基于 OpenGL 的三维立体图形渲染

学院 ：信息学院

专业：计算机科学与技术

学号：20201060561

姓名：彭莎淋

摘要

真实感图形绘制是计算机图形学的一个重要组成部分，其综合利用数学、物理学、计算机科学和其他学科知识在计算机图形设备上生成带色彩的具有真实感的图形。而自然景物是会受到环境光、材料、反射光、纹理等因素影响的，所以真实感图形的生成对于可见面的颜色的计算至关重要，计算可见面的颜色即根据基于光学物理的光照模型计算可见面投射到观察者眼中的光亮度大小和颜色分量，并将它转化为适合图形设备的颜色值，从而确定投影画面上每一象素的颜色值，最终生成图形。

本论文主要研究三维图形的纹理渲染和光照模型，通过这两项算法对三维图形进行简单的绘制，向真实感图形靠近。纹理渲染需要先生成纹理矩阵，再将纹理映射到图像表面；光照模型需要考虑光照射到物体上产生的反射光、光透过物体产生的透射光，对于透射光需要考虑物体的材料，结合这几个因素才能够使图形更接近真实物体。本文将以三维图形机器人、茶壶为例，通过 OpenGL 语言研究纹理渲染、光照对于真实感图形生成的影响。

关键字：三维渲染，OpenGL，纹理映射，光照模型，材质，阴影

目录

第一章 绪论

1.1 实验背景

计算机图形学的主要研究内容是利用计算机对图形数据描述、图形处理、图形计算、图形显示的理论和算法。其研究的图形主要由点、线、面、体等元素构成，还包括灰度、色彩、线宽、线型等元素。真实感图形的生成是计算机图形学的重点，能够真实地反应现实生活中的物体是其目标。

在计算机图形学中，想要真实地反应自然物体，需要从以下四个步骤完成：一是建模，即利用一定的数学方法建立所需三维场景的集合描述，场景的集合描述将直接影响图形的复杂性和图形绘制的计算耗费；二是将三维几何模型经过一定的变换转为二维平面透视投影图；三是确定场景中所有的可见面，运用消隐算法将不可见面消去；四是计算场景中可见面的颜色，根据基于光学物理的光照模型计算可见面投射到观察者眼中的光亮度大小和颜色分量，并将其转换成适合图形设备的颜色值，确定投影画面上的像素颜色值，形成最终的图形。

针对第四步骤——计算可见面的颜色，包括纹理渲染、光照模型、纹理贴图、光线跟踪等算法。其中，纹理渲染是指将创建的纹理渲染到纹理对象上；光照模型分为简单光照模型和复杂光照模型，简单光照模型即假定图形表面是光滑的且其材料为理想材料，当光照射到图形表面时只能产生反射光，而复杂光照模型还需要考虑周围环境光对图形表面的影响。但现实生活中影响图形表现的因素远比这复杂，想要让生成的图形更接近现实，深入研究其影响因素必不可少。

综上所述，从计算图形学领域来说，研究纹理渲染、光照模型对于真实感图形的生成具有重要研究意义。

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，渲染过程须加入纹理、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。

第二章 相关技术介绍

2.1 开发工具

本实验基于 code blocks 编程软件平台，配置 OpenGL API 接口调用图形函数。OpenGL 是用于渲染 2D、3D 矢量图形的跨语言、跨平台的应用程序编程接口（API）。这个接口由近 350 个不同的函数调用组成，用来绘制从简单的图形比特到复杂的三维景象。作为独立于操作系统的开放的三维图形的软件开发包，在其基础上开发的应用程序能够简单方便的移植于各种平台。OpenGL 常用于 CAD、虚拟现实、

科学可视化程序和电子游戏开发。其具有强大的通用性和可移植性，可以将其轻松的移植在多个不同的平台上进行二次开发；能够转换 3D 图形设计软件制作的模型文件；配备了高级图形库：Open Inventor、Cosmo3D、Optimizer 等。

2.2 程序设计及实现目的及基本模块介绍

2.2.1 纹理图案

采用数组的形式存储纹理图案，定义一个[32][32][4]的三足码数组存储 32*32 的二维表面纹理图案，每个纹理颜色由四个 RGBA 分量指定，实现纹理图像、密度、颜色的生成及存储。

