**《计算机图形学实验》综合实验报告**

**题目OpenGL实现三维图形的真实感绘制**

**学 号 20201120454**

**姓 名 曾文**

**指导教师 钱文华**

**日 期 2022年6月15日**

摘要：计算机图形学(Computer Graphics，简称CG)是一种使用数学算法将[二维](https://baike.baidu.com/item/%E4%BA%8C%E7%BB%B4/380405" \t "https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%9B%BE%E5%BD%A2%E5%AD%A6/_blank)或[三维图形](https://baike.baidu.com/item/%E4%B8%89%E7%BB%B4%E5%9B%BE%E5%BD%A2/5612976" \t "https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%9B%BE%E5%BD%A2%E5%AD%A6/_blank)转化为计算机显示器的[栅格](https://baike.baidu.com/item/%E6%A0%85%E6%A0%BC/7368256" \t "https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%9B%BE%E5%BD%A2%E5%AD%A6/_blank)形式的科学。简单地说，计算机图形学的主要研究内容就是研究如何在计算机中表示图形、以及利用计算机进行图形的计算、处理和显示的相关[原理](https://baike.baidu.com/item/%E5%8E%9F%E7%90%86/85014" \t "https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%9B%BE%E5%BD%A2%E5%AD%A6/_blank)与[算法](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95/209025" \t "https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%9B%BE%E5%BD%A2%E5%AD%A6/_blank)。经过一个学期的学习，这次综合试验进行的是真实感图形的绘制。要运用到光照、纹理等知识，对三维图形进行渲染。

关键词：OpenGL,三维图形渲染，纹理映射，光照，真实感

目录

**[一、实验背景及实验内容 6](#_Toc7183)**

[（一）实验背景 6](#_Toc5795)

[（二）实验内容 6](#_Toc3741)

**[二、开发工具，程序设计及实现目的及基本模块介绍 7](#_Toc18947)**

[（一） 开发工具 7](#_Toc24195)

[（二） 程序设计 7](#_Toc31451)

[（三） 实现目的 7](#_Toc20093)

[（四） 基本模块介绍 7](#_Toc30476)

**[三、关键算法的理论介绍和程序实现步骤 7](#_Toc31403)**

**[四、 实验结果 9](#_Toc215)**

[（一） 运行截图 9](#_Toc24685)

[(二) 结果分析 14](#_Toc15310)

**[五、实验体会及小结 14](#_Toc2544)**

**[六、参考文献 15](#_Toc21163)**

**[七、附录 15](#_Toc9026)**

一、实验背景及实验内容

（一）实验背景

计算机图形学的核心目标在于创建有效的视觉交流。在科学领域，图形学可以将科学成果通过可视化的方式展示给公众；在娱乐领域，如在PC游戏、[手机游戏](https://baike.baidu.com/item/%E6%89%8B%E6%9C%BA%E6%B8%B8%E6%88%8F/1379" \t "https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%9B%BE%E5%BD%A2%E5%AD%A6/_blank)、[3D电影](https://baike.baidu.com/item/3D%E7%94%B5%E5%BD%B1" \t "https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%9B%BE%E5%BD%A2%E5%AD%A6/_blank)与电影特效中，计算机图形学发挥着越来越重要的作用；在创意或艺术创作、商业广告、产品设计等行业，图形学也起着重要的基础作用。而在科学领域中，这一点是在1987年关于科学计算可视报告中才被重点提出。该报告引用了Richard Hamming在1962年的经典论断：“计算的目的是洞察事物的本质，而不是获得数字。”报告中提到了计算机图形学在帮助人脑从图形图像的角度理解事物本质的重要作用，因为图形图像比单纯数字具有更强的洞察力。计算机图形学核心目标(视觉交流)可以分解为三个基本任务：表示、交互、绘制，即如何在计算机中“交互”地“表示”、“绘制”出丰富多彩的主、客观世界。这里的“表示”是如何将主、客观世界放到计算机中去——二维、三维对象的表示与建模；而“绘制”是指如何将计算机中的对象用一种直观形象的图形图像方式表现出来——二维、三维对象的绘制：“交互”是指通过计算机输入、输出设备，以有效的方式实现“表示”与“绘制”的技术。其中，“表示”是计算机图形学的“数据层”，是物体或对象在计算机中的各种几何表示；“绘制”是计算机图形学的“视图层”，指将图形学的数据显示、展现出来。“表示”是建模、输入，“绘制”是显示、输出。“交互”是计算机图形学的“控制层”，它负责完成有效的对象输入与输出任务，解决与用户的交互问题。经过一个学期的学习之后，本次实验使用opengl进行真实感图形的绘制。

（二）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。

二、开发工具，程序设计及实现目的及基本模块介绍

1. 开发工具

开发工具为OpenGL,Microsoft Visual Stdio 2022 社区版，采用的开发语言为C++.

