Name: Ralenski Doucet
Contents: Rendering Geometry

- **Render Plane see Image I:**

To be able to generate a plane you need a function that takes in one argument that is of type int.
The int's name should be size. When defining the function you need to make 4 vertices just as follows. **1st** Vertex A = Vertex(glm::vec4(-size, size, 0, 1), glm::vec4(1, 0, 0, 1), glm::vec2(0, 0));
2nd Vertex B = Vertex(glm::vec4(size, size, 0, 1), glm::vec4(1, 0, 0, 1), glm::vec2(1, 0));
3rd Vertex C = Vertex(glm::vec4(size, -size, 0, 1), glm::vec4(1, 0, 0, 1), glm::vec2(1, 1));
4th Vertex D = Vertex(glm::vec4(-size, -size, 0, 1), glm::vec4(1, 0, 0, 1), glm::vec2(0, 1));
Then you create a new vector of type vertex and assign the new vector the value of A,B,C,D.
Then you want to return the vector you just made.

Image I:

```
std::vector<Vertex> RenderingGeometryApp::genPlane(int size)
{

    Vertex A = Vertex(glm::vec4(-size, size, 0, 1), glm::vec4(1, 0, 0, 1), glm::vec2(0, 0));
    Vertex B = Vertex(glm::vec4(size, size, 0, 1), glm::vec4(1, 0, 0, 1), glm::vec2(1, 0));
    Vertex C = Vertex(glm::vec4(size, -size, 0, 1), glm::vec4(1, 0, 0, 1), glm::vec2(1, 1));
    Vertex D = Vertex(glm::vec4(-size, -size, 0, 1), glm::vec4(1, 0, 0, 1), glm::vec2(0, 1));
    std::vector<Vertex> PlaneVertices = { A,B,C,D };
    return PlaneVertices;
}
```

- **Render Cube: see image II:**

To be able to render a cube you need a make a function of type std::vector<Vertex>. That takes in one argument of type std::vector<Vertex> named vertices. Then you assign and push back the vertices just like as follows.
vertices.push_back(Vertex(glm::vec4(0, 1, 1, 1), glm::vec4(1), glm::vec2(0)));
vertices.push_back(Vertex(glm::vec4(1, 1, 1, 1), glm::vec4(1), glm::vec2(0)));
vertices.push_back(Vertex(glm::vec4(1, 0, 1, 1), glm::vec4(1), glm::vec2(0)));
vertices.push_back(Vertex(glm::vec4(0, 0, 1, 1), glm::vec4(1), glm::vec2(0)))
vertices.push_back(Vertex(glm::vec4(0, 0, 0, 1), glm::vec4(1), glm::vec2(0)));
vertices.push_back(Vertex(glm::vec4(1, 0, 0, 1), glm::vec4(1), glm::vec2(0)));
vertices.push_back(Vertex(glm::vec4(1, 1, 0, 1), glm::vec4(1), glm::vec2(0)));
vertices.push_back(Vertex(glm::vec4(0, 1, 0, 1), glm::vec4(1), glm::vec2(0)));
vertices.push_back(Vertex(glm::vec4(0, 1, 1, 1), glm::vec4(1), glm::vec2(0)));
vertices.push_back(Vertex(glm::vec4(1, 1, 1, 1), glm::vec4(1), glm::vec2(0)));
vertices.push_back(Vertex(glm::vec4(1, 1, 0, 1), glm::vec4(1), glm::vec2(0)));
vertices.push_back(Vertex(glm::vec4(1, 0, 0, 1), glm::vec4(1), glm::vec2(0)));

