Assignment 2: Geometric Modeling

NAME: LUOCHENQI
STUDENT NUMBER: 14784547
EMAIL: LUOOCHQ@SHANGHAITECH.EDU.CN

ACM Reference format:
Name: luochenqi
student number: 14784547
email: luoochq@shanghaitech.edu.cn. YYYY. Assignment 2: Geometric Modeling. VV, NNN, Article AA (MM YYYY), 2 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the last assignment, you have been familiar with the basic graphics programs with OpenGL, where you should have been able to render simple geometric objects. In this assignment, we go further based on assignment 1 that we construct more complex geometric objects, basically with Bézier surfaces. The rendering of the geometries is the same as in assignment 1 where you can choose to either shade the objects with OpenGL fixed pipeline (Gourand shading) or implement per-pixel Phong shading in shaders as the optional work in assignment 1 to give better visual results. What you really need to do in this assignment is how the complex geometries could be constructed by yourself and how to texture them. You will be required to implement the Bézier surface construction method with a particular triangulation algorithm to draw complex shapes. In addition, you need to texture the Bézier surface using OpenGL texture mapping techniques. This report summarize the method to create the geometric modeling.

2 IMPLEMENTATION DETAILS

This part will show how to create my program step by step.

2.1 Bézier curve construction

(1) drawBezierSurface
(2) we need to calculate bezier surface
(3) $B(t) = \sum_{i=0}^{n} \beta_i b_{i,n}(t)$

```
GLFWwindow* window = glfwCreateWindow(800,
  600, "myopengl", NULL, NULL);
```

(4) prepare my render engine.It is called render loop.it will be always running until the end of glfw.Function glfwPollEvents will check if there is some triggering events to update window state and call the corresponding callback function.

```
while (!glfwWindowShouldClose(window))
{
    glfwSwapBuffers(window);
    glfwPollEvents();
}
```

2.2 Create a simple shader

Opengl have vertex shader and fragment shader ,vertex shader is responsible for the vertex input,and fragment shader is responsible for the color of the pixel.

Vertex Shader

A vertex coordinate is transformed by following matrix,

$$V_{clip} = M_{projection} * M_{view} * M_{model} * M_{local}$$

At the beginning ,we define the coordinate which should be the local coordinates.And we multiply a model matrix to transform it into the world space.Then we multiply a view matrix to camera system.And we mutiply a projection matrix to adjust to -1.0 to 1.0.
So our vertex shader is constructed by three variable of GLSL language,gl_Position is most important to decide the location of the vertices.

```
FragPos = vec3(model * vec4(aPos, 1.0));
Normal = mat3(transpose(inverse(model))) * aNormal;
gl_Position = projection * view * vec4(FragPos, 1.0);
```

Fragment Shader

The easiest way to show the color is below.
FragColor = vec4(1.0f,0.8f,1.0f,1.0f);

2.3 Create a cube

In this section,I try to create a 3D cube in OpenGL.As we all know,a cube is make up of 6 squares.So we only need to create 6 squares.Also we know triangles is most efficient in OpenGL.So we try to construct the square with two triangle.Then we put the vertex into the float vertices[].And I bind it with a VAO and VBO.
Finally,in while (!glfwWindowShouldClose(window)) loop,we do the command

```
glBindVertexArray(cubeVAO);
glDrawArrays(GL_TRIANGLES, 0, 36);
```

And the cube is appeared.

2.4 Create a tetrahedron

Tetrahedron is the same as cube,we only need to give vertices coordinate of four triangular plane to vertex shader and it will put out the tetrahedron.

DOI: 10.1145/nnnnnnn.nnnnnnn

2.5 Create a sphere

When it comes to create a sphere,we can also use the same method as before.We create the vertices and give it to the vertex shader and done.
we try to use the parameter equation to describe a sphere.
$x = x_0 + rsin\theta cos\phi$
$y = y_0 + rsin\theta sin\phi$
$z = z_0 + rcos\theta$
And we set the center to be $(0, 0, 0)$ and radius to be 1 for simplicity.

2.6 Camera

Now we construct the camera system.we define the camera class and set up a few camera attributes ,Position,Front,Up and Right vectors.And I need to set up our Lookat function.I initialize it to be

```
view = glm::lookAt(Position, Position +
Front, Up);
```

Now we set up the Front and Right and Up in function UpdateCameraVectors

```
glm::vec3 front;
front.x = cos(glm::radians(Yaw)) * cos(glm::
radians(Pitch));
front.y = sin(glm::radians(Pitch));
front.z = sin(glm::radians(Yaw)) * cos(glm::
radians(Pitch));
Front=glm::normalize(front);
Right=glm::normalize(glm::cross(Front,WorldUp));
Up= glm::normalize(glm::cross(Right,Front));
```

Now to control the keyboard input,once we press the WS,I hope to move forward and backward,and once we press AD,I hope it rotate around the object.
WS is easier,we just add or minusa vector of velocity multiply Front and it is done. While AD is more difficult,because we need to lookat the object and rotate the camera,so we need to let Position vector add or minus the mutilply of cross of Front and up vector and velocity.And we need let Front equals minus Position.
The following code is left rotation around the objects.

```
Position -= glm::cross(Front,Up)*velocity;
Front = -Position ;
```

2.7 Phong shading

The natural light is consisted with Ambient lighting,Diffuse lighting and specular light.
Ambient Light
The ambient light is the most simple.

```
vec3 result = ambient * objectColor;
```

Diffuse Light
Now we need to calculate the diffuse lighting,we need normal vector of the vertex,and we need to calculate the differential vector between fragment vertex and light position.Then we dot the distance differential vector and normal vector to get the diffuse lighting.Finally we multiply diffuse lighting to light color.

```
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(lightPos - FragPos);
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * lightColor;
```

Specular Light
When we calculate specular light ,we prefer the camera angle of view.The same as diffuse light,Firstly,we calculate the differential vector.Secondly we dot this vector to reflector vector and we get its 32power to get its shiniess.

```
float specularStrength = 0.5;
vec3 viewDir = normalize(viewPos - FragPos);
vec3 reflectDir = reflect(-lightDir, norm);
float spec = pow(max(dot(viewDir, reflectDir),
 0.0), 32);
vec3 specular = specularStrength * spec * lightColor;
```

3 RESULT