAttrib);
64
65     glGenBuffers(1, &uvbuffer);
66     glBindBuffer(GL_ARRAY_BUFFER, uvbuffer);
67     glBufferData(GL_ARRAY_BUFFER, indexed_uvs.size() * sizeof(glm::vec2), &indexed_uvs[0], GL_STATIC_DRAW);
68     glVertexAttribPointer(texAttrib, 2, GL_FLOAT, GL_FALSE, 0, (void*)0);
69     glEnableVertexAttribArray(texAttrib);
70
71     glGenBuffers(1, &elementbuffer);
72     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, elementbuffer);
73     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(unsigned short), &indices[0], GL_STATIC_DRAW);
74
75     sf::Texture::bind(&texture);
76
77     glBindVertexArray(0);
78 }
79
80 void Head::render()
81 {
82     glUseProgram(shader.id);
```

```

84     glm::mat4 model = glm::mat4(1.0f);
85     model = glm::rotate(model, rotation, glm::vec3(0, 1, 0));
86     model = glm::translate(model, glm::vec3(0.0f, -offset, 0.0f));
87
88     glUniformMatrix4fv(proj_id, 1, GL_FALSE, glm::value_ptr(cam->proj.mat()));
89     glUniformMatrix4fv(view_id, 1, GL_FALSE, glm::value_ptr(cam->view.mat()));
90     glUniformMatrix4fv(model_id, 1, GL_FALSE, glm::value_ptr(model));
91
92     glUniform3f(light_id, cam->position.x, cam->position.y, cam->position.z);
93
94     glBindVertexArray(vao);
95     glDrawElements(GL_TRIANGLES, indices.size(), GL_UNSIGNED_SHORT, (void*)0);
96 }

```

Listing 12: head (./code/head.cpp)

```

1 #pragma once
2 #include <SFML/Graphics.hpp>
3 #include <SFML/OpenGL.hpp>
4 #include <vector>
5 #include "camera.h"
6
7 class Head {
8 public:
9     Head();
10    Head(Camera *);
11    void render();
12    float rotation;
13 private:
14     std::vector<unsigned short> indices;
15     GLuint vao;
16     GLuint vertexbuffer;
17     GLuint uvbuffer;
18     GLuint normalbuffer;
19     GLuint elementbuffer;
20     GLuint shader_id;
21     GLuint model_id;
22     GLuint view_id;
23     GLuint proj_id;
24     GLuint texture_id;
25     sf::Texture texture;
26     GLuint light_id;
27     Camera *cam;
28     float offset;
29 };

```

Listing 13: head (./code/head.h)

```

1 #include <GL/glew.h>
2 #include <SFML/Graphics.hpp>
3 #include <SFML/OpenGL.hpp>
4 #include <glm/glm.hpp>
5 #include <glm/gtc/noise.hpp>
6 #include <glm/gtc/matrix_transform.hpp>
7 #include <glm/gtc/type_ptr.hpp>
8 #include "camera.h"
9 #include "shader.h"
10 #include "grid.h"
11
12 Grid::Grid() {}
13
14 Grid::Grid(Camera *_cam)
15 {
16     cam = _cam;
17
18     #define GRID_SIZE 20
19
20     for(int k = -GRID_SIZE; k<GRID_SIZE; k++) {
21         // start of x line / color / end of x line / color
22         verts.push_back(k); verts.push_back(0); verts.push_back(-GRID_SIZE);
23         verts.push_back(1); verts.push_back(0); verts.push_back(0);
24         verts.push_back(k); verts.push_back(0); verts.push_back(GRID_SIZE);
25         verts.push_back(1); verts.push_back(0); verts.push_back(0);
26         // start of z line / color / end of z line / color
27         verts.push_back(-GRID_SIZE); verts.push_back(0); verts.push_back(k);
28         verts.push_back(0); verts.push_back(0); verts.push_back(1);
29         verts.push_back(GRID_SIZE); verts.push_back(0); verts.push_back(k);
30         verts.push_back(0); verts.push_back(0); verts.push_back(1);
31     }
32
33     glGenVertexArrays(1, &vao);
34     glBindVertexArray(vao);
35
36     glGenBuffers(1, &vbo);
37     glBindBuffer(GL_ARRAY_BUFFER, vbo);
38     glBufferData(GL_ARRAY_BUFFER, verts.size()*sizeof(float), verts.data(), GL_STATIC_DRAW);
39
40     printf("[grid] Loading shaders\n");
41     shader_id = LoadShaders("grid.vert.glsl", "grid.frag.glsl");
42
43     mvp_id = glGetUniformLocation(shader_id, "mvp");
44     GLint posAttrib = glGetAttribLocation(shader_id, "position");
45     GLint colAttrib = glGetAttribLocation(shader_id, "color");
46     printf("[grid] %d %d\n", posAttrib, colAttrib);
47     glEnableVertexAttribArray(posAttrib);
48     glEnableVertexAttribArray(colAttrib);
49     glVertexAttribPointer(posAttrib, 3, GL_FLOAT, GL_FALSE, 6*sizeof(GLfloat), 0);
50     glVertexAttribPointer(colAttrib, 3, GL_FLOAT, GL_FALSE, 6*sizeof(GLfloat), (void*)(3*sizeof(GLfloat)));
51
52 }

```

