专 业 ： 数 媒 1801
姓名：__杨锐_____
学号：3180101941_____
日期：__2020/3/20_____
地点：_____

浙江大学实验报告

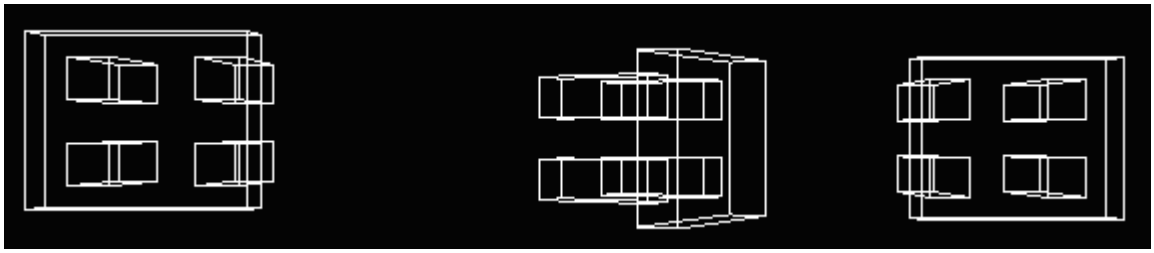
课程名称：____计算机图形学____ 指导老师：_____唐敏_____成绩：_____
实验名称：____OpenGL 矩阵_____ 实验类型：_____基础实验_____同组学生姓名：_____

