中山大學
SUN YAT-SEN UNIVERSITY

《计算机图形学》

实验报告

（作业二）

学 院 名 称	：	数据科学与计算机学院
专业（班级）	：	17 计科 7 班
学 生 姓 名	：	张富瑞
学　　号	：	17341197
时　　间	：	2019 年 12 月 6 日

一.实验要求

2.1 绘制你的机器人

使用 GL_POINTS, GL_TRIANGLES, GL_QUADS, GL_POLYGON 等基本图元，结合平移、旋转函数绘制一个机器人。

机器人应该有头、躯干、四肢等基本部分。

2.2 绘制机器人的运动线路

使用平移、旋转函数使你的机器人沿固定线路运动。

线路可以是圆或任意其他闭合路径。

2.3 绘制机器人的动作

使用平移、旋转函数绘制机器人的动作。

机器人在运动过程中应具有摆臂及抬腿两个基本动作。

二.实验过程

1.绘制机器人的头部

我的机器人的头部是一个球体，所以首先要编写一个绘制球体的函数。

```
void drawSphere(float r, float M, float N)
{
    float step_z = PI / M;              //每次在 z 轴上增加的角度
    float step_xy = 2 * PI / N;         //每次在 x,y 平面上增加的
角度
    float x[4], y[4], z[4];

    float angle_z = 0.0;
    float angle_xy = 0.0;
    int i = 0, j = 0;
    glColor3f(0.95f, 0.82f, 0.20f);
    glBegin(GL_QUADS);
    for(i = 0; i < M; i++)
    {
        angle_z = i * step_z;           //角度递增

        for(j = 0; j < N; j++)
        {
            angle_xy = j * step_xy;     //角度递增
            //绘制空间中的四边形
            x[0] = r * sin(angle_z) * cos(angle_xy);
            y[0] = r * sin(angle_z) * sin(angle_xy);
            z[0] = r * cos(angle_z);

            x[1] = r * sin(angle_z + step_z) * cos(angle_xy);
            y[1] = r * sin(angle_z + step_z) * sin(angle_xy);
```

```
            z[1] = r * cos(angle_z + step_z);

            x[2] = r * sin(angle_z + step_z) * cos(angle_xy +
step_xy);
            y[2] = r * sin(angle_z + step_z) * sin(angle_xy +
step_xy);
            z[2] = r * cos(angle_z + step_z);

            x[3] = r * sin(angle_z) * cos(angle_xy + step_xy);
            y[3] = r * sin(angle_z) * sin(angle_xy + step_xy);
            z[3] = r * cos(angle_z);
            for(int k = 0; k < 4; k++)
            {
                glVertex3f(x[k], y[k], z[k]);
            }
        }
    }
    glEnd();
}
```

其中球的中心是在(0,0)点上。绘制球的方法是利用球的极坐标方程来得到球上一些点的坐标，然后利用微分的思想，将球的表面进行分块，绘制多边形，这样在分块之后，随着N的增大，球就会显得更圆。这里我选择分别分成M,N块，通过极坐标计算在空间坐标系中的坐标，选取绘图的模式为GL_QUADS,然后在当前的四个点上绘制四边形。绘制完成后，就是一个球体。

