深 圳 大 学 实 验 报 告

课程名称：计算机图形学

实验项目名称：实验四　带纹理的 OBJ 文件读取和显示

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：林宪亮　学号：2022150130　班级：国际班

实验时间：2024 年 11 月 19 日 -- 2024 年 12 月 02 日

实验报告提交时间：2024 年　11 月　25 日

教务部制

实验目的与要求：

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
4. 在程序中读取带纹理的 obj 文件，载入相应的纹理图片文件，将带纹理的模型显示在程序窗口中。

实验过程及内容：

1. 存储传入 GPU 的数据

```
// @TODO Task1 根据每个三角面片的顶点下标存储要传入GPU的数据
for (int i = 0; i < faces.size(); i++)
{
    // 坐标
    points.push_back(vertex_positions[faces[i].x]);
    points.push_back(vertex_positions[faces[i].y]);
    points.push_back(vertex_positions[faces[i].z]);

    // 颜色
    colors.push_back(vertex_colors[color_index[i].x]);
    colors.push_back(vertex_colors[color_index[i].y]);
    colors.push_back(vertex_colors[color_index[i].z]);
```

图 1 storeFacesPoints 函数 1

说明：

我的目标是根据 faces 中存储的顶点索引，从模型的顶点、颜色、法向量和纹理等属性数据中提取具体值，并将这些数据按照顶点顺序排列后存储到 GPU 缓冲区中，以便进行渲染操作。我使用了一个 for 循环来遍历 faces 容器，其中每个 faces[i] 都包含了一个三角形面片的三个顶点索引（x, y, z）。这些索引用于指向具体的顶点属性，例如坐标、颜色、法向量等。通过 faces[i].x, faces[i].y, 和 faces[i].z 这三个索引，我从 vertex_positions 中提取对应的三维坐标，并依次存储到 points 容器中。points 将成为 GPU 顶点缓冲区的一部分。对于颜色，我从 vertex_colors 中提取对应索引的颜色值。这里的 color_index 是面片的颜色索引，与 faces 中的顶点一一对应。我按顺序将每个顶点的颜色存储到 colors 容器中。

```
    // 法向量
    if (vertex_normals.size() != 0)
    {
        normals.push_back(vertex_normals[normal_index[i].x]);
        normals.push_back(vertex_normals[normal_index[i].y]);
        normals.push_back(vertex_normals[normal_index[i].z]);
    }

    // 纹理
    if (vertex_textures.size() != 0)
    {
        textures.push_back(vertex_textures[texture_index[i].x]);
        textures.push_back(vertex_textures[texture_index[i].y]);
        textures.push_back(vertex_textures[texture_index[i].z]);
    }

}
```

图 2 storeFacesPoints 函数 2

如果模型包含法向量数据（vertex_normals.size() != 0），我会从 vertex_normals 中根据 normal_index 提取对应的法向量值，并存储到 normals 容器中。这部分数据主要用于光照计算。同样，如果模型包含纹理坐标数据（vertex_textures.size() != 0），我会从 vertex_textures 中根据 texture_index 提取对应的纹理坐标值，并存储到 textures 容器中。这些纹理数据将用于贴图处理。

2. 读取 obj 文件

```
while (std::getline(fin, line))
{
    std::istringstream sin(line);
    std::string type;
    sin >> type;

    if (type == "v") // 解析顶点坐标
    {
        vertex_positions.push_back(parseVec3(sin));
    }
    else if (type == "vn") // 解析法向量
    {
        glm::vec3 normal = parseVec3(sin);
        vertex_normals.push_back(normal);
    }
    else if (type == "vt") // 解析纹理坐标
    {
        vertex_textures.push_back(parseVec2(sin));
    }
    else if (type == "f") // 解析面信息
    {
        parseFace(sin, faces, texture_index, normal_index);
    }
}

// 默认将法向量作为顶点颜色，索引保持一致
vertex_colors = vertex_normals;
color_index = normal_index;
storeFacesPoints();
```