2.2.2 纹理映射

通过调用 glEnable( )函数启动纹理映射，将纹理映射到三维图形上，还可以调用 glTexParmeteri( )函数来确定如何把纹理象素映射成像素，以减少贴图过程中产生的图像失真。由于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，采用重复处理方式，它们也被称为环绕模式。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。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），因此，定义 glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR) glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)指定采用线性插值的方法放大和缩小二维纹理。

2.2.3 光照模型

先定义光源的颜色值，用一维数组存储环境光、漫射光、镜面光的颜色值；定义光照属性的模型，使用 glLightModelfv( )函数，定义整个场景的环境光的 RGBA 强度和镜面反射角度是如何计算的；使用 glLightfv( )函数设置环境光、漫射光、镜面光、光源位置；使用 glEnable( )函数启动光照和光源。该模块实现了选择光照模型，设置光源颜色及位置、环境光、漫射光及镜面光等功能。

2.2.4 明暗处理

在计算机图形学中，光滑的曲面表面常用多边形予以逼近和表示，而每个小多边形轮廓（或内部）就用单一的颜色或许多不同的颜色来勾画（或填充），这

种处理方式就称为明暗处理。在 OpenGL 中，用单一颜色处理的称为平面明暗处理（Flat Shading），用许多不同颜色处理的称为光滑明暗处理（Smooth Shading），也称为 Gourand 明暗处理（Gourand Shading）。本实验采用光滑明暗处理方式，其所有点的法向由内插生成（双线性插值法），使得颜色与颜色之间具有一定连续性。

2.2.5 材质

OpenGL 用材料对光的红、绿、蓝三原色的反射率来近似定义材料的颜色。像光源一样，材料颜色也分成环境、漫反射和镜面反射成分，它们决定了材料对环境光、漫反射光和镜面反射光的反射程度。在进行光照计算时，材料对环境光的反射率与每个进入光源的环境光结合，对漫反射光的反射率与每个进入光源的漫反射光结合，对镜面光的反射率与每个进入光源的镜面反射光结合。对环境光与漫反射光的反射程度决定了材料的颜色，并且它们很相似。对镜面反射光的反射率通常是白色或灰色（即对镜面反射光中红、绿、蓝的反射率相同）。镜面反射高光最亮的地方将变成具有光源镜面光强度的颜色。先定义材质的具体参数值，包括环境光颜色、漫反射光颜色、镜面光颜色；使用 glMaterialfv( )函数实现材质定义，包括环境光、漫反射光、镜面反射光、镜面指数参数。

材质 RGB 值与光源 RGB 值的关系为：若 OpenGL 的光源颜色为（LR、LG、LB），材质颜色为（MR、MG、MB），在忽略所有其他反射效果的情况下，最终到达眼睛的光的颜色为（LR*MR、LG*MG、LB*MB）。

2.2.6 三维模型生成

机器人的模型采用基本模型包括实心球体、实心长方体，通过将画球、画长方体、画半球封装在函数内，在 display 函数中通过传入位置参数和大小参数调用函数，绘制机器人模型；而茶壶则采用 glutSolidTeapot( )函数绘制实心茶壶，给定茶壶大小参数。

绘制三维图形时还需要指定视角位置，通过 gluLookAt()函数定义，不然后面的 gluPerspective()函数无法操作视角。

2.2.7 交互设计

鼠标、键盘交互设计包括鼠标点击窗口实现环境光颜色的改变、键盘操作变换物体方向、鼠标点击不放可移动物体。

鼠标点击时，环境光随机生成，采用 rand( )函数随机生成参数；键盘控制时，按下按键修改对应旋转变换参数、平移变换参数，实现三维物体的移动；设置一个自动旋转函数，实现每次角度参数变化。

第三章 关键算法介绍

3.1 纹理映射

1、原理介绍

在传统的几何物体构造技术中，通常只能简单的表示景物的外部形状，而对于物体表面所具有的微观细节则无法有效的表示，真实感图形绘制技术利用纹理图像来描述景物表面各点处的反射属性，成功模拟出景物表面的丰富纹理细节。在纹理映射技术中，我们将纹理图像作为信息输入，通过在纹理和物体之间建立相应的映射关系，从而可以在简单的三维几何物体上映射图像，用以在物体上产生具有真实感的图案、纹理及花纹。纹理映射可分为两类:一-类用于改变物体表:面的图案和颜色;--类用来改变物体表面的几何属性(如利用法向量扰动以产生凹凸不平的效果)。利用纹理映射可以在不增加场景描述复杂度，不显著增加计算量的前提下，大幅度地提高图形的真实感。在真实感绘制中，二维纹理是最常用的纹理。纹理映射把二维纹理变换到三维物体的表面，成为计算机显示的最终图像。假设二维纹理函数定义在(u, v)平面上。一般地二维纹理函数的定义域是单位正方，如下式所示的二维纹理的函数:

$$g(u,v)=\begin{cases}0, \lfloor u\times 8, v\times 8\rfloor 为偶数\\ 0，\ \lfloor u\times 8, v\times 8\rfloor 为奇数\end{cases}（0\le u\le 1, 0\le v\le 1）$$

