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 吴嘉楷 学号 2022150168

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. 在 MeshPainter.cpp 中，参照 vPosition 等变量的传递方法，将纹理坐标相关代码补全。

但实际上，留空代码中已存在将纹理坐标传入着色器的相关代码，并不需要再对此进行修改。

代码截图：

```
// @TODO: Task1 将纹理坐标传入着色器
// 获得纹理坐标的位置
object.tLocation = glGetAttribLocation(object.program, "vTexture");
// 启用顶点属性数组，用于传递纹理坐标
glEnableVertexAttribArray(object.tLocation);
// 传递纹理坐标
glVertexAttribPointer(object.tLocation, 2,
    GL_FLOAT, GL_FALSE, 0,
    BUFFER_OFFSET( ( points.size() + colors.size() + normals.size())  * sizeof(glm::vec3)));
```

图 1 将纹理坐标传入着色器

代码说明：

glGetAttribLocation 函数从着色器程序 object.program 中获取与纹理坐标相关联的属性位置，并将其存储在 object.tLocation 中。这个位置标识了着色器中用于接收纹理坐标的数据槽。

然后， glEnableVertexAttribArray 函数启用这个顶点属性数组，使 OpenGL 能够使用它来传递顶点的纹理坐标。这样做的目的是让管线处理纹理时，着色器可以使用传入的坐标。

glVertexAttribPointer 函数定义了顶点属性数组的布局。它将 object.tLocation 绑定到指定的缓冲数据，描述了如何解释传递的数据。其中，参数 2 指定了每个顶点有两个浮点数来描述纹理坐标，GL_FLOAT 表示数据的类型，GL_FALSE 表示数据不需要归一化，0 是数据之间的步长（数据是紧密排列的）。

最后， BUFFER_OFFSET 指定了从缓冲区数据中计算纹理坐标的起始位置。这个偏移量是根据顶点数据（位置、颜色、法线）的大小来计算的，用于确保纹理坐标从正确的内存位置读取。

2. 参考圆柱体的函数写法，以及圆盘的表面展开，在 TriMesh.cpp 中补全 generateDisk 函数，生成一个圆盘。

源代码：

```
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 = 0.0; // 表示圆盘在 z = 0 的平面上
    for (int i = 0; i < num_samples; i++) {
        float theta = i * step; // 弧度
        float x = radius * cos(theta); // x 坐标
        float y = radius * sin(theta); // y 坐标
        vertex_positions.push_back(glm::vec3(x, y, z));// 添加顶点坐标
        vertex_normals.push_back(glm::vec3(0, 0, 1));// 添加法向量
        vertex_colors.push_back(glm::vec3(0, 0, 1));// 添加颜色
    }
    // 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));
    // 生成三角形面片，每个三角形面片由中心点和相邻两个顶点构成
    for (int i = 0; i < num_samples; i++) {
        // 面片
        faces.push_back(vec3i(i, (i+1) % num_samples, num_samples));
        // 将 0 到 360° 映射到 UV 坐标的 0 到 1
        for (int j = 0; j < 2; j++) {
            float theta = (i + j) * step;
            float x = cos(theta) / 2.0 + 0.5; // 除以 2 是为了确保将半径映射到 0 到 1
            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));// 中心点的纹理坐标
        texture_index.push_back(vec3i(3 * i, 3 * i + 1, 3 * i + 2));// 对应的三角面片的纹理坐标的下标
    }
    // 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;
    // 存储面片顶点数据
    storeFacesPoints();
}
```

代码说明：

generateDisk 的功能是生成一个位于 z = 0 平面的圆盘网格。generateDisk 函数使用 num_division 来决定圆盘的分割数目，并使用 radius 来设置圆盘的半径。开始时调用 cleanData 来清除先前生成的数据，以确保可以从头生成新的圆盘。

每个切片的弧度通过 2 * M_PI / num_samples 来计算，随后利用一个循环遍历所有的分割点。通过 cos 和 sin 函数计算每个顶点的 x 和 y 坐标，并存储在 vertex_positions 中。所有顶点都位于 z = 0 平面上，法向量设为 (0, 0, 1)，表示指向正 z 轴，默认颜色为 (0, 0, 1)，使用蓝色。

为圆心添加一个顶点 (0, 0, 0)，法向量和颜色与边缘顶点相同。这个中心点用于帮助创建三角形面片，使每个面片由圆心和两个相邻的边缘顶点组成。面片索引存储在 faces 中，每个面片的三个顶点由当前边缘点、下一个边缘点和圆心点构成，其中 (i + 1) % num_samples 确保最后一个点与第一个点相连。

纹理坐标的计算将单位圆上的顶点映射到 [0, 1] 的范围内。使用 cos 和 sin 将边缘顶点的坐标转换为纹理坐标，并添加到 vertex_textures。中心点的纹理坐标被设为 (0.5, 0.5)，确保其位于纹理的中心位置。texture_index 存储与每个面片相关联的纹理坐标索引。