```

54 void Grid::render()
55 {
56     glUseProgram(shader.id);
57
58     glm::mat4 mvp = cam->proj_mat() * cam->view_mat() * glm::mat4(1.0f);
59     glUniformMatrix4fv(mvp.id, 1, GL_FALSE, glm::value_ptr(mvp));
60
61     glBindVertexArray(vao);
62     glDrawArrays(GL_LINES, 0, verts.size()/6 - 2);
63 }

```

Listing 14: grid (.../code/grid.cpp)

```

1 #pragma once
2 #include <SFML/Graphics.hpp>
3 #include <SFML/OpenGL.hpp>
4 #include <vector>
5 #include "camera.h"
6
7 class Grid {
8 public:
9     Grid();
10    Grid(Camera*);
11    void render();
12 private:
13     Camera *cam;
14     std::vector<float> verts;
15     GLuint vao;
16     GLuint vbo;
17     GLuint shader.id;
18     GLuint mvp.id;
19 };

```

Listing 15: grid (.../code/grid.h)

```

1 #include <glm/glm.hpp>
2 #include <glm/gtc/matrix_transform.hpp>
3
4 #include "camera.h"
5
6 Camera::Camera()
7 {
8     distance = 50.0f;
9     target = glm::vec3(0.0f);
10    angle.horz = 45.0f;
11    angle.vert = 20.0f;
12    fov = 45.0f;
13    _width = 800;
14    _height = 600;
15    update_vectors();
16 }
17
18 void Camera::increase_angle_v(float angle)
19 {
20     angle.vert += angle;
21     update_vectors();
22 }
23
24 void Camera::increase_angle_h(float angle)
25 {
26     angle.horz += angle;
27     update_vectors();
28 }
29
30 void Camera::increase_distance(float dist)
31 {
32     distance += dist;
33     update_vectors();
34 }
35
36 void Camera::update_vectors()
37 {
38     direction = glm::vec3(
39         cos(angle.vert) * sin(angle.horz),
40         sin(angle.vert),
41         cos(angle.vert) * cos(angle.horz)
42     );
43
44     right = glm::vec3(
45         sin(angle.horz - 3.14f/2.0f),
46         0,
47         cos(angle.horz - 3.14f/2.0f)
48     );
49
50     up = glm::cross(right, direction);
51
52     position = distance * direction;
53 }
54
55 glm::mat4 Camera::view_mat(void)
56 {
57     return glm::lookAt(
58         target + (distance * direction),
59         target,
60         up);
61 }
62
63 glm::mat4 Camera::proj_mat(void)
64 {
65     return glm::perspective(fov, (_width/(_height, 0.01f, 10000.0f));

```

```

66     }
68 void Camera::set.size(int width, int height)
69 {
70     _width = width;
71     _height = height;
72 }
```

Listing 16: camera (./code/camera.cpp)