图 2 readobj

说明：

这段代码的目标是读取 .obj 文件的每一行，根据不同的类型（如顶点坐标、法向量、纹理坐标和面信息）解析并存储相应的数据。这些数据之后将被传递到 GPU 进行渲染处理。首先，我使用 std::getline(fin, line) 逐行读取 .obj 文件中的数据。在每一行中，我用 std::istringstream 解析字符串，通过 sin >> type 获取当前行的类型（即首个关键字，如 v、vn、vt 或 f），然后根据不同的类型执行不同的解析操作。当 type == "v" 时，我知道这一行包含的是顶点坐标数据。因此，我调用 parseVec3(sin) 来解析一个三维坐标，并将解析到的坐标存储到 vertex_positions 容器中。当 type == "vn" 时，表示这一行包含法向量数据。和顶点坐标解析类似，我调用 parseVec3(sin) 来解析法向量，并将其存储到 vertex_normals 容器中。当 type == "vt" 时，这一行包含的是纹理坐标。我调用 parseVec2(sin) 来解析二维纹理坐标并将其存储到 vertex_textures 容器中。当 type == "f" 时，表示这一行是面数据。每个面由多个顶点组成，并且每个顶点可以包含多个属性，如位置、法向量和纹理坐标。在此部分，我调用 parseFace(sin, faces, texture_index, normal_index) 来解析面信息，解析后的数据将存储在 faces、texture_index 和 normal_index 中。在读取完成所有面数据后，我将 vertex_normals（法向量）作为默认的顶点颜色，索引也与法向量保持一致。这个操作将法向量作为颜色传递给 GPU，通常用于光照计算或着色。

```
glm::vec3 parseVec3(std::istringstream& sin)
{
    GLfloat x, y, z;
    sin >> x >> y >> z;
    return glm::vec3(x, y, z);
}
glm::vec2 parseVec2(std::istringstream& sin)
{
    GLfloat u, v;
    sin >> u >> v;
    return glm::vec2(u, v);
}
void parseFace(
    std::istringstream& sin,
    std::vector<vec3i>& faces,
    std::vector<vec3i>& texture_index,
    std::vector<vec3i>& normal_index)
{
    int vertex[3], texture[3], normal[3];
    char slash;

    for (int i = 0; i < 3; i++) // 处理三角形的三个顶点
    {
        sin >> vertex[i] >> slash >> texture[i] >> slash >> normal[i];
        vertex[i]--; texture[i]--; normal[i]--; // 索引从 1 转为 0
    }

    faces.push_back(vec3i(vertex[0], vertex[1], vertex[2]));
    texture_index.push_back(vec3i(texture[0], texture[1], texture[2]));
    normal_index.push_back(vec3i(normal[0], normal[1], normal[2]));
}
```

图 3 辅助函数

3. 文件读取与绘制

```
    // 创建桌子模型
TriMesh* table = new TriMesh();
table->setNormalize(true);  // 归一化模型尺寸
table->readObj("./assets/table.obj");  // 读取桌子模型文件

// 设置桌子的旋转、位移和缩放属性
table->setTranslation(glm::vec3(-0.8, -0.2, 0.0));  // 偏移
table->setRotation(glm::vec3(-90.0, 0.0, 0.0));     // 沿X轴旋转-90度
table->setScale(glm::vec3(2.0, 2.0, 2.0));          // 放大2倍

// 添加桌子到绘制器并指定纹理和着色器
painter->addMesh(table, "mesh_a", "./assets/table.png", vshader, fshader);
meshList.push_back(table);  // 保存到容器以便统一管理内存

// 创建娃娃模型
TriMesh* wawa = new TriMesh();
wawa->setNormalize(true);  // 归一化模型尺寸
wawa->readObj("./assets/wawa.obj");  // 读取娃娃模型文件

// 设置娃娃的旋转、位移和缩放属性
wawa->setTranslation(glm::vec3(0.8, 0.0, 0.0));  // 向右移
wawa->setRotation(glm::vec3(-90.0, 0.0, 0.0));  // 沿X轴旋转-90度
wawa->setScale(glm::vec3(2.0, 2.0, 2.0));        // 放大2倍

// 添加娃娃到绘制器并指定纹理和着色器
painter->addMesh(wawa, "mesh_b", "./assets/wawa.png", vshader, fshader);
meshList.push_back(wawa);  // 保存到容器以便统一管理内存

// 设置背景颜色为白色
glClearColor(1.0, 1.0, 1.0, 1.0);
```