$\lfloor x\rfloor$表示不大于 x 的最大整数。

2、程序实现步骤

1）准备纹理，这里准备的是棋盘纹理，结果存储在纹理数组中；

2）确定纹理映射的对象；

3）将制作好的纹理通过 glTexImage2D 等等函数映射到指定对象上。

其变换过程可如下图所示：

图例 纹理映射中纹理空间、物体空间和像素空间的变换

3.2 光照模型

当光照射到一个物体表面上时，会出现三种情形。首先，光可以通过物体表面向空间反射，产生反射光。其次，对于透明体，光可以穿透该物体并从另一端射出，产生透射光。最后，部分光将被物体表面吸收而转换成热。在上述三部分

光中，仅仅是透射光和反射光能够进入人眼产生视觉效果。这里介绍的简单光照模型只考虑被照明物体表面的反射光影响，假定物体表面光滑不透明且由理想材料构成，环境假设为由白光照明。

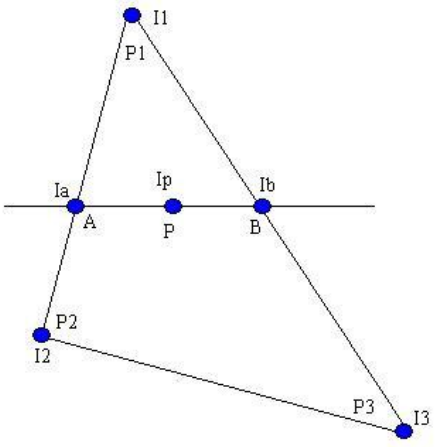
一般来说，反射光可以分成三个分量， 即环境反射、漫反射和镜面反射 。环境反射分量假定入射光均匀地从周围环境入射至景物表面并等量地向各个方向反射出去，通常物体表面还会受到从周围环境来的反射光（如来自地面、天空、墙壁等的反射光）的照射，这些光常统称为环境光（ Ambient Light ）；漫反射分量表示特定光源在景物表面的反射光中那些向空间各方向均匀反射出去的光，这些光常称为漫射光（ Diffuse Light ）；镜面反射光为朝一定方向的反射光，如一个点光源照射一个金属球时会在球面上形成一块特别亮的区域，呈现所谓“高光（ Highlight ）”，它是光源在金属球面上产生的镜面反射光（ Specular Light ）。对于较光滑物体，其镜面反射光的高光区域小而亮；相反，粗糙表面的镜面反射光呈发散状态，其高光区域大而不亮。

简单光照计算公式如下，只考虑环境光、漫射光、镜面反射光：

$$I = I_e + I_d + I_s$$
$$= I_a K_a + I_p [K_d (L \cdot N) + K_s (V \cdot R)^n]$$

3.3 明暗处理算法

Gouraud 明暗处理通常算法为：先用多边形顶点的光强线性插值出当前扫描线与多边形边交点处的光强，然后再用交点的光强线插值处扫描线位于多边形内区段上每一象素处的光强值。图中显示出一条扫描线与多边形相交，交线的端点是 A 点和 B 点，P 点是扫描线上位于多边形内的任一点，多边形三个顶点的光强分别为 I1、I2 和 I3.取 A 点的光强 Ia 为 I1 和 I2 的线性插值，B 点的光强 Ib 为 I1 和 I3 的线性插值，P 点的光强 Ip 则为 Ia 和 Ib 的线性插值。采用 Gouraud 明暗处理不但可以使用多边形表示的曲面光强连续，而且计算量很小。这种算法还可以以增量的形式改进，且能用硬件直接实现算法，从而广泛用于计算机实时图形生成。

