序　号：13

教学上机实验报告

课 程 名 称：计 算 机 图 形 学

任 课 教 师：徐 文 鹏

学　　号：311809000608

姓　　名：王荣胜

专 业 班 级：计实验 1801 班

2020 ～ 2021 学年第 1 学期

河南理工大学

教学上机实验报告评价分值标准

序号	评价指标	分值	评价等级及参考分值					评价分
			优	良	中	合格	差	
1	实验报告书写规范、字迹工整，内容完整充实	20	20	17	15	13	6	
2	实验过程叙述详细、概念正确，语言表达准确，结构严谨，条理清楚，逻辑性强，自己努力完成，没有抄袭	30	30	26	23	20	10	
3	对实验过程中存在的问题分析详细透彻、全面、规范，结合实验内容，有自己的个人见解和想法，给出解决办法	30	30	26	23	20	10	
4	实验结果、分析和结论正确	20	20	17	15	13	6	
总得分								

签名（盖章）：

日期：　　年 月 日

说明：

学生有应按照要求按时参加上机，完成教师布置的上机任务。学生每次上机结束后，应该及时按照实验要求和上机操作情况认真填写上机实验报告，并在课程结束后以班级为单位按照学号从小到大顺序排序后上交给任课教师。

河南理工大学上机实验报告

学年：_2020-2021__　　第_1_学期　　上机时间__2020.11.25

专业班级__计实验 1801 班__　　学号_311809000608_　　姓名_王荣胜

课程名称：

计算机图形学

实验项目名称：

直线光栅化

实验目的：

1. 理解基本图形元素光栅化的基本原理；
2. 掌握基本图形元素光栅化方法，如中点方法，Bresenham 方法；
3. 利用 OpenGL 实现基本图形元素的光栅化算法。

实验内容与主要代码：

(1) 根据所给的直线光栅化的示范源程序，在计算机上编译运行，输出正确结果。

(2) 指出示范程序采用的算法，以此为基础将其改造为中点线算法或 Bresenham 算法，写入实验报告。

(3) 根据示范代码，将其改造为圆的光栅化算法，写入实验报告。

(4) 了解和使用 OpenGL 的生成直线的命令，来验证程序运行结果。

绘制点：

```
void myDisplay(void)
{
    // 请在此添加你的代码
    /********** Begin ********/
     glClearColor(0.0,0.0,0.0,0.0);
     glPointSize(3);
     glBegin(GL_POINTS);
     glColor3f(1.0f,0.0f,0.0f); glVertex2f(-0.4f,-0.4f);
     glColor3f(0.0f,1.0f,0.0f); glVertex2f(0.0f,0.0f);
     glColor3f(0.0f,0.0f,1.0f); glVertex2f(0.4f,0.4f);
     glEnd();
    /********** End **********/

     glFlush();
}
```

绘制光栅图：

```
void myDisplay(void)
{
    // 请在此添加你的代码
    /********** Begin ********/
     glClearColor(0.0, 0.0, 0.0, 0.0);
     glClear(GL_COLOR_BUFFER_BIT);

     glColor3f(1.0f, 1.0f, 1.0f);
     glRectf(-0.5f, -0.5f, 0.5f, 0.5f);

```

```
    glBegin(GL_TRIANGLES);
    glColor3f(1.0f, 0.0f, 0.0f);    glVertex2f(0.0f, 1.0f);
    glColor3f(0.0f, 1.0f, 0.0f);    glVertex2f(0.8f, -0.5f);
    glColor3f(0.0f, 0.0f, 1.0f);    glVertex2f(-0.8f, -0.5f);
    glEnd();

    glPointSize(3);
    glBegin(GL_POINTS);
    glColor3f(1.0f, 0.0f, 0.0f);    glVertex2f(-0.4f, -0.4f);
    glColor3f(0.0f, 1.0f, 0.0f);    glVertex2f(0.0f, 0.0f);
    glColor3f(0.0f, 0.0f, 1.0f);    glVertex2f(0.4f, 0.4f);
    glEnd();
   /********** End **********/

    glFlush();
}
```

绘制直线：

```
void myDisplay(void)
{
    // 请在此添加你的代码
    /********** Begin ********/
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f (1.0f, 0.0f, 0.0f);
        glRectf(25.0, 25.0, 75.0, 75.0);
        glPointSize(10);
        glBegin (GL_POINTS);
        glColor3f (0.0f, 1.0f, 0.0f);   glVertex2f (0.0f, 0.0f);
        glEnd ();


        glBegin (GL_LINES);
        glColor3f (0.0f, 1.0f, 0.0f);   glVertex2f (100.0f, 0.0f);
        glColor3f (0.0f, 1.0f, 0.0f);   glVertex2f (180.0f, 240.0f);
        glEnd ();

   /********** End **********/
    glFlush();
}
```

DDA（数值微分算法）算法是一个增量算法。增量算法：在一个迭代算法中，每一步的 x、y 值是用前一步的值加上一个增量来获得。

通过各行各列象素中心构造一组虚拟网格线。按直线从起点到终点的顺序计算直线与各垂直网格线的交点，然后根据误差项的符号确定该列象素中与此交点最近的象素。（见图 A.1）

DDA 需要考虑所画直线的斜率 k：

当 |k|<1 , x 每增加 1，y 增加 k。

当 |k|>1 , y 每增加 1，x 增加 1/k。

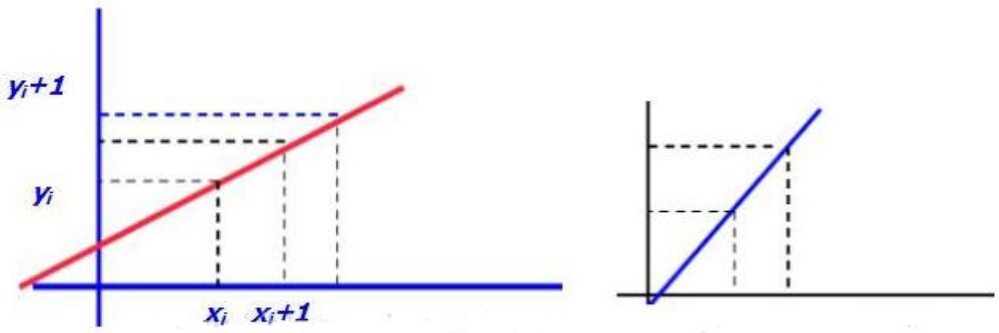


图 A.1 DDA 算法

OpenGL 画线的基础知识：

（1）数学上的直线没有宽度，但 OpenGL 的直线则是有宽度的。同时，OpenGL 的直线必须是有限长度，而不是像数学概念那样是无限的。可以认为，OpenGL 的“直线”概念与数学上的“线段”接近，它可以由两个端点来确定。这里的线由一系列顶点顺次连结而成，有闭合和不闭合两种。

前面的实验已经知道如何绘“点”，那么 OpenGL 是如何知道拿这些顶点来做什么呢？是一个一个的画出来，还是连成线？或者构成一个多边形？或是做其它事情呢？为了解决这一问题，OpenGL 要求：指定顶点的命令必须包含在 glBegin 函数之后，glEnd 函数之前（否则指定的顶点将被忽略），并由 glBegin 来指明如何使用这些点。

例如：

```
1.  glBegin(GL_POINTS);
2.      glVertex2f(0.0f, 0.0f);
3.      glVertex2f(0.5f, 0.0f);
4.  glEnd();
```

则这两个点将分别被画出来。如果将 GL_POINTS 替换成 GL_LINES，则两个点将被认为是直线的两个端点，OpenGL 将会画出一条直线。还可以指定更多的顶点，然后画出更复杂的图形。另一方面，glBegin 支持的方式除了 GL_POINTS 和 GL_LINES，还有 GL_LINE_STRIP，GL_LINE_LOOP，GL_TRIANGLES，GL_TRIANGLE_STRIP，GL_TRIANGLE_FAN 等，每种方式的大致效果如图 A.2 所示：

