中山大學
SUN YAT-SEN UNIVERSITY

《计算机图形学》

（作业二）

学 院 名 称	**：**	数据科学与计算机学院
专业（班级）	**：**	17 计算机科学与技术
姓名学号	**：**	17341067 江金昱

一、算法原理

1. 机器人的绘制原理

使用$glBegin(POLYGON)$绘制一个$1X1X1$的立方体，然后使用$glScalef(x,y,z)$对画出来的立方体进行拉伸可以创造各种大小的长方体，分别调用绘制可创造处机器人的头、躯干、四肢等部分。

在绘制好机器人的各部件后，使用$glTranslatef(x,y,z)$对机器人的各个部件进行平移，使得机器人的头、躯干、四肢等都摆放在相应正确的位置。每绘制一个部件都要在外层加上$glPushMatrix()$和$glPopMatrix()$对，以防止对改部件的操作影响到其他部件

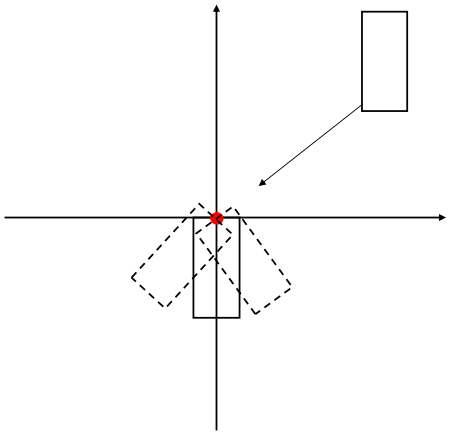
2. 机器人沿圆运动原理

将 **1** 中提到的机器人封装好，以便调用整个函数绘制完成的机器人。

为了使得机器人沿圆运动（以绕着原点运动为例），首先需要将机器人平移（假设平移到点$(-30,0,0)$），然后将机器人绕着 y 轴旋转。 由于在 QT 中设置了定时器，即每隔一段时间重新渲染屏幕，因此我们可以设定旋转角度，记为$angle$。每次定时器调用 $paintGL()$ 渲染的时候，我们就递增$angle$变量，这样就可以在视觉上产生机器人绕着原点旋转的效果。

3. 机器人摆臂和抬腿原理

以 **2** 中提到的机器人平移到点$(-30,0,0)$为例。此时可以分析得到，为了使得手臂和腿旋转，应该使这两个部件绕着x轴旋转，因为手臂和腿是在$y-z$平面内进行摆动的。那么如何进行摆动呢？以手的摆动为例，在平移前需要将手平移至原点，如下图所示。



在原点旋转完成后，再平移相同长度至原来的位置即可。腿的摆动是相同原理的。

4. obj 模型载入原理

在载入 OBJ 模型前，首先要了解一下.obj 文件以及附带的材质文件.mtl 的格式。

.obj 文件格式	
标识符	含义
v x y z	v 表示顶点，后面三个数字表示 x,y,z 的坐标
mtllib XXX.mtl	指定该 obj 文件使用的材质文件名称为 XXX.mtl
vn x y z	vn 表示法向量, 后面三个数字表示法向量的 x,y,z 坐标
vt x y z	vt 表示纹理，后面三个数字表示纹理坐标
g XXX	表示面的分组
usemtl XXXX	表示使用材质文件里的哪一个材质
s off / on	s 表示平滑组
f 1/1/1 2/2/2 3/3/3 4/4/4	f 表示面，后面以'/'隔开的数字分别代表顶点索引/纹理索引/法向量索引

.mtl 件格式	
标识符	含义
newmtl XXX	定义新材质，名为 XXX
Ns	反射指数
d	渐隐指数
Tr	透明度
Tf	滤光透射率
illum	指定材质的光照模型
Ka	环境光
Kd	散射光
Ks	镜面光
map_Kd	后面跟纹理图片