一、实验目的和要求

在 OpenGL 编程基础上，通过实现实验内容，掌握 OpenGL 的矩阵使用，并验证课程中矩阵变换的内容。

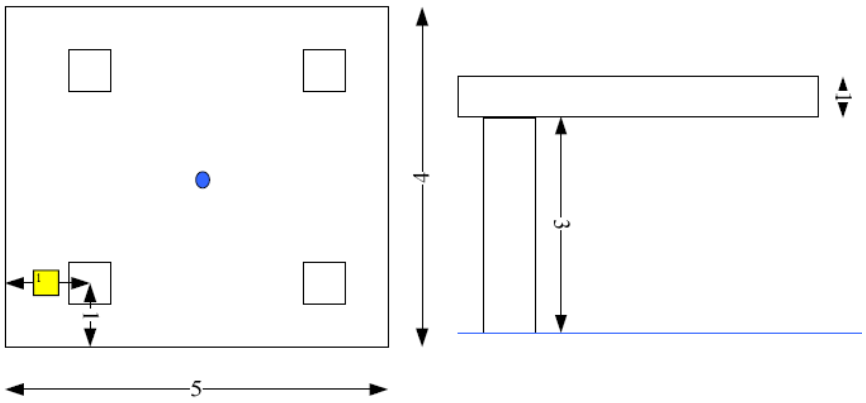
二、实验内容和原理

使用 Visual Studio C++编译已有项目工程，并修改代码生成以下图形（参考示例答案）：



其中最左边的桌子循环上移（即匀速上移到一定位置后回到原点继续匀速上移），中间的桌子不断旋转，最右边的桌子循环缩小（即不断缩小到一定大小后回归原来大小继续缩小）。

桌子的模型尺寸如下：



三、主要仪器设备

Visual Studio C++
Ex2 工程

四、操作方法和实验步骤

五、实验数据记录和处理

六、实验结果与分析

七、讨论、心得

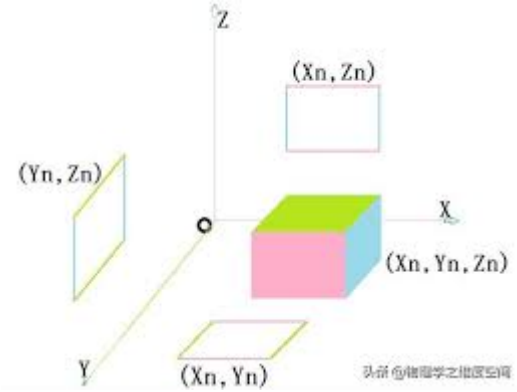
装 订 线

四、操作方法和实验步骤

0.实验分析

这次实验是分别绘制平移、旋转、缩放的三张桌子，根据已有三角形代码和提示，主要分两步进行：构建一个绘制桌子的函数，在redraw函数设置三种变换的参数。

(1)首先对于桌子的绘制，简单观察我们可以发现，其由五个立方体组成（一个桌面+四条桌腿），因此我们可以构建一个绘制立方体的通用函数，通过设置参数来实现桌子的绘制。对于一个任意的立方体，我们可以通过设置互为对角顶点的三个顶点坐标共9个参数确定，而由于在这里我们的立方体各面都平行于坐标平面，因此只需6个参数。



---

(2)其次是对于三种变换参数的设定。在opengl中无论是二维还是三维图形的变换，都是通过左×一个变换矩阵完成的，这也和线性代数中基本观点一致：矩阵代表坐标系变换。

同时，在opengl中，变换矩阵为4×4矩阵， 向量为1×4矩阵， 而非3×3和1×3，个人认为原因是对于平移来说，我们必须要额外的一列来放置位移值，具体见下面的矩阵变换

平移：函数`glTranslatef(GLfloat x, GLfloaty, GLfloat z)`三种参数分别代表沿xyz三个方向的平移量（单位为像素），变换原理如下：

$$\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$

通过矩阵点乘我们可以很容易得到结果为

$$\begin{Bmatrix} x+dx \\ y+dy \\ z+dz \\ 1 \end{Bmatrix}$$

如果原始矩阵为3×3，那么我们就没有地方放置平移向量(dx, dy, dz)了

缩放：`glScalef(GLfloat x, GLfloat y, GLfloatz);`参数为各坐标缩放程度。严格的来说，缩放矩阵可以仅通过3×3实现，当然我们没必要这么做，因为我们只需要设置单位矩阵的$[1,1],[2,2],[3,3]$为对应比例即可

旋转：`glRotatef(GLfloat theta, GLfloatx, GLfloat y, GLfloat z)`参数为角度和旋转轴。旋转比起平移和缩放来说有些复杂，原因是旋转涉及的坐标运算有些繁琐，特别是对于一条任意的轴来说。而在实际中我们大多数是绕三个坐标轴作旋转，因此这里给出本次实验所需的绕y轴旋转矩阵：

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot x + \sin\theta \cdot z \\ y \\ -\sin\theta \cdot x + \cos\theta \cdot z \\ 1 \end{pmatrix}$$

这和上次实验绘制五角星时我用到的一条向量旋转α度的公式也是一样的。

---

(3)动画效果的实现主要来自于reshape()函数，我对其中每一个函数的具体原理还不是很了解，我的理解是`glutDisplayFunc(redraw)`函数作单次画面的绘制，当执行完一次后，调用`glutReshapeFunc(reshape)`回调函数，在这里我们可以修改每次绘制时的视角、画面尺寸等参数，然后又回到单次绘制函数，这样达到每执行完一次就刷新一下平移/旋转/缩放因子，最终实现连续画面。

1.单个立方体绘制

通过传入参数，设定所有顶点坐标，再将六个面所对应坐标数字放入6×4数组，最后按照对应顶点，依次绘制出六个面

```
void Draw_Cube(GLfloat x1 , GLfloat x2 , GLfloat y1 , GLfloat y2 , GLfloat z1 , GLfloat z
        //设置立方体的8个顶点坐标
        GLfloat V[8][3]{
                x1 , y1 , z1,
                x1 , y2 , z1,
                x2 , y2 , z1,
                x2 , y1 , z1,
                x2 , y1 , z2,
                x2 , y2 , z2,
                x1 , y2 , z2,
                x1 , y1 , z2,
        };
        //设置立方体的六个面对应顶点
        GLint Planes[6][4]{
                0 , 1 , 2 , 3,
                4 , 5 , 6 , 7,
                2 , 3 , 4 , 5,
                0 , 1 , 6 , 7,
                1 , 2 , 5 , 6,
                0 , 3 , 4 , 7,
        };
        glBegin(GL_QUADS);
        for (int i = 0; i < 6; ++i)
                for (int j = 0; j < 4; ++j)
                        glVertex3fv(V[Planes[i][j]]);
        glEnd();
}
```