1. 程序设计

本程序拥有13个函数（包含main函数），大多数操作由OpenGL库函数实现。用户可通过键盘鼠标对程序进行控制，无需输入数据。

1. 实现目的

绘制一个经典的OpenGL犹他茶壶，用户可以对其进行旋转、纹理填充、不同颜色照射等操作。

1. 基本模块介绍

纹理模块：包括纹理图样定义函数makeStripeImage（）和纹理映射到图形的函数texture()。

光源、材质、真实感等初始化模块：主要由函数init（）和setupRederingState（）实现。

坐标系旋转模块：主要由函数translate()实现。

绘图模块：由函数display（）实现。

窗口变更模块：由函数changeSize（）实现。

用户菜单模块：由函数menuFun()和setMenu()实现。

键鼠交互控制模块：由函数keyFunc()、mouseFunc()和mouseMove()实现。

三、关键算法的理论介绍和程序实现步骤

1. 理论介绍

纹理映射算法：纹理映射（Texture Mapping）是将纹理空间中的纹理像素映射到屏幕空间中的像素的过程。

在三维图形中，纹理映射（Texture Mapping）的方法运用得最广，尤其描述具有真实感的物体。比如绘制一面[砖墙](https://baike.baidu.com/item/%E7%A0%96%E5%A2%99/8853276" \t "https://baike.baidu.com/item/%E7%BA%B9%E7%90%86%E6%98%A0%E5%B0%84/_blank)，就可以使用一幅具有真实感的图像或者照片作为纹理贴到一个矩形上，这样，一面逼真的砖墙就画好了。如果不用纹理映射的方法，这墙上的每一块砖都要作为一个独立的多边形来绘制。另外，纹理映射能够保证在变换多边形时，多边形上的纹理也会随之变化。例如，用[透视投影](https://baike.baidu.com/item/%E9%80%8F%E8%A7%86%E6%8A%95%E5%BD%B1" \t "https://baike.baidu.com/item/%E7%BA%B9%E7%90%86%E6%98%A0%E5%B0%84/_blank)模式观察墙面时，离视点远的墙壁的砖块的尺寸就会缩小，而离视点近的就会大些，这些是符合视觉规律的。此外，纹理映射也被用在其他一些领域。如飞行仿真中常把一大片植被的[图像映射](https://baike.baidu.com/item/%E5%9B%BE%E5%83%8F%E6%98%A0%E5%B0%84/11027539" \t "https://baike.baidu.com/item/%E7%BA%B9%E7%90%86%E6%98%A0%E5%B0%84/_blank)到一些大多边形上用以表示地面，或者用大理石、木材等自然物质的图像作为纹理映射到多边形上表示相应的物体。纹理对象通过一个单独的数字来标识。这允许[OpenGL](https://baike.baidu.com/item/OpenGL" \t "https://baike.baidu.com/item/%E7%BA%B9%E7%90%86%E6%98%A0%E5%B0%84/_blank)硬件能够在内存中保存多个纹理，而不是每次使用的时候再加载它们，从而减少了运算量，提高了速度。

纹理映射是真实感图像制作的一个重要部分，运用它可以方便的制作出极具真实感的图形而不必花过多时间来考虑物体的表面细节。然而纹理加载的过程可能会影响程序运行速度，当纹理图像非常大时，这种情况尤为明显。如何妥善的管理纹理，减少不必要的开销，是[系统优化](https://baike.baidu.com/item/%E7%B3%BB%E7%BB%9F%E4%BC%98%E5%8C%96/10072100" \t "https://baike.baidu.com/item/%E7%BA%B9%E7%90%86%E6%98%A0%E5%B0%84/_blank)时必须考虑的一个问题。还好，OpenGL提供了纹理对象对象管理技术来解决上述问题。与[显示列表](https://baike.baidu.com/item/%E6%98%BE%E7%A4%BA%E5%88%97%E8%A1%A8" \t "https://baike.baidu.com/item/%E7%BA%B9%E7%90%86%E6%98%A0%E5%B0%84/_blank)一样，纹理对象通过一个单独的数字来标识。这允许OpenGL硬件能够在内存中保存多个纹理，而不是每次使用的时候再加载它们，从而减少了运算量，提高了速度。

