练 习 题 报 告

课程名称 计算机图形学

项目名称 简单可扩展曲面纹理映射

学　　院 计算机与软件学院

专　　业 软件工程（腾班）

指导教师 熊卫丹

报 告 人 洪子敬 学号 2022155033

一、 练习目的

1. 了解三维曲面和纹理映基本知识
2. 了解从图片文件载入纹理数据基本步骤
3. 掌握三维曲面绘制过程中纹理坐标和几何坐标的使用

二．练习完成过程及主要代码说明

实验要求：完成 main.cpp、MeshPainter.cpp 和 TriMesh.cpp 中的 TO-DO 内容，实现曲面绘制和纹理设置。

Task1：在 MeshPainter.cpp 中，将纹理坐标传入着色器，参照所给代码的传入方式补全代码。

解答：将 openGL 中的纹理坐标传入着色器指定变量“vTexture”中并进行绑定，传入对应的数据（注意这里有顶点坐标、法向量和颜色三个数组）；详细代码如下所示：

```
// @TODO: Task1 将纹理坐标传入着色器
object.tLocation = glGetAttribLocation(object.program, "vTexture");
glEnableVertexAttribArray(object.tLocation);
glVertexAttribPointer(object.tLocation, 2,
    GL_FLOAT, GL_FALSE, 0,
    BUFFER_OFFSET((points.size() + normals.size() + colors.size()) * sizeof(glm::vec3)));
```

HZJ

Task2：在 TriMesh.cpp 中，参照圆柱体顶点和纹理坐标的生成，实现圆盘和圆锥的生成，补充完成 generateDisk 函数和 generateCone 函数。

解答：已知纹理空间是个 0-1、0-1 的二维空间，这就需要将顶点坐标和纹理坐标进行映射；

对于圆盘：首先绘制圆盘的形状，对于圆周上的点，接受指定的划分三角形个数和半径，用极坐标的形式计算顶点坐标：

$$x = radius * \cos(theta)$$

$$y = radius * \sin(theta)$$

我们将圆盘定在 x、y 平面，则 z 为 0，此时我们得到所有圆周顶点坐标，最后再加上圆心：

$$glm :: vec3(0.0, 0.0, 0.0)$$

而对于法向量，由于圆盘在 x、y 平面，所以法向量方向我们简单的定为：

$$glm :: vec3(0.0, 0.0, 1.0)$$

则对于颜色的生成，我们用法向量生成颜色即可，同样使用：

$$glm :: vec3(0.0, 0.0, 1.0)$$

接下来就是为每个面片添加纹理坐标，每个面片由圆周上两个点和圆心的连线形成，至于纹

理坐标，我们需要对顶点坐标进行空间映射：**先缩放成单位圆再移动到纹理中心**，操作如下：

$$x = \cos(theta)\ /2.0 + 0.5$$

$$y = \sin(theta)\ /2.0 + 0.5$$

由于纹理坐标是二维坐标系，所以上面计算出的坐标打包成二维数组进行存储，同时将对应的面片坐标与纹理坐标进行绑定即完成了纹理的绘制；详细代码如下所示：

```
void TriMesh::generateDisk(int num_division, float radius)
{
    cleanData();
    // @TODO:  Task2 请在此添加代码生成圆盘
    int num_samples = num_division;
    float step = 2 * M_PI / num_samples; // 每个切片的弧度
    float z = 0;
    //生成下表面顶点，法向量和颜色
    for (int i = 0; i < num_samples; i++)
    {
        float r_r_r = i * step;
        float x = radius * cos(r_r_r);
        float y = radius * sin(r_r_r);

        vertex_positions.push_back(glm::vec3(x, y, z));
        //圆盘法向量为z方向
        vertex_normals.push_back(glm::vec3(0, 0, 1));
        //根据法向量生成颜色
        vertex_colors.push_back(glm::vec3(0, 0, 1));
    }

    vertex_positions.push_back(glm::vec3(0.0, 0.0, 0.0)); // 添加圆心
    vertex_normals.push_back(glm::vec3(0, 0, 1));       //添加圆心法向量
    vertex_colors.push_back(glm::vec3(0, 0, 1));       // 添加圆心颜色（白色）
```

HZJ

```
//生成三角形面片
for (int i = 0; i < num_samples; i++)
{
    faces.push_back(vec3i(i, (i+1)%num_samples, num_samples));

    for (int j = 0; j < 2; j++) {
        float theta = (i + j) * step;
        //纹理映射到单位正方形（纹理空间）
        //纹理空间0-1与0-1的正方形区域
        float x = cos(theta) / 2.0 + 0.5;
        float y = sin(theta) / 2.0 + 0.5;

        //添加纹理坐标
        vertex_textures.push_back(glm::vec2(x, y));
    }
    vertex_textures.push_back(glm::vec2(0.5, 0.5));//中心点纹理

    // 对应的三角面片的纹理坐标的下标
    texture_index.push_back(vec3i(3 * i, 3 * i + 1, 3 * i + 2));
}

// 三角面片的每个顶点的法向量的下标，这里和顶点坐标的下标 faces是一致的，所以我们用faces就行
normal_index = faces;
// 三角面片的每个顶点的颜色的下标
color_index = faces;

storeFacesPoints();
```