Name: Ralenski Doucet
Contents: Rendering Geometry

vertices.push_back(Vertex(glm::vec4(0, 1, 0, 1), glm::vec4(1), glm::vec2(0)));
vertices.push_back(Vertex(glm::vec4(0, 0, 0, 1), glm::vec4(1), glm::vec2(0)));
Then you want to return the vertices.
Image II:

```
std::vector<Vertex> RenderingGeometryApp::genCube(std::vector<Vertex> vertices)
{
    //Front
    vertices.push_back(Vertex(glm::vec4(0, 1, 1, 1), glm::vec4(1), glm::vec2(0)));
    vertices.push_back(Vertex(glm::vec4(1, 1, 1, 1), glm::vec4(1), glm::vec2(0)));
    vertices.push_back(Vertex(glm::vec4(1, 0, 1, 1), glm::vec4(1), glm::vec2(0)));
    vertices.push_back(Vertex(glm::vec4(0, 0, 1, 1), glm::vec4(1), glm::vec2(0)));
   //Bottom
    vertices.push_back(Vertex(glm::vec4(0, 0, 0, 1), glm::vec4(1), glm::vec2(0)));
    vertices.push_back(Vertex(glm::vec4(1, 0, 0, 1), glm::vec4(1), glm::vec2(0)));
    //Back
    vertices.push_back(Vertex(glm::vec4(1, 1, 0, 1), glm::vec4(1), glm::vec2(0)));
    vertices.push_back(Vertex(glm::vec4(0, 1, 0, 1), glm::vec4(1), glm::vec2(0)));
    //Top
    vertices.push_back(Vertex(glm::vec4(0, 1, 1, 1), glm::vec4(1), glm::vec2(0)));
    vertices.push_back(Vertex(glm::vec4(1, 1, 1, 1), glm::vec4(1), glm::vec2(0)));
     //Right
    vertices.push_back(Vertex(glm::vec4(1, 1, 0, 1), glm::vec4(1), glm::vec2(0)));
    vertices.push_back(Vertex(glm::vec4(1, 0, 0, 1), glm::vec4(1), glm::vec2(0)));
     //Left
    vertices.push_back(Vertex(glm::vec4(0, 1, 0, 1), glm::vec4(1), glm::vec2(0)));
    vertices.push_back(Vertex(glm::vec4(0, 0, 0, 1), glm::vec4(1), glm::vec2(0)));
    return vertices;
}
```

- **Render A sphere see Images III,IV,V:**

To be able to render a sphere first you need a the ability to make a half circle.
To make a half circle you need to do as follows.
1st:declare an std::vector of type <glm::vec4> named CircleVerts.
2nd:you declare a for loop as follows (float i=0;i<np;i++).
3rd: inside of the for loop you want to declare 2 floats just like as follows **float 1** float angle = glm::pi<float>() / ((float) np - 1); **Float 2** float theta = i * angle;
4th:you push back each vertices in CircleVerts by do as follows:
CircleVerts.push_back(glm::vec4(glm::cos(theta)*radius, glm::sin(theta)*radius,0,1))
5th: you then return CircleVerts;
Image III:

Name: Ralenski Doucet
Contents: Rendering Geometry

```
std::vector<glm::vec4> RenderingGeometryApp::genHalfCircle(int np, double radius)
{
    //1st two arguments int np(Number of Points); double radius;
    //2nd declare number of points;
    //3rd declare local varaible that will represent an vertex's position.
    std::vector<glm::vec4>CircleVerts;

    for (float i=0;i<np;i++)
    {
        //calculate (angle or theta) in for loop.
        //angle is equals the answer of (3.14/number of points)
        float angle = glm::pi<float>() / ((float) np - 1);
        float theta = i * angle;

        //push back each vertice in the vertex _points->
        //that shows each generated portion of the half circle
        CircleVerts.push_back(glm::vec4(glm::cos(theta)*radius, glm::sin(theta)*radius, 0, 1));

    }
    return CircleVerts;

}
```

See Image IV
To be able to render a sphere you also need to generate the spheres indices by do as follows. To generate sphere indices you need a function that returns a std vector of type Vertex that has two arguments for the number of points and the number of meridians.
When defining the function you want to do as follows.
1st: you create a new std vector of type unsigned int named sphere indices.
2nd:declare 3 new unsigned ints start,bottom left,bottom right.
3rd: declare this for loop : for(int r = 0;r < number of meridians;r++)
4th: inside of the for loop you want to assign start to be equal to r * np.
5th: Also inside of the for loop you declare a nested for loop.
6th: declare the for loop as follows: for (int p =0; p < np; p++)
7th: inside of the nested for loop you want to assign bottom left to be equal to start + p. Then bottom right to be equal to bottom left +np.
8th: then you want to pushback sphere indices bottom left and bottom right
9th: then you return sphere indices.
Image IV:

