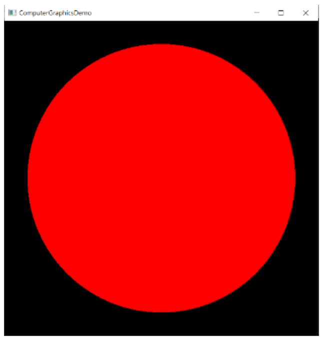
HW2 Guideline

Task 1 :

Follow the steps to draw a sphere by GLSL.



Step 1 : Initialization and callback function setting. (in main function)

1. Initialize glut
2. Initialize glew
3. Initialize shader (You need to implement)
4. Callback functions (display, reshape, keyboard, idle….)
5. Call glutMainLoop() to enter the GLUT event processing loop.

```
int main(int argc, char** argv)
{
    //1.Initialize glut
    glutInit(&argc, argv);
    glutInitWindowSize(windowSize[0], windowSize[1]);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutCreateWindow("ComputerGraphicsDemo");
    //2.Initialize glew
    glewInit();
    //3.Initialize shader (You need to implement)
    shaderInit();
    //4.Callback functions
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutIdleFunc(idle);
    //5.Enter the GLUT event processing loop.
    glutMainLoop();

    return 0;
}
```

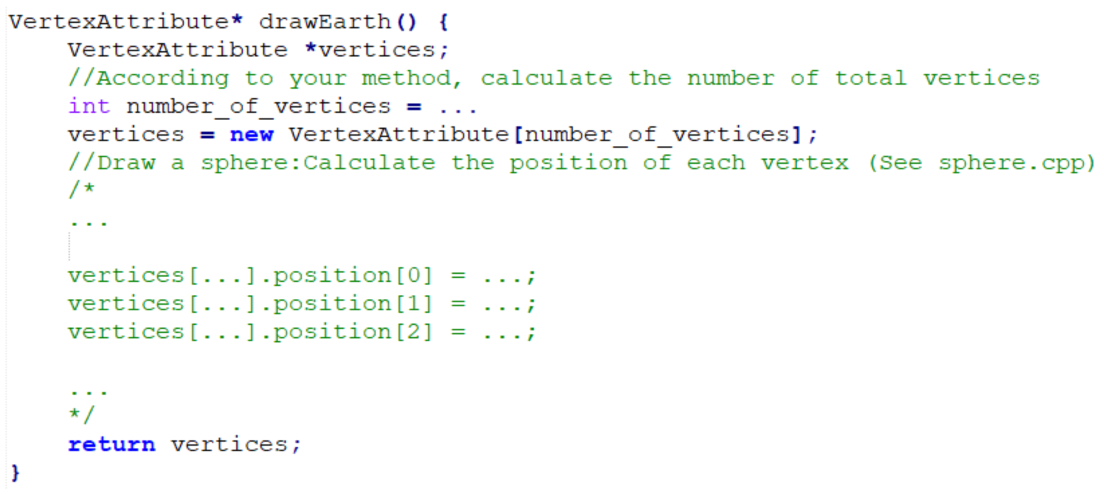
Step 2 : Shader initialization (in shaderInit() function)

1. Call createShader() function twice to create two shaders : vertex shader and fragment shader. (createShader() function is in shader.h)
2. Call createProgram() to create a program (createProgram() function is in shader.h)
3. Generate one vertex buffer object (VBO) and bind it
4. Get vertices from drawEarth() function in VertexAttribute type. VertexAttribute can be defined as

   ```
   struct VertexAttribute
   {
       GLfloat position[3];
   };
   ```
5. Call glBufferData() to copy vertex data to the buffer object
6. Enable the VertexAttributeArray and call glVertexAttribPointer() to link the vertex buffer with the vertex shader input.
7. Remember to unbind the vertex buffer.

```
void shaderInit() {
    //1.Call createShader() function twice to create two shaders :
    //  vertex shader and fragment shader.
    //( createShader() function is in shader.h )
    GLuint vert = createShader("Shaders/example.vert", "vertex");
    GLuint frag = createShader(...);
    //2.Call createProgram() to create a program
    //( createProgram() function is in shader.h )
    program = createProgram(...);
    //3.Generate one vertex buffer object (VBO) and bind it
    glGenBuffers(...);
    glBindBuffer(...);
    //4.Get vertices from drawEarth() in VertexAttribute type.
    VertexAttribute *vertices;
    vertices = drawEarth(); // You need to implement drawEarth()
    //5.Call glBufferData() to copy vertex data to the buffer object
    glBufferData(...);
    //6.Enable the VertexAttributeArray and call glVertexAttribPointer()
    // to link the vertex buffer with the vertex shader input.
    glEnableVertexAttribArray(...);
    glVertexAttribPointer(...);
    //7.Remember to unbind the vertex buffer.
    glBindBuffer(...);
}
```