2.桌面绘制

这里要注意桌腿和桌面的相对关系

```
void Draw_Table() {
        Draw_Cube(0.0, 1.0, 0.0, 0.8, 0.6, 0.8);//Table Top
        //Table Legs
        Draw_Cube(0.1, 0.3, 0.1, 0.3, 0.0, 0.6);
        Draw_Cube(0.1, 0.3, 0.5, 0.7, 0.0, 0.6);
        Draw_Cube(0.5, 0.7, 0.1, 0.3, 0.0, 0.6);
        Draw_Cube(0.5, 0.7, 0.5, 0.7, 0.0, 0.6);
}
```

3.变换

opengl同时实现多种变换的原理是使用矩阵栈，以平移为例，
`glPushMatrix();`往矩阵栈中push一个4×4单位矩阵(即不做任何变换)，

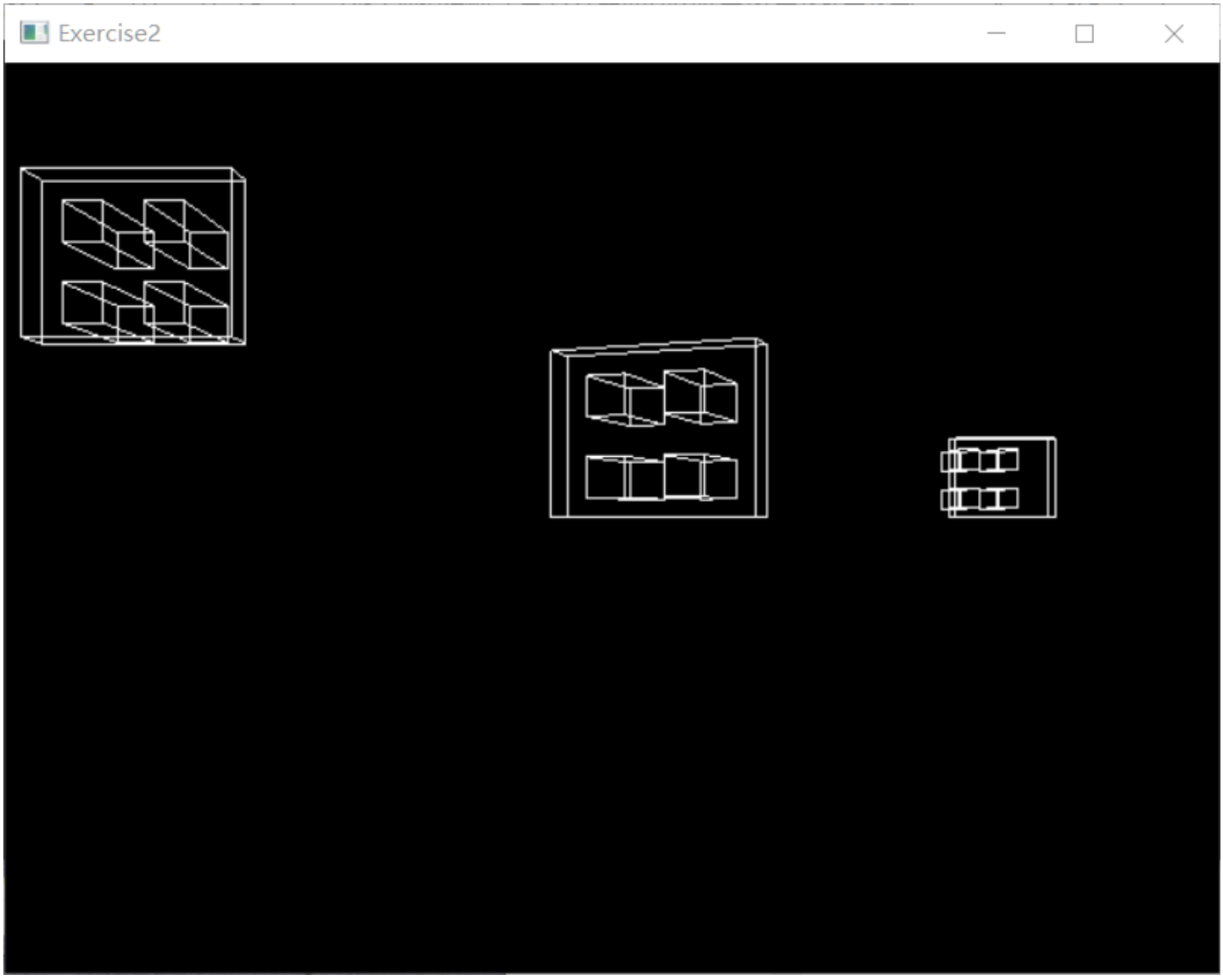
`glTranslatef(-2.8f, 0.0f, -6.0f);` 将矩阵平移到起始位置，
`glTranslatef(0.0f, fTranslate, 0.0f);` 作一次平移
`Draw_Table();` 在当前矩阵位置绘制一个table
`glPopMatrix();` pop当前矩阵，该次变换结束