2.绘制立方体

我的机器人的其他部分都是由立方体组成，所以我选择先写出一个函数，来绘制一个基本的立方体，然后利用glScalef的方法来进行缩放，形成颜色和长宽高不同的躯体。

```
void draw_cube()
{
    glPushMatrix();
    glBegin(GL_QUADS);

    // 绘制上面
    glVertex3f(2.0f, 2.0f, 2.0f);
    glVertex3f(2.0f, 2.0f, -2.0f);
    glVertex3f(-2.0f, 2.0f, -2.0f);
    glVertex3f(-2.0f, 2.0f, 2.0f);
    // 绘制下面
    glVertex3f(2.0f, -2.0f, 2.0f);
    glVertex3f(2.0f, -2.0f, -2.0f);
    glVertex3f(-2.0f, -2.0f, -2.0f);
```

```
    glVertex3f(-2.0f, -2.0f, 2.0f);
    // 绘制前面
    glVertex3f(2.0f, 2.0f, 2.0f);
    glVertex3f(-2.0f, 2.0f, 2.0f);
    glVertex3f(-2.0f, -2.0f, 2.0f);
    glVertex3f(2.0f, -2.0f, 2.0f);
    // 绘制后面
    glVertex3f(2.0f, 2.0f, -2.0f);
    glVertex3f(-2.0f, 2.0f, -2.0f);
    glVertex3f(-2.0f, -2.0f, -2.0f);
    glVertex3f(2.0f, -2.0f, -2.0f);
    // 绘制左面
    glVertex3f(-2.0f, 2.0f, 2.0f);
    glVertex3f(-2.0f, 2.0f, -2.0f);
    glVertex3f(-2.0f, -2.0f, -2.0f);
    glVertex3f(-2.0f, -2.0f, 2.0f);
    // 绘制右面
    glVertex3f(2.0f, 2.0f, 2.0f);
    glVertex3f(2.0f, 2.0f, -2.0f);
    glVertex3f(2.0f, -2.0f, -2.0f);
    glVertex3f(2.0f, -2.0f, 2.0f);

    glEnd();
    glPopMatrix();
}
```

这里绘制的是以(0,0,0)为中心的立方体，边长为4。

绘制机器人的其他部分就只需要进行，颜色以及缩放的比例的修改，然后进行拼接。

```
void draw_sth(float x,float y,float z,float a,float b,float c)
{
    glPushMatrix();
    glColor3f(a,b,c);
    glScalef(x, y, z);
    draw_cube();  //调用绘制最基本的立方体函数
    glPopMatrix();
}
```

3.视角准备

```
void MyGLWidget::paintGL()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
```

```
    glLoadIdentity();
    gluPerspective(85, width()/height(), 1, 500);//设置透视投影

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0, 80, 80, 40*cos(2*PI/360*th), 0, 40*sin(2*PI/360*th), 0, 1, 0);//定义视图
```

这里首先清除颜色缓冲，然后将当前矩阵指定为投影矩阵，并把矩阵设为单位矩阵，设置好合适的透视投影，视角设为85°。之后切换到模型视图矩阵，设置好单位矩阵，定义视图，其中观察位置设为(0,80,80)，目标位置为机器人的当前位置。这样可以跟着机器人的移动改变视角。看起来更有立体感。

4.路线的绘制

```
 //绘制机器人行走的路径，路径为r=40 的圆，中心为(0,0,0)
    glColor3f(1.0f, 0.0f, 0.0f);
    glBegin(GL_LINE_LOOP);
    for (int i=0; i<360; ++i)
    {
        double theta = 2*PI/360*i;
        glVertex3f(40 * cos(theta), 0,40 * sin(theta));
        //printf("%f %f\n", cos(theta), sin(theta));

    }
    glEnd();
```

这里和绘制头部的球体方法类似，也是利用微分的思想来进行圆的绘制。先利用极坐标的方程来得到绘制的点的坐标，然后将模式改为GL_LINE_LOOP的模式，绘制好之后就变成了圆。

