作业报告

院系：数据科学与计算机学院　　专业：计算机类　　科目：计算机图形学
姓名：郑康泽　　学号：17341213

一. 作业题目

绘制一个沿固定线路运动的机器人

绘制过程中的平移、旋转可以使用 glTranslate()、glRotate()、glMultMatrix()等函数。

二. 作业要求

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. **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。
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即抬腿两个基本动作。
4. **（选做）加分项-模型载入**
   - 使用 mesh 模型（如 obj 文件）载入机器人模型。
   - 或载入其他 mesh 模型围绕机器人运动。

三. 主要算法及代码

1. **绘制你的机器人**
   1) 实现过程：
      A) 因为用各种比例的长方体来代表机器人的头、躯干以及四肢，所以需要一个基本图元，就是正方体。利用 GL_POLYGON 绘制一个中心在原点、边长为 2 的实心正方体；
      B) 在 A) 步骤的正方体的基础，通过放缩，形成机器人的各部分。设计头在(x, y, z)上的放大比例为$(2, 2, 2)$，躯干在(x, y, z)的放大比例为$(3, 5, 2)$，手臂在(x, y, z)的放大比例为$(1, 6, 1)$，腿在(x, y, z)的放大比例为$(1, 7, 1)$；
      C) 有了 B) 步骤中的机器人的各部分，现在只要通过平移到正确位置即可形成机器人的大致模样。首先确定机器人是在 xOz 平面上运动，并且$z \geq 0$，所以机器人的腿的长方体的在 z 轴上最小的坐标应该是 0，因为该长方体在 z 轴上最小的坐标为-7，所以需要朝 z 轴正方向移动 7。并且机器人的腿有两条，所以在 x 轴上也需要平移，因为躯干的长方体在 x 轴的坐标范围为-3 到 3，而腿的长方体在 x 轴的坐标范围为-1 到 1，所以调整左腿（我们的视角的左边）的长方体朝 x 轴负方向移动 2，使得左腿的长方体的左面与躯干的长方体的左面相平，同理，调整右腿的长方体朝 x 轴正方向移动 2。这样我们就调整好了腿部。
      D) 经过 C) 步骤后，腿的长方体在 y 轴上最大坐标为 14，而没有调整过的躯干的

长方体在 y 轴上最小坐标为-5，所以将躯干的长方体朝 y 轴的正方向移动 19，使得躯干的长方体的下面与腿的长方体的上面相平；

E) 经过 D) 步骤后，躯干的长方体在 y 轴上最大坐标为 24，而没有调整过的头的长方体在 y 轴上最小坐标为-2，所以将头的长方体朝 y 轴的正方向移动 26，使得躯干的长方体的上面与头的长方体的下面相平；

F) 经过 D) 步骤后，躯干的长方体在 y 轴上最大坐标为 24，而没有调整过的手臂的长方体在 y 轴上最大坐标为 6，所以将手臂的长方体朝 y 轴的正方向移动 18，同机器人的腿的移动原理，将左手臂的长方体朝 x 轴负方向移动 4，将右手臂的长方体朝 x 轴正方向移动 4，这样就调整完了机器人各部分的位置了。

2) 代码展示：

A) 绘制中心在原点，边长为 2 的正方体：

```
/*######################################################
##   函数: draw_cube
##   函数描述:  绘制中心在原点，边长为2的正方体
##   参数描述:  无
######################################################*/
void draw_cube()
{
    glPushMatrix();
    glBegin(GL_POLYGON);
    // 正面
    glVertex3f(1.0f, 1.0f, 1.0f);
    glVertex3f(-1.0f, 1.0f, 1.0f);
    glVertex3f(-1.0f, -1.0f, 1.0f);
    glVertex3f(1.0f, -1.0f, 1.0f);
    // 后面
    glVertex3f(1.0f, 1.0f, -1.0f);
    glVertex3f(-1.0f, 1.0f, -1.0f);
    glVertex3f(-1.0f, -1.0f, -1.0f);
    glVertex3f(1.0f, -1.0f, -1.0f);
    // 上面
    glVertex3f(1.0f, 1.0f, 1.0f);
    glVertex3f(1.0f, 1.0f, -1.0f);
    glVertex3f(-1.0f, 1.0f, -1.0f);
    glVertex3f(-1.0f, 1.0f, 1.0f);
    // 下面
    glVertex3f(1.0f, -1.0f, 1.0f);
    glVertex3f(1.0f, -1.0f, -1.0f);
    glVertex3f(-1.0f, -1.0f, -1.0f);
    glVertex3f(-1.0f, -1.0f, 1.0f);
    // 右面
    glVertex3f(1.0f, 1.0f, 1.0f);
    glVertex3f(1.0f, 1.0f, -1.0f);
    glVertex3f(1.0f, -1.0f, -1.0f);
    glVertex3f(1.0f, -1.0f, 1.0f);
    // 左面
    glVertex3f(-1.0f, 1.0f, 1.0f);
    glVertex3f(-1.0f, 1.0f, -1.0f);
    glVertex3f(-1.0f, -1.0f, -1.0f);
    glVertex3f(-1.0f, -1.0f, 1.0f);
    glEnd();
    glPopMatrix();
}
```