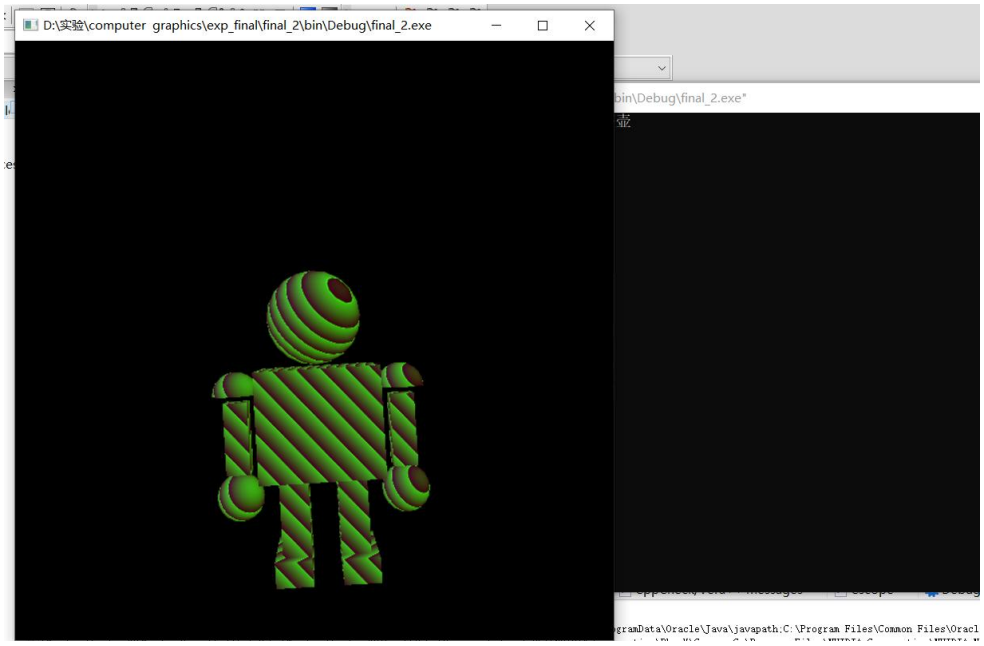


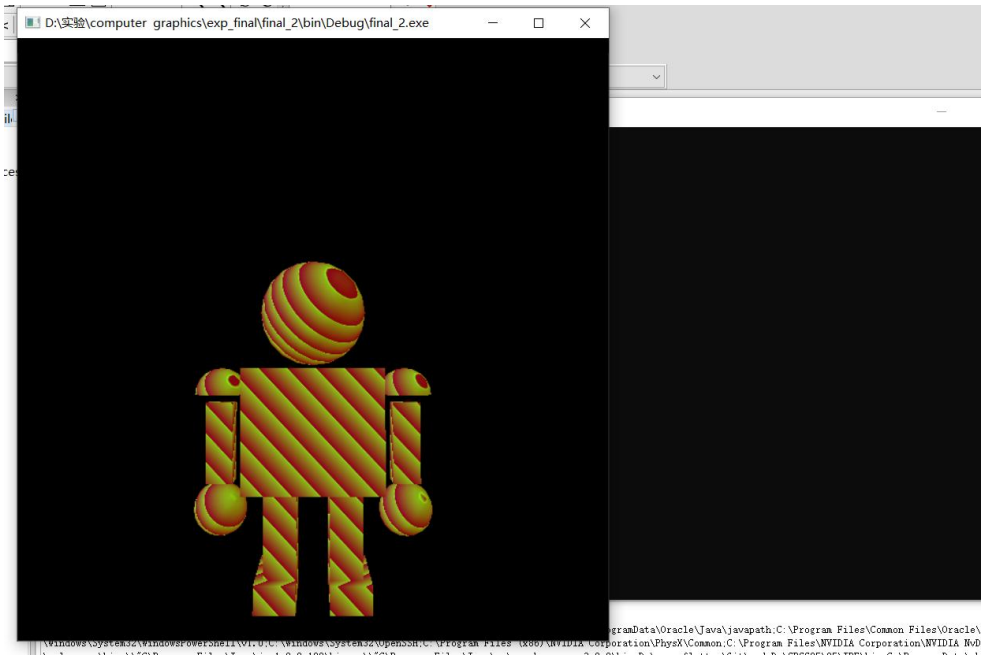
3.4 光线跟踪算法

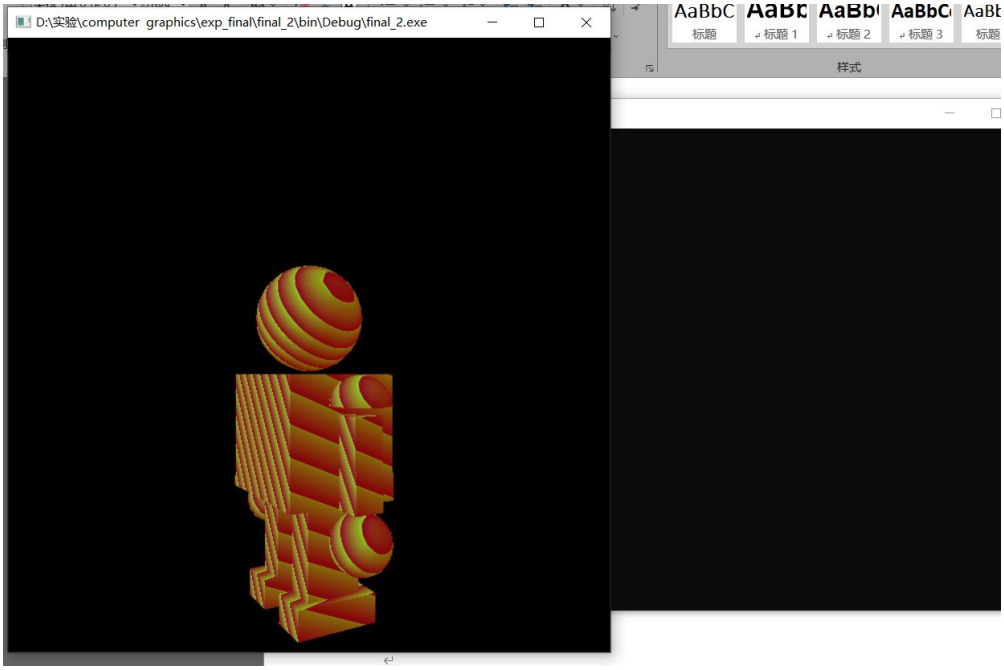
自然界中光线的传播过程：光源->物体表面->物体表面->人眼。光线跟踪是自然界光照明物理过程的近似逆过程，即逆向跟踪从光源发出的光经环境景物间的多次反射、折射后投射到景物表面，最终进入人眼的过程。