Name: Ralenski Doucet
Contents: Rendering Geometry

```
std::vector<unsigned int> RenderingGeometryApp::genSphereIndices(int np, int numofM)
{
    std::vector<unsigned int> Sphereindices;
    unsigned int start;
    unsigned int bottom_left;
    unsigned int bottom_right;
    for (int r = 0; r < numofM; r++)
    {
        start = r * np;
        for (int p = 0; p < np; p++)
        {
            bottom_left = start + p;
            bottom_right = bottom_left + np;
            Sphereindices.push_back(bottom_left);
            Sphereindices.push_back(bottom_right);
        }
        Sphereindices.push_back(0xFFFF);
    }
    return Sphereindices;

}
```

See Image V:
Finally to render a sphere do as follows.
You need to make a function of type std vector the vector is or type glm vec4
The function takes in two arguments a std vector of type glm vec4 and an unsigned int for number of meridians(numofM). When defining the function you create a new std vector of type glm vec4 named sphere points. Then you declare a for loop just like as follows: for (int i = 0; i < numofM; i++). Then inside of the for loop you want to declare two new floats .**Float 1** spheresplice and assign it the value of glm pi * 2 divided by numofM. **Float 2** theta equals the value of i * spheresplice. Then create a nested for loop just like as follows: int j = 0 ; j < points.size();j++). Inside of the nested for loop declare 3 new floats: **float 1** x is equal to points[j].x. **Float 2** y is equal to points[j].y * cos(theta) + points[j].z * sin(theta);
Float z = points[j].z * cos(theta) + points[j].y * sin(theta). Then you push back sphere points . And the you return sphere points.

Name: Ralenski Doucet
Contents: Rendering Geometry

Image V:

```cpp
std::vector<glm::vec4> RenderingGeometryApp::genSphere(std::vector<glm::vec4>points, unsigned int numofM)
{

    std::vector<glm::vec4> SpherePoints;
    for (int i = 0; i < numofM + 1; i++)
    {
        float sphereSlice = (glm::pi<float>() * 2) / (float)numofM;
        float theta = i * sphereSlice;
        for (int j = 0; j < points.size(); j++)
        {
            float X = points[j].x;
            float Y = points[j].y * cos(theta) + points[j].z * -sin(theta);
            float Z = points[j].z * cos(theta) + points[j].y * sin(theta);
            glm::vec4 point = glm::vec4(X, Y, Z, 1);
            SpherePoints.push_back(point);
        }
    }
    return SpherePoints;

}
```

- **Shaders Load from a separate file: See Image VI:**

To be able to load shaders from a separate file first you need a seperate file with the shader information to load from. So create a new text file with the shader information inside of it. You also need to create a load method.The following is how I created my load function.
1st: I Created a new variable of type File named file and one more new variable of type errno_t named err.
2nd: The value of err was assigned to fopen_s(&file, Filename,”r”).
3rd: I created a new char named mstring that is an array the size 500.
4th: I declare a while loop with the conditions as follows (std::fgets(mstring, sizeof mstring,file))
5th: inside of the body of the while loop declare a if statement just as follows (shadertype==Shader::Shader_type::VERTEX).
6th: Inside of the body of the if statement do as follows: vsSourceString.append(mstring).
7th: Declare a else if statement inside of the while loop just as follows**:**(shadertype== Shader::ShaderType::FRAGMENT).
8th: Inside of the body for the else if statement do as follows:fsSourceString.append(mstring)
9th: close of your while loop and do as follows: vsSource = vsSourceString.c_str(); fsSource = fsSourceString.c_str();
10th: And last step is to return true .

Image VI:

Name: Ralenski Doucet
Contents: Rendering Geometry

```
bool Shader::load(const char *Filename, Shader::SHADER_TYPE shadertype)
{
    errno_t err;
    FILE *file;
    err = fopen_s(&file, Filename, "r");
    char mstring[500];
    while(std::fgets(mstring, sizeof mstring, file))
    {
        if (shadertype == Shader::SHADER_TYPE::VERTEX)
        {
            vsSourceString.append(mstring);
        }
        else if (shadertype == Shader::SHADER_TYPE::FRAGMENT)
        {
            fsSourceString.append(mstring);
        }

    }
    vsSource = vsSourceString.c_str();
    fsSource = fsSourceString.c_str();
    return true;
}
```