B) 绘制头部的长方体（不包含平移，下同）：

```
/*########################################################
##   函数: draw_head
##   函数描述:  绘制头部正方体
##   参数描述:  无
########################################################*/
void draw_head()
{
    glPushMatrix();
    // 颜色
    glColor3f(0.0f, 1.0f, 0.0f);
    // 放大比例
    glScalef(2.0f, 2.0f, 2.0f);
    draw_cube();
    glPopMatrix();
}
```

C) 绘制躯干的长方体：

```
/*#####################################################
##   函数: draw_body
##   函数描述:  绘制躯干部正方体
##   参数描述:  无
#####################################################*/
void draw_body()
{
    glPushMatrix();
    // 颜色
    glColor3f(1.0f, 1.0f, 1.0f);
    // 放大比例
    glScalef(3.0f, 5.0f, 2.0f);
    draw_cube();
    glPopMatrix();
}
```

D) 绘制手的长方体：

```
/*#####################################################
##   函数: draw_arm
##   函数描述:  绘制手臂正方体
##   参数描述:  无
#####################################################*/
void draw_arm()
{
    glPushMatrix();
    //glColor3f(1.0f, 0.84f, 0.0f);
    // 放大比例
    glScalef(1.0f, 6.0f, 1.0f);
    draw_cube();
    glPopMatrix();
}
```

E) 绘制腿的长方体：

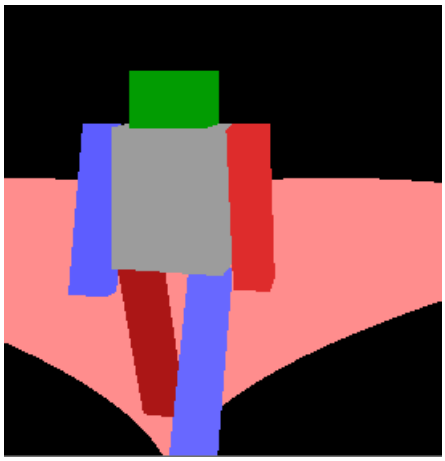
```
/*#####################################################
##   函数: draw_leg
##   函数描述:  绘制腿部正方体
##   参数描述:  无
#####################################################*/
void draw_leg()
{
    glPushMatrix();
    //glColor3f(0.0f, 0.0f, 1.0f);
    // 放大比例
    glScalef(1.0f, 7.0f, 1.0f);
    draw_cube();
    glPopMatrix();
}
```

F) 在 paintGL()中调用上述函数形成机器人：

```
// 画头
glPushMatrix();
glTranslatef(0.0f, 26.0f, 0.0f);
draw_head();
glPopMatrix();

// 画躯干
glPushMatrix();
glTranslatef(0.0f, 19.0f, 0.0f);
draw_body();
glPopMatrix();

// 画左手
glPushMatrix();
glTranslatef(4.0f, 18.0f, 0.0f);
glColor3f(1.0f, 0.0f, 0.0f);
draw_arm();
glPopMatrix();
```

```
// 画右手
glPushMatrix();
glTranslatef(-4.0f, 18.0f, 0.0f);
glColor3f(0.0f, 0.0f, 1.0f);
draw_arm();
glPopMatrix();

// 画左腿
glPushMatrix();
glTranslatef(2.0f, 7.0f, 0.0f);
glColor3f(0.0f, 0.0f, 1.0f);
draw_leg();
glPopMatrix();

// 画右腿
glPushMatrix();
glTranslatef(-2.0f, 7.0f, 0.0f);
glColor3f(1.0f, 0.0f, 0.0f);
draw_leg();
glPopMatrix();
```