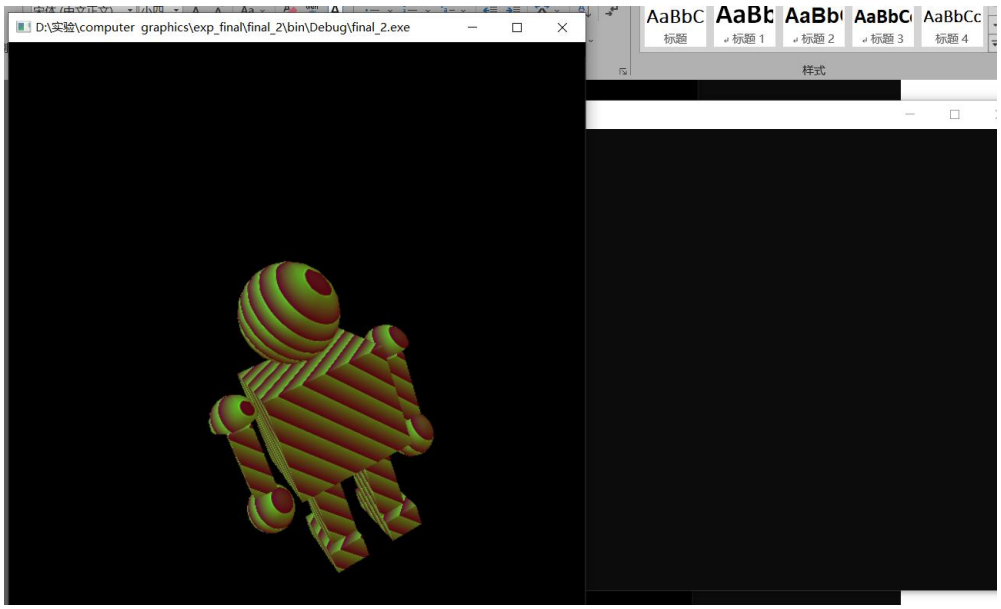
第四章 实验结果

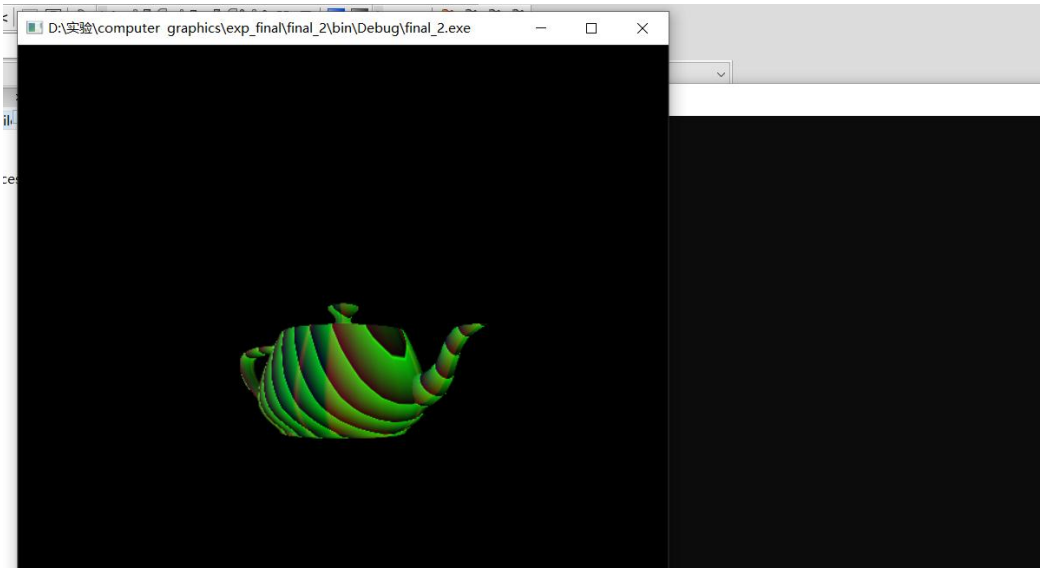
4.1 运行结果截图






D:\实验\computer graphics\exp_final\final_2\bin\Debug\final_2.exe


D:\实验\computer graphics\exp_final\final_2\bin\Debug\final_2.exe



4.2 结果分析

通过运行结果可知，基本的纹理能产生了，鼠标、键盘的交互控制也能够实现了，通过鼠标点击切换环境光也能够实现。但生成的纹理还是不太真实，只是能够生成纹理了，但生成的纹理还是不够自然，对纹理矩阵的生成还有待提升；对镜面反射、环境光、漫反射各个参数值的确定还有待研究，虽然能够产生镜面反射了，但是其产生了光不属于物体，当修改参数后，其产生的光就不太明显，不太能看出其产生了镜面反射。对透明材质和物体阴影产生还没有实现，这部分还需要仔细学习研究。

第五章 总结

真实感图形一直都是计算机图形学的核心内容之一，是最能体现学科魅力的一个分支。对于本实验，通过研究纹理映射和光照模型算法，对真实感图形的可见面颜色进行研究。纹理映射算法是将纹理图像作为信息输入，通过在纹理和物体之间建立相应的映射关系，从而可以在简单的三维几何物体上映射图像。光照明模型即根据光学物理的有关定律，计算景物表面上任一点投向观察者眼中的光亮度的大小和色彩组成的公式。

在本次实验中，我学到了如何生成纹理矩阵，如何将纹理图像存储在三片码数组中，如何启动纹理映射，如何将纹理图案贴到三维物体上，如何启动光照明模型，如何设置环境光、漫射光、镜面反射光的颜色参数，如何设置材质的环境光、漫射光、镜面反射光的反射参数。光源的 RGB 性质与材质的 RGB 性质不同，对于光源，R、G、B 值等于 R、G、B 对其最大强度的百分比，对于材质，R、G、B 值为材质对光的 R、G、B 成分的反射率。而最终到达眼睛的光为材质颜色矩阵与光源颜色矩阵相乘的结果，若有两束光，其最终的颜色就为两束光对应的颜色值相加。

最开始不清楚如何定义纹理图案、也不清楚各个函数的用法以及各个参数的含义，最后通过阅读课本和利用网络搜索学到了纹理图像的生成、纹理映像函数、

光照相关函数、材质相关函数的定义及用法。最终利用这些知识生成了纹理，并将纹理映射到所建立的三维图形上，并实现了图形的变换。

参考文献

[1]赵冬琴,王晓帆.基于光线跟踪算法的真实感图形的生成[J].山西电子技术,2016(06):15-17.
[2]梁梦洁. 基于 OpenGL 的真实感三维海面模拟[D].西安电子科技大学,2013.

