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 计算机科学与技术

指导教师 周虹

报 告 人 林浩晟　学号 2022280310

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

1. 首先更改窗口的大小以及窗口名称；

```
// 配置窗口属性
GLFWwindow* window = glfwCreateWindow(900, 900, "2022280310_林浩晟_实验4.1", NULL, NULL);
if (window == NULL)
{
    std::cout << "Failed to create GLFW window" << std::endl;
    glfwTerminate();
    return -1;
}
```

2. 仿照 cylinder 部分编写该部分的代码；首先新建 TriMesh 类，定义为 disk；调用对应的函数生成物体，随后设置平移、旋转、缩放等参数，再设置环境光、漫反射、镜面反射等光反射参数；最后将对应的纹理贴图传入 painter，将这个类放入 meshList 容器中；cone 物体同理。

```
TriMesh* disk = new TriMesh();
// @TODO: Task2 生成圆盘并贴图
disk->generateDisk(100, 0.1);
disk->setTranslation(glm::vec3(0, 0.0, 0.0));
disk->setRotation(glm::vec3(0, 0.0, 0.0));
disk->setScale(glm::vec3(2.0, 2.0, 1.0));

disk->setAmbient(glm::vec4(0.2, 0.2, 0.2, 1.0)); // 环境光
disk->setDiffuse(glm::vec4(0.7, 0.7, 0.7, 1.0)); // 漫反射
disk->setSpecular(glm::vec4(0.2, 0.2, 0.2, 1.0)); // 镜面反射
disk->setShininess(1.0); //高光系数
painter->addMesh(disk, "mesh_b", "./assets/disk.jpg", vshader, fshader);
meshList.push_back(disk);

// @TODO: Task2 生成圆锥并贴图
TriMesh* cone = new TriMesh();
cone->generateCone(100,0.1, 0.5);

cone->setTranslation(glm::vec3(0.5, -0.2, 0.0));  // 平移
cone->setRotation(glm::vec3(-90.0, 0.0, 0.0));     // 旋转
cone->setScale(glm::vec3(1.5, 1.0, 0.7));          // 缩放

// 设置圆锥的材质属性
cone->setAmbient(glm::vec4(0.2, 0.2, 0.2, 1.0));  // 环境光
cone->setDiffuse(glm::vec4(0.7, 0.7, 0.7, 1.0));  // 漫反射
cone->setSpecular(glm::vec4(0.2, 0.2, 0.2, 1.0)); // 镜面反射
cone->setShininess(1.0);
painter->addMesh(cone, "mesh_c", "./assets/cone.jpg", vshader, fshader);
meshList.push_back(cone);
```

3. 再打开 TriMesh 文件，编写 generateDisk 函数；首先，使用 cleanData()函数清空之前的模型数据。然后计算每个切片的弧度 step，这是通过将 2π 除以切片数目得到的，用于后续计算顶点坐标；随后通过循环生成下表面的顶点坐标、法向量和颜色，其中顶点坐标是通过极坐标系转换得到的，而 z 坐标设为 0 是表示圆盘在 xy 平面上；生成的数据被加入到相应的数组中；之后添加中心点的顶点坐标、法向量和颜色以构成圆盘的中心；另一个循环用于生成三角面片，每个矩形由两个三角形面片构成，循环中生成顶点索引并映射 UV 坐标；纹理坐标被添加到 vertex_textures 数组中，通过将极坐标映射到 UV 坐标的 0-1 范围内得到；然后我们存储三角形面片的每个顶点的纹理坐标的下标，并将法向量和颜色的下标设为与顶点坐标的下标相同；最后调用 storeFacesPoints()函数存储面片的顶点坐标即可。

```cpp
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

    // 生成下表面的顶点坐标，法向量和颜色
    float z = 0;
    for (int i = 0; i < num_samples; i++)
    {
        float theta = i * step;
        float x = radius * cos(theta);
        float y = radius * sin(theta);

        // 添加顶点坐标
        vertex_positions.push_back(glm::vec3(x, y, z));
        // 添加法向量
        vertex_normals.push_back(glm::vec3(0, 0, 1));
        // 使用法向量生成颜色，你也可以根据需要自定义颜色生成方式
        vertex_colors.push_back(glm::vec3(0, 0, 1));
    }

    // 中心点
    vertex_positions.push_back(glm::vec3(0, 0, 0));
    vertex_normals.push_back(glm::vec3(0, 0, 1));
    vertex_colors.push_back(glm::vec3(0, 0, 1));

    // 生成三角面片，每个矩形由两个三角形面片构成
    for (int i = 0; i < num_samples; i++)
    {
        // 面片1
        faces.push_back(vec3i(i, (i + 1) % num_samples, num_samples));

        // 将0-360度映射到UV坐标的0-1
        for (int j = 0; j < 2; j++)
        {
            float theta = (i + j) * step;
            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));
        }

        // 中心点的纹理坐标
        vertex_textures.push_back(glm::vec2(0.5, 0.5));

        // 对应的三角面片的每个顶点的纹理坐标的下标
        texture_index.push_back(vec3i(3 * i, 3 * i + 1, 3 * i + 2));
    }
    // 法向量和颜色的索引与顶点索引一致
    normal_index = faces;
    color_index = faces;

    storeFacesPoints();
}
```

4. 再编写 generateCone 函数；首先通过调用 cleanData()函数清空之前的模型数据。然后根据用户指定的切片数量计算每个切片的弧度 step，用于生成顶点坐标；随后通过循环遍历每个切片，计算并添加底部圆形的顶点坐标、法向量和颜色到顶点数组中；然后，添加圆锥尖端的顶点坐标、法向量和颜色；；使用循环生成连接底部和尖端的侧面三角面片，并将它们添加到面片数组中；接着为每个面片的顶点添加纹理坐标，并存储纹理坐标和法向量的索引；其中面片的颜色索引与顶点坐标的下标相同；最后调用 storeFacesPoints()函数存储面片的顶点坐标。

```
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;
// 生成圆锥底部的顶点坐标、法向量和颜色
    float z = 0;
    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);

        // 添加底部顶点坐标
        vertex_positions.push_back(glm::vec3(x, y, z));
        // 计算法向量并添加
        vertex_normals.push_back(normalize(glm::vec3(x, y, 0)));
        // 添加颜色，这里采用法向量作为颜色
        vertex_colors.push_back(normalize(glm::vec3(x, y, 0)));
    }

    // 添加圆锥尖端的顶点坐标、法向量和颜色
    vertex_positions.push_back(glm::vec3(0, 0, height));
    vertex_normals.push_back(glm::vec3(0, 0, 1));
    vertex_colors.push_back(glm::vec3(0, 0, 1));

    // 生成圆锥侧面的三角面片
    for (int i = 0; i < num_samples; i++)
    {
        // 三角面片
        faces.push_back(vec3i(num_samples, (i) % (num_samples), (i + 1) % (num_samples)));

        // 添加纹理坐标，这里采用简单的映射方式
        vertex_textures.push_back(glm::vec2(0.5, 1 - 0));
        vertex_textures.push_back(glm::vec2(1.0 * (i) / num_samples, 1 - 1));
        vertex_textures.push_back(glm::vec2(1.0 * (i + 1) / num_samples, 1 - 1));

        // 存储三角面片的每个顶点的纹理坐标的下标
        texture_index.push_back(vec3i(3 * i, 3 * i + 1, 3 * i + 2));
    }

    // 法向量和颜色的索引与顶点索引一致
    normal_index = faces;
    color_index = faces;

    storeFacesPoints();
}
```

5. 实验结果

可以看到结果和预期一致