法向量和颜色的索引与 faces 相同，因为它们在每个面片的顶点处是相同的。最后，通过调用 storeFacesPoints，所有生成的顶点和面片数据都被保存，以便后续绘制。

3. 参考圆柱体的函数写法，以及圆锥的表面展开，在 TriMesh.cpp 中补充 generateCone 函数，生成一个圆锥。

源代码：

```
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 theta = i * step; // 弧度
        float x = radius * cos(theta); // x 坐标
        float y = radius * sin(theta); // y 坐标
        vertex_positions.push_back(glm::vec3(x, y, z));  // 添加底部顶点坐标
        vertex_normals.push_back(normalize(glm::vec3(x, y, z)));  // 计算法向量并添加
        vertex_colors.push_back(normalize(glm::vec3(x, y, z)));    // 添加颜色，这里颜色和法向量一样
    }
    // 生成圆锥顶部的顶点坐标，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));
    // 生成侧面的三角形面片，每个三角形面片由底部相邻两个顶点和顶部顶点构成
    for (int i = 0; i < num_samples; i++) {
        // 三角形面片
        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));
```

```
            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));
            // 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));
        }
        // 三角面片的每个顶点的法向量的下标，与顶点坐标的下标 faces 一致
        normal_index = faces;
        // 三角面片的每个顶点的颜色的下标，与顶点坐标的下标 faces 一致
        color_index = faces;
        // 存储面片的顶点坐标
        storeFacesPoints();
}
```

代码说明：

generateCone 函数用于生成一个圆锥体网格。圆锥体的底部位于 z = 0 平面，顶点在 z = height 位置，num_division 指定了底部圆的分割数，radius 定义了底部的半径。

一开始通过 cleanData 清空之前的数据，确保生成新的网格。然后根据 num_samples = num_division 计算分割角度步长 step，遍历每个分割点，使用 cos 和 sin 函数计算底部圆的顶点坐标 (x, y, 0)，并将它们添加到 vertex_positions 中。法向量 vertex_normals 使用归一化的 (x, y, 0)，方向沿着**从圆心到顶点**的向外方向。颜色数据 vertex_colors 也设置为相同的**归一化**向量，作为顶点颜色。

在所有底部顶点生成完毕后，代码在圆锥顶部添加一个顶点 (0, 0, height)，其法向量为 (0, 0, 1)，颜色设置为蓝色 (0, 0, 1)。

接下来，使用一个循环生成侧面的三角形面片。每个三角形由底部相邻的两个顶点和顶部顶点构成。faces 数组存储每个三角形的三个顶点索引，其中 num_samples 索引表示顶部顶点，i % num_samples 和 (i + 1) % num_samples 处理底部的两个相邻顶点，确保最后一个顶点与第一个顶点相连。

纹理坐标使用**简单的映射**方式。顶部顶点纹理坐标被设置为 (0.5, 1)，底部两个顶点的纹理坐标基于 i 和 i + 1 的位置按比例分布在 [0, 1] 范围内。vertex_textures 数组存储所有纹理坐标，texture_index 存储三角面片顶点的纹理坐标索引。

由于每个顶点的法向量和颜色都与顶点数据索引一致，normal_index 和 color_index 直接设置为 faces。最后，storeFacesPoints 函数调用用于保存所有顶点和面片数据，以便后续渲染。

4. 在 main.cpp 文件的 init 函数中将圆盘创建出来。圆盘的纹理图片在 assets 文件夹中，叫 disk.jpg。

代码截图：

```
TriMesh* disk = new TriMesh();
// @TODO: Task2 生成圆盘并贴图
disk->generateDisk(100, 0.1); // 两个参数分别为分段数和半径
// 设置物体的变换属性
disk->setTranslation(glm::vec3(0.0, 0.0, 0.0));  // 平移
disk->setRotation(glm::vec3(0.0, 0.0, 0.0));     // 旋转
disk->setScale(glm::vec3(2.0, 2.0, 1.0));         // 缩放
// 将圆盘添加到Painter中，指定纹理与着色器
painter->addMesh(disk, "mesh_b", "./assets/disk.jpg", vshader, fshader);
// 将圆盘对象添加到对象列表
meshList.push_back(disk);
```

图 2 创建圆盘并贴图

代码说明：

这里创建了一个新的 TriMesh 对象 disk，并生成一个圆盘网格并应用纹理。首先，调用 generateDisk 方法，传入两个参数 100 和 0.1，分别指定圆盘的分割数为 100（即将圆盘分割成的等分数量）和圆盘的半径。这样便可以生成一个半径为 0.1 的圆盘，它有足够的分割来呈现平滑的圆形外观。