1. 程序实现步骤

第一步:定义纹理对象

#define imageWidth 50

void makeStripeImage(void)

{

int j;

for (j = 0; j < imageWidth; j++)

{

stripeImage[3 \* j] =255;

stripeImage[3 \* j + 1] = 255;

stripeImage[3 \* j + 2] = 255;

}

}

第二步：选择纹理对象，完成该纹理对象的定义。

glTexImage1D(GL\_TEXTURE\_1D, 0, 3, imageWidth, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, stripeImage);

glTexGeni(GL\_S, GL\_TEXTURE\_GEN\_MODE, GL\_OBJECT\_LINEAR);

glTexGenfv(GL\_S, GL\_OBJECT\_PLANE, sgenparams);

第三步:为图形加载相应的纹理。

glEnable(GL\_TEXTURE\_GEN\_S);

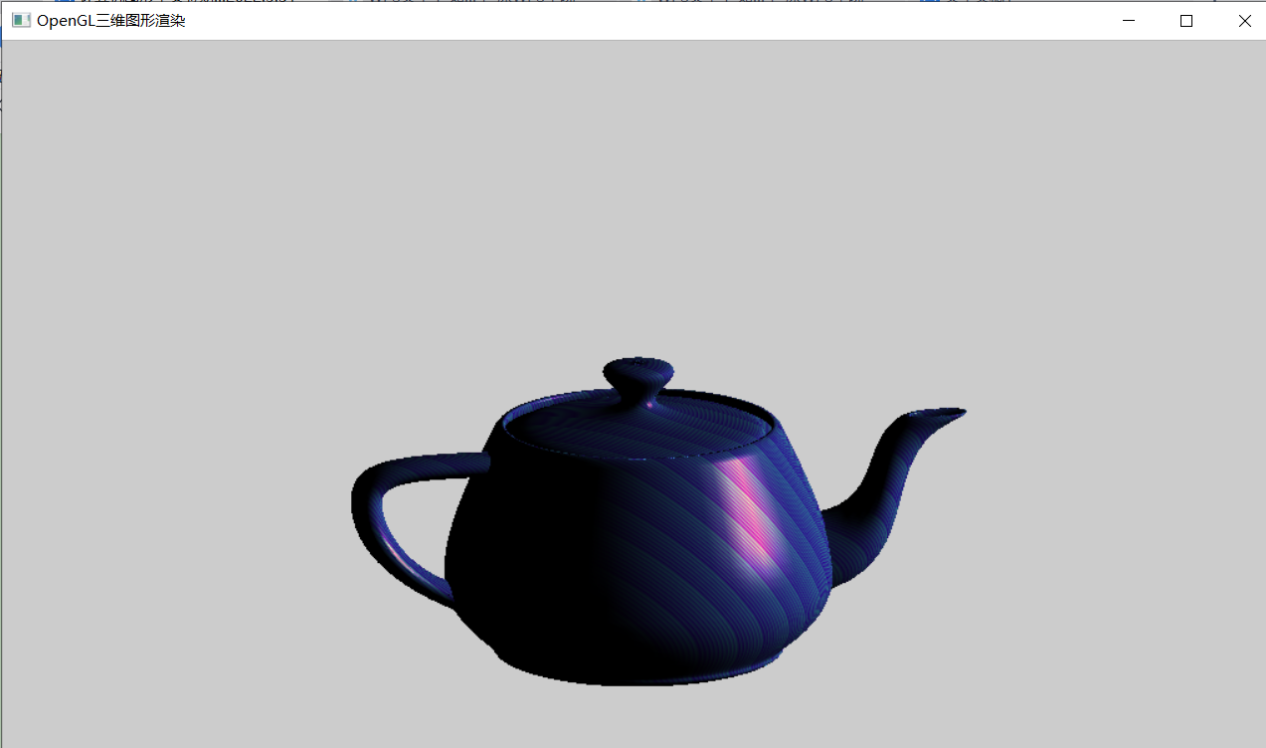
glEnable(GL\_TEXTURE\_1D);