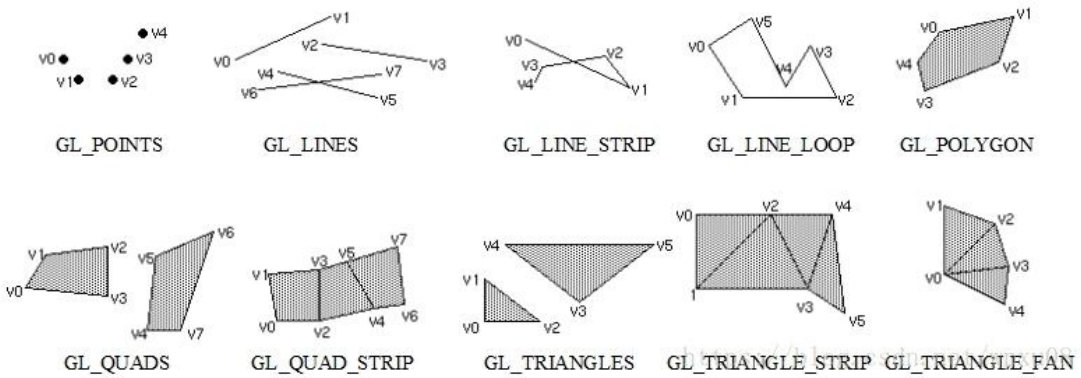


图 A.3 OpenGL 几何图元类型

（2）首次打开窗口、移动窗口和改变窗口大小时，窗口系统都将发送一个事件，以通知程序员。如果使用的是 GLUT，通知将自动完成，并调用向 glutReshapeFunc()注册的函数。该函数必须完成下列工作：

- 重新建立用作新渲染画布的矩形区域；
- 定义绘制物体时使用的坐标系。

如：

```
1.  void Reshape(int w, int h){
2.      glViewport(0, 0, (GLsizei) w, (GLsizei) h);
3.      glMatrixMode(GL_PROJECTION);
4.      glLoadIdentity();
5.      gluOrtho2D(0.0, (GLdouble) w, 0.0, (GLdouble) h);
6.  }
```

在 GLUT 内部，将给该函数传递两个参数：窗口被移动或修改大小后的宽度和高度，单位为像素。glViewport()调整像素矩形，用于绘制整个窗口。接下来三个函数调整绘图坐标系，使左下角位置为（0， 0），右上角为（w, h）。

```
1.  void LineDDA(int x0,int y0,int x1,int y1/*,int color*/)
2.  {
3.      int  x, dy, dx, y;
4.      float m;
5.      dx=x1-x0;
6.      dy=y1-y0;
7.      m=dy/dx;
8.      y=y0;
9.
10.     glColor3f (1.0f, 1.0f, 0.0f);
11.     glPointSize(1);
12.     for(x=x0;x<=x1; x++)
13.     {
14.         glBegin (GL_POINTS);
15.         glVertex2i (x, (int)(y+0.5));
16.         glEnd ();
17.         y+=m;
18.     }
19. }
```

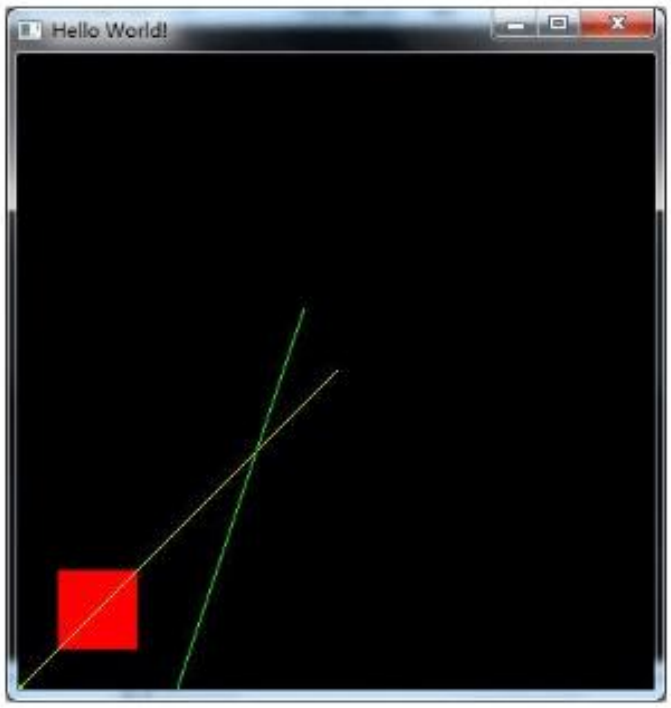
注： glShadeModel 选择平坦或光滑渐变模式。GL_SMOOTH 为缺省值，为光滑渐变模式，GL_FLAT 为平坦渐变模式。

中点线算法：

```
1.  void MidPLine(int x0, int y0, int x1, int y1)
2.  {
3.   // 请在此添加你的代码
4.     /********** Begin ********/
5.
6.      int dx, dy, dt, db, d, x, y;
7.      dx = x1- x0;
8.      dy = y1 - y0;
9.      d = dx - 2*dy;
```

```
    dt = 2*dx - 2*dy;
    db = -2*dy;
    x = x0; y = y0;
    glColor3f(0.0f, 3.0f, 0.0f);
   glPointSize(1);
   glBegin(GL_POINTS);
   glVertex2i(x, y);glEnd();
    while (x < x1)
    {
        if (d < 0)
        {
            x++;
            y++;
            d += dt;
        }
        else
        {
            x++;
            d += db;
        }


     glBegin(GL_POINTS);
   glVertex2i(x, y);glEnd();
    }
   /********** End **********/
    glFlush();
}
```

实验结果与分析：



错误指出：在书本中代码呈现出的直线是一条斜率为 1 的线，而按照数据，线条斜率为 1.5，分析其原因是在 LineDDA 函数中，m,x,y 等值的类型都为 int 型，造成了数据的丢失，所以斜率发生了变化。

在这次实验中，学习了画线的方法，一个点一个点的画出，通过初始结点和末结点的坐标，来决策下一个结点的坐标。对计算机中的画图方法有了进一层理解，对像素点也有了多一些的理解，希望以后可以更多的了解。

EduCoder 实训得分：96.5

任课老师评语：

签名：_____

日期：_____年___月___日

实验类别：专业　　实验要求：必修

实验类型：设计　　实验者类型：本科生

河南理工大学上机实验报告

学年：2020-2021　　第 1 学期　　上机时间 2020.12.2

专业班级 计实验 1801 班　　学号 311809000608　　姓名 王荣胜

课程名称：

计算机图形学

实验项目名称：

二维几何变换

实验目的：

1.阅读实验原理，掌握 OpenGL 程序平移、旋转、缩放变换的方法。

2.根据示范代码，完成实验作业。

实验内容与主要代码：

(1) OpenGL 下的几何变换

在 OpenGL 的核心库中，每一种几何变换都有一个独立的函数，所有变换都在三维空间中定义。

- 平移矩阵构造函数为 glTranslate<f,d>(tx, ty, tz)，作用是把当前矩阵和一个表示移动物体的矩阵相乘。tx、ty、tz 指定这个移动物体的矩阵，它们可以是任意的实数值，后缀为 f（单精度浮点 float）或 d（双精度浮点 double），对于二维应用来说，tz=0.0。

```
1.  void drawSquare(void)                    //绘制中心在原点，边长为 2 的正方形
2.  {
3.      // 请在此添加你的代码
4.      /********** Begin ********/
5.       glBegin(GL_POLYGON);
6.       glVertex2f(-1.0f, -1.0f);
7.       glVertex2f(1.0f, -1.0f);
8.       glVertex2f(1.0f, 1.0f);
9.       glVertex2f(1.0f, 1.0f);
10.      glVertex2f(-1.0f, 1.0f);
11.      glEnd();
12.     /********** End **********/
13. }
14.
15. void myDraw(void)                         //二维几何变换
16. {
17.     // 请在此添加你的代码
18.     /********** Begin ********/
19.      glClear(GL_COLOR_BUFFER_BIT);
20.      glLoadIdentity();
21.      glPushMatrix();
22.      glTranslatef(0.0f,2.0f,0.0f);
23.      glScalef(3.0,0.5,1.0);
24.      glColor3f(1.0,1.0,1.0);
25.      drawSquare();
26.      glPopMatrix();
27.
28.      glLoadIdentity();
29.      glColor3f(1.0,0.0,0.0);
30.      drawSquare();
31.
32.     /********** End **********/
33.      glFlush();
```

```
34. }
```