图 4 init

在这段代码中，我首先创建了桌子和娃娃模型。对于每个模型，我使用 TriMesh 类来初始化，并通过调用 setNormalize(true) 方法将模型的尺寸归一化，以确保它们的大小一致，便于后续的变换操作。接着，我通过 readObj 方法分别读取了桌子和娃娃的 .obj 文件，这些文件包含了模型的顶点、法线、纹理等信息。

随后，我设置了每个模型的变换属性。对于桌子，我将它沿 X、Y 和 Z 轴进行了平移、旋转和缩放操作。具体来说，我将桌子向左偏移了 0.8 单位，沿 X 轴旋转了 -90 度，并将它的尺寸放大了 2 倍。对于娃娃模型，我进行了类似的变换，但位置偏移是向右移动了 0.8 单位。

接下来，我将这两个模型添加到绘制器中，并指定了它们的纹理和着色器。通过调用 addMesh 方法，我将模型与对应的纹理文件（桌子用 table.png，娃娃用 wawa.png）以及着色器程序一起传递给绘制器。为了便于后续管理，我还将这两个模型保存到了 meshList 容器中。

最后，我设置了背景颜色为白色，通过 glClearColor(1.0, 1.0, 1.0, 1.0) 来确保渲染时背景是白色的。这样，整个模型的加载、变换和渲染准备过程就完成了。

4. 实验结果

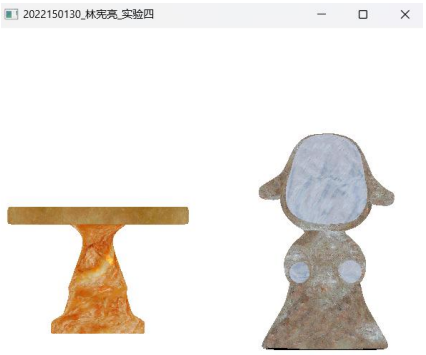

图 5 结果图 1

调整角度：

图 6 结果图 2

图 7 结果图 3

调整距离：

图 8 结果图 4

实验成功!!!

深圳大学学生实验报告用纸

实验结论：

在这次实验中，我深入了解了如何读取带纹理的 3D 模型文件，并将其正确地渲染到屏幕上。尽管实验的目标明确，但在实现过程中遇到了一些困难，尤其是在处理数据读取、存储和渲染这几个环节时，经过反复调试和优化，我对计算机图形学的知识有了更深的理解和掌握。

举个例子，在开始实验时，最初的挑战是如何正确解析 OBJ 文件中的不同数据。OBJ 文件通常包含多个部分，例如顶点坐标、法向量、纹理坐标和面信息，而每个部分都有其独特的格式。刚开始时，我并不完全清楚如何处理每一行的数据格式，特别是面信息的解析，面片数据包含了顶点、纹理坐标和法向量的索引，必须确保这些索引对应正确的数组位置。为了解决这个问题，我花了一些时间参考了实验 4.1 中的代码，并对 readObj 函数进行了改进。通过逐行解析文件并根据不同的类型（如“v”、“vn”、“vt”和 “f”）来分类处理，我成功地将每部分数据存储到相应的容器中.尽管在实验中遇到了不少困难，但通过不断查阅文档、调试和尝试不同的方法，我最终都一一克服了这些问题。在整个过程中，我深刻认识到，编写高效的图形程序不仅仅需要良好的编码习惯，还需要对底层原理有一定的了解。通过这次实验，我进一步加深了对计算机图形学的理解，特别是 OBJ 文件格式的处理、数据的正确存储与传输以及如何优化渲染效率等方面。

总的来说，尽管遇到了许多困难，但每一次解决问题的过程都让我收获颇丰。这次实验不仅帮助我提升了对图形编程的理解，也增强了我调试和优化程序的能力。这些经验对于我未来深入学习计算机图形学和游戏开发将起到重要的推动作用。

指导教师批阅意见：

成绩评定：

指导教师签字：

年　　月　　日

备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。