1. 实验结果
2. 运行截图

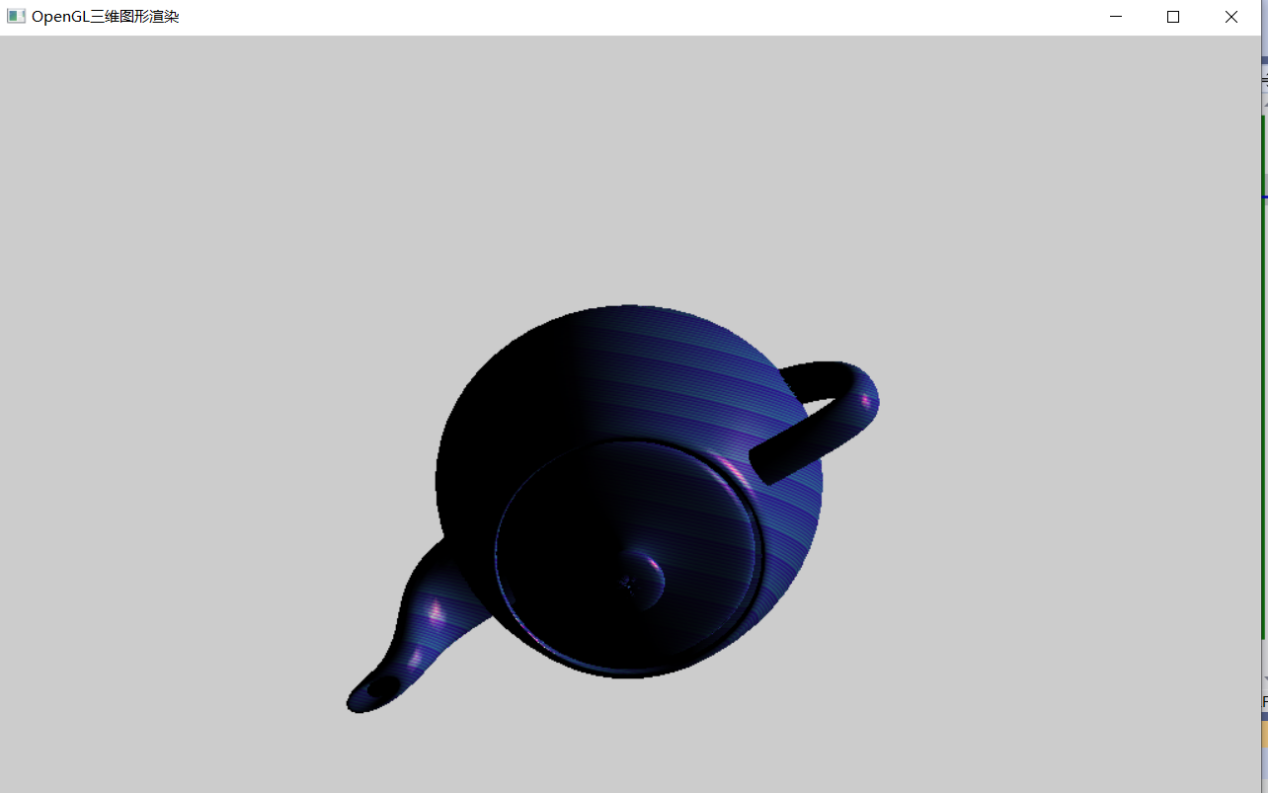
线框图：



实体图：



旋转：



为了让光照效果更加明显，我更改了纹理颜色：



光源向下移动：



光源向上移动：



光源向外移动：



光源向内移动：



环境光调为品红色：



1. 结果分析

从实验的运行结果来说，程序能正常运行，也能得到较好的预期效果。但是由于我没有找到一个较好的纹理配色，导致光照效果看起来不太明显。由于窗口参考系的原因，菜单中选择光源向下移动后，其实在我们看来光源是向上移动的，茶壶的顶部被照亮。程序初始的运行结果为一个茶壶线框图，点击鼠标右键，就会跳出选项菜单，用户可以自由的对光照颜色，光源位置进行选择。旋转方面，用户可以通过鼠标左键来拖动旋转，也可以用键盘上的方向键来控制旋转。