- 旋转矩阵构造函数为 glRotate<f,d>(theta, vx, vy, vz)，作用是把当前矩阵和一个表示旋转物体的矩阵相乘。theta，vx，vy，vz 指定这个旋转物体的矩阵，物体将围绕(0,0,0)到(x,y,z)的直线以逆时针旋转，参数 theta 表示旋转的角度。向量 v=(vx,vy,vz)的分量可以是任意的实数值，该向量用于定义通过坐标原点的旋转轴的方向，后缀为 f（单精度浮点 float）或 d（双精度浮点 double），对于二维旋转来说，vx=0.0，vy=0.0，vz=1.0。

```
1.  void drawSquare(void)                    //绘制中心在原点，边长为 2 的正方
    形
2.  {
3.     // 请在此添加你的代码
4.     /********** Begin ********/
5.      glBegin(GL_POLYGON);
6.      glVertex2f(-1.0f, -1.0f);
7.      glVertex2f(1.0f, -1.0f);
8.      glVertex2f(1.0f, 1.0f);
9.      glVertex2f(1.0f, 1.0f);
10.     glVertex2f(-1.0f, 1.0f);
11.     glEnd();
12.    /********** End **********/
13. }
14.
15. void myDraw(void)                         //二维几何变换
16. {
17.    // 请在此添加你的代码
18.    /********** Begin ********/
19.     glTranslatef(-3.0,0.0,0.0);
20.     glPushMatrix();
21.     glRotatef(45.0,0.0,0.0,1.0);
22.     glColor3f(0.0,1.0,0.0);
23.     drawSquare();
24.     glPopMatrix();
25.
26.
27.     glTranslatef(3.0,0.0,0.0);
28.     glPushMatrix();
29.     glRotatef(0.0,0.0,0.0,1.0);
30.     glColor3f(1.0,0.0,0.0);
31.     drawSquare();
32.     glPopMatrix();
33.
34.
35.
36.     glTranslatef(3.0,0.0,0.0);
37.     glPushMatrix();
38.     glRotatef(45.0,0.0,0.0,1.0);
39.     glColor3f(0.0,0.7,0.0);
40.     drawSquare();
41.     glPopMatrix();
42.
43.     glPopMatrix();
44.    /********** End **********/
45.     glFlush();
46. }
```

- 缩放矩阵构造函数为 glScale<f,d>(sx, sy, sz)，作用是把当前矩阵和一个表

示缩放物体的矩阵相乘。sx，sy，sz 指定这个缩放物体的矩阵，分别表示在 x，y，z 方向上的缩放比例，它们可以是任意的实数值，当缩放参数为负值时，该函数为反射矩阵，缩放相对于原点进行，后缀为 f（单精度浮点 float）或 d（双精度浮点 double）。

注意这里都是说“把当前矩阵和一个表示移动<旋转, 缩放>物体的矩阵相乘”，而不是直接说“这个函数就是旋转”或者“这个函数就是移动”，这是有原因的，马上就会讲到。

假设当前矩阵为单位矩阵，然后先乘以一个表示旋转的矩阵 R，再乘以一个表示移动的矩阵 T，最后得到的矩阵再乘上每一个顶点的坐标矩阵 v。那么，经过变换得到的顶点坐标就是((RT)v)。由于矩阵乘法满足结合率，((RT)v) = R(Tv))，换句话说，实际上是先进行移动，然后进行旋转。即：实际变换的顺序与代码中写的顺序是相反的。由于“先移动后旋转”和“先旋转后移动”得到的结果很可能不同，初学的时候需要特别注意这一点。

(2) OpenGL 下的各种变换简介

我们生活在一个三维的世界，如果要观察一个物体，我们可以：

① 从不同的位置去观察它（人运动，选定某个位置去看）。（视图变换）

② 移动或者旋转它，当然了，如果它只是计算机里面的物体，我们还可以放大或缩小它（物体运动，让人看它的不同部分）。（模型变换）

③ 如果把物体画下来，我们可以选择是否需要一种“近大远小”的透视效果。另外，我们可能只希望看到物体的一部分，而不是全部（指定看的范围）。（投影变换）

④ 我们可能希望把整个看到的图形画下来，但它只占据纸张的一部分，而不是全部（指定在显示器窗口的那个位置显示）。（视口变换）

这些，都可以在 OpenGL 中实现。

```
1.  glMatrixMode(GL_MODELVIEW);
```

该语句指定一个 4×4 的建模矩阵作为当前矩阵。

通常，我们需要在进行变换前把当前矩阵设置为单位矩阵。把当前矩阵设置为单位矩阵的函数为：

```
1.  glLoadIdentity();
```

我们在进行矩阵操作时，有可能需要先保存某个矩阵，过一段时间再恢复它。当我们需要保存时，调用 glPushMatrix()函数，它相当于把当前矩阵压入堆栈。当需要恢复最近一次的保存时，调用 glPopMatrix()函数，它相当于从堆栈栈顶弹出一个矩阵为当前矩阵。OpenGL 规定堆栈至少可以容纳 32 个矩阵，某些 OpenGL 实现中，堆栈的容量实际上超过了 32 个。因此不必过于担心矩阵的容量问题。

通常，用这种先保存后恢复的措施，比先变换再逆变换要更方便、更快速。

注意：模型视图矩阵和投影矩阵都有相应的堆栈。使用 glMatrixMode 来指定当前操作的究竟是模型视图矩阵还是投影矩阵。

（3）实验代码

```
1.  void drawDiamond(void)                          //绘制一个菱形
2.  {
3.      // 请在此添加你的代码
```

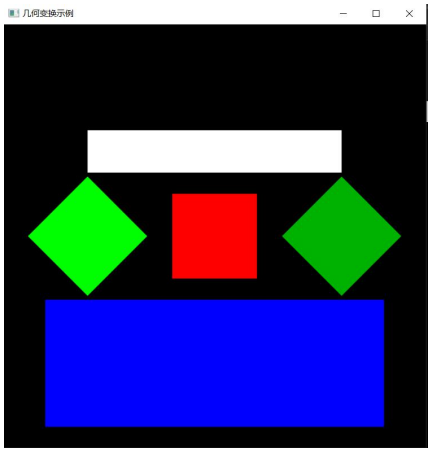
```
    /********** Begin ********/
     glBegin(GL_POLYGON);
     glVertex2f(0.0f,0.0f);
     glVertex2f(1.0f,2.0f);
     glVertex2f(0.0f,4.0f);
     glVertex2f(-1.0f,2.0f);
     glEnd();
    /********** End **********/
}

void myDraw(void)                                   //二维几何变换
{
    // 请在此添加你的代码
    /********** Begin ********/
      glClear(GL_COLOR_BUFFER_BIT);
     glLoadIdentity();
     glColor3f(1.0,0.0,0.0);
     drawDiamond();
     glRotatef(120.0,0.0,0.0,1.0);
     glColor3f(0.0,1.0,0.0);
     drawDiamond();
     glRotatef(120.0,0.0,0.0,1.0);
     glColor3f(0.0,0.0,1.0);
     drawDiamond();
     glFlush();
    /********** End **********/
     glFlush();
}
```

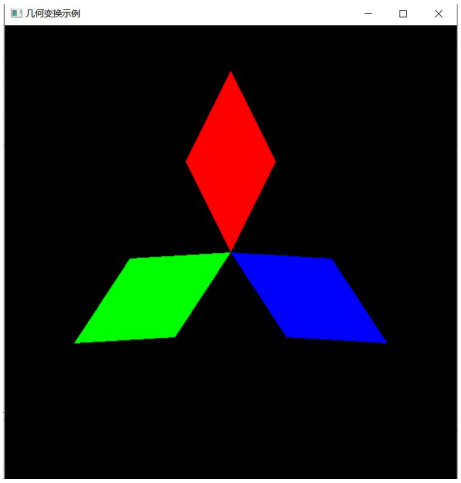
（4）实验提高

```
 void drawSquare(void)                              //绘制中心在原点，边长为 2 的正方形
{
    // 请在此添加你的代码
    /********** Begin ********/
     glBegin(GL_POLYGON);
     glVertex2f(-1.0f,-1.0f);
     glVertex2f(1.0f,-1.0f);
     glVertex2f(1.0f,1.0f);
     glVertex2f(-1.0f,1.0f);
     glEnd();
    /********** End **********/
}

void myDraw(void)                                   //二维几何变换
{
    // 请在此添加你的代码
    /********** Begin ********/
     glClear(GL_COLOR_BUFFER_BIT);
     glLoadIdentity();

     glPushMatrix();
     glTranslatef(0.0f,2.0f,0.0f);
     glScalef(3.0,0.5,1.0);
     glColor3f(1.0,1.0,1.0);
     drawSquare();
     glPopMatrix();



     glPushMatrix();
     glTranslatef(-3.0,0.0,0.0);