3) 结果展示：（展示的是运动的机器人，因为改回不动的太麻烦了）



2. 绘制机器人的运动线路

1) 实现思路：

A) 我设计的是心形的路线，首先需要知道心形的参数方程如下：

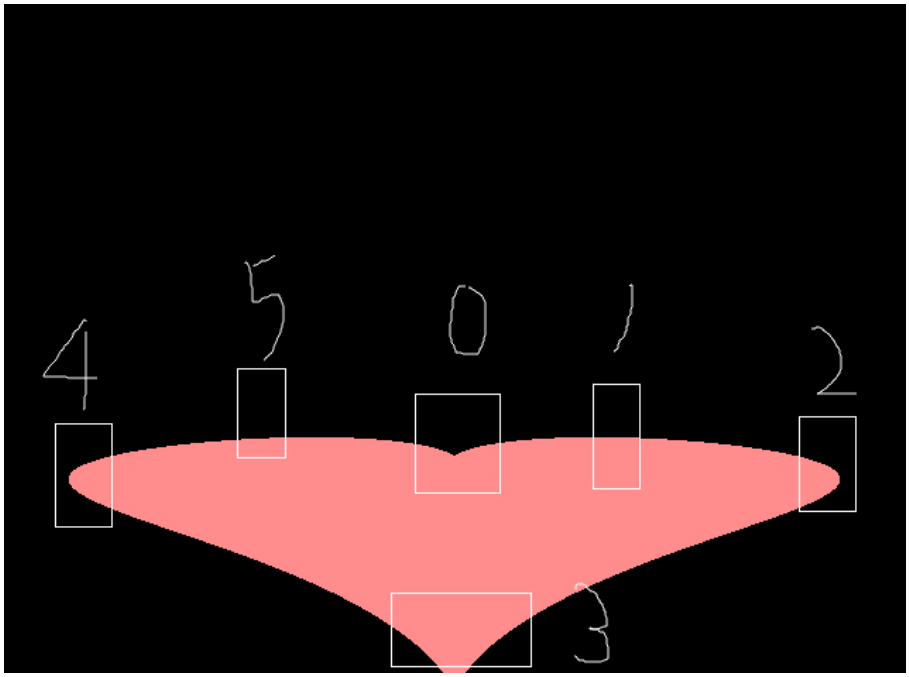
$$\begin{cases} x(\theta) = R * 16 * \sin^3\theta \\ y(\theta) = R * (-13 * \cos\theta + 5 * \cos 2\theta + 2 * \cos 3\theta + 4 * \cos 4\theta) \end{cases}, 0 \le \theta \le 2\pi$$

画路径很简单，for 循环迭代每个θ，然后算出每个θ对应的x和y，然后利用 GL_POLYGON 绘出心形；

B) 对于机器人，只需要利用上次θ，加上每一次的增量，得到当前的θ'，然后算出

θ'对应的x和y，然后通过平移整个机器人到目标位置即可，不过在本次作业，是在 xOz 平面，所以将以上y改成z即可；

C) 最难的是机器人的朝向问题，单纯利用dz/dx是不行的，因为会出现转的角度刚好相反，即差个 180°，呈现的效果也是非常滑稽，整个机器人在瞎转。所以需要分析dz/dx和旋转的角度的关系，首先看下图：



假设机器人从 0 出发，并且假设机器人初始状态是面向我们即朝向z轴正方向。通过编号将爱心分成 6 段，首先处理这六个特殊点，这 6 段可以通过 B) 中的θ'分开。对于 0 和 3 这两个点，dx和 dz都为 0，不能通过atan (dx/dz)求得斜率并得知机器人朝向，所以设定机器人在 0 的时候朝向 x 轴正方向即绕 y 轴逆时针转 90°，在 3 的时候朝向 x 轴负方向即绕 y 轴顺时针转 90°；对于 1 和 5 这两个点，dy为 0，机器人的朝向应是一样的，所以机器人绕 y 轴逆时针旋转 90°；对于 2 和 4 这两个点，dx为 0，机器人的朝向应是相反的，在 2 时机器人不用旋转，在 4 时机器人绕 y 轴旋转 180°。