五、实验体会及小结

这次实验的综合性很强，在绘制图形的基础上，需要对其进行纹理映射，光源照射，总的来说，就是让我们绘制出的图形尽可能的做到真实。

在这次实验中，我还是遇到了不少的困难。大体的设计方面，我想用教材上的例子绘制可以旋转的茶壶，再对其进行相应的处理。但是只能做到输入角度进行旋转。这当然不够好，我自然的想起将键鼠交互函数放进来，以达到交互控制的目的。这个过程还是有些许困难，毕竟参数，对象等都已经改变，还要增加新的功能。还有一个问题就是这次实验的实现功能比较多，我最初的想法是直接将渲染过后的茶壶显示给用户。转念一想，这样的话用户不能直观的感受到图形由简单的线条变为接近真实的过程。于是，我决定让用户自己来选择是否进行纹理映射，是否光照，光源放在哪。这个部分我本来打算做到键盘控制函数中，让用户按下相应的键位来执行相应的操作。可是，那样不够直观。这时，我想到了Java中学过的GUI界面。如果能用鼠标右键呼出菜单，再进行相应操作，就会显得既简单又直观。在网上查阅了相关资料后，得知OpenGL是有相关的菜单函数的，只要配合上switch-case语句，就能制作出简易的菜单了。这些菜单函数也是我的新收获吧。

总而言之，这学期的计算机图形学课程让我学会了很多。我知道了我电脑上的简单壁纸也是十分珍贵的。电子产品呈现给我们的图像，都是无数学者优化算法，减小误差才得到的。硬件不断升级，分辨率不断提高，也离不开这些研究者的大力推动。想想那些好莱坞大片，能用计算机合成那么逼真的场景，那么震撼的特效，真是很了不起。计算机图形学的内容很多，一个学期可能只能让我们入门，想要更加深入，就得再以后的时间里多下功夫。

六、参考文献

[1]谢志鹏,陈锻生.用 OpenGL开发三维交互式虚拟场景.计算机应用，2000.

[2]彭群生,鲍虎军，金小刚.计算机真实感图形的算法基础.北京:科学出版社.

[3]乔林.费广正OpenGL程序设计.北京:人民邮电出版社,2000.

[4]廖朵朵,张华军OpenGL三维图形程序设计.北京:星球地图出版社，2003.

[5]向世明.OpenGL编程与实例:北京:电子工业出版社, 1999: 125-126

[6]向世明.VisualC++数字图像与图形处理.北京:电子工业出版社,2002.

七、附录

#include<GL/glut.h>

#include<stdio.h>

#define imageWidth 50

//系统屏幕大小初始化

GLint SCREEN\_WIDTH = 0;

GLint SCREEN\_HEIGHT = 0;

//窗口大小初始化

GLint window\_width = 1024;

GLint window\_height = 768;

//中心坐标计算

#define MIDWIDTH window\_width/2

#define MIDHEIGHT window\_height/2

//设置模型旋转角度

GLfloat xAngle = 0.0f;

GLfloat yAngle = 0.0f;

//设置模型初始距离

GLfloat distancex;

GLfloat distancey;

//受支持的点大小范围

GLfloat sizes[2];

//受支持的点大小增量

GLfloat step;

//出现的模型是茶壶

GLint flag = 1;

//线框|实体：状态默认为线框

GLint lsr = 0;

GLboolean mouseLeftDown = false; //鼠标左键按下状态，默认是未按下

//当前鼠标位置

GLint mousex;

GLint mousey;

//默认启用光照

GLboolean lightEnable = true;