由于 obj 文件有身体分块的注释，如下图所示

```
#
# object Chest_Belly
#
```

每次遇到这样的注释，都要做好模型的分块，这样就能区分身体不同部分的部件。

obj 文件的读取方式为，一次读取一行，通过判断行首的标识符，判断读入的是什么属性，然后将点坐标或者索引存储到对应的$vector$容器里即可。

文件载入完毕后，可以使用$glBegin(GL_POLYGON)$进行面的绘制，使用$glMaterialfv()$进行物体表面材质的设置（还需要打开光照，使用$glLightfv()$设置光照），用$glVertex3fv()$设置顶点坐标，用$glTexCoord3fv()$设置纹理信息，用$glNormal3fv()$设置法线信息，使用$glShadeModel(GL_SMOOTH)$使得着色均匀。

5. 纹理设置

纹理设置的流程：

1. 载入图片文件到内存，可以用一个数组存储
2. 使用$glGenTextures()$生成 ID
3. 将图片绑定到 ID 并加载
4. 在绘制纹理前要$glEnable(GL_TEXTURE_2D)$打开纹理，并且设置相应的纹理坐标

二、实现过程与运行结果

1. 机器人的绘制

首先要创建一个通用的绘制标准$1X1X1$立方体的函数

```
/*########################################################
##  函数: DrawCube
##  函数描述： 绘制一个立方体，立方体大小为 1X1X1 在(0,0,0)到(1,1,1)的范围内
##  参数描述：
##  x:沿着 x 轴移动的长度
##  y:沿着 y 轴移动的长度
##  z:沿着 z 轴移动的长度
########################################################*/
void DrawCube(float x, float y, float z)
{
    glPushMatrix();
    glTranslatef(x, y, z);
```

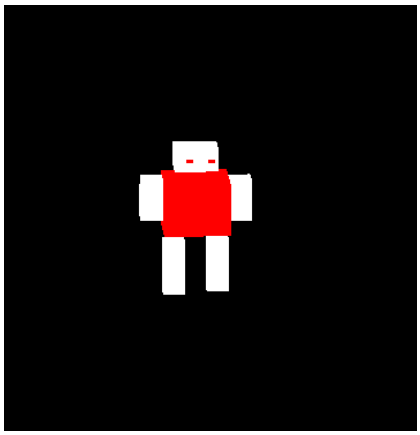
```
    glBegin(GL_POLYGON);
    //底面
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(0.0f, 0.0f, 1.0f);
    glVertex3f(1.0f, 0.0f, 1.0f);
    glVertex3f(1.0f, 0.0f, 0.0f);
    //背面
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(1.0f, 0.0f, 0.0f);
    glVertex3f(1.0f, 1.0f, 0.0f);
    glVertex3f(0.0f, 1.0f, 0.0f);
    //左面
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(0.0f, 1.0f, 0.0f);
    glVertex3f(0.0f, 1.0f, 1.0f);
    glVertex3f(0.0f, 0.0f, 1.0f);
    //右面
    glVertex3f(1.0f, 0.0f, 0.0f);
    glVertex3f(1.0f, 0.0f, 1.0f);
    glVertex3f(1.0f, 1.0f, 1.0f);
    glVertex3f(1.0f, 1.0f, 0.0f);
     //顶面
    glVertex3f(0.0f, 1.0f, 0.0f);
    glVertex3f(0.0f, 1.0f, 1.0f);
    glVertex3f(1.0f, 1.0f, 1.0f);
    glVertex3f(1.0f, 1.0f, 0.0f);
    //正面
    glVertex3f(0.0f, 0.0f, 1.0f);
    glVertex3f(1.0f, 0.0f, 1.0f);
    glVertex3f(1.0f, 1.0f, 1.0f);
    glVertex3f(0.0f, 1.0f, 1.0f);
    glEnd();
    glPopMatrix();
}
```

然后，调用上述$DrawCube()$函数再加以$glScalef()$即可绘制出任意大小的长方体，下面以绘制手臂为例展示，其他包括头、躯干、手臂等原理相似，就不一一赘述。

```
/*#######################################################
##  函数: DrawArm
##  函数描述： 绘制机器人的手臂
##  参数描述：
##  x:沿着 x 轴移动的长度
```

```
##  y:沿着 y 轴移动的长度
##  z:沿着 z 轴移动的长度
#########################################################*/
void DrawArm(float x, float y, float z)
{
    glPushMatrix();
    // 白色
    glColor3f(1.0f, 1.0f, 1.0f);
    glTranslatef(x, y, z);
    glScalef(1.0f, 4.0f, 1.0f);
    DrawCube(0.0f, 0.0f, 0.0f);
    glPopMatrix();
}
```