```

1 #pragma once
3
5 class Camera
6 {
7     public:
8         Camera();
9         glm::vec3 position;
10        glm::vec3 direction;
11        glm::vec3 right;
12        glm::vec3 up;
13        glm::vec3 target;
14        glm::mat4 proj.mat(void);
15        glm::mat4 view.mat(void);
16        void increase.angle.v(float);
17        void increase.angle.h(float);
18        void increase.distance(float);
19        void set.size(int, int);
20        float distance;
21        float angle.horz;
22        float angle.vert;
23        float fov;
24     private:
25        void update.vectors(void);
26        int _width;
27        int _height;
28 };
```

Listing 17: camera (./code/camera.h)

```

1 #include <vector>
2 #include <stdio.h>
3 #include <string>
4 #include <cstring>
6
6 #include <glm/glm.hpp>
8
8 #include "objloader.h"
10
10 // Include Assimp
11 #include <assimp/Importer.hpp>           // C++ importer interface
12 #include <assimp/scene.hpp>              // Output data structure
13 #include <assimp/postprocess.hpp>          // Post processing flags
15
15 bool load_obj(
16     const char * path,
17     std::vector<unsigned short> & indices,
18     std::vector<glm::vec3> & vertices,
19     std::vector<glm::vec2> & uvs,
20     std::vector<glm::vec3> & normals
21 ){
23
23     Assimp::Importer importer;
25
25     const aiScene* scene = importer.ReadFile(path, 0/* aiProcess_JoinIdenticalVertices | aiProcess_SortByPType */);
26     if( !scene ) {
27         fprintf(stderr, importer.GetErrorString());
28         getchar();
29         return false;
30     }
31     const aiMesh* mesh = scene->mMeshes[0]; // In this simple example code we always use the first mesh (in OBJ files there is often only one anyway)
33
33     // Fill vertices positions
34     printf("[assimp] filling vertice positions\n");
35     vertices.reserve(mesh->mNumVertices);
36     for(unsigned int i=0; i<mesh->mNumVertices; i++){
37         aiVector3D pos = mesh->mVertices[i];
38         vertices.push_back(glm::vec3(pos.x, pos.y, pos.z));
39     }
41
41     // Fill vertices texture coordinates
42     printf("[assimp] filling vertice tex coords\n");
43     uvs.reserve(mesh->mNumVertices);
44     for(unsigned int i=0; i<mesh->mNumVertices; i++){
45         aiVector3D UW = mesh->mTextureCoords[0][i]; // Assume only 1 set of UV coords; Assimp supports 8 UV sets.
46         uvs.push_back(glm::vec2(UW.x, UW.y));
47     }
49
49     // Fill vertices normals
50     printf("[assimp] filling vertice normals\n");
51     normals.reserve(mesh->mNumVertices);
52     for(unsigned int i=0; i<mesh->mNumVertices; i++){
53         aiVector3D n = mesh->mNormals[i];
54         normals.push_back(glm::vec3(n.x, n.y, n.z));
55     }
57
57     // Fill face indices
58     printf("[assimp] filling face indices\n");
59     indices.reserve(3*mesh->mNumFaces);
60     for (unsigned int i=0; i<mesh->mNumFaces; i++) {
```

```

61         // Assume the model has only triangles.
62         indices.push_back(mesh->mFaces[i].mIndices[0]);
63         indices.push_back(mesh->mFaces[i].mIndices[1]);
64         indices.push_back(mesh->mFaces[i].mIndices[2]);
65     }
66
67     // The "scene" pointer will be deleted automatically by "importer"
68     return true;
69 }
```

Listing 18: objloader (../code/objloader.cpp)

```

1 #ifndef OBJLOADER_H
2 #define OBJLOADER_H
3
4 bool load_obj(
5     const char * path,
6     std::vector<unsigned short> & indices,
7     std::vector<glm::vec3> & vertices,
8     std::vector<glm::vec2> & uvs,
9     std::vector<glm::vec3> & normals
10 );
11
12 #endif
```

Listing 19: objloader (../code/objloader.h)