```

```
    glPushMatrix();
    glRotatef(45.0,0.0,0.0,1.0);
    glColor3f(0.0,1.0,0.0);
    drawSquare();
    glPopMatrix();


    glTranslatef(3.0,0.0,0.0);

    glPushMatrix();
    glRotatef(0.0,0.0,0.0,1.0);
    glColor3f(1.0,0.0,0.0);
    drawSquare();
    glPopMatrix();


    glTranslatef(3.0,0.0,0.0);
    glPushMatrix();
    glRotatef(45.0,0.0,0.0,1.0);
    glColor3f(0.0,0.7,0.0);
    drawSquare();
    glPopMatrix();


    glPopMatrix();
    glTranslatef(0.0,-3.0,0.0);
    glScalef(4.0,1.5,1.0);
    glColor3f(0.0,0.0,1.0);
    drawSquare();
   /********** End **********/
    glFlush();
}
```

实验结果与分析：





本次实验我通过 OpenGL 程序实现了图像的平移、旋转、缩放变换的方法，在实验之后对图像的变换有了更深刻的认识，让我产生了更大的兴趣。

EduCoder 实训得分：95.0

任课老师评语：

签名：_____

日期：_____年___月___日

实验类别：专业　　实验要求：必修

实验类型：设计　　实验者类型：本科生

河南理工大学上机实验报告

学年：_2020-2021__　　第_1_学期　　上机时间__2020.12.9

专业班级__计实验 1801 班__　　学号_311809000608__　　姓名_王荣胜

课程名称：

计算机图形学

实验项目名称：

立方体三位观察

实验目的：

1.通过示范代码 1 的立方体实例，理解巩固点的透视投影变换知识；

2.通过示范代码 1 的立方体实例，了解 OpenGL 实体显示的基本原理与方法；

3.通过示范代码 2 的立方体实例，学习 OpenGL 观察变换函数、投影变换函数的设置与使用方法；

实验内容与主要代码：

一、模型变换

1.背景色为黑色，用 glclearcolor 来完成；

2.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，以中心为绘制原点，设置前景色为红色 glColor3f(1.0, 0.0, 0.0)，绘制单位立方体线框，用 glutWireCube(1.0)完成;

3.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，设置前景色为黑色 glColor3f(0.0, 1.0, 0.0)，设置线宽为 2.0 用 glLineWidth(2.0)完成，将原单位立方体线框沿 X 轴正方向平移 2.0；

4.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，沿 X 轴负方向平移 2.0，设置前景色为蓝色 glColor3f(0.0, 0.0, 1.0)，绘制单位立方体实体用 glutSolidCube(1.0)完成；

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    /*观察变换*/
    gluLookAt(x0, yy, z0, xref, yref, zref, Vx, Vy, Vz);        //指定三维观察参数

    // 请在此添加你的代码
    /********** Begin ********/
    glPushMatrix();
    glColor3f(1.0,0.0,0.0);
    glScalef(1.0, 1.0, 1.0);
    glutWireCube(1.0);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(2.0,0.0,0.0);
    glColor3f(0.0,1.0,0.0);
    glLineWidth(2.0);
    glScalef(1.0, 1.0, 1.0);
    glutWireCube(1.0);
```

```
    glPopMatrix();

    glPushMatrix();
    glTranslatef(-2.0,0.0,0.0);
    glColor3f(0.0,0.0,1.0);
    glScalef(1.0, 1.0, 1.0);
    glutSolidCube(1.0);
    glPopMatrix();

    /********** End **********/
    glFlush();
}
```

二、立方体视图变换

1.背景色为黑色，用 glclearcolor 来完成；

2.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，以中心为绘制原点，设置前景色为红色 glColor3f(1.0, 0.0, 0.0)，绘制单位立方体线框，用 glutWireCube(1.0)完成;

3.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，设置前景色为黑色 glColor3f(0.0, 1.0, 0.0)，设置线宽为 2.0 用 glLineWidth(2.0)完成，将原单位立方体线框沿 X 轴正方向平移 2.0；

4.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，沿 X 轴负方向平移 2.0，设置前景色为蓝色 glColor3f(0.0, 0.0, 1.0)，绘制单位立方体实体用 glutSolidCube(1.0)完成；

5.视点改为（1.0,1.5,8.0)，观察中心改为在(0, 0 ,0)，向上矢量改为(0, 1, 0)；

6. 将 glFrustum(xwMin, xwMax, ywMin, ywMax, dnear, dfar); 换为透视投影 gluPerspective (fovy,aspect,zNear,zFar)函数，参数分别为（视角，宽高比，近处，远处）。要求参数为 gluPerspective(45, 1, 1, 100)。

```
GLint winWidth = 400, winHeight =400 ;        //设置初始化窗口大小

/*观察坐标系参数设置*/
GLfloat x0 = 1.0, yy = 1.5, z0 = 8.0;       //设置观察坐标系原点
GLfloat xref = 0.0, yref = 0.0, zref = 0.0; //设置观察坐标系参考点（视点）
GLfloat Vx = 0.0, Vy = 1.0, Vz = 0.0;       //设置观察坐标系向上向量（y 轴）

/*观察体参数设置 */
GLfloat xwMin = -1.0, ywMin = -1.0, xwMax = 1.0, ywMax = 1.0;//设置裁剪窗口坐标范围
GLfloat dnear = 1.5, dfar = 20.0;          //设置远、近裁剪面深度范围
```

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    /*观察变换*/
    gluLookAt(x0, yy, z0, xref, yref, zref, Vx, Vy, Vz);        //指定三维观察参数

    // 请在此添加你的代码
```

```
    /********** Begin ********/

    glPushMatrix();
    glColor3f(1.0,0.0,0.0);
    glScalef(1.0, 1.0, 1.0);
    glutWireCube(1.0);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(2.0,0.0,0.0);
    glColor3f(0.0,1.0,0.0);
    glLineWidth(2.0);
    glScalef(1.0, 1.0, 1.0);
    glutWireCube(1.0);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(-2.0,0.0,0.0);
    glColor3f(0.0,0.0,1.0);
    glScalef(1.0, 1.0, 1.0);
    glutSolidCube(1.0);
    glPopMatrix();
    /********** End **********/
    glFlush();
}
```

三、立方体三点透视

1.背景色为黑色，用 glclearcolor 来完成；

2.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，以中心为绘制原点，设置前景色为红色 glColor3f(1.0, 0.0, 0.0)，绘制单位立方体线框，用 glutWireCube(1.0)完成;

3.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，设置前景色为黑色 glColor3f(0.0, 1.0, 0.0)，设置线宽为 2.0 用 glLineWidth(2.0)完成，将原单位立方体线框沿 X 轴正方向平移 2.0；

4.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，沿 X 轴负方向平移 2.0，设置前景色为蓝色 glColor3f(0.0, 0.0, 1.0)，绘制单位立方体实体用 glutSolidCube(1.0)完成；

5.由图可知，中间红色为一点透视。右边绿色和左边蓝色为两点透视。通过 glRotatef()旋转绿色立方体来，完成蓝色立方体的三点透视。将绿色立方体绕 X 轴旋转+30 度。

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    /*观察变换*/
    gluLookAt(x0, yy, z0, xref, yref, zref, Vx, Vy, Vz);        //指定三维观察参数

    // 请在此添加你的代码
    /********** Begin ********/
    glPushMatrix();
    glColor3f(1.0,0.0,0.0);
    glScalef(1.0, 1.0, 1.0);
```

```
    glutWireCube(1.0);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(2.0,0.0,0.0);
    glColor3f(0.0,1.0,0.0);
    glLineWidth(2.0);
    glRotatef(30,1,0,0);
    glScalef(1.0, 1.0, 1.0);
    glutWireCube(1.0);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(-2.0,0.0,0.0);
    glColor3f(0.0,0.0,1.0);
    glScalef(1.0, 1.0, 1.0);
    glutSolidCube(1.0);
    glPopMatrix();