处理完了特殊点，再来处理各个区间。在 0 到 1 区间和 4 到 5 区间是一样的，计算$\theta = \text{atan}\left(\frac{dx}{dz}\right) * 180/\pi$，这个$\theta$应该是负的，因为$dz$为负，这里我们不能让机器人直接绕 y 轴逆时针旋转 θ，因为这样是绕 y 轴顺时针旋转$-\theta$，所以我们要让机器人绕 y 轴逆时针旋转$180^\circ + \theta$；在 1 到 2 区间和 5 到 0 区间也是一样的，同样计算θ，这个θ是正的，这里我们可以让机器人直接绕 y 轴逆时针旋转θ；在 2 到 3 区间，计算θ，这个θ是负的，应该让机器人绕 y 轴逆时针旋转θ，相当于绕 y 轴顺时针旋转$-\theta$；在 3 到 4 区间上，计算θ，这个θ是正的，应该让机器人绕 y 轴顺时针旋转θ，也可以绕 y 轴逆时针旋转$\theta - 180^\circ$。

经过上述处理，机器人的旋转看起来正常多了。注意，以上旋转都是机器人在原点的时候，还没平移到爱心相应的位置上。

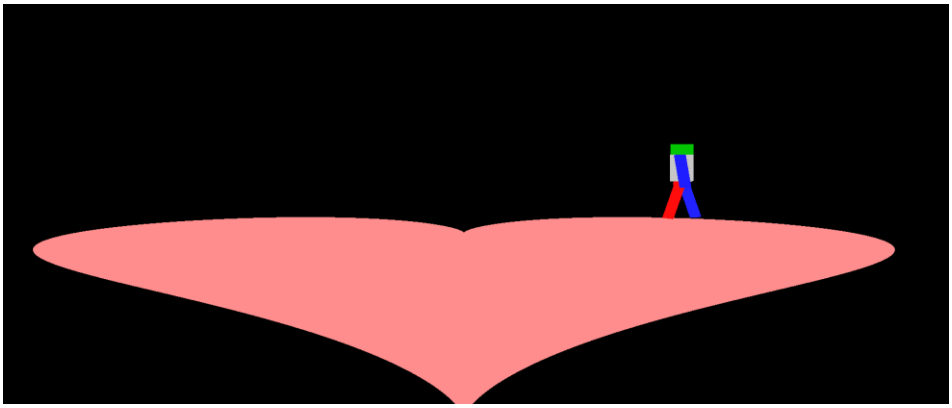
2) 代码展示：

```
// 更新当前运动轨迹的参数即角度theta
cur_angle = (cur_angle + step)>359?0:(cur_angle + step);
// 计算坐标
double theta = 2 * PI * cur_angle / 360;
cur_x = 4 * 16 * pow(sin(theta), 3);
cur_z = 4 * (-13*cos(theta) + 5*cos(2*theta) + 2*cos(3*theta) + cos(4*theta));

// 画路径
draw_path();

// 将机器人移动当前运动到坐标
glTranslated(cur_x, cur_y, cur_z);
```

```
// 调整朝向
double dx = 4 * 16 * 3 * pow(sin(theta), 2) * cos(theta) * PI / 180;
double dy = 4 * (13*sin(theta) - 10*sin(2*theta) - 6*sin(3*theta) - 4*sin(4*theta)) * PI / 180;
// 爱心中轴两点
if (fabs(dx) < 1e-3 && fabs(dy) < 1e-3)
{
    if (int(cur_angle) == 0)
        glRotatef(90, 0, 1, 0);
    else
        glRotatef(-90, 0, 1, 0);
}
// 爱心两个z轴坐标最小的两个点
else if (fabs(dy) < 1e-3)
    glRotated(90, 0, 1, 0);
// 爱心最两边的两个点
else if (fabs(dx) < 1e-3)
{
    if (int(cur_angle) == 270)
        glRotated(180, 0, 1, 0);
}
```

```
else
{
    if ((int(cur_angle) > 0 && int(cur_angle) < 90) ||
            ((int(cur_angle) > 270 && int(cur_angle) < 360)))
    {
        if (dx * dy < 0)
            glRotated(atan(dx/dy) * 180 / PI + 180, 0, 1, 0);
        else
            glRotated(atan(dx/dy) * 180 / PI, 0, 1, 0);
    }
    else if ((int(cur_angle) > 90 && int(cur_angle) < 180))
        glRotated(atan(dx/dy) * 180 / PI, 0, 1, 0);
    else
        glRotated(atan(dx/dy) * 180 / PI - 180, 0, 1, 0);
}
```