```

1 #include <stdio.h>
2 #include <string>
3 #include <vector>
4 #include <iostream>
5 #include <fstream>
6 #include <algorithm>
7 using namespace std;
8
9 #include <stdlib.h>
10 #include <string.h>
11
12 #include <GL/glew.h>
13
14 #include "shader.h"
15
16 GLuint LoadShaders(const char * vertex_file_path ,const char * fragment_file_path){
17
18     // Create the shaders
19     GLuint VertexShaderID = glCreateShader(GL_VERTEX_SHADER);
20     GLuint FragmentShaderID = glCreateShader(GL_FRAGMENT_SHADER);
21
22     // Read the Vertex Shader code from the file
23     std::string VertexShaderCode;
24     std::ifstream VertexShaderStream(vertex_file_path , std::ios::in);
25     if(VertexShaderStream.is.open()){
26         std::string Line = "";
27         while(getline(VertexShaderStream , Line))
28             VertexShaderCode += "\n" + Line;
29         VertexShaderStream.close();
30     }else{
31         printf("Impossible to open %. Are you in the right directory ? Don't forget to read the FAQ !\n", vertex_file_path);
32         return 0;
33     }
34
35     // Read the Fragment Shader code from the file
36     std::string FragmentShaderCode;
37     std::ifstream FragmentShaderStream(fragment_file_path , std::ios::in);
38     if(FragmentShaderStream.is.open()){
39         std::string Line = "";
40         while(getline(FragmentShaderStream , Line))
41             FragmentShaderCode += "\n" + Line;
42         FragmentShaderStream.close();
43     }
44
45     GLint Result = GL_FALSE;
46     int InfoLogLength;
47
48     // Compile Vertex Shader
49     printf("Compiling shader : %s\n", vertex_file_path);
50     char const * VertexSourcePointer = VertexShaderCode.c_str();
51     glShaderSource(VertexShaderID , 1 , &VertexSourcePointer , NULL);
52     glCompileShader(VertexShaderID);
53
54     // Check Vertex Shader
55     glGetShaderiv(VertexShaderID , GL_COMPILE_STATUS , &Result);
56     glGetShaderiv(VertexShaderID , GL_INFO_LOG_LENGTH , &InfoLogLength);
57     if ( InfoLogLength > 0 ){
58         std::vector<char> VertexShaderErrorMessage(InfoLogLength+1);
59         glGetShaderInfoLog(VertexShaderID , InfoLogLength , NULL , &VertexShaderErrorMessage[0]);
60         printf("%s\n" , &VertexShaderErrorMessage[0]);
61     }
62
63     // Compile Fragment Shader
64     printf("Compiling shader : %s\n" , fragment_file_path);
65     char const * FragmentSourcePointer = FragmentShaderCode.c_str();
66     glShaderSource(FragmentShaderID , 1 , &FragmentSourcePointer , NULL);
67     glCompileShader(FragmentShaderID);
68
69     // Check Fragment Shader
70     glGetShaderiv(FragmentShaderID , GL_COMPILE_STATUS , &Result);
71     glGetShaderiv(FragmentShaderID , GL_INFO_LOG_LENGTH , &InfoLogLength);
72     if ( InfoLogLength > 0 ){
73         std::vector<char> FragmentShaderErrorMessage(InfoLogLength+1);
74         glGetShaderInfoLog(FragmentShaderID , InfoLogLength , NULL , &FragmentShaderErrorMessage[0]);
75     }
76 }
```

```

75         printf("%s\n", &FragmentShaderErrorMessage[0]);
76     }
77
78     // Link the program
79     printf("Linking program\n");
80     GLuint ProgramID = glCreateProgram();
81     glAttachShader(ProgramID, VertexShaderID);
82     glAttachShader(ProgramID, FragmentShaderID);
83     glLinkProgram(ProgramID);
84
85     // Check the program
86     glGetProgramiv(ProgramID, GLLINK_STATUS, &Result);
87     glGetProgramiv(ProgramID, GLINFO_LOG_LENGTH, &InfoLogLength);
88     if (InfoLogLength > 0) {
89         std::vector<char> ProgramErrorMessage(InfoLogLength + 1);
90         glGetProgramInfoLog(ProgramID, InfoLogLength, NULL, &ProgramErrorMessage[0]);
91         printf("%s\n", &ProgramErrorMessage[0]);
92     }
93
94     glDeleteShader(VertexShaderID);
95     glDeleteShader(FragmentShaderID);
96
97     return ProgramID;
98 }
```

Listing 20: shader (./code/shader.cpp)

```

1 #pragma once
2 GLuint LoadShaders(const char * vertex_file_path ,const char * fragment_file_path);
```

Listing 21: shader (./code/shader.h)