Step 3 : Implement the drawEarth() function to set up the vertices of the sphere. (see the sphere.cpp for reference)



```
VertexAttribute* drawEarth() {
    VertexAttribute *vertices;
    //According to your method, calculate the number of total vertices
    int number_of_vertices = ...
    vertices = new VertexAttribute[number_of_vertices];
    //Draw a sphere:Calculate the position of each vertex (See sphere.cpp)
    /*
    ...

    vertices[...].position[0] = ...;
    vertices[...].position[1] = ...;
    vertices[...].position[2] = ...;

    ...
    */
    return vertices;
}
```

Step 4 : Implement the display() function.

1. Clear the color buffer and the depth buffer
2. Modeling and viewing transformation
3. Projection transformation
4. Viewport transformation
5. Let the sphere rotate
6. Get the Projection and ModelView matrix and store in pmtx and mmtx.
7. Get the location of uniform variable(Projection, ModelView)
8. Install the program object as part of current rendering state
9. Pass the projection (and modelview) matrix into vertex shader through uniform variable
10. Render the primitives in vertex array
11. Set the current rendering state to NULL

```
void display()
{
    //1.Clear the color buffer and the depth buffer
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glClearDepth(1.0f);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //2.Modeling and viewing transformation
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0.0f, 0.0f, 3.0f,// eye
        0.0f, 0.0f, 0.0f,      // center
        0.0f, 1.0f, 0.0f);     // up
    //3.Projection transformation
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45, (GLfloat)512 / (GLfloat)512, 1, 500);
    //4.Viewport transformation
    glViewport(0, 0, windowSize[0], windowSize[1]);

    //5.Let the sphere rotate
    glMatrixMode(GL_MODELVIEW);
    glRotatef(...);

    //6.Get the Projection and ModelView matrix and store in pmtx and mmtx.
    GLfloat pmtx[16];
    GLfloat mmtx[16];
    glGetFloatv(GL_PROJECTION_MATRIX, pmtx);
    glGetFloatv(GL_MODELVIEW_MATRIX, mmtx);

    //7.Get the location of uniform variable(Projection,ModelView) in vertex shader
    GLint pmatLoc = glGetUniformLocation(program, "Projection");
    GLint mmatLoc = glGetUniformLocation(...);
    //8.Install the program object as part of current rendering state
    glUseProgram(program);
    //9.Pass the projection (and modelview) matrix
    //  into vertex shader through uniform variable
    glUniformMatrix4fv(pmatLoc, 1, GL_FALSE, pmtx);
    glUniformMatrix4fv(...);
    //10.Render the primitives in vertex array
    glDrawArrays(...);

    //11.Set the current rendering state to NULL
    glUseProgram(0);
    glPopMatrix();
    glutSwapBuffers();
}
```

Step 5 : Implement the vertex shader.

1. Get the position of each vertex by location
2. Declare uniform variables of projection matrix and modelview matrix, and their value were set and changed in main.cpp
3. Each vertex needs to do modelview and projection transform here

```
#version 430
//1.Get the position of each vertex by location
layout(location = 0) in vec3 position;
//2.Declare uniform variables of projection matrix and modelview matrix,
//  and their value were set and changed in main.cpp
uniform mat4 Projection;
uniform mat4 ModelView;

void main() {
  //3.Each vertex needs to do modelview and projection transform here
  gl_Position = ...
}
```

Step 6 : Implement the fragment shader.

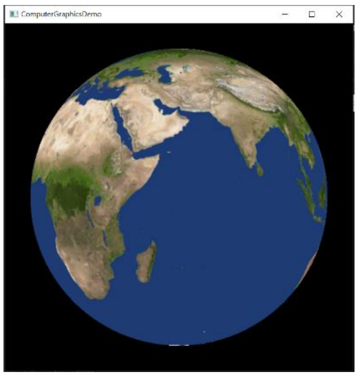
1. Set up the output variable
2. Set up the output color of each pixel

```
#version 430
//1.Set up the output variable
out vec4 frag_color;

void main() {
    //2.Set up the output color of each pixel
    frag_color = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Task 2 :

Modify some codes to add texture on the sphere.



Step 1 : Add textureInit() for texture initialization in main() function

```
//3.Initialize shader (You need to implement)
shaderInit();
//Task2: Add textureInit() for texture initialization
textureInit();
//4.Callback functions
```

Step 2 : Texture initialization (in textureInit() function)

1. Enable texture
2. Load the texture
3. Generate texture
4. Set the texture warping parameters
5. Specifies a texture environment
6. Generate a 2D texture image
7. Set the texture filtering parameters