机器人绘制结果展示:



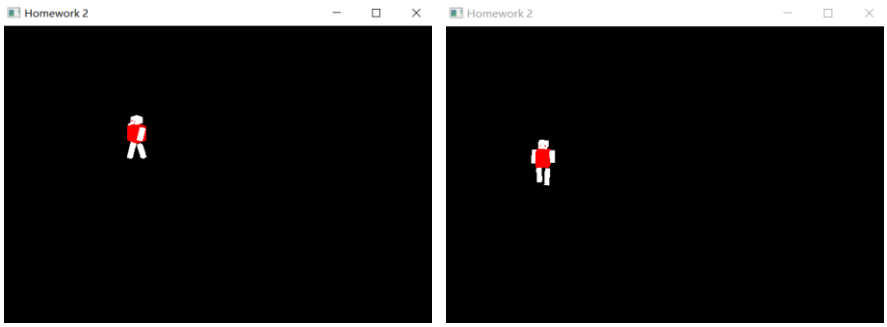
2. 机器人沿圆运动以及照相机和视景体的设置

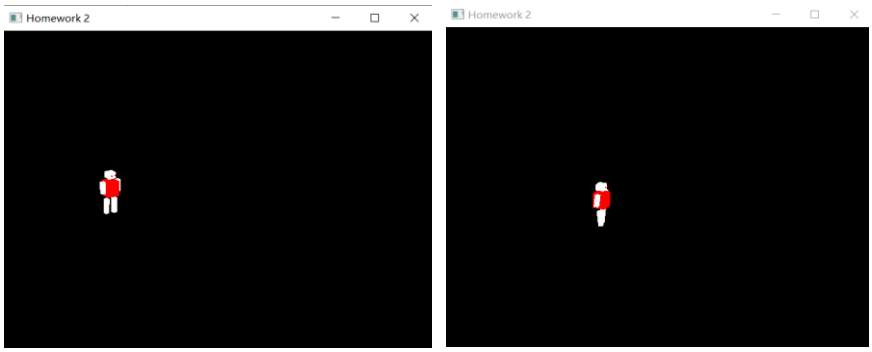
在绘制机器人之前，需要调用$glRotatef()$和$glTranslatef()$对物体进行先平移后旋转，下面我展示了机器人平移到$(0,0,-30)$后绕着原点旋转的代码，并且相应地设置了视景体和照相机。

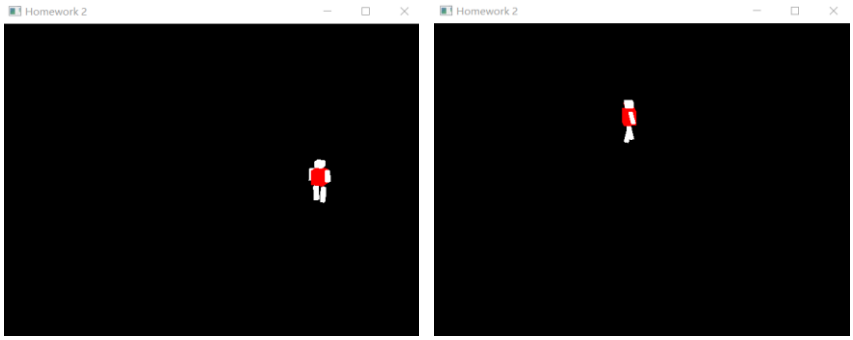
```
/*########################################################
##  函数: paintGL
##  函数描述:  绘图函数，实现图形绘制，会被 update()函数调用
##             设置照相机，视景体，调用画机器人的模型，
##             实现机器人绕着原点运动
##  参数描述:  无
#########################################################*/
void MyGLWidget::paintGL()
{
    // 启用深度检测
    glEnable(GL_DEPTH_TEST);
```

```
    // 设置清理色为黑色
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    // 清理颜色与深度缓存
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    // 将当前矩阵设置为单位矩阵
    glLoadIdentity();
    // 创建正交平行的视景体
    glOrtho(-50.0f, 50.0f, -50.0f, 50.0f, -50.0f, 50.0f);
    gluLookAt(0.0f,1.5f,3.0f,    0.0f,0.0f,0.0f, 0.0f, 1.0f,-1.0f);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    // 每次更新视图，递增旋转角度
    angle = angle + 0.5f;
    if (angle >= 360.0f)
        angle = 0.0f;
    glPushMatrix();
    glLoadIdentity();
    // 绕着 y 轴旋转机器人
    glRotatef(angle, 0.0f, 1.0f, 0.0f);
    // 平移至(0,0,-30)
    glTranslatef(0.0f, 0.0f, -30.0f);
    // 绘制机器人
    DrawRobot(0.0f, 0.0f, 0.0f);
    glPopMatrix();
    glFlush();
}
```