    /********** End **********/
    glFlush();
}
```

四、立方体平行投影

1.背景色为黑色，用 glclearcolor 来完成；

2.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，以中心为绘制原点，设置前景色为红色 glColor3f(1.0, 0.0, 0.0)，绘制单位立方体线框，用 glutWireCube(1.0)完成;

3.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，设置前景色为黑色 glColor3f(0.0, 1.0, 0.0)，设置线宽为 2.0 用 glLineWidth(2.0)完成，将原单位立方体线框沿 X 轴正方向平移 2.0；

4.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，沿 X 轴负方向平移 2.0，设置前景色为蓝色 glColor3f(0.0, 0.0, 1.0)，绘制单位立方体实体用 glutSolidCube(1.0)完成；

5.进行平行投影调用 glOrtho()函数，坐标为(左，右，下，上，近，远)，坐标为 glOrtho(-3.0, 3.0, -3.0, 3.0,-100.0, 100.0);

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    /*观察变换*/
    gluLookAt(x0, yy, z0, xref, yref, zref, Vx, Vy, Vz);        //指定三维观察参数

    // 请在此添加你的代码
    /********** Begin ********/
    glPushMatrix();
    glColor3f(1.0,0.0,0.0);
    glScalef(1.0, 1.0, 1.0);
    glutWireCube(1.0);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(2.0,0.0,0.0);
```

```
    glColor3f(0.0,1.0,0.0);
    glLineWidth(2.0);
    //glRotatef(30,1,0,0);
    glScalef(1.0, 1.0, 1.0);
    glutWireCube(1.0);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(-2.0,0.0,0.0);
    glColor3f(0.0,0.0,1.0);
    glScalef(1.0, 1.0, 1.0);
    glutSolidCube(1.0);
    glPopMatrix();


    /********** End **********/
    glFlush();
}
```

五、立方体视口变换

1.背景色为黑色，用 glclearcolor 来完成；

2.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，以中心为绘制原点，设置前景色为红色 glColor3f(1.0, 0.0, 0.0)，绘制单位立方体线框，用 glutWireCube(1.0)完成;

3.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，设置前景色为黑色 glColor3f(0.0, 1.0, 0.0)，设置线宽为 2.0 用 glLineWidth(2.0)完成，将原单位立方体线框沿 X 轴正方向平移 2.0；

4.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，沿 X 轴负方向右平移 2.0，设置前景色为蓝色 glColor3f(0.0, 0.0, 1.0)，绘制单位立方体实体用 glutSolidCube(1.0)完成；

5.进行视口变换视口高为 800，宽为 400。实验内调整 winWidth 和 winHeight 来设置初始化窗口大小；

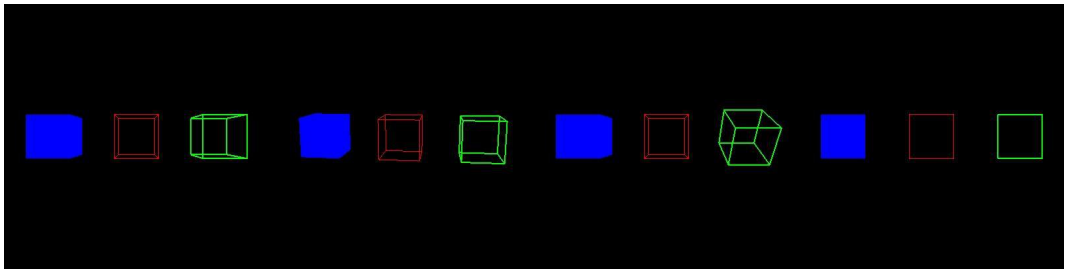
6.调用透视投影 gluperspective()函数，参数为 gluPerspective(45, 2, 1, 100)要求宽高比为 2；

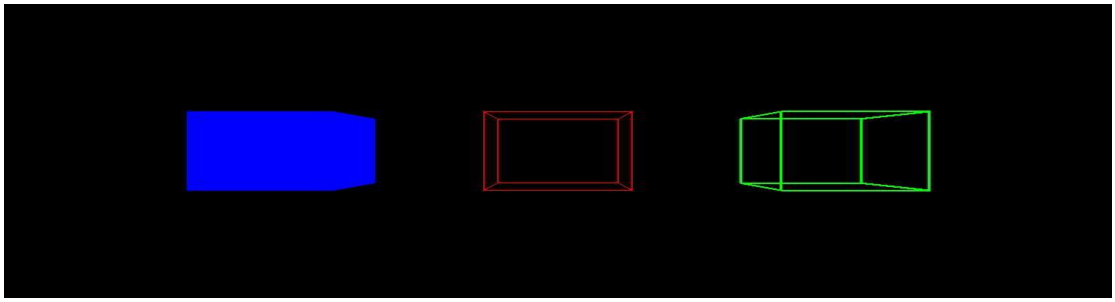
7.在 main 函数中用 glutInitWindowSize()调整视口窗口大小。

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    /*观察变换*/
    gluLookAt(x0, yy, z0, xref, yref, zref, Vx, Vy, Vz);        //指定三维观察参数

    // 请在此添加你的代码
    /********** Begin ********/
    glPushMatrix();
    glColor3f(1.0,0.0,0.0);
    glScalef(1.0, 1.0, 1.0);
    glutWireCube(1.0);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(2.0,0.0,0.0);
    glColor3f(0.0,1.0,0.0);
```

```
    glLineWidth(2.0);
    //glRotatef(30,1,0,0);
    glScalef(1.0, 1.0, 1.0);
    glutWireCube(1.0);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(-2.0,0.0,0.0);
    glColor3f(0.0,0.0,1.0);
    glScalef(1.0, 1.0, 1.0);
    glutSolidCube(1.0);
    glPopMatrix();
    /********** End **********/
    glFlush();
}
```

实验结果与分析：





本次实验我通过 OpenGL 程序实现掌握 OpenGL 程序的模型视图变换，OpenGL 三维图形显示与观察的原理与实现。在实验之后对图像的变换有了更深刻的认识，让我产生了更大的兴趣。

EduCoder 实训得分：96.1

任课老师评语：

签名：_____

日期：_____年___月___日

实验类别：专业　　实验要求：必修

实验类型：设计　　实验者类型：本科生

河南理工大学上机实验报告

学年：2020-2021　　第 1 学期　　上机时间 2020.12.16

专业班级 计实验 1801 班　　学号 311809000608　　姓名 王荣胜

课程名称：

计算机图形学

实验项目名称：

三维造型

实验目的：

1．了解简单实体构建的过程；熟悉视点观察函数的设置和使用;熟悉 3D 图形变换的设置和使用;进一步熟悉基本 3D 图元的绘制。

2．熟悉视点观察函数的设置和使用；熟悉 3D 图形变换的设置和使用；进一步熟悉基本 3D 图元的绘制。

3．了解曲线的生成原理；掌握几种常见的曲线生成算法，利用 VC+OpenGL 实现 Bezier 曲线生成算法和 B 样条曲线算法。

实验内容与主要代码：

一、　简单实体的构建

1.背景色为黑色，用 glclearcolor 来完成；

2.以中心为绘制原点，构建一个三棱锥；

3.创建二维数组用来存取三棱锥顶点坐标，顶点坐标分别为{-1,0,1},{1,0,1},{0,0,-0.7},{0,1.7,0}；

4.创建二维数组用来存取三棱锥面的颜色，分别为{0,1,0},{1,0,0},{1,1,0},{0,0,1}；

5.创建二维数组用来存取三棱锥顶点的序号；

6.运用 glPushMatrix()函数和 glPopMatrix()函数进行矩阵操作，沿 Z 轴负方向平移-3.0f，沿 Y 轴方向平移 0.2f,然后将三棱锥绕 X 轴方向选择 95 度

```
GLfloat points1[4][3] = {{-1,0,1},{1,0,1},{0,0,-0.7},{0,1.7,0}};
GLfloat Colors1[4][3] = {{0,1,0},{1,0,0},{1,1,0},{0,0,1}};
int vertice1[4][3] = {{0,1,2},{1,2,3},{0,2,3},{0,1,3}};

