《计算机图形学实验》综合实验报告

题目 基于 OpenGL 的纹理映射茶壶

学 号 20201060277

姓 名 张天翊

指导教师 钱文华

日 期 2022 年 6 月 18 日

目录

摘　要：

在本次综合实验中，调用了 OpenGL 的纹理映射方法，指定并启用二维纹理的函数，建立物体表面上点与纹理空间的对应关系——明确定点对应的纹理坐标，从而生成了一个具有克莱因蓝与天蓝色条纹相错的茶壶

关键词：　OpenGL；纹理渲染；线性插值

1 实验背景与内容

1.1 实验背景

渲染简单的理解可以是这样：将三维物体或三维场景的描述转化为一幅二维图像，生成的二维图像能很好的反映三维物体或三维场景。

当一个图元被光栅化为一堆零个或多个片段的时候，插值、贴图和着色阶段就在片段属性需要的时候插值，执行一系列的贴图和数学操作，然后为每个片段确定一个最终的颜色。除了确定片段的最终颜色，这个阶段还确定一个新的深度，或者甚至丢弃这个片段以避免更新帧缓存对应的像素。允许这个阶段可能丢弃片段，这个阶段为它接收到的每个输入片段产生一个或不产生着过色的片段。

1.2 实验内容

利用 Visual C++, OpenGL, Java 等工具，实现三维图形渲染，自定义三维图形，三维图形不能仅仅是简单的茶壶、球体、圆柱体、圆锥体等图形，渲染过程需加入纹理、色彩、光照、阴影、透明等效果，可采用光线跟踪、光照明模型、纹理贴图、纹理映射等算法。

2 开发工具与程序设计

2.1 开发工具

硬件：

笔记本电脑 HUAWEI MateBook14 2020 版本

操作系统：

windows11

所用软件：

CodeBlocks 2017

所需预编译头文件：

<windows.h>、<GL/glut.h>、<cstdlib>、<cstdio>、<cmath>

网络环境：

无要求

2.2 程序设计

2.2.1 实现目的

自定义茶壶，实现茶壶的三维图形渲染。渲染过程选择性加入纹理、色彩、光照、阴影、透明等效果，采用光线跟踪、光照明模型、纹理贴图、纹理映射等算法。

2.2.2 基本模块介绍

模块	功能
makeStripeImage 函数	建立物体表面上点与纹理空间的对应关系
init 函数	建立纹理渲染，生成光照模型等
display 函数	指定创建窗口时其显示模式的类型，画茶壶
reshape 函数	定义当窗口的形状改变事件发生时 调用的处理函数
idle 函数	标记重新绘制当前窗口，在下一个显示回调 只产生单一的重新显示回调
main 主函数	调用各个函数，生成图形

3 关键算法的理论介绍及程序实现步骤

3.1 理论介绍

用于指定二维的函数为：

Void glTexImage2D(GLenum target, Glint level, Glint components, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *texels);

其中，参数 target 取值一般为 GL_TEXTURE_2D，与二维的纹理相对应。参数 Level 表示纹理多分辨率层数，通常取值为 0，表示只有一种分辨率。参数 components 的可能取值为 1～4 的整数以及多种符号常量(如 GL_RGBA)，表示纹理元素中存储的哪些分量（RGBA 颜色、深度等）在纹理映射中被使用，1 表示使用 R 颜色分量，2 表示使用 R 和 A 颜色分量，3 表示使用 RGB 颜色分量，4 表示使用 RGBA 颜色分量。参数 width，height，depth 分别指定纹理的宽度、高度、深度。参数 format 和 type 表示给出的图像数据的数据格式和数据类型，这两个参数的取值都是符号常量（比如 format 指定为 GL_RGBA,type 指定为 GL_UNSIGNED_BYTE，参数 texels 指向内存中指定的纹理图像数据。

在定义了纹理之后，需要启用纹理的函数：

glEnable(GL_TEXTURE_2D);

在启用纹理之后，需要建立物体表面上点与纹理空间的对应关系，即在绘制基本图元时，在 glVertex 函数调用之前调用 glTexCoord 函数，明确指定当前顶点所对应的纹理坐标，例如：

```
glBegin（GL_TRIANGLES）;
    glTexCoord2f(0.0, 0.0); glVertex2f(0.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex2f(15.0, 15.0);
glTexCoord2f(1.0, 0.0); glVertex2f(30.0, 0.0);
glEnd();
```