HZJ

对于圆锥：同样地我们需要绘制其底面圆周，指定划分面片数量和半径后，其他计算方法均一样；但添加圆锥顶端坐标时注意其坐标是：

$$glm :: vec3(0.0,0.0,height)$$

法向量和颜色生成均为：

$$glm :: vec3(0.0,0.0,1.0)$$

不同的在于纹理坐标的添加，此时面片的坐标是顶端和底面的两个顶点的连线，面片的三角形刚好在纹理空间中，此时顶端的纹理坐标是：

$$glm :: vec2(0.5,1.0)$$

按照面片顺序计算对应的纹理坐标，并进行顶点坐标的绑定即可；详细代码如下所示：

```
void TriMesh::generateCone(int num_division, float radius, float height)
{
    cleanData();
    // @TODO: Task2 请在此添加代码生成圆锥体
    int num_samples = num_division;
    float step = 2 * M_PI / num_samples;
    float z = 0;
    //生成圆锥底部顶点坐标、颜色和法向量
    for (int i = 0; i < num_samples; i++) {
        float angle = i * step;
        float x = radius * cos(angle);
        float y = radius * sin(angle);
        vertex_positions.push_back(glm::vec3(x, y, z));
        vertex_normals.push_back(normalize(glm::vec3(x, y, 0)));
        vertex_colors.push_back(normalize(glm::vec3(x, y, 0)));
    }
    //添加圆锥顶点坐标、法向量和颜色
    vertex_positions.push_back(glm::vec3(0.0, 0.0, height));
    vertex_normals.push_back(glm::vec3(0, 0, 1));
    vertex_colors.push_back(glm::vec3(0, 0, 1));
```

HZJ

```
//生成面片
for (int i = 0; i < num_samples; i++) {
    faces.push_back(vec3i(num_samples, i, (i + 1) % num_samples));
    //添加纹理坐标
    vertex_textures.push_back(glm::vec2(0.5, 1));
    vertex_textures.push_back(glm::vec2(1.0*i/num_samples, 0));
    vertex_textures.push_back(glm::vec2(1.0*(i+1)/num_samples, 0));

    // 对应的三角面片的纹理坐标的下标
    texture_index.push_back(vec3i(3 * i, 3 * i + 1, 3 * i + 2));
}

// 三角面片的每个顶点的法向量的下标，这里和顶点坐标的下标 faces是一致
normal_index = faces;
// 三角面片的每个顶点的颜色的下标
color_index = faces;

storeFacesPoints();
```

HZJ

Task3：在 main.cpp 的 init 函数中，生成圆盘和圆锥并进行贴图和参数设置。

这里我们调用前面编写好的两个函数生成圆盘和圆锥；设置圆盘分为 100 份，半径为 0.1，设置圆锥底面分为 100 份，半径为 0.1，高为 0.3；设置圆盘的初始位置为(0.0,0.0,0.0)，对其所在的 x、y 平面进行放大 2 倍，设置圆锥初始位置为$(0.5,-0.2,0.0)$；最后将纹理图片加到对应的绘画板上即可；详细代码如下所示：

```
TriMesh* cone = new TriMesh();
// @TODO: Task2 生成圆锥并贴图
cone->generateCone(100, 0.1, 0.3);
// 设置物体的旋转位移
cone->setTranslation(glm::vec3(0.5, -0.2, 0.0));
cone->setRotation(glm::vec3(-90.0, 0.0, 0.0));
cone->setScale(glm::vec3(1.5, 1.0, 1.5));
// 设置材质
cone->setAmbient(glm::vec4(0.2, 0.2, 0.2, 1.0));
cone->setDiffuse(glm::vec4(0.7, 0.7, 0.7, 1.0));
cone->setSpecular(glm::vec4(0.2, 0.2, 0.2, 1.0));
cone->setShininess(1.0);
// 加到painter中
painter->addMesh(cone, "mesh_c", "./assets/cone.jpg", vshader, fshader);
meshList.push_back(cone);
```

HZJ

```
TriMesh* disk = new TriMesh();
// @TODO: Task2 生成圆盘并贴图
disk->generateDisk(100, 0.1);
// 设置物体的旋转位移
disk->setTranslation(glm::vec3(0.0, 0.0, 0.0));
disk->setRotation(glm::vec3(0.0, 0.0, 0.0));
disk->setScale(glm::vec3(2.0, 2.0, 1.0));
// 设置材质
disk->setAmbient(glm::vec4(0.2, 0.2, 0.2, 1.0));
disk->setDiffuse(glm::vec4(0.7, 0.7, 0.7, 1.0));
disk->setSpecular(glm::vec4(0.2, 0.2, 0.2, 1.0));
disk->setShininess(1.0);
// 加到painter中
painter->addMesh(disk, "mesh_b", "./assets/disk.jpg", vshader, fshader);
meshList.push_back(disk);
```

HZJ

结果展示：运行上述代码，部分效果如下所示：