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    // 请在此添加你的代码
    /********** Begin ********/
    glPushMatrix();
    glTranslatef(0.0f,0.2f,-3.0f);  //平移至左侧
    glRotatef(95.0,1.0,0.0,0.0);
    /********** End **********/
    Create();                   //三棱锥
    glPopMatrix();
    glutSwapBuffers();
}
```

二、　简单机器人构建

1.背景色为黑色，用 glclearcolor 来完成；

2.用 gluSolidCube(GLdouble size)绘制立方体实体模型，要求尺寸大小为 0.5；

3.绘制脖子，将立方实体设为白色 glColor3f(1.0,1.0,1.0),沿 Y 轴正方向平移

1.4f，调用 glScalef()缩放函数参数为 glScalef(0.5,0.7,0.5)；
3.绘制头部，将立方实体设为黄色 glColor3f(1.0, 0.5, 0.2),沿 Y 轴正方向平移 1.9f，调用 glScalef()缩放函数参数为 glScalef(1.5,1.5,0.5)；
4.绘制身体，将立方实体设为红色 glColor3f(1.0,0.0,0.0),沿 Y 轴正方向平移 0.25f，调用 glScalef()函数将立方实体 X 轴方向变为原来的 4 倍，Y 轴方向变为原来的 4 倍，Z 轴方向变为原来的 0.5 倍；
5.绘制两个手臂，将立方实体设为 glColor3f(1.0, 1.0, 0.0);将左手臂移动坐标为 glTranslatef(-1.25f,0.5f,0.0f)，右手臂移动坐标为 glTranslatef(1.25f,0.5f,0.0f)，调用 glScalef()函数将立方体 Y 轴方向变为原来的 3 倍，Z 轴方向变为原来的 0.5 倍；
6.绘制两个机器人的腿部，将立方实体设为蓝色 glColor3f(0.5,0.5,1.5),将左腿移动坐标为 glTranslatef(-0.5f,-1.5f,0.0f)，右腿移动坐标为 glTranslatef(0.5f,-1.5f,0.0f)，调用 glScalef()函数将立方体 Y 轴方向变为原来的 3 倍，Z 轴方向变为原来的 0.5 倍；
7.绘制机器人的双手，用圆球绘制。球体的半径为 0.25，球心在原点，经线和纬线数目为 50。将球体设置为 glColor3f(1.0, 0.5, 0.2)。将小球分别移 glTranslatef(-1.25f,-0.5f,0.0f)glTranslatef(1.25f,-0.5f,0.0f)作为左手和右手。将两个小球进行放缩 glScalef(1,1.5,1)，变为椭圆。
8.视点改为（2.0,2.0,5.0），观察中心改为在(0, 0 ,0)，向上矢量改为(0, 1, 0)；

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    /*观察变换*/
    gluLookAt(x0, yy, z0, xref, yref, zref, Vx, Vy, Vz);        //指定三维观察参数

    // 请在此添加你的代码
    /********** Begin ********/

    glPushMatrix();
    glColor3f(1.0,1.0,1.0);
    glTranslatef(0.0f,1.4f,0.0f); //脖子
    glScalef(0.5,0.7,0.5);
    glutSolidCube(0.5);
    glPopMatrix();


    glPushMatrix();
    glColor3f(1.0, 0.5, 0.2);
    glTranslatef(0.0f,1.9f,0.0f); //头
    glScalef(1.5,1.5,0.5);
    glutSolidCube(0.5);
    glPopMatrix();

    glPushMatrix();
    glColor3f(1.0,0.0,0.0);
    glTranslatef(0.0f,0.25f,0.0f); //身体
    glScalef(4.0,4.0,0.5);
    glutSolidCube(0.5);
    glPopMatrix();

```

```
    glPushMatrix();
    glColor3f(1.0, 1.0, 0.0);
    glTranslatef(-1.25f,0.5f,0.0f); //手臂
    glScalef(1.0,3.0,0.5);
    glutSolidCube(0.5);
    glPopMatrix();

    glPushMatrix();
    glColor3f(1.0, 1.0, 0.0);
    glTranslatef(1.25f,0.5f,0.0f);
    glScalef(1.0,3.0,0.5);
    glutSolidCube(0.5);
    glPopMatrix();

    glPushMatrix();
    glColor3f(0.5,0.5,1.5);//退
    glTranslatef(-0.5f,-1.5f,0.0f);
    glScalef(1.0,3.0,0.5);
    glutSolidCube(0.5);
    glPopMatrix();

    glPushMatrix();
    glColor3f(0.5,0.5,1.5);
    glTranslatef(0.5f,-1.5f,0.0f);
    glScalef(1.0,3.0,0.5);
    glutSolidCube(0.5);
    glPopMatrix();

    GLUquadricObj *sphere;  //定义二次曲面对象
    sphere=gluNewQuadric();

    glPushMatrix();
    glColor3f(1.0, 0.5, 0.2);
    glTranslatef(-1.25f,-0.5f,0.0f);
    glScalef(1,1.5,1);
    gluSphere(sphere,0.25,50,50);
    glPopMatrix();

    glPushMatrix();
    glColor3f(1.0, 0.5, 0.2);
    glTranslatef(1.25f,-0.5f,0.0f);
    glScalef(1,1.5,1);
    gluSphere(sphere,0.25,50,50);
    glPopMatrix();

    /********** End **********/
    glFlush();
}
```

三、 Bezier 曲线和 B 样条曲线

1.背景色为黑色，用 glclearcolor 来完成；

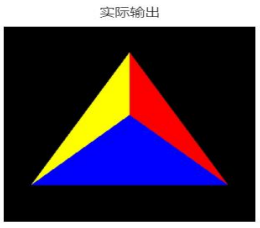
2.结合公式及代码示范代码了解 Bezier 曲线生成原理与算法实现；

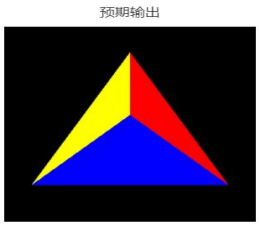
3.参考代码文件中的 Bezier 曲线生成代码，根据 B 样条公式，修改 B 样条曲线代码来实现 B 样条的生成。要求 B 样条的控制顶点坐标为 Point pt2[4]={ { 450, 100}, { 540,300},{650, 320}, {690, 120} }。

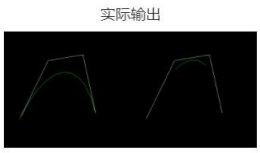
```
Point pt2[4]={ { 450, 100}, { 540,300},{650, 320}, {690, 120}  };

float a0, a1, a2, a3, b0, b1, b2, b3;

```

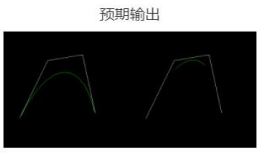
```
5.      a0 = pt2[0].x + 4 * pt2[1].x + pt2[2].x;
6.      a1 = -3 * pt2[0].x + 3 * pt2[2].x;
7.      a2 = 3 * pt2[0].x - 6 * pt2[1].x + 3 * pt2[2].x;
8.      a3 = -pt2[0].x + 3 * pt2[1].x - 3 * pt2[2].x + pt2[3].x;
9.      b0 = pt2[0].y + 4 * pt2[1].y + pt2[2].y;
10.     b1 = -3 * pt2[0].y + 3 * pt2[2].y;
11.     b2 = 3 * pt2[0].y - 6 * pt2[1].y + 3 * pt2[2].y;
12.     b3 = -pt2[0].y + 3 * pt2[1].y - 3 * pt2[2].y + pt2[3].y;
13.
14.     float t = 0;
15.     float dt = 0.01;
16.     for (int i = 0; t < 1.1; t += 0.1, i++)
17.     {
18.         bspt[i].x =  ( a0 + a1 * t + a2 * t * t + a3 * t * t * t ) / 6;
19.         bspt[i].y =  ( b0 + b1 * t + b2 * t * t + b3 * t * t * t ) / 6;
20.     }
21. }
```

实验结果与分析：









本次实验我通过 OpenGL 程序实现熟悉视点观察函数的设置和使用、3D 图形变换的设置和使用、基本 3D 图元的绘制。在实验之后对图像的变换有了更深刻的认识，让我产生了更大的兴趣。