```
void textureInit()
{
    //1. Enable texture
    glEnable(GL_TEXTURE_2D);
    //2. Load the texture
    FIBITMAP* pImage = FreeImage_Load(...);
    FIBITMAP* p32BitsImage = FreeImage_ConvertTo32Bits(pImage);
    int iWidth = FreeImage_GetWidth(pImage);
    int iHeight = FreeImage_GetHeight(pImage);
    //3. Generate texture
    glGenTextures(...);
    glBindTexture(...);
    //4. Set the texture warping parameters
    glTexParameteri(...);
    glTexParameteri(...);
    //5. Specifies a texture environment.
    glTexEnvf(...);
    //6. Generate a two-dimensional texture image
    glTexImage2D(...);
    //7. Set the texture filtering parameters
    glTexParameteri(...);
    glTexParameteri(...);

    glBindTexture(GL_TEXTURE_2D, 0);
}
```

Step 3 : Modify the VertexAttribute type to store texture coordinate

```
struct VertexAttribute
{
    GLfloat position[3];
    //store texture coordinate
    GLfloat texcoord[2];
};
```

Step 4 : Calculate texture coordinate of each vertex and store them in drawEarth() function

```
/*
...

vertices[...].position[0] = ...;
vertices[...].position[1] = ...;
vertices[...].position[2] = ...;
vertices[...].texcoord[0] = ...;
vertices[...].texcoord[1] = ...;

...
*/
```

Step 5 : Add another VertexAttribPointer to point to the vertex coordinate (in shaderInit() function)

```
//6.Enable the VertexAttributeArray and call glVertexAttribPointer()
// to link the vertex buffer with the vertex shader input.
glEnableVertexAttribArray(...);
glVertexAttribPointer(...);

//Task2:Add another VertexAttribPointer to point to the vertex coordinate
glEnableVertexAttribArray(...);
glVertexAttribPointer(...);
```

Step 6 : Bind and active the texture (in display() function)

1. Get the location of uniform variable in fragment shader
2. Active, bind and set up the value of uniform variable in fragment shader
3. Remember to unbind the texture

```
//7.Get the location of uniform variable(Projection,ModelView) in vertex shader
GLint pmatLoc = glGetUniformLocation(program, "Projection");
GLint mmatLoc = glGetUniformLocation(...);
//Task2:Get the location of uniform variable in fragment shader
GLint texLoc = glGetUniformLocation(program, "Texture");

//8.Install the program object as part of current rendering state
glUseProgram(program);

//9.Pass the projection (and modelview) matrix
//  into vertex shader through uniform variable
glUniformMatrix4fv(pmatLoc, 1, GL_FALSE, pmtx);
glUniformMatrix4fv(...);

//Task2:Active, bind and set up the value of uniform variable in fragment shader
glActiveTexture(...);
glBindTexture(...);
glUniform1i(...);

//10.Render the primitives in vertex array
glDrawArrays(...);

//Task2:Remember to unbind the texture
glBindTexture(GL_TEXTURE_2D, NULL);
```

Step 7 : Modify the vertex shader

1. Get the texture coordinate of each vertex by location
2. Set up the output variable to pass the texture coordinate to the fragment shader. The name of the variable should be the same as the input of the fragment shader
3. Pass the texture coordinate

```
#version 430
//1.Get the position of each vertex by location
layout(location = 0) in vec3 position;

//Task2:Get the texture coordinate of each vertex by location
layout(location = 1) in vec2 texcoord;
//Task2:Set up the output variable to pass the texture coordinate to the fragment shader
//      The name of the variable should be the same as the input of the fragment shader
out vec2 frag_UV;

//2.Declare uniform variables of projection matrix and modelview matrix,
//  and their value were set and changed in main.cpp
uniform mat4 Projection;
uniform mat4 ModelView;

void main() {
  //Task2:Pass the texture coordinate
  frag_UV = ...
  //3.Each vertex needs to do modelview and projection transform here
  gl_Position = ...
}
```

Step 8 : Modify the fragment shader

1. Declare the uniform variable with type sampler2D. Its value was set in main.cpp
2. Set up the input variable to get the texture coordinate passed from vertex shader
3. Modify the output “frag_color” with the texture

```
#version 430
//1.Set up the output variable
out vec4 frag_color;

//Task2:Declare the uniform variable with type sampler2D
//      Its value was set in main.cpp
uniform sampler2D Texture;
//Task2:Set up the input variable to get the texture coordinate passed from vertex shader
in vec2 frag_UV;

void main() {
    //Task2:Modify the output "frag_color" with the texture
    frag_color = ...
}
```