//定义光源的颜色和位置

GLfloat ambient[] = { 0.0f, 0.0f, 0.0f, 1.0f }; //环境光

GLfloat position[] = { 10.0f, 3.0f, 2.0f, 0.0f }; //光源位置

GLfloat matEmission[] = { 1.0f, 1.0f, 1.0f, 1.0 }; //反射光颜色

GLfloat diffuse[] = { 1.0, 1.0, 1.0, 1.0 }; //漫射光

GLfloat specular[] = { 1.0f, 1.0f, 1.0f, 1.0f }; //镜面反射

GLfloat mat\_ambient[] = { 0.7f, 0.7f, 0.7f, 1.0f }; // 环境颜色

GLfloat mat\_diffuse[] = { 0.2f, 0.5f, 0.8f, 1.0f }; // 散射颜色

GLfloat mat\_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f }; // 镜面反射颜色

GLfloat high\_shininess[] = { 100.0f }; // 镜面反射指数为100.0

GLfloat no\_mat[] = { 0.0f, 0.0f, 0.0f, 1.0f }; // 无材质颜色

float xrot;

GLubyte stripeImage[3 \* imageWidth];

//定义纹理图像

void makeStripeImage(void)

{

int j;

for (j = 0; j < imageWidth; j++)

{

stripeImage[3 \* j] =999;

stripeImage[3 \* j + 1] = 699/4\*j;

stripeImage[3 \* j + 2] = 699;

}

}

/\* 参数设置 \*/

GLfloat sgenparams[] = { 1.0, 1.0, 1.0, 0.0 };

void texture() {

glClearColor(0.0, 0.0, 0.0, 0.0);

glShadeModel(GL\_SMOOTH);//光暗处理

makeStripeImage();//绘制纹理

//函数设定从内存中读取纹理图并放到屏幕上的方式

//指定内存中每个象素行起始的排列要求为字节排列

glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1);

//定义纹理环境参数：调整当前亮度和颜色信息，使之适应纹理图像

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

//纹理绕转使用重复方式

glTexParameterf(GL\_TEXTURE\_1D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

//定义纹理放大和缩小函数均为GL\_LINEAR

glTexParameterf(GL\_TEXTURE\_1D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexParameterf(GL\_TEXTURE\_1D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

//定义纹理

glTexImage1D(GL\_TEXTURE\_1D, 0, 3, imageWidth, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, stripeImage);

//控制纹理坐标的生成

//指定单值纹理生成参数

glTexGeni(GL\_S, GL\_TEXTURE\_GEN\_MODE, GL\_OBJECT\_LINEAR);

//指定纹理坐标生成函数,系数由sgenparams指定

glTexGenfv(GL\_S, GL\_OBJECT\_PLANE, sgenparams);

glEnable(GL\_TEXTURE\_GEN\_S);//开启纹理坐标映射

glEnable(GL\_TEXTURE\_1D);//开启纹理

}

void init() { //光源、材质、真实感初始化

//选择光照模型

GLfloat local\_view[] = { 0.0 };

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);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, diffuse); //设置材料反射指数（漫反射）->确定最终颜色

//设置光源位置

glLightfv(GL\_LIGHT0, GL\_POSITION, position);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambient);

glLightModelfv(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, local\_view);

glLightModelfv(GL\_LIGHT\_MODEL\_TWO\_SIDE, local\_view);

glMaterialfv(GL\_FRONT, GL\_EMISSION, matEmission); // 用来模拟物体发光的效果，但这不是光源

//在OpenGL中，必须明确指出光照是否有效或无效。如果光照无效，则只是简单地将当前颜色映射到当前顶点上去

glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_ambient);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

glMaterialfv(GL\_FRONT, GL\_SHININESS, high\_shininess);

glMaterialfv(GL\_FRONT, GL\_EMISSION, no\_mat);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, mat\_specular);

glEnable(GL\_DEPTH\_TEST); //深度测试

//启动光照

if (lightEnable) {

glEnable(GL\_LIGHTING);

//启用指定光源

glEnable(GL\_LIGHT0);

}

else

glDisable(GL\_LIGHTING);

}

void translate() //位移旋转操作

{

glColor3f(0.0f, 1.0f, 1.0f);

//glScalef(times, times, times);

glTranslatef(distancex, distancey, 0);

//坐标系绕x轴旋转xAngle度

glRotatef(xAngle, 1.0f, 0.0f, 0.0f);

//坐标系绕y轴旋转yAngle度

glRotatef(yAngle, 0.0f, 1.0f, 0.0f);

}

void display() //绘图函数

{

init();

//初始化窗口背景颜色

glClearColor(0.8f, 0.8f, 0.8f, 0.75f);

//将模板缓冲区值全部清理为1

glClearStencil(1);

