计算机图形学

（Computer Graphics）

实验题目：OpenGL 中的模型显示和纹理坐标定义

目录：

实验报告：

1　实验内容描述，使用模型定义顶点的属性信息：法向量、纹理坐标的描述

在实验中，我是用的顶点结构体由 顶点数据、纹理坐标和法向量三个数据成员组成

```
struct Vertex
{
    Vector3f m_pos;
    Vector2f m_tex;
    Vector3f m_nor;

    inline Vertex() { ... }

    Vertex(Vector3f pos) { ... }

    Vertex(Vector3f pos, Vector3f nor) { ... }

    Vertex(Vector3f pos, Vector3f nor,Vector2f tex) { ... }
};
```

图 1 顶点结构体

其中，顶点坐标由三个 float 数据组成，分别代表其(x,y,z)坐标值，纹理坐标由两个 float 数据组成，分别代表其(u,v)坐标，法向量由三个 float 数据组成，共同组成其向量值。

2　实验功能算法描述，即纹理材质是如何建立、如何载入、如何使用的

2.1　纹理是使用 blender 对物体进行 uv 展开后，使用纹理图片进行贴图得到的。

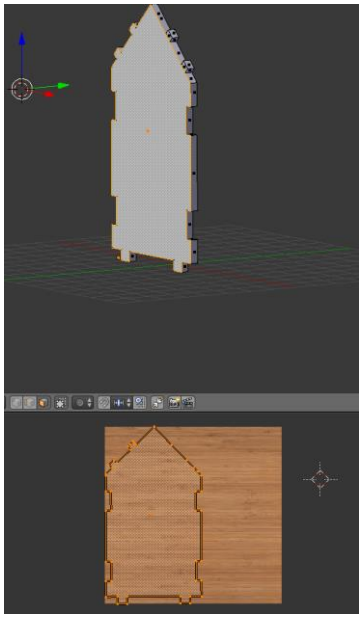

图 2 blender 软件进行贴图的过程

2.2 纹理的载入仍然和实验一一样，使用自己的一个 ObjLoader 类，对 obj 文件进行读取，然后将其放入顶点数组中的对应属性中。

```
class ObjLoader
{
private:
    vector<Vertex> VertexBuffer; //顶点
    vector<unsigned int> VertexIndexBuffer; //顶点索引
    vector<Vector3f> NormalBuffer; //法向量
    vector<unsigned int> NormalIndexBuffer; //法向量索引
    vector<Vector2f> TextureBuffer; //纹理坐标

    vector<Vertex> AllVertexBuffer;
public:
    void load(const char* filename);
    vector<Vertex> getVertexBuffer();
    vector<unsigned int> getIndexBuffer();

    vector<Vertex> getAllVertexBuffer();
};
```

图 3 自己实现的 obj 文件加载类

```
void ObjLoader::load(const char * filename)
{
    ifstream f(filename, ifstream::in);
    if (f.is_open())
    {
        string line;
        string token;
        while (getline(f, line))
        {
            if (line.find(" ") == string::npos)
            {
                continue;
            }
            token = line.substr(0, line.find(" "));
            line = line.substr(line.find(" ") + 1, line.length());
            if (token == "v") { ... }
            else if (token == "vt") { ... }
            else if (token == "vn") { ... }
            else if (token == "f") { ... }
            else
            {
                continue;
            }
        }

        f.close();
    }
}
```

图 4 加载类的对外接口的具体实现

其中可以看到，根据 obj 文件数据的标识字母，如”vt”即为纹理坐标，”f”即为面的数据，里面每个数据点都由 “顶点坐标索引/纹理坐标索引/法向量坐标索引” 这样的格式组成，做对应的语义分析之后放入对应的定点结构体数据中，完成载入。

2.3 纹理的使用，使用简单包装过的纹理类，每一个纹理类的对象都有纹理对象、纹理目标、纹理图片路径作为其数据成员。

```
class Texture
{
public:
    Texture(GLenum TextureTarget, const std::string& FileName);

    bool Load();

    void Bind(GLenum TextureUnit);

private:
    std::string m_fileName;
    GLenum m_textureTarget; //纹理目标
    GLuint m_textureObj; //纹理对象
    //Magick::Image* m_pImage;
    //Magick::Blob m_blob;
};
```

图 5 纹理类

里面使用了 SOIL(Simple OpenGL Image Loader)实现了图片数据的读入。

```
int width, height,channels;
unsigned char* image = SOIL_load_image(m_fileName.c_str(), &width, &height, &channels, SOIL_LOAD_RGBA);
```

图 6 使用 SOIL 读入文件数据

封装实现了绑定图片数据，激活纹理单元并绑定纹理对象等功能。

```
if (channels == 3)
{
    glTexImage2D(m_textureTarget, 0, GL_RGB, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);
}
else if (channels == 4)
{
    glTexImage2D(m_textureTarget, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);
}
```

图 7 绑定数据

```
void Texture::Bind(GLenum TextureUnit)
{
    glActiveTexture(TextureUnit);
    glBindTexture(m_textureTarget, m_textureObj);
}
```

图 8 激活纹理单元，并绑定纹理对象及目标

3 实验 shader 程序描述，即 vertex shader 和 fragment shader 的程序代码及说明

3.1 Vertex Shader

在顶点着色器中，直接将传入的顶点的纹理坐标传给片元着色器。

```
layout(location = 0) in vec3 Position;  //传入的顶点坐标
layout(location = 1) in vec2 TexturePosition;   //传入的纹理坐标
layout(location = 2) in vec3 Normal; //传入的顶点法向量
```

图 9 读取从顶点结构体中包含的顶点数据

```
out vec2 TextureAfterTrans; //输出的纹理坐标
```

图 10 指定输出的纹理坐标

3.2 Fragment Shader

```
in vec2 TextureAfterTrans;
```

图 11 得到从顶点着色器中插值之后传来的纹理坐标

```
FragColor = texture2D(gSampler, TextureAfterTrans.st);
```

图 12 输出的片源颜色即为纹理采样器从纹理图片中取得的像素值

4 实验结果，要贴实验结果图

成功将纹理贴在导入的模型上。

图 13 纹理贴图效果（正、侧视图）

5 心得体会

在这次实验中，由于需要导入的模型经过贴图，因此在导出模型之后，obj 文件中多了 uv 坐标和法向量，根据这种变化，自己进一步修改了 ObjLoader，看到了原始的 uv 坐标数据之后，加深了对纹理贴图的理解。

同样，在 shader 语言中也有了新的认识和收获，实验一中每一个顶点只有顶点坐标一个数据，因此结构叫简单，现在每一个顶点有 3 个数据，因此，在绑定数据上也认真学习了一下，绑定数据时候，还需要注意数据偏移的计算。

在读取纹理图片的时候也接触了 Magick++和 SOIL 这两个库，最后顺利将图片进行读入。

这次实验让我对纹理有了基本的认识，并且学会了在 opengl 中导入带纹理的图像，受益匪浅。