```

1 #version 330 core
2
3 // Input vertex data, different for all executions of this shader.
4 in vec3 vert.pos.model;
5 in vec3 vert.nor.model;
6 in vec2 vert.tex;
7
8 // Output data ; will be interpolated for each fragment.
9 out vec2 UV;
10 out vec3 vert.pos.world;
11 out vec3 vert.nor.cam;
12 out vec3 eye.dir.cam;
13 out vec3 light.dir.cam;
14
15 // Values that stay constant for the whole mesh.
16 uniform mat4 M;
17 uniform mat4 V;
18 uniform mat4 P;
19 uniform vec3 light.pos.world;
20
21 void main() {
22
23     // Output position of the vertex, in clip space : MVP * position
24     gl_Position = P * V * M * vec4(vert.pos.model,1);
25
26     // Position of the vertex, in worldspace : M * position
27     vert.pos.world = (M * vec4(vert.pos.model,1)).xyz;
28
29     // Vector that goes from the vertex to the camera, in camera space.
30     // In camera space, the camera is at the origin (0,0,0).
31     vec3 vert.pos.cam = (V * M * vec4(vert.pos.model,1)).xyz;
32     eye.dir.cam = vec3(0,0,0) - vert.pos.cam;
33
34     // Vector that goes from the vertex to the light, in camera space. M is omitted because it's identity.
35     vec3 light.pos.cam = (V * vec4(light.pos.world,1)).xyz;
36     light.dir.cam = light.pos.cam + eye.dir.cam;
37
38     // Normal of the the vertex, in camera space
39     vert.nor.cam = (V * M * vec4(vert.nor.model,0)).xyz; // Only correct if ModelMatrix does not scale the model ! Use its inverse transpose if not.
40
41     // UV of the vertex. No special space for this one.
42     UV = vert.tex;
43 }
```

Listing 22: head vertex shader (./code/head_vert.glsl)

```

1 #version 330 core
2
3 // Interpolated values from the fragex shaders
4 in vec2 UV;
5 in vec3 vert.pos.world;
6 in vec3 vert.nor.cam;
7 in vec3 eye.dir.cam;
8 in vec3 light.dir.cam;
9
10 // Ouput data
11 out vec3 color;
12
13 // Values that stay constant for the whole mesh.
14 uniform sampler2D tex;
15 uniform mat4 MV;
16 uniform vec3 light.pos.world;
17
18 void main() {
19
```

```

20      // Light emission properties
21      vec3 light_color = vec3(1,1,1);
22      float light_power = 1500.0f;
23
24      // Material properties
25      vec3 diffuse_color = texture2D(tex, UV).rgb;
26      vec3 ambient_color = vec3(0.1,0.1,0.1) * diffuse_color;
27      vec3 specular_color = vec3(0.3,0.3,0.3);
28
29      // Distance to the light
30      float distance = length(light_pos_world - vert_pos_world );
31
32      vec3 n = normalize(vert_nor_cam);
33      vec3 l = normalize(light_dir_cam);
34      float cosTheta = clamp(dot(n,l), 0, 1);
35
36      vec3 E = normalize(eye_dir_cam);
37      vec3 R = reflect(-l,n);
38      float cosAlpha = clamp(dot(E,R), 0, 1);
39
40      color = ambient_color +
41          diffuse_color * light_color * light_power * cosTheta / (distance*distance) +
42          specular_color * light_color * light_power * pow(cosAlpha,.5) / (distance*distance);
43
44  }

```

Listing 23: head fragment shader ([..../code/head_frag.gls1](#))

```

1 #version 150
3
4 in vec3 position;
5 in vec3 color;
6 uniform mat4 mvp;
7
8 out vec3 Color;
9
10 void main()
11 {
12     Color = color;
13     gl_Position = mvp * vec4(position, 1);
14 }

```

Listing 24: grid vertex shader ([..../code/grid_vert.gls1](#))

```

1 #version 150
3
4 in vec3 Color;
5 out vec4 out_color;
6
7 void main()
8 {
9     out_color = vec4(Color, 1.0);
}

```

Listing 25: grid fragment shader ([..../code/grid_frag.gls1](#))