其图元内部点的纹理坐标利用顶点处的纹理坐标采用线性插值的方法计算出来。

在 OpenGL 中，纹理坐标的范围被指定在[0,1]之间，而在使用映射函数进行纹理坐标计算时，有可能得到不在[0,1]之间的坐标。此时 OpenGL 有两种处理方式，一种是截断，另一种是重复，它们被称为环绕模式。在截断模式（GL_CLAMP）

中，将大于 1.0 的纹理坐标设置为 1.0，将小于 0.0 的纹理坐标设置为 0.0。在重复模式（GL_REPEAT）中，如果纹理坐标不在[0,1]之间，则将纹理坐标值的整数部分舍弃，只使用小数部分，这样使纹理图像在物体表面重复出现。例如，使用下面的函数：

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);

glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S, GL_REPEAT);

分别指定二维纹理中 s 坐标采用截断或重复处理方式。

另外，在变换和纹理映射后，屏幕上的一个像素可能对应纹理元素的一小部分（放大），也可能对应大量的处理元素（缩小）。在 OpenGL 中，允许指定多种方式来决定如何完成像素与纹理元素对应的计算方法（滤波）。比如，下面的函数可以指定放大和缩小的滤波方法：

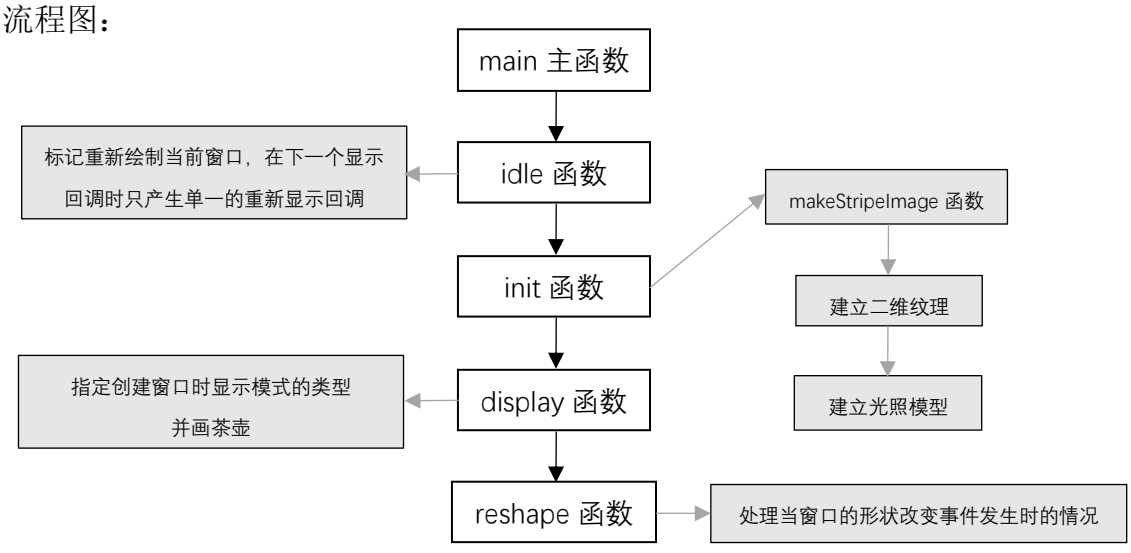
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);

其中，glTexParameteri 函数的第一个参数指定使用的是一维、二维或三维纹理；第二个参数为 GL_TEXTURE_MAG_FILTER 或 GL_TEXTURE_MIN_FILTER，指出要指定缩小还是放大滤波算法；最后一个参数指定滤波的方法。

3.2 流程介绍

流程图：



4 实验结果

4.1 运行截图

4.2 结果与问题分析

利用二维纹理渲染的知识，通过设定 makeStripeImage 函数内 stripeImage 数组的值设定了茶壶的颜色——克莱因蓝色为底，天蓝色作为渐变条形花纹。利用光照渲染实现了茶壶的立体感。通过设定 display 函数内 glRotatef(roangles,X, Y, Z)中 X, Y, Z 的值，可以预设茶壶自转的角度。

问题在于：不能实现鼠标键盘控制茶壶旋转，只能在代码中预设固定的旋转角度。没有实现三维渲染或图片贴图，且只能生成简单的茶壶图形。

5 实验体会及小结

在本次实验过程中，我不可避免地遇到了一些问题，但也学习到了很多知识。

修改开源的代码后生成图形的结果往往和预料的不一致，调用键盘鼠标函数以实现图形交互式旋转的想法也因此失败。这意味着我对计算机图形学的知识掌握不够牢固，还需要继续学习。

同时，我在不断地尝试不同代码和 CSDN 等一系列网站中的实战举例中，学习并熟练掌握了光照模型、一维和二维纹理渲染的使用方法。

总而言之，这次实验是对本学期学习成果的综合锻炼，让我对计算机图形学有了更深刻、更具体的认识，拓宽了眼界，为以后专业性的学习奠定坚实的基础。

6 参考文献

[1] 马丹，阳凡林，崔晓东，等. 基于 OpenGL 的海底地形三维渲染方法［J］. 山东科技大学学报（自然科学版）：99-106.