随后，对圆盘进行变换设置。setTranslation 将圆盘平移到 (0.0, 0.0, 0.0)，也就是说，圆盘的中心放置在原点。setRotation 将旋转设置为 (0.0, 0.0, 0.0)，即没有任何旋转变换。setScale 调整圆盘的缩放比例，将 x 和 y 方向放大为两倍，而 z 方向不变。这样可以使圆盘变宽，形成一个扁平的椭圆状。

接下来，使用 painter->addMesh 方法将圆盘添加到 Painter 对象中，并为圆盘指定纹理和着色器。这里 "mesh_b" 是一个标识符，"./assets/disk.jpg" 是用于圆盘的纹理图片路径，而 vshader 和 fshader 分别是顶点着色器和片段着色器，用于渲染圆盘的外观。

最后，将 disk 对象添加到 meshList，这是一个存储所有网格对象的列表，便于后续统一管理、回收和删除我们创建的物体对象。

5. 在 main.cpp 文件的 init 函数中将圆锥创建出来。圆锥的纹理图片在 assets 文件夹中，叫 cone.jpg。

代码截图：

```
TriMesh* cone = new TriMesh();
// @TODO: Task2 生成圆锥并贴图
// 生成圆锥并贴图
cone->generateCone(100, 0.1, 0.5);
// 设置圆锥的变换属性
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, 1.5, 0.7));          // 缩放
// 将圆锥添加到Painter中，指定纹理与着色器
painter->addMesh(cone, "mesh_c", "./assets/cone.jpg", vshader, fshader);
// 将圆锥对象添加到对象列表
meshList.push_back(cone);
```

图 3 创建圆锥并贴图

代码说明：

这里用于创建一个新的 TriMesh 对象 cone，生成一个圆锥网格，并应用纹理和设置变换属性。

首先，调用 generateCone 方法，传入三个参数 100、0.1 和 0.5，分别代表圆锥的分割数、底部半径和圆锥的高度。这会生成一个底部半径为 0.1、高度为 0.5 的圆锥，并将其分割为 100 个面片以确保外观平滑。

接下来，对圆锥设置几何变换。setTranslation 将圆锥平移到位置 (0.5, -0.2, 0.0)，这将圆锥移动到 x 轴的正方向和 y 轴的负方向位置，同时保持在 z 轴的原点平面。setRotation 对圆锥进行旋转，将 -90.0 度应用于 x 轴方向，意味着将圆锥向下倾斜 90 度进行调整，以使其尖端指向正 y 轴。setScale 设置缩放比例，将 x 和 y 方向放大为 1.5 倍，而 z 方向缩小为 0.7 倍，调整圆锥的形状使其更宽而矮。

随后，通过 painter->addMesh 将圆锥对象添加到 Painter 中，并指定纹理和着色器。"mesh_c" 是一个标识符，用于引用此对象，"./assets/cone.jpg" 指定了用于圆锥的纹理图片路径，vshader 和 fshader 是顶点和片段着色器，用于渲染圆锥。

最后，将 cone 添加到 meshList 中，方便管理、回收和删除我们创建的物体对象。

6. 修改窗口的标题和尺寸大小

```
// 配置窗口属性
GLFWwindow* window = glfwCreateWindow(700, 700, "2022150168_吴嘉楷_实验4.1", NULL, NULL);
```

图 4 配置窗口属性

7. 程序运行结果

运行程序，不进行任何键盘鼠标交互：

图 5 初始效果

多次按下键盘“u”键，调整 rotate 角度：

图 6 调整旋转角度后的效果

多次按下键盘“i”键，调整 up 角度：

图 7 调整 up 角度后的效果

综合使用键盘“u”、“i”、“o”键，综合调整角度以及物体与相机的距离：

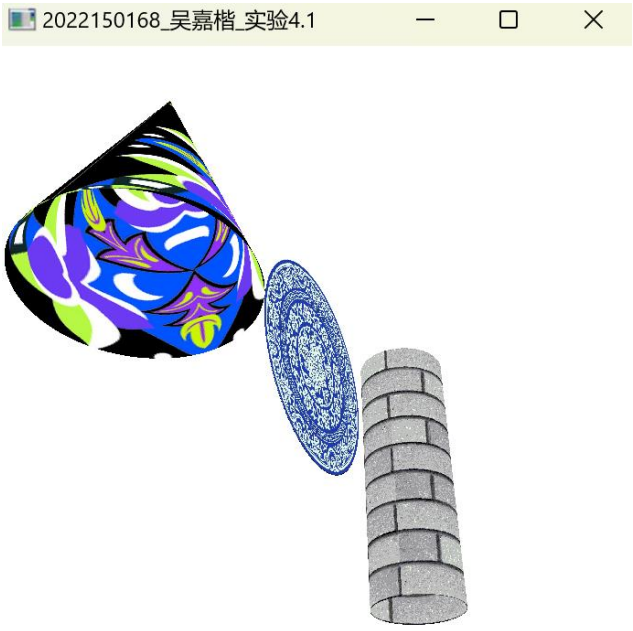

图 8 调整旋转、up 角度以及相机距离后的效果