附录

```
#include <GL/glut.h>
#include <cstdlib>
#include <cstdio>
#include <cmath>
#define SOLID 1
#define WIRE 2
#define stripeImageWidth 32
#define stripeImageHeight 32

int moveX,moveY;
int spinX = 0;
int spinY = 0;
int des = 0;
char choice;

//纹理矩阵
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)(i * 10 - 3);
			stripeImage[i][j][1] = (GLubyte)(j * 6 - 4);
			stripeImage[i][j][2] = (GLubyte)70;
			stripeImage[i][j][3] = (GLubyte)255;
		}
	}
}
static GLfloat xequalzero[] = { 1.0, 1.0, 1.0, 0.0 };
static GLfloat slanted[] = { 0.0, 1.0, 0.0, 0.0 };
```

```
static GLfloat *currentCoeff;
static float roangles = 60.0f;
void init(void)
{
    //纹理渲染
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glEnable(GL_DEPTH_TEST);//启动深度测试
    glShadeModel(GL_SMOOTH);//明暗处理
    makeStripeImage();
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    //纹理过滤
    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;
    //纹理生成
    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
    glTexGenfv(GL_S, GL_EYE_PLANE, currentCoeff);
    glEnable(GL_TEXTURE_GEN_S);  //启动贴图
    glEnable(GL_TEXTURE_2D);//启动纹理映射
    glEnable(GL_LIGHTING);  //启动光照
    glEnable(GL_LIGHT0);    //启动 0 号光源
    glEnable(GL_AUTO_NORMAL);  //把光反射到各个方向
    glEnable(GL_NORMALIZE);  //在转换之后和光照之前将法线向量标准化成单位长度

    //材质
    GLfloat mat_ambient[] = { 0.8, 0.8, 0.2, 1.0 };
    GLfloat mat_diffuse[] = { 0.1, 0.8, 0.5, 1.0 };
    GLfloat mat_specular[] = { 0.0, 0.0, 0.0, 1.0 };
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialf(GL_FRONT, GL_SHININESS, 100.0);

    //定义光源的颜色和位置
   GLfloat ambient[] = { 0.0, 0.8, 0.1, 0.1 };
   GLfloat diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat specular[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat position[] = { -80.0, 50.0, 25.0, 1.0 };
   //选择光照模型
```

```
    GLfloat lmodel_ambient[] = { 0.4, 0.4, 0.4, 1.0 };
    GLfloat local_view[] = { 50.0 };
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_SMOOTH);
    //设置环境光
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
    //设置漫射光
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
    //镜面光
    glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
    //设置光源位置
    glLightfv(GL_LIGHT0, GL_POSITION, position);
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
    glLightModelfv(GL_LIGHT_MODEL_LOCAL_VIEWER, local_view);
    //启动光照
    glEnable(GL_LIGHTING);
    //启用光源
    glEnable(GL_LIGHT0);
}
//画球
void drawBall(double R, double x, double y,double z, int MODE) {
      glPushMatrix();
      glTranslated(x,y,z);
      if (MODE == SOLID) {
            glutSolidSphere(R,20,20);   //实心图形
      } else if (MODE ==WIRE) {
            glutWireSphere(R,20,20);     //线框图
      }
      glPopMatrix();
}
//画半球
void drawHalfBall(double R, double x, double y,double z, int MODE) {
      glPushMatrix();
      glTranslated(x,y,z);
      GLdouble eqn[4]={0.0, 1.0, 0.0, 0.0};
      glClipPlane(GL_CLIP_PLANE0,eqn);
      glEnable(GL_CLIP_PLANE0);
      if (MODE == SOLID) {
            glutSolidSphere(R,20,20);   //实心图形
      } else if (MODE ==WIRE) {
            glutWireSphere(R,20,20);   //线框图
      }
      glDisable(GL_CLIP_PLANE0);
      glPopMatrix();
```

```
}
//画长方体
void drawSkewed(double l, double w, double h, double x, double y, double z, int MODE) {
	glPushMatrix();
	glScaled(l, w, h);
	glTranslated(x, y, z);
	if (MODE == SOLID) {
		glutSolidCube(1);   //实心图形
	} else if (MODE ==WIRE) {