//开启使用模板测试并且更新模版缓存

glEnable(GL\_STENCIL\_TEST);

//把整个窗口清理为当前清理颜色：黑色。清除深度缓冲区、模板缓冲区

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT | GL\_STENCIL\_BUFFER\_BIT);

//glPushMatrix()和glPopMatrix()的配对使用可以消除上一次的变换对本次变换的影响。

//使本次变换是以世界坐标系的原点为参考点进行

glPushMatrix();

//进行平滑处理

glEnable(GL\_POINT\_SMOOTH); //点 反走样，使平滑

glHint(GL\_POINT\_SMOOTH, GL\_NICEST); //指示平滑程度为最高质量

glEnable(GL\_LINE\_SMOOTH); //线反走样

glHint(GL\_LINE\_SMOOTH, GL\_NICEST);

glEnable(GL\_POLYGON\_SMOOTH); //多边形反走样

glHint(GL\_POLYGON\_SMOOTH, GL\_NICEST);

switch (flag) {

case 1:

translate();

if (lsr == 0) {

glutWireTeapot(6.5f);

}

else if (lsr == 1) {

glutSolidTeapot(6.5f);

}

break;

}

//恢复压入栈的Matrix

glPopMatrix();

//交换两个缓冲区的指针

glutSwapBuffers();

}

void setupRederingState(void) { //渲染初始化

//设置清理颜色为黑色

glClearColor(0.0f, 0.0, 0.0, 1.0f);

//设置绘画颜色为绿色

glColor3f(1.0f, 1.0f, 1.0f);

//使能深度测试

glEnable(GL\_DEPTH\_TEST);

}

void changeSize(GLint width, GLint height) //窗口改变尺寸时调用

{

//横宽比率

GLfloat ratio;

//设置坐标系为x(-100.0f,100.0f)、y(-100.0f,100.0f)、z(-100.0f,100.0f)

GLfloat coordinatesize = 16.0f;

//窗口宽高为零直接返回

if ((width == 0) || (height == 0))

return;

//设置视口和窗口大小一致

glViewport(0, 0, width, height);

//对投影矩阵应用随后的矩阵操作

glMatrixMode(GL\_PROJECTION);

//重置当前指定的矩阵为单位矩阵

glLoadIdentity();

ratio = (GLfloat)width / (GLfloat)height;

//正交投影

if (width < height)

glOrtho(-coordinatesize, coordinatesize, -coordinatesize / ratio, coordinatesize / ratio, -coordinatesize, coordinatesize);

else

glOrtho(-coordinatesize \* ratio, coordinatesize \* ratio, -coordinatesize, coordinatesize, -coordinatesize, coordinatesize);

//对模型视图矩阵堆栈应用随后的矩阵操作

glMatrixMode(GL\_MODELVIEW);

//重置当前指定的矩阵为单位矩阵

glLoadIdentity();

}

void menuFunc(int key) //菜单回调函数

{

switch (key) {

case 1:

lsr = 0;

break;

case 2:

lsr = 1;

break;

case 3:

ambient[0] = 1.0f; ambient[1] = 0.0f; ambient[2] = 0.0f;

break;

case 4:

ambient[0] = 1.0f; ambient[1] = 0.0f; ambient[2] = 1.0f;

break;

case 5:

ambient[0] = 0.0f; ambient[1] = 1.0f; ambient[2] = 0.0f;

break;

case 6:

ambient[0] = 0.0f; ambient[1] = 0.0f; ambient[2] = 1.0f;

break;

case 7:

ambient[0] = 1.0f; ambient[1] = 1.0f; ambient[2] = 1.0f;

break;

case 8:

ambient[0] = 0.0f; ambient[1] = 0.0f; ambient[2] = 0.0f;

break;

case 9:

lightEnable = true;

break;

case 10:

lightEnable = false;

break;

case 11:

position[0] -= 10.0;

break;

case 12:

position[0] += 10.0;

break;

case 13:

position[2] += 10.0;

break;

case 14:

position[2] -= 10.0;

break;

}

glutPostRedisplay();