通过每次预先push一个矩阵，就能实现多种绘制互不干扰。

```
#define PI 3.1415926
float fTranslate = 0.0f;
float fRotate = 0.0f;
float fScale = 1.0f;
```

```
void redraw()
{
        // If want display in wireframe mode
        //      glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
        glClear(GL_COLOR_BUFFER_BIT);
        glLoadIdentity();// Reset The Current Modelview Matrix

        glPushMatrix();
        glTranslatef(-2.8f, 0.0f, -6.0f);// Place the triangle Left
        glTranslatef(0.0f, fTranslate, 0.0f);// Translate in Y direction
        Draw_Table();// Draw Table
        glPopMatrix();

        glPushMatrix();
        glTranslatef(0.0f, 0.0f, -6.0f);// Place the triangle at Center
        glRotatef(fRotate, 0, 1.0f, 0);// Rotate around Y axis
        //glTranslatef(-0.5f, 0.0f, 0.0f);
        //glRotatef(GLfloat theta, GLfloatx, GLfloat y, GLfloat z);参数为角度和旋转轴
        Draw_Table();// Draw Table
        glPopMatrix();

        glPushMatrix();
        glTranslatef(2.0f, 0.0f, -6.0f);// Place the triangle at Right
        glScalef(fScale, fScale, fScale);// glScalef(GLfloat x, GLfloat y, GLfloatz);
    //参数为各坐标缩放程度
        Draw_Table();// Draw Table
        glPopMatrix();

        fTranslate += 0.001f;
        fRotate += (0.5/PI);
        fScale -= 0.001f;

        if (fTranslate > 0.9f) fTranslate = 0.0f;
        if (fScale < 0.4f) fScale = 1.0f;
        glutSwapBuffers();
}
```

五、结果



六、心得分析

有了上次使用opengl的经验，这次就对函数细节没有那么纠结了，感觉更多的是在复习大一学的线代😂，特别是对于矩阵的理解，以前学的时候只是单纯的去进行计算，并不是很懂其中的几何本质，也没有去思考。这次为了更好理解opengl变换中的原理，在网上找了些资料，认真学习了一下，对于以前学过的矩阵的性质如行列式、秩、逆矩阵等有了更直观的理解，也对OpenGL中函数原型有了更深入理解。

也花了一些功夫在www.khronos.org网站上阅读像`glMatrixMode()` `glPolygonMode()` `gluPerspective()` `glutReshapeFunc(reshape)`这些函数的源代码，虽然懂了大概怎么用，但总有一种云里雾里的感觉，可能现在计算机基础还不够吧 😅

总的来说，通过这次实验所用到的变换，差不多可以做出一些比较有趣的画面了，比如流星、降雨等，当然要做出好的效果，仅有这些是不够的，纹理和光照等进阶知识还需要进一步学习。