结果展示:







3. 机器人的摆臂和抬腿

按照原理部分提到的，先将手臂/腿移到原点进行旋转，旋转完毕然后平移回原位置。

手臂和腿的摆动浮动相反，用旋转角度$angle$的正负来控制

```
/*#########################################################
##  函数: DrawRobot
##  函数描述： 绘制机器人的头、身体、腿、手，并且实现腿和手的转动
##  参数描述：
##  x:沿着 x 轴移动的长度
##  y:沿着 y 轴移动的长度
##  z:沿着 z 轴移动的长度
#########################################################*/
void DrawRobot(float x, float y, float z)
{
    // 机器人腿的状态
    // true 表示向前摆动,false 表示向后摆动
    static bool leg1 = true;
    static bool leg2 = false;
    // 机器人手臂的状态
    // true 表示向前摆动,false 表示向后摆动
    static bool arm1 = false;
    static bool arm2 = true;
    glPushMatrix();
    // 将机器人放置的位置
    glTranslatef(x, y, z);
    // 头放在身体的正上方
```

```
DrawHead(0.5f,12.0f,0.5f);
DrawBody(0.0f,6.0f,0.0f);
// 画眼睛,眼睛放在头的中间
// 左眼
DrawEye(0.2f,13.0f,1.0f);
DrawEye(0.2f,13.0f,2.0f);
//----------右手------------
glPushMatrix();
//如果手臂向前移动,就增大角度,否则减小角度
if (arm1){
    armAngle[0] = armAngle[0] + 0.5f;
}
else
    armAngle[0] = armAngle[0] - 0.5f;
//一旦手臂在一个方向上达到了其最大角度值就将其反转
if (armAngle[0] >= 15.0f)
    arm1 = false;
if (armAngle[0] <= -15.0f)
    arm1 = true;
// 外面一层 1 7 1 是移回原点旋转
glTranslatef(1.0f,7.0f,-1.0f);
// 将长方体移动到 x-y 轴的第三象限和第四象限中间
glTranslatef(0.5f,4.0f,0.0f);
// 绕着 z 轴旋转，因为是在 x-y 平面内旋转的
glRotatef(armAngle[0], 0.0f, 0.0f, 1.0f);
glTranslatef(-0.5f,-4.0f,0.0f);
glTranslatef(-1.0f,-7.0f,1.0f);
DrawArm(1.0f,7.0f,-1.0f);
glPopMatrix();
//----------------------------
//-------------左手-------------
glPushMatrix();
//如果手臂向前移动,就增大角度,否则减小角度
if (arm2)
    armAngle[1] = armAngle[1] + 0.5f;
else
    armAngle[1] = armAngle[1] - 0.5f;
//一旦手臂在一个方向上达到了其最大角度值就将其反转
if (armAngle[1] >= 15.0f)
    arm2 = false;
if (armAngle[1] <= -15.0f)
    arm2 = true;
glTranslatef(1.0f,7.0f,3.0f);
glTranslatef(0.5f,4.0f,0.0f);
```

```
// 绕着 z 轴旋转，因为是在 x-y 平面内旋转的
glRotatef(armAngle[1], 0.0f, 0.0f, 1.0f);
glTranslatef(-0.5f,-4.0f,0.0f);
glTranslatef(-1.0f,-7.0f,-3.0f);
DrawArm(1.0f,7.0f,3.0f);
glPopMatrix();
//---------------------------

//---------右腿------------
glPushMatrix();
//如果腿向前移动,就增大角度,否则就减小角度
if (leg1)
    legAngle[0] = legAngle[0] + 0.5f;
else
    legAngle[0] = legAngle[0] - 0.5f;
//一旦腿在一个方向达到了最大的角度值,就将其反转
if (legAngle[0] >= 15.0f)
    leg1 = false;
if (legAngle[0] <= -15.0f)
    leg1 = true;
glTranslatef(1.0f,0.0f,0.0f);
glTranslatef(0.5f,5.0f,0.0f);
// 绕着 z 轴旋转，因为是在 x-y 平面内旋转的
glRotatef(legAngle[0], 0.0f, 0.0f, 1.0f);
glTranslatef(-0.5f,-5.0f,0.0f);
glTranslatef(-1.0f,0.0f,0.0f);
// 画腿
DrawLeg(1.0f,0.0f,0.0f);
glPopMatrix();
//-----------------------

//----------左腿-----------
glPushMatrix();
//如果腿向前移动,就增大角度,否则就减小角度
if (leg2)
    legAngle[1] = legAngle[1] + 0.5f;
else
    legAngle[1] = legAngle[1] - 0.5f;
//一旦腿在一个方向达到了最大的角度值,就将其反转
if (legAngle[1] >= 15.0f)
    leg2 = false;
if (legAngle[1] <= -15.0f)
    leg2 = true;
glTranslatef(1.0f,0.0f,2.0f);
```

```
    glTranslatef(0.5f,5.0f,0.0f);
    // 绕着 z 轴旋转，因为是在 x-y 平面内旋转的
    glRotatef(legAngle[1], 0.0f, 0.0f, 1.0f);
    glTranslatef(-0.5f,-5.0f,0.0f);
    glTranslatef(-1.0f,0.0f,-2.0f);
    // 画腿
    DrawLeg(1.0f,0.0f,2.0f);
    glPopMatrix();
    //----------------------

    glPopMatrix();
}
```

