**附件一：**

**作业报告**

**三维图形变换**

课程名称：计算机图形学

任课教师：王 欣

学 号：16201319

姓 名：徐晋国

2023年1月14日

1. **程序功能及操作简介**

**功能一：用改良的Bresenham算法实现两点间画线**

**功能二：在二维平面上显示立方体的斜二测投影、斜等轴投影和正投影**

**功能三：通过键鼠操作，实现立方体三个轴向的旋转和平移**

**操作：具体如UI界面左边显示，内容如下：**

1. **按下三个按钮可以在斜二测投影、斜等轴投影、正投影之间任意切换**
2. **w，s键使立方体沿X轴前后移动**
3. **a，d键使立方体沿Y轴前后移动**
4. **t，g键使立方体沿Z轴前后移动**
5. **按住Ctrl+鼠标左键拖动，绕X轴正向旋转**
6. **按住Shift+鼠标左键拖动，绕X轴逆向旋转**
7. **按住Ctrl+鼠标中键拖动，绕Y轴正向旋转**
8. **按住Shift+鼠标中键拖动，绕Y轴逆向旋转**
9. **按住Ctrl+鼠标右键拖动，绕Z轴正向旋转**
10. **按住Shift+鼠标右键拖动，绕Z轴逆向旋转**
11. **核心代码及运行效果截图**

使用python语言，代码总体框架如下图，定义了Bresenham函数和Cube和UI两个类，使得主函数非常简洁明了。



**下面介绍几个核心的代码**

1. **Bresenham函数**

对书上的示例进行了优化，可以对任意斜率的两点路径进行计算。该函数以数组的形式输出路径点的横纵坐标，便于后面的画图。

1. # Bresenham算法计算路径点坐标
2. **def** Bresenham(X0, Y0, X1, Y1):
3. x = np.array([])
4. y = np.array([])
5. dx = abs(X1 - X0)
6. dy = abs(Y1 - Y0)
7. **if** X0 < X1:
8. sx = 1
9. **else**:
10. sx = -1
11. **if** Y0 < Y1:
12. sy = 1
13. **else**:
14. sy = -1
15. **if** dx > dy:
16. erro = dx / 2
17. **else**:
18. erro = -dy / 2
19. **while** X0 != X1 **or** Y0 != Y1:
20. x = np.append(x, X0)
21. y = np.append(y, Y0)
22. e2 = erro
23. **if** e2 > -dx:
24. erro -= dy
25. X0 += sx
26. **if** e2 < dy:
27. erro += dx
28. Y0 += sy
29. **return** x, y

**2.投影函数**

通过投影矩阵，计算出空间点投影在xy平面上的坐标（m,n），根据推导，斜二测投影矩阵为[[1,0,0,0],[0,1,0,0],[-0.354,-0.354,0,0].[0,0,0,1]]；

斜等轴投影矩阵为[[1,0,0,0],[0,0.707,0,0],[0,707,1,0,0],[0,0,0,1]]；

正投影矩阵为[[1,0,0,0],[0,1,0,0],[0,0,0,0],[0,0,0,1]]。

1. # 斜二测投影
2. **def** Oblique\_Twomeasure\_Projection(self):
3. self.m = np.around(self.X - 0.354 \* self.Z)
4. self.n = np.around(self.Y - 0.354 \* self.Z)
6. # 斜等轴投影
7. **def** Oblique\_Isometric\_Projection(self):
8. self.m = np.around(self.X + 0.707 \* self.Z)
9. self.n = np.around(self.Y + 0.707 \* self.Z)
11. # 正投影
12. **def** Orthographic\_Projection(self):
13. self.m = np.around(self.X)
14. self.n = np.around(self.Y)

**3.平移函数**

只需要将原来空间点的坐标加上三个方向的变化量，就实现了坐标的平移。

1. **def** Translation(self, Dx, Dy, Dz):
2. self.X += Dx
3. self.Y += Dy
4. self.Z += Dz