5.机器人身体各部分的绘制

由于绘制的代码重复，这只展示左手的绘制，同时讲解左手摆动的原理。

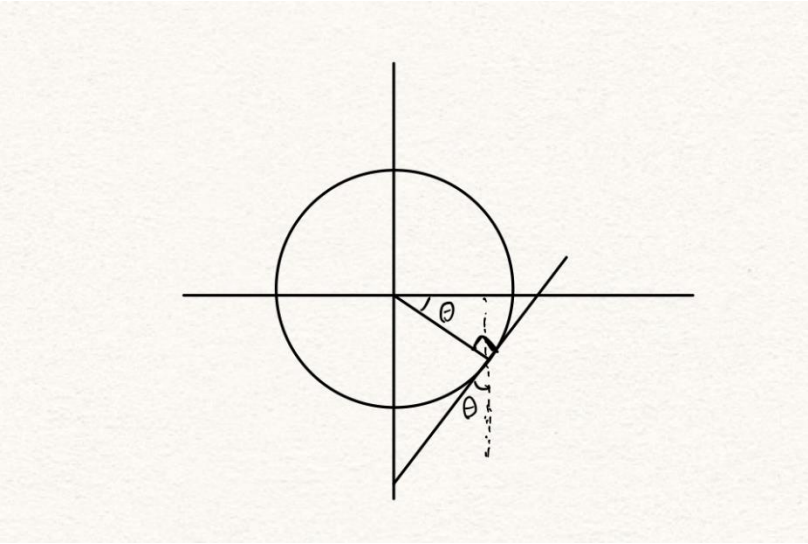
```
//绘制左手
    glPushMatrix();
    glTranslatef(6.0f,30.0f,0.0f);
    left_arm_angle+=arm_angle*left_arm_plus; //进行手部的摆动
    if (left_arm_angle*left_arm_plus >= arm_bound) //判断手部摆动的方向
    {
        left_arm_plus*=(-1);
    }
    glTranslatef(0.0f,6.0f,0.0f);
    glRotatef(left_arm_angle,1.0f,0.0f,0.0f);
```

```
    glTranslatef(0.0f,-6.0f,0.0f);
    draw_sth(1.0f, 3.0f, 1.0f,0.35f, 0.82f, 0.69f);
    glPopMatrix();
```

由于Opengl的特性，我们首先在(0,0,0)处使用之前写好的函数来进行左手的绘制，由于绘制好之后的左手是中心为(0，0，0)，这里为了体现手臂的摆动，我们将手臂下移，使得手臂的上界处于ZOX平面，然后旋转到指定的角度。角度这里，我们首先是已经规定了手臂初始的角度，以及可以摆动的最大角度和每一次绘制，角度的改变量。每一次绘制之后，下一次的角度都会进行改变。同时在达到边界之后，手臂转动的方向也会进行改变，这里我们进行一次判定即可。然后我们将旋转完成后的手臂由平移回手臂该放置的位置。机器人的身体的其他部分也是同样得到。

6.机器人位置的平移

由于我选择的线路是一个圆，类似于手臂的摆动，我用一个值来记录下当前的角度，然后利用极坐标的方式来得到机器人在圆上应该处于的位置，之后进行平移。同时我们可以根据角度来算出圆的切线角度，这样我们可以实现机器人的转向。之后再进行角度的增加。



由上图我们可以看出切线的角度为θ和圆上的角度相同。所以我们只需要对绘制好的整体的机器人逆时针旋转θ度即可。

```
//利用极坐标的方法，将机器人旋转到圆的切线角度，在将机器人放置到圆的指定位置上
    th%=360;
    th+=1;
    printf("%f %f %f\n", float(th),cos(2*PI/360*th), sin(2*PI/360*th));
    glTranslatef(40*cos(2*PI/360*th),0,40*sin(2*PI/360*th));
    glRotatef(-th,0.0f,1.0f,0.0f);
```