3) 结果展示：



3. 绘制机器人的动作

1) 实现思路：

我设计手臂摆的最大角度为 15°，腿抬的最大角度为 20°。因为摆臂和抬腿的过程都是相同的，所以这里只说明怎么摆臂。首先我们知道两只手摆的方向是不一样的，所以需要记录左右手臂的运动状态，是向前还是向后，在代码中是 arm_state 记录状

态，向前用 1 表示，向后用-1 表示；需要记录当前摆臂的角度，在代码中是 arm_angle 记录；然后每次先判断摆臂的角度|arm_angle|是否已经到达最大角度 max_angle，如果达到了，就需要调整 arm_state 为-arm_state，开始朝反方向摆臂，接下来就是利用 arm_state 更新当前 arm_angle。确定好了摆臂角度，我们需要先将手臂的长方体旋转 arm_angle 度，然后在平移到机器人正确的位置。对于抬腿的动作也是同理的，只不过当左手向前摆的时候，左腿应该是向后的，即同边的手腿的摆动方向是相反的。

2) 代码展示：

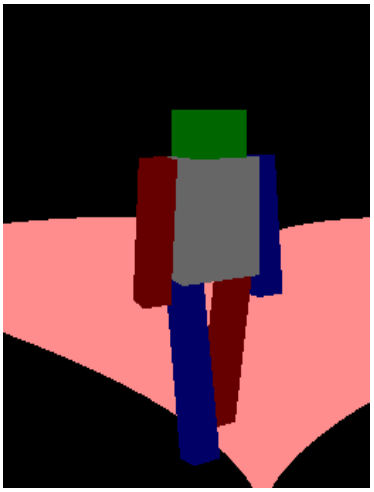
```
// 画左手
glPushMatrix();
glTranslatef(4.0f, 18.0f, 0.0f);
// 往前摆臂到最大角度，开始往后
if (left_arm_state == 1 && left_arm_angle >= max_arm_angle)
    left_arm_state = -1;
// 往后摆臂到最大角度，开始往前
if (left_arm_state == -1 && left_arm_angle <= -max_arm_angle)
    left_arm_state = 1;
// 更新摆的角度
left_arm_angle += left_arm_state * per_arm_angle;
// 旋转
glTranslatef(0.0f, 6.0f, 0.0f);
glRotatef(left_arm_angle, 1.0f, 0.0f, 0.0f);
glTranslatef(0.0f, -6.0f, 0.0f);
glColor3f(1.0f, 0.0f, 0.0f);
draw_arm();
glPopMatrix();
```