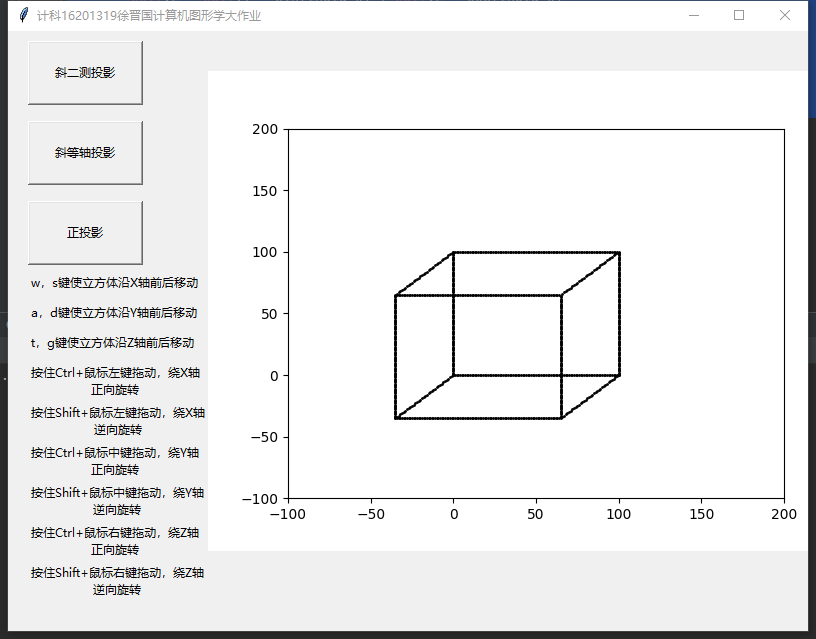
**4.旋转函数**

通过旋转矩阵，输入绕三个轴旋转的角度，依次对坐标进行旋转变换即可。

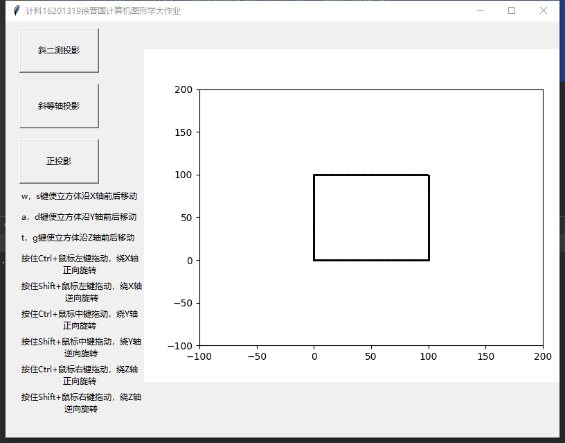
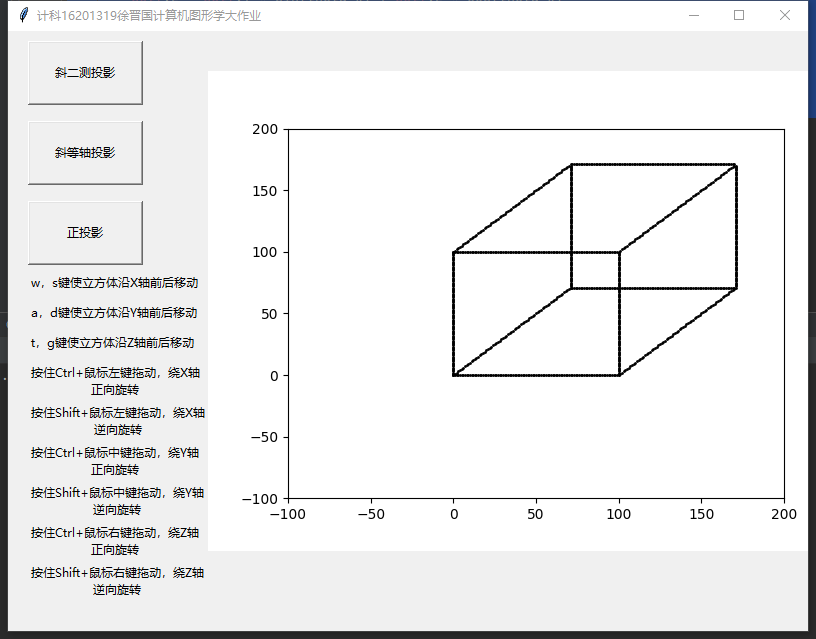
1. # 旋转变换
2. **def** Rotation(self, theta\_x, theta\_y, theta\_z):
3. # 绕x轴旋转
4. self.Y = self.Y \* cos(theta\_x) - self.Z \* sin(theta\_x)
5. self.Z = self.Y \* sin(theta\_x) + self.Z \* cos(theta\_x)
6. # 绕y轴旋转
7. self.X = self.X \* cos(theta\_y) + self.Z \* sin(theta\_y)
8. self.Z = -self.X \* sin(theta\_y) + self.Z \* cos(theta\_y)
9. # 绕z轴旋转
10. self.X = self.X \* cos(theta\_z) - self.X \* sin(theta\_z)
11. self.Y = self.X \* sin(theta\_z) + self.Y \* cos(theta\_z)