}

void setMenu() //设置菜单

{

int slrMenu;

int mainMenu;

int colorMenu;

int lightMenu;

int lightPositionMenu;

slrMenu = glutCreateMenu(menuFunc);

glutAddMenuEntry("线框", 1);

glutAddMenuEntry("实体", 2);

colorMenu = glutCreateMenu(menuFunc);

glutAddMenuEntry("红色", 3);

glutAddMenuEntry("品红色", 4);

glutAddMenuEntry("绿色", 5);

glutAddMenuEntry("蓝色", 6);

glutAddMenuEntry("白色", 7);

glutAddMenuEntry("黑色", 8);

lightMenu = glutCreateMenu(menuFunc);

glutAddMenuEntry("启用", 9);

glutAddMenuEntry("关闭", 10);

lightPositionMenu = glutCreateMenu(menuFunc);

glutAddMenuEntry("光源向下移动", 11);

glutAddMenuEntry("光源向上移动", 12);

glutAddMenuEntry("光源向外移动", 13);

glutAddMenuEntry("光源向内移动", 14);

mainMenu = glutCreateMenu(menuFunc);

glutAddSubMenu("线框|实体", slrMenu);

glutAddSubMenu("环境光颜色选择", colorMenu);

glutAddSubMenu("启用光照", lightMenu);

glutAddSubMenu("光照位置", lightPositionMenu);

//将菜单链接到鼠标右键上

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}

void keyFunc(int key, int x, int y) //键盘控制

{

//旋转

if (key == GLUT\_KEY\_UP)

xAngle -= 5.0f; //绕x轴顺时针转

else if (key == GLUT\_KEY\_DOWN)

xAngle += 5.0f; //绕x轴逆时针转

else if (key == GLUT\_KEY\_LEFT)

yAngle -= 5.0f; //绕y轴顺时针转

else if (key == GLUT\_KEY\_RIGHT)

yAngle += 5.0f; //绕y轴逆时针转

glutPostRedisplay();

}

void mouseFunc(int key, int state, int x, int y) //鼠标点击事件

{

if (key == 0)

if (state == 0)

mouseLeftDown = true;

else if (state == GLUT\_UP)

mouseLeftDown = false;

}

void mouseMove(int x, int y) //鼠标拖动则在相应位置进行旋转

{

mousex = x;

mousey = y;

if ((x > 0 && x < MIDWIDTH / 2) && (y > MIDHEIGHT / 2 && y < MIDHEIGHT \* 3 / 2))

yAngle -= 1.0f; //向上转

if ((x<window\_width && x>MIDWIDTH \* 3 / 2) && (y > MIDHEIGHT / 2 && y < MIDHEIGHT \* 3 / 2))

yAngle += 1.0f; //向下转

if ((y > 0 && y < MIDHEIGHT / 2) && (x > MIDWIDTH / 2 && x < MIDWIDTH \* 3 / 2))

xAngle -= 1.0f; //向左转

if ((x > MIDWIDTH / 2 && x < MIDWIDTH \* 3 / 2) && (y<window\_height && y > MIDHEIGHT \* 3 / 2))

xAngle += 1.0f; //向右转

glutPostRedisplay();

}

int main(int argc, char\* argv[])

{

glutInit(&argc, argv);//完成系统初始化工作

//设置初始显示模式：使用 双缓冲区 、RGBA颜色模式、 深缓冲区 、 模板缓冲区

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_DEPTH | GLUT\_STENCIL);

//设置屏幕大小为系统获取的宽度和高度

SCREEN\_WIDTH = glutGet(GLUT\_SCREEN\_WIDTH);

SCREEN\_HEIGHT = glutGet(GLUT\_SCREEN\_HEIGHT);

//设置窗口名称

glutCreateWindow("OpenGL三维图形渲染");

//设置窗口的大小

glutReshapeWindow(window\_width, window\_height);

//窗口居中显示

glutPositionWindow((SCREEN\_WIDTH - window\_width) / 2, (SCREEN\_HEIGHT - window\_height) / 2);

//设置菜单操作

setMenu();

//设置窗口大小改变后执行的函数

glutReshapeFunc(changeSize);

//显示回调函数

glutDisplayFunc(display);

//键鼠函数

glutSpecialFunc(keyFunc);

glutMouseFunc(mouseFunc);

glutMotionFunc(mouseMove);

//设置渲染参数

setupRederingState();

//纹理映射

texture();

//设置主循环

glutMainLoop();

}