EduCoder 实训得分：95.0

任课老师评语：

签名：_____

日期：_____年___月___日

实验类别：专业　　实验要求：必修

实验类型：设计　　实验者类型：本科生

河南理工大学上机实验报告

学年：2020-2021　　第 1 学期　　上机时间 2020.12.23

专业班级 计实验 1801 班　　学号 311809000608　　姓名 王荣胜

课程名称：

计算机图形学

实验项目名称：

着色

实验目的：

为在场景中增加光照，需要执行以下步骤。

1.设置一个或多个光源，设定它的有关属性；

2.选择一种光照模型；

3.设置物体的材料属性。

实验内容与主要代码：

一、 OpenGL 球体漫反射

给球体添加环境光，漫反射和镜面反射等效果，每个参数的含义已在代码中给出。

1.左侧圆球只有漫反射；

2.中间圆球有环境光和漫反射；

3.右侧圆球有彩色环境光和漫反射。

```
void display(void)
{
    GLfloat no_mat[]={0.0,0.0,0.0,1.0};                    //没有光
    GLfloat mat_ambient[]={0.7,0.7,0.7,1.0};               //环境光
    GLfloat mat_ambient_color[]={0.8, 0.8, 0.2, 1.0};      //彩色环境光
    GLfloat mat_diffuse[]={0.1,0.5,0.8,1.0};               //漫反射
    GLfloat mat_specular[]={1.0,1.0,1.0,1.0};              //镜面反射
    GLfloat no_shininess[] = {0.0};                        //没有镜面反射
    GLfloat low_shininess[]={5.0};                         //低镜面反射
    GLfloat high_shininess[]={100.0};                      //高镜面反射
    GLfloat mat_emission[]={0.3,0.2,0.2,0.0};              //材料辐射光颜色

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

     glPushMatrix();
     glTranslatef(-3.0, 0.0, 0.0);
     // 请在此添加你的代码，左侧圆球只有漫反射
     /********** Begin ********/
     glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
     glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);


     /********** End *********/
     glutSolidSphere(1.0, 50, 50);
     glPopMatrix();

     glPushMatrix();
     // 请在此添加你的代码，中间圆球有环境光和漫反射
     /********** Begin ********/
```

```
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);


    /********** End *********/
    glutSolidSphere(1.0, 50, 50);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(3.0, 0.0, 0.0);
    // 请在此添加你的代码，右侧圆球有彩色环境光和漫反射
    /********** Begin ********/
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_color);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);

    /********** End *********/
    glutSolidSphere(1.0, 50, 50);
    glPopMatrix();


    glFlush();
    glutPostRedisplay();
}
```

二、 OpenGL 球体镜面反射

给球体添加环境光，漫反射和镜面反射等效果，每个参数的含义已在代码中给出。

1.左侧圆球有环境光、漫反射和镜面低反射；

2.中间圆球有环境光、漫反射和镜面高反射；

3.右侧圆球有彩色环境光、漫反射、镜面高反射和材料辐射。

```
void display(void)
{
   GLfloat no_mat[]={0.0,0.0,0.0,1.0};                    //没有光
   GLfloat mat_ambient[]={0.7,0.7,0.7,1.0};               //环境光
   GLfloat mat_ambient_color[]={0.8, 0.8, 0.2, 1.0};      //彩色环境光
   GLfloat mat_diffuse[]={0.1,0.5,0.8,1.0};               //漫反射
   GLfloat mat_specular[]={1.0,1.0,1.0,1.0};              //镜面反射
   GLfloat no_shininess[] = {0.0};                        //没有镜面反射
   GLfloat low_shininess[]={5.0};                         //低镜面反射
   GLfloat high_shininess[]={100.0};                      //高镜面反射
   GLfloat mat_emission[]={0.3,0.2,0.2,0.0};              //材料辐射光颜色

   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glPushMatrix();
    glTranslatef(-3.0, 0.0, 0.0);
    // 请在此添加你的代码，左侧圆球有环境光、漫反射镜面低反射
    /********** Begin ********/
    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,low_shininess);

    /********** End *********/
```

```
    glutSolidSphere(1.0, 50, 50);
    glPopMatrix();

    glPushMatrix();
    // 请在此添加你的代码，中间圆球有环境光、漫反射镜面高反射
    /********** Begin ********/
    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);

    /********** End *********/
    glutSolidSphere(1.0, 50, 50);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(3.0, 0.0, 0.0);
    // 请在此添加你的代码，右侧圆球有彩色环境光、漫反射、镜面高反射和材料辐射
    /********** Begin ********/
    glMaterialfv(GL_FRONT,GL_AMBIENT,mat_ambient_color);
    glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);
    glMaterialfv(GL_FRONT,GL_SHININESS,high_shininess);
    glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
    glMaterialfv(GL_FRONT,GL_EMISSION,mat_emission);

    /********** End *********/
    glutSolidSphere(1.0, 50, 50);
    glPopMatrix();



    glFlush();
    glutPostRedisplay();
}
```

三、 OpenGL 茶壶光照

1.背景色为黑色，用 glclearcolor 来完成；

2.在合适的地方修改代码，使茶壶进行着色,每个茶壶材质参数和光照的参数已给出；

3.左边茶壶只有环境光和自身的发射光 RGBA 强度；

4.中间的茶壶有环境光、漫反射光和自身的发射光 RGBA 强度；

5.右边的茶壶有环境光、漫反射光、镜面反射光和自身的发射光 RGBA 强度；

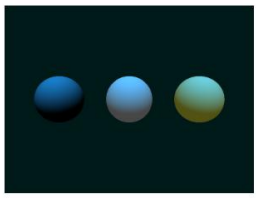
```
//定义左边茶壶材质系数
GLfloat left_ambient[] = { 5.0f, 0.0f, 0.0f, 1.0f };
GLfloat left_diffuse[] = { 0.8f, 0.8f, 0.0f, 1.0f };
GLfloat left_specular[] = { 1.0f, 1.0f, 0.0f, 1.0f };
GLfloat left_emission[] = { 0.1f, 0.0f, 0.0f, 1.0f };

//定义中间茶壶材质系数
GLfloat mid_ambient[] = { 0.0f, 0.2f, 0.0f, 1.0f };
GLfloat mid_diffuse[] = { 0.0f, 0.8f, 0.0f, 1.0f };
GLfloat mid_specular[] = { 0.0f, 1.0f, 0.0f, 1.0f };
GLfloat mid_shininess[] = { 80.0f };
GLfloat mid_emission[] = { 0.0f, 0.1f, 0.0f, 1.0f };

//定义右边茶壶材质系数
GLfloat right_ambient[] = { 0.0f, 0.0f, 0.1f, 1.0f };
```

```
GLfloat right_diffuse[] = { 0.0f, 0.0f, 0.8f, 1.0f };
GLfloat right_specular[] = { 0.0f, 0.0f, 0.9f, 1.0f };
GLfloat right_shininess[] = { 50.0f };
GLfloat right_emission[] = { 0.0f, 0.1f, 0.0f, 1.0f };

void myInit(void)
{

    GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
    GLfloat white_light[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_ambient[] = { 0.2 , 0.2 , 0.2 , 1.0 };


    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_SMOOTH);

    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, white_light);
    glLightfv(GL_LIGHT0, GL_SPECULAR, white_light);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);

}

void myDisplay(void)
{

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glPushMatrix();
    // 请在此添加你的代码，中间茶壶
    /********** Begin ********/
    glMaterialfv(GL_FRONT, GL_AMBIENT, mid_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mid_diffuse);
    //glMaterialfv(GL_FRONT, GL_SPECULAR, mid_specular);
    //glMateriali(GL_FRONT, GL_SHININESS, mid_shininess[0]);
    glMaterialfv(GL_FRONT, GL_EMISSION, mid_emission);
    /********** End *********/
    glutSolidTeapot(0.5);
    glPopMatrix();
    glFlush();


    glPushMatrix();
    glTranslatef(2.0,0.0,0.0);
    // 请在此添加你的代码,右边茶壶
    /********** Begin ********/
    glMaterialfv(GL_FRONT, GL_AMBIENT, right_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, right_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, right_specular);
    glMateriali(GL_FRONT, GL_SHININESS, right_shininess[0]);
    glMaterialfv(GL_FRONT, GL_EMISSION, right_emission);
    /********** End *********/
    glutSolidTeapot(0.5);
    glPopMatrix();
    glFlush();


    glPushMatrix();
    glDisable(GL_LIGHT0);
    glTranslatef(-2.0,0.0,0.0);
    // 请在此添加你的代码，左边茶壶
```

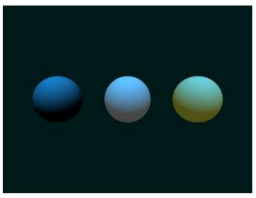
```
80.     /********** Begin ********/
81.     glMaterialfv(GL_FRONT, GL_AMBIENT, left_ambient);
82.     //glMaterialfv(GL_FRONT, GL_DIFFUSE, left_diffuse);
83.     //glMaterialfv(GL_FRONT, GL_SPECULAR, left_specular);
84.     glMaterialfv(GL_FRONT, GL_EMISSION, left_emission);
85.     /********** End *********/
86.     glutSolidTeapot(0.5);
87.     glPopMatrix();
88.     glFlush();
89. }
```