**运行效果截图**

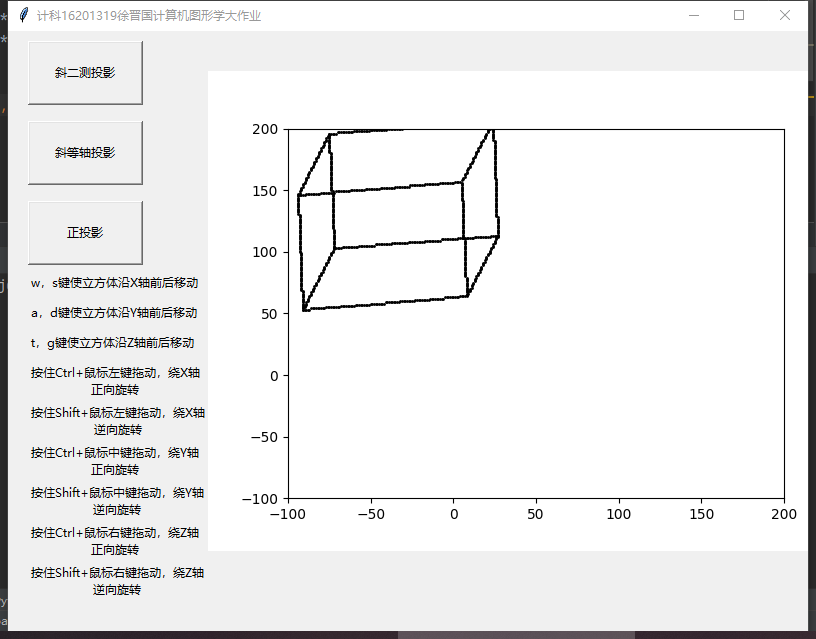
开始，默认以斜二测投影显示：



点击左侧按钮切换投影，功能正常：



在经过平移旋转一系列操作后，正常显示：



**三、技术难点分析**

1.如何画图？

因为不能调用现成的画线函数，又要模拟像素的连点成线，所以采用了matplotlib库中绘制散点图的函数。将Bresenham函数输出的路径点绘制成散点图，这样一来在视觉上就有了直线的效果，模拟了像素的连点成线。但要求是路径点数量必须足够多，否则会有强烈的锯齿感，所以在初始化立方体参数的时候，我将立方体的边长设置为100，画布的横纵坐标固定为（-100,200）区间，显示效果好。

2.如何更新图像？

我是用python中的tkinter库来做图形化界面的，为了使键鼠操作绑定事件的结果能显示在画布上，我设置了一个死循环调用plot（）函数，每20毫秒清除上一幅图像并根据新的坐标重新绘图。由于程序计算的速度很快而且问题的规模也很小，肉眼看起来就是比较连贯的效果。