[2] 罗仁, 李子轩. 基于 OpenGL 的三维实时渲染引擎. 软件开发与应用 2021 年第 18 期——1003-9767（2021）18-112-04

[3] 尚游. OpenGL 高级图形编程指南[M]. 哈尔滨：哈尔滨工程大学出版社,1999:41

[4] 张肇同. 基于 OpenGL 的三维模型读取与动态观察[J].科技视界, 2017(34):4-5

7 附录

源代码：

```
#include <windows.h>
```

```
#include <GL/glut.h>
#include <cstdlib>
#include <cstdio>
#include <cmath>
#define stripeImageWidth 32
#define stripeImageHeight 32
GLubyte stripeImage[stripeImageWidth][stripeImageHeight][4];
void makeStripeImage(void)
{
    int i, j;
    for (i = 0; i < stripeImageWidth; i++)
    {
        for (j = 0; j < stripeImageHeight; j++)
        {
            stripeImage[i][j][0] = (GLubyte)0;
            stripeImage[i][j][1] = (GLubyte)(j * 6 - 1);
            stripeImage[i][j][2] = (GLubyte)200;
            stripeImage[i][j][3] = (GLubyte)(i * 4 - 1);
        }
    }
}
static GLfloat xequalzero[] = { 1.0, 1.0, 1.0, 1.0 };
static GLfloat slanted[] = { 1.0, 1.0, 1.0, 1.0 };
static GLfloat *currentCoeff;
static GLenum currentPlane;
static GLint currentGenMode;
static float roangles;
void init(void)
{
    glClearColor(1.0, 1.0, 1.0, 1.0);
```

```
glEnable(GL_DEPTH_TEST);
glShadeModel(GL_SMOOTH);
makeStripeImage();
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
glTexImage2D(GL_TEXTURE_2D, 0, 4, stripeImageWidth, stripeImageHeight,
0, GL_RGBA, GL_UNSIGNED_BYTE, stripeImage);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE);
currentCoeff = xequalzero;
currentGenMode = GL_OBJECT_LINEAR;
currentPlane = GL_OBJECT_PLANE;
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, currentGenMode);
glTexGenfv(GL_S, currentPlane, currentCoeff);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_2D);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_AUTO_NORMAL);
glEnable(GL_NORMALIZE);
glMaterialf(GL_FRONT, GL_SHININESS, 64.0);
roangles = 90.0f;
//光照
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat mat_shininess[] = { 90.0 };
```

```
    GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };      //光源位置
    GLfloat white_light[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat Light_Model_Ambient[] = { 0.6, 0.6, 0.6, 1.0 };
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);      //设置材料反射指数（纯镜面反射）
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);  //设置材料反射指数（材料反射指数）
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);    //建立光源 （光源位置）
    glLightfv(GL_LIGHT0, GL_DIFFUSE, white_light);          //建立光源 （漫反射光分量强度）
    glLightfv(GL_LIGHT0, GL_SPECULAR, white_light);         //建立光源 （折射光分量强度）
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, Light_Model_Ambient);
}
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glRotatef(roangles, 0.0, 1.0, 0.0);
    glutSolidTeapot(2.0);
    glPopMatrix();
    glFlush();
}
void reshape(int w, int h) {
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
        glOrtho(-3.5, 3.5, -3.5*(GLfloat)h / (GLfloat)w, 3.5*(GLfloat)h / (GLfloat)w,
-3.5, 3.5);
```

```
        else
            glOrtho(-3.5*(GLfloat)w / (GLfloat)h, 3.5*(GLfloat)w / (GLfloat)h, -3.5, 3.5,
-3.5, 3.5);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}
void idle()
{
        roangles += 0.05f;
        glutPostRedisplay();
}
int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(600, 600);
        glutInitWindowPosition(250, 250);
        glutCreateWindow("计算机图形学期末综合——三维图形渲染");
        glutIdleFunc(idle);
        init();
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutMainLoop();
        return 0;
}
```