实验结果与分析：

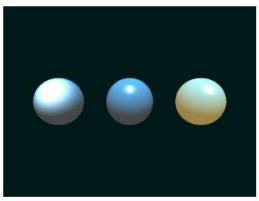
实际输出



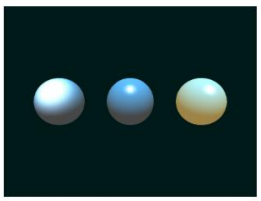
预期输出



实际输出

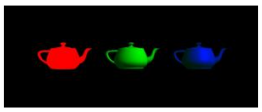


预期输出



实际输出

预期输出



本次实验我通过 OpenGL 程序实现球的各种反射和光照模型。在实验之后对着色有了更深刻的认识，让我产生了更大的兴趣。本次实验较为简单，我们可以在编写完成第一个程序代码后，对新程序代码进行光照或者模型的改变即可完成实验，这次试验使我们能更好的渲染模型，并且更好在实践中应用。

EduCoder 实训得分：97.3

任课老师评语：

签名：_____

日期：_____年___月___日

实验类别：专业　　实验要求：必修

实验类型：设计　　实验者类型：本科生

河南理工大学上机实验报告

学年：2020-2021　　第 1 学期　　上机时间 2020.12.30

专业班级 计实验 1801 班　　学号 311809000608　　姓名 王荣胜

课程名称：

计算机图形学

实验项目名称：

纹理映射

实验目的：

1. 学习真实感图形绘制技术
2. 读取或生成纹理图形数据
3. 将纹理映射到物体表面
4. 纹理映射是将纹理空间中的纹理像素映射到屏幕空间像素的过程

实验内容与主要代码：

一、正方形纹理映射

为在场景中添加纹理映射，需要执行以下步骤。

1.定义纹理，利用矩形图像进行贴图是二维纹理贴图中常用的方法；

2.纹理获取，利用函数直接设置各种纹理像素点的 RGB 值；

3.纹理坐标，定义纹理与几何坐标对应关系的纹理坐标；

4.纹理控制，定义纹理如何包裹物体的表面；

5.纹理映射方式，用函数可以设置纹理映射的方式。

```
1. 给两个正方形添加纹理映射。
2. 1.设置控制纹理映射的函数，参数分别如下：
3.
4. glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
5. glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
6. glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,GL_NEAREST);
7. glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,GL_NEAREST);
8. 2.定义二维纹理贴图函数，参数分别如下：
9.
10. glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA, checkImageWidth, checkImageHeight,0
    ,
11. GL_RGBA, GL_UNSIGNED_BYTE, checkImage);
12. 3.设置纹理映射方式，函数参数如下：
13.
14. glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
15. 4.设置纹理坐标，函数参数如下：
16.
17.     glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
18.     glTexCoord2f(0.0, 1.0); glVertex3f(-2.0, 1.0, 0.0);
19.     glTexCoord2f(1.0, 1.0); glVertex3f(0.0, 1.0, 0.0);
20.     glTexCoord2f(1.0, 0.0); glVertex3f(0.0, -1.0, 0.0);
21.     glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
22.     glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
23.     glTexCoord2f(1.0, 1.0); glVertex3f(2.41421, 1.0, -1.41421);
24.     glTexCoord2f(1.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);
```

```
void init(void)
{
    glClearColor(0.5, 2.0, 0.5, 0.0);
    glShadeModel(GL_FLAT);
    glEnable(GL_DEPTH_TEST);

    makeCheckImage();
    glBindTexture(GL_TEXTURE_2D, texName);
    //********1.2.请输入代码设置控制纹理映射函数和纹理的定义函数********//
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,GL_NEAREST);


   //*********************************************************//

}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_TEXTURE_2D);
    //********3.请输入代码进行纹理映射方式设置*********//
    glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA, checkImageWidth, checkImageHeight,0,
GL_RGBA, GL_UNSIGNED_BYTE, checkImage);

    //*********************************************//
    glBindTexture(GL_TEXTURE_2D, texName);
    glBegin(GL_QUADS);
    //********4.请输入代码进行纹理坐标设置**********//
    glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-2.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(0.0, 1.0, 0.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(0.0, -1.0, 0.0);
    glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(2.41421, 1.0, -1.41421);
    glTexCoord2f(1.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);


    //*********************************************//
    glEnd();
    glFlush();
    glDisable(GL_TEXTURE_2D);
}
```

二、球体纹理映射

为在场景中添加纹理映射，需要执行以下步骤。

1.定义纹理，利用矩形图像进行贴图是二维纹理贴图中常用的方法；

2.纹理获取，利用函数直接设置各种纹理像素点的 RGB 值；

3.纹理坐标，定义纹理与几何坐标对应关系的纹理坐标；

4.纹理控制，定义纹理如何包裹物体的表面；

5.纹理映射方式，用函数可以设置纹理映射的方式。

给球体添加纹理映射。
1.设置纹理映射控制的函数，参数分别如下：

```
    glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

2.定义一维纹理贴图函数，参数分别如下：

```
    glTexImage1D(GL_TEXTURE_1D, 0, 4, stripeImageWidth, 0, GL_RGBA, GL_UNSIGNED_BYTE,
stripeImage);
```

3.设置纹理映射方式，函数参数如下：

```
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
```

4.设置纹理坐标自动生成函数参数如下：

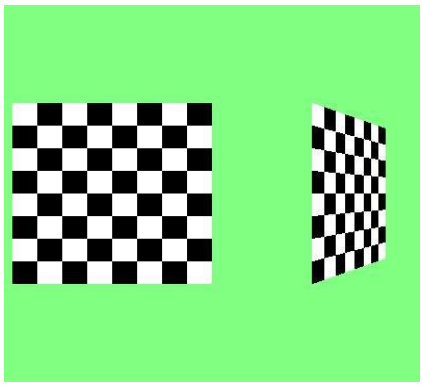
```
    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE,  GL_OBJECT_LINEAR);
    glTexGenfv(GL_S, currentPlane, currentCoeff);
```

```
void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);
    makeStripeImage();
    //********1.2.请输入代码设置纹理映射控制函数和纹理的定义函数********//
    glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);


    glTexImage1D(GL_TEXTURE_1D, 0, 4, stripeImageWidth, 0, GL_RGBA, GL_UNSIGNED_BYTE,
stripeImage);


    //*********************************************************//


    //********3.请输入代码进行纹理映射方式设置*********//


    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

    //*****************************************//
    currentCoeff = xequalzero;
    currentGenMode = GL_OBJECT_LINEAR;
    currentPlane = GL_OBJECT_PLANE;
    //****************4.自动纹理坐标生成功能函数******************//


    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE,  GL_OBJECT_LINEAR);
    glTexGenfv(GL_S, currentPlane, currentCoeff);

    //*********************************************************//
    glEnable(GL_TEXTURE_GEN_S);
```

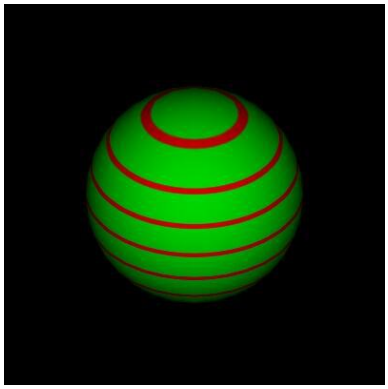
```
36.     glEnable(GL_TEXTURE_1D);
37.     glEnable(GL_LIGHTING);
38.     glEnable(GL_LIGHT0);
39.     glEnable(GL_AUTO_NORMAL);
40.     glEnable(GL_NORMALIZE);
41.     glMaterialf(GL_FRONT, GL_SHININESS, 64.0);
42.     roangles = 45.0f;
43. }
```

实验结果与分析：

实验一运行结果与实际输出：



实验二运行结果与实际输出：



本次实验我通过 OpenGL 程序实现读取或生成纹理图形数据、将纹理映射到物体表面、纹理映射是将纹理空间中的纹理像素映射到屏幕空间像素的过程。在实验之后对纹理映射有了更深刻的认识，让我产生了更大的兴趣。

EduCoder 实训得分：97.3

任课老师评语：

签名：_____

日期：_____年___月___日

实验类别：专业　　实验要求：必修

实验类型：设计　　实验者类型：本科生