		glutWireCube(1);    //线框图
	}
	glPopMatrix();
}
//机器人
void display_roboto(void) {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	//圆点放坐标中心
	glLoadIdentity();
	gluLookAt(-2.0, -1.0, 20.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);//定义视角位置
	glPushMatrix();
	glRotated(spinX, 0, 1, 0);
	glRotated(spinY, 1, 0, 0);
	glRotatef(roangles, 0.0, 1.0, 0.0);
	glTranslated(0, 0, des);

	//头
	drawBall(2, 0, 1, 0, SOLID);
	//身体
	drawSkewed(5, 4.4, 4, 0, -0.75, 0, SOLID);
	//肩膀
	drawHalfBall(1, 3.5, -2.1, 0, SOLID);
	drawHalfBall(1, -3.5, -2.1, 0, SOLID);
	//胳膊
	drawSkewed(1, 3, 1, 3.5, -1.3, 0, SOLID);
	drawSkewed(1, 3, 1, -3.5, -1.3, 0, SOLID);
	//手
	drawBall(1, 3.5, -6.4, 0, SOLID);
	drawBall(1, -3.5, -6.4, 0, SOLID);
	//腿
	drawSkewed(1.2, 3, 2, 1, -2.4, 0, SOLID);
	drawSkewed(1.2, 3, 2, -1, -2.4, 0, SOLID);
	//脚
	drawSkewed(1.5, 1, 3, 0.9, -9.2, 0, SOLID);
	drawSkewed(1.5, 1, 3, -0.9, -9.2, 0, SOLID);
```

```
	glPopMatrix();
	glFlush();
}
//茶壶
void display_teapot(void)
{
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	//圆点放坐标中心
	glLoadIdentity();
	gluLookAt(-2.0, -1.0, 20.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);//定义视角位置
	glPushMatrix();
	glRotated(spinX, 0, 1, 0);
	glRotated(spinY, 1, 0, 0);
	glRotatef(roangles, 0.0, 1.0, 0.0);
	glTranslated(0, 0, des);
	glutSolidTeapot(3);	//茶壶

	 glPopMatrix();
	glFlush();
}
//鼠标点击事件，设置光照
void mouseClick(int btn, int state, int x, int y) {
	moveX = x;
	moveY = y;
	GLfloat ambient[] = { (float)rand() / RAND_MAX, (float)rand() / RAND_MAX, (float)rand()
/ RAND_MAX, 0.1 };
	 //设置环境光
	glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
	//启用光源
	glEnable(GL_LIGHT0);
}
//键盘事件，移动、旋转物体
void keyPressed(unsigned char key, int x, int y) {
	switch (key) {
	case 'A':
		spinX -= 2;
		break;
	case 'D':
		 spinX += 2;
		break;
	case 'W':
		des += 2;
		break;
```

```
        case 'S':
            des -= 2;
            break;
        }
        glutPostRedisplay();
}
// 鼠标移动事件，点击鼠标不放可移动物体
void mouseMove(int x, int y) {
        int dx = x - moveX;
        int dy = y - moveY;
        spinX += dx;
        spinY += dy;
        glutPostRedisplay();
        moveX = x;
        moveY = y;
}
//窗口重定义函数
void reshape(int w, int h) {
        //定义视口大小
        glViewport(0, 0, (GLsizei)w, (GLsizei)h);
        //投影显示
        glMatrixMode(GL_PROJECTION);
        //坐标原点在屏幕中心
        glLoadIdentity();
        //操作模型视角
        gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 40.0);
        glMatrixMode(GL_MODELVIEW);
}
//自动旋转函数
void idle()
{
        roangles += 0.01f;
        glutPostRedisplay();
}
int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(600, 600);
        glutInitWindowPosition(100, 100);
        glutCreateWindow(argv[0]);
        init();
        do{
        printf("请选择需要查看的图形：1、机器人 2、茶壶 \n");
        scanf("%c",&choice);
```

```
    getchar();
    if(choice=='1'){
        glutDisplayFunc(display_roboto);
        glutReshapeFunc(reshape);
        //鼠标点击事件，鼠标点击或者松开时调用
        glutMouseFunc(mouseClick);
        //鼠标移动事件，鼠标按下并移动时调用
        glutMotionFunc(mouseMove);
        //键盘事件
        glutKeyboardFunc(keyPressed);
    }else if(choice=='2'){
        glutDisplayFunc(display_teapot);
        //自动旋转
        glutIdleFunc(idle);
        //窗口重定义
        glutReshapeFunc(reshape);
    }else{
    printf("输入有误\n");
    }
    }while(choice != '1' && choice != '2');
    glutMainLoop();
    return 0;
}
```