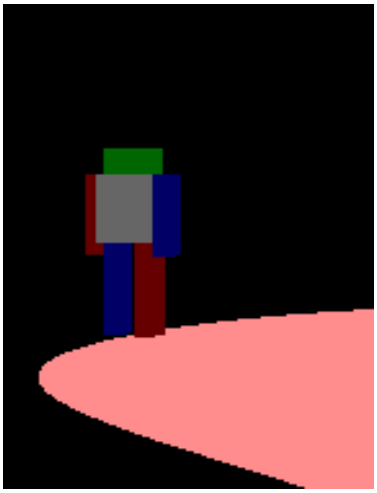
```
// 画右手
glPushMatrix();
glTranslatef(-4.0f, 18.0f, 0.0f);
// 往前摆臂到最大角度，开始往后
if (right_arm_state == 1 && right_arm_angle >= max_arm_angle)
    right_arm_state = -1;
// 往后摆臂到最大角度，开始往前
if (right_arm_state == -1 && right_arm_angle <= -max_arm_angle)
    right_arm_state = 1;
// 更新摆的角度
right_arm_angle += right_arm_state * per_arm_angle;
// 旋转
glTranslatef(0.0f, 6.0f, 0.0f);
glRotatef(right_arm_angle, 1.0f, 0.0f, 0.0f);
glTranslatef(0.0f, -6.0f, 0.0f);
glColor3f(0.0f, 0.0f, 1.0f);
draw_arm();
glPopMatrix();
```

```
// 画左腿
glPushMatrix();
glTranslatef(2.0f, 7.0f, 0.0f);
// 往前抬腿到最大角度，开始往后
if (left_leg_state == 1 && left_leg_angle >= max_leg_angle)
    left_leg_state = -1;
// 往后抬腿到最大角度，开始往前
if (left_leg_state == -1 && left_leg_angle <= -max_leg_angle)
    left_leg_state = 1;
// 更新抬的角度
left_leg_angle += left_leg_state * per_leg_angle;
// 旋转
glTranslatef(0.0f, 7.0f, 0.0f);
glRotatef(left_leg_angle, 1.0f, 0.0f, 0.0f);
glTranslatef(0.0f, -7.0f, 0.0f);
glColor3f(0.0f, 0.0f, 1.0f);
draw_leg();
glPopMatrix();
```

```
// 画右腿
glPushMatrix();
glTranslatef(-2.0f, 7.0f, 0.0f);
// 往前抬腿到最大角度，开始往后
if (right_leg_state == 1 && right_leg_angle >= max_leg_angle)
    right_leg_state = -1;
// 往后抬腿到最大角度，开始往前
if (right_leg_state == -1 && right_leg_angle <= -max_leg_angle)
    right_leg_state = 1;
// 更新抬的角度
right_leg_angle += right_leg_state * per_leg_angle;
// 旋转
glTranslatef(0.0f, 7.0f, 0.0f);
glRotatef(right_leg_angle, 1.0f, 0.0f, 0.0f);
glTranslatef(0.0f, -7.0f, 0.0f);
glColor3f(1.0f, 0.0f, 0.0f);
draw_leg();
glPopMatrix();
```