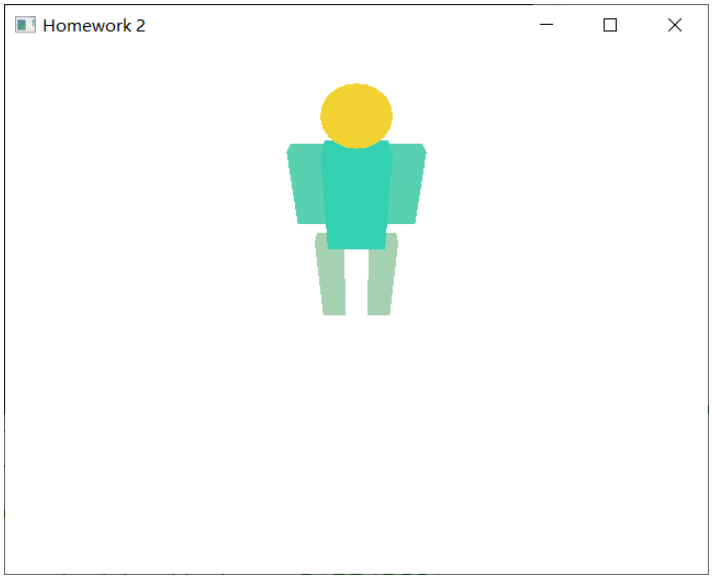
7.总体流程

由于opengl的特性，我们是先绘制机器人的身体。就是利用我前面已经讲到的函数，在朝向z轴的方向上进行绘制出一个拥有完整形体的机器人。这里的机器人拥有不同的摆臂和摆腿，然后根据我绘制的路线，将机器人整体平移到指定的位置，每一次绘制的角度递增，在路线上的位置也不同，同时根据角度来进行机器人朝向的计算，旋转朝向，这样以后根

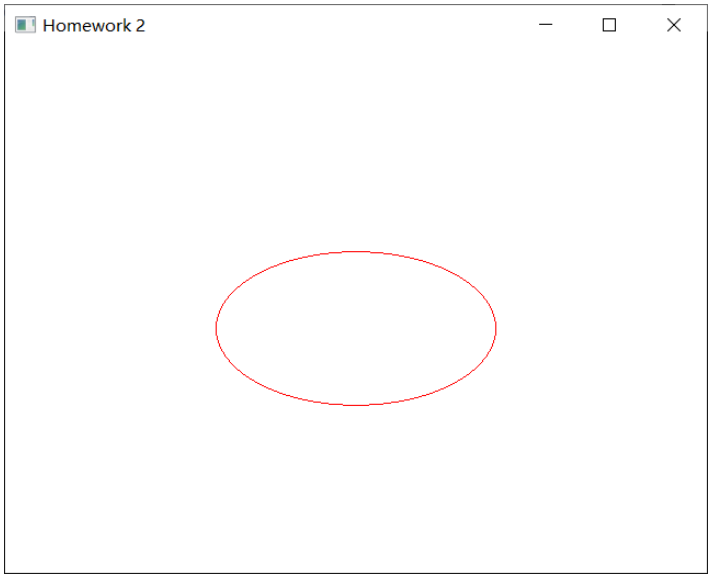
据循环的绘制，我们就可以看出绕着路线行走的机器人。

三.实验结果

初始的机器人



初始绘制的路线



在路线上行走的机器人

四.实验总结

这次实验总的来说算不上难(也有可能我选择完成的方式不够难)，但是在实际的编写过程中也遇到了许多困难。

由于编写opengl的程序较少，所以在实现这次作业之前对于如何完成机器人的行走没有太多的思路，经过一番的查询资料，我才逐渐意识到，由于我们要实现的那段代码是会循环着执行，所以我们才可以看到动态的行走。通过不同循环次数中的位置和角度的变换。

同时在写出第一个画立方体的函数之后，我发现由于视角的设置问题，我看到的只是一团空白。虽然理论课上也进行过学习，但是没有较多的实际编写，所以我还是没有能够很好的设置视角。又是一番查询资料，才逐渐完成好初始视角的设置的问题。

绘制立方体这些东西还算是没有什么难度，但是对于绘制球体，我一开始也是没有什么很好的想法。经过同学的提示，我开始理解了利用极坐标的方程以及微分的思想绘制球体，虽然画好之后觉得没啥难度，但是在画成之前，还是觉得比较困难。画好了球体之后，画圆的路线就没有难度了，之后的实现都是谈不上困难。只是opengl的操作特性，由后往前写代码让人有些难受。

经过这次实验，我对于opengl的使用有了更深的理解，以前觉得十分困难的操作，现在也能够进行编写。