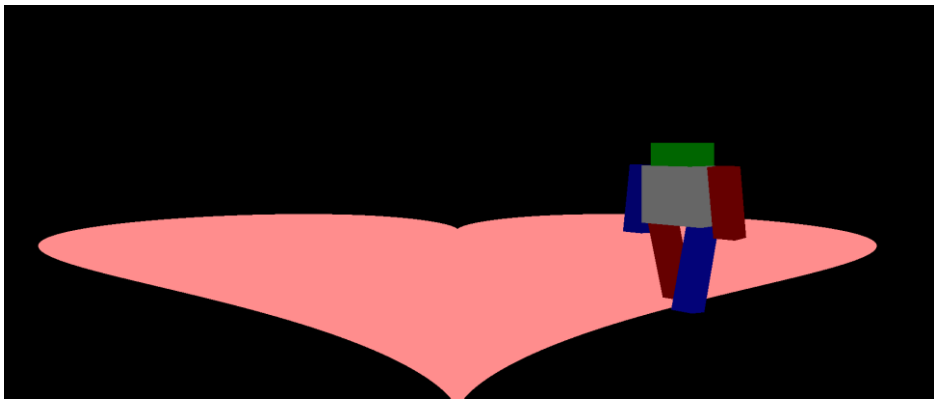
3） 结果展示

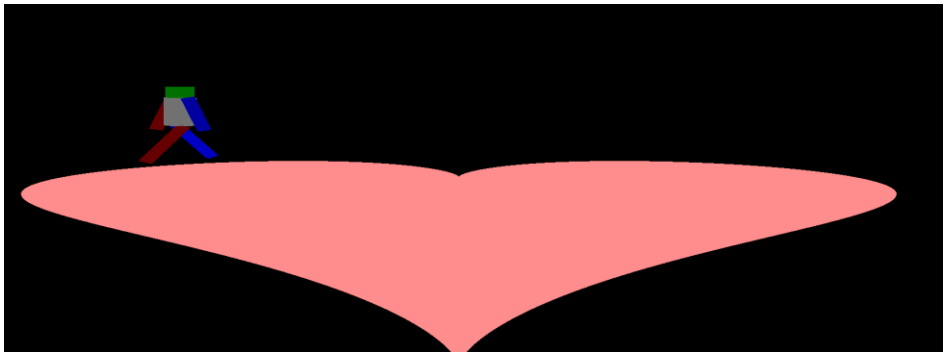


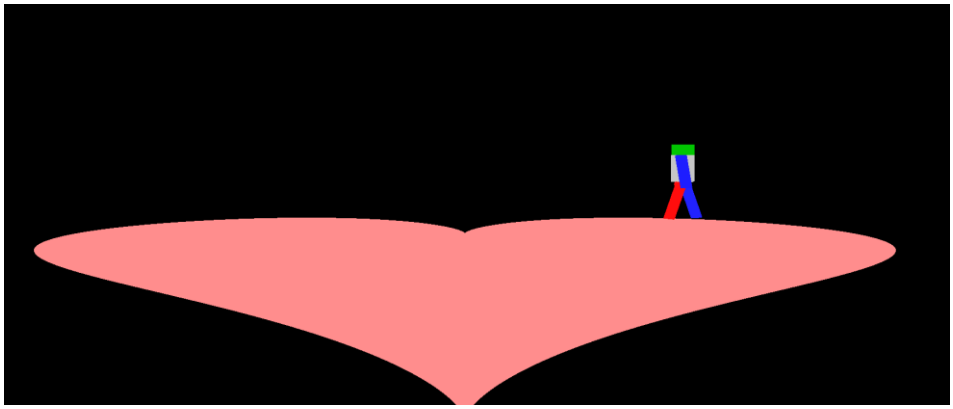


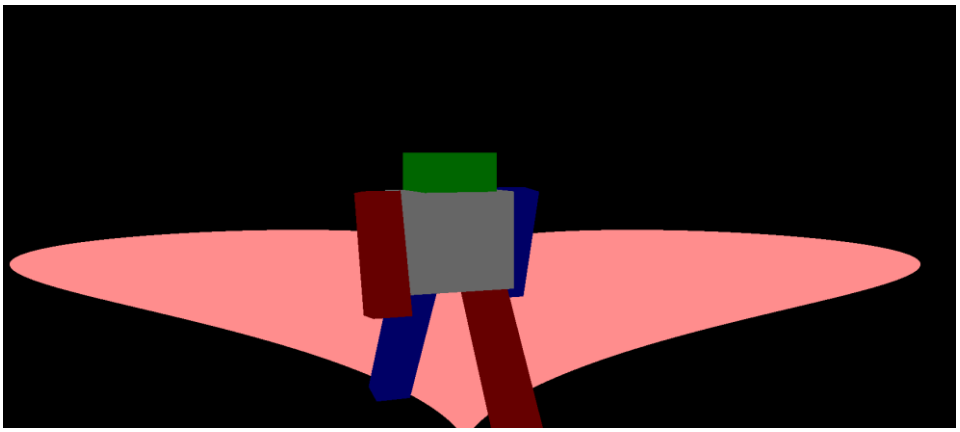
四． 运行结果截图

为了能够更好的显示出 3D 效果，我加了光照，立体的感觉也确实加强了。









五. 困难及解决

中山大學 SUN YAT-SEN UNIVERSITY

作业报告

1. 机器人朝向问题：
   这个问题比较烦躁，主要是因为是在立体中，想起来有点别扭，并且在这里要把 x 轴当作 y 轴，z 轴当作 x 轴，但最终还是解决了，具体解决方法请见 2.2)。
2. 添加光照后颜色只有灰白色：
   通过百度解决了，只需要添加语句glEnable(GL_COLOR_MATERIAL)即可，即利用颜色作用材料。
3. 照相机的摆放位置和投影方式：
   照相机的摆放位置其实慢慢尝试即可找到一个合适的位置，投影方式要选择透视投影，不能选择平行投影，效果没有透视投影好。这个困难主要在于需要时间去尝试。