4. obj 模型的载入

4.1 读入 obj 文件和 mtl 文件

这一部分的代码实在是太长了，就不贴了。

代码在"*objloader.h*"文件里。

贴一部分说明大致思想:

即一行一行地读取文件，通过行首字符来判断读进来地是注释、顶点坐标、纹理坐标、法线坐标还是面。

```
/*#######################################################
##  函数: loadObjModel
##  函数描述：加载 objFileName 指定的 obj 文件
##  参数描述：
##  objFileName 是 obj 文件的路径
#######################################################*/
void objModel::loadObjModel(const char * objFileName)
{
    objModel* tempModel = new objModel;
    ifstream infile(objFileName,ios::in);
    // 判断文件是否存在
    if(!infile){
        cerr<<"open error."<<endl;
        // 退出程序
        exit(1);
    }
    // 接受无关信息
    string temp;
    // 读取一行的数据
    char szoneLine[256];
```

```
    // 循环，当没有读取完毕时
    while(infile)
    {
        infile.getline(szoneLine,256);
//        cout<<szoneLine<<endl;
        // 该行不为空
        if (strlen(szoneLine) > 0)
        {
            // 获取 v 开头的数据
            if (szoneLine[0] == 'v')
            {
                // 数据存储到 stringstream 流
                stringstream ssOneLine(szoneLine);
                // 纹理信息
                if (szoneLine[1] == 't')
                {
                    // 总纹理坐标数+1
                    cnum+=1;
                    // 过滤标识符 vt
                    ssOneLine >> temp;
                    // 纹理坐标向量
                    Float3 tempTexcoord;
                    // 存储纹理坐标
                    ssOneLine >> tempTexcoord.Data[0] >>
tempTexcoord.Data[1]>>tempTexcoord.Data[2];

                    // 将纹理坐标存入容器
                    tempModel->texcoord.push_back(tempTexcoord);
                    tempModel->coordnum+=1;
                    tempModel->exist_texture=true;
                }
```

每次读到类似注释“# ChestBelly“就说明下面的坐标是另一个身体部件了，这下就要把之前读入的身体部件加好名称放到列表里。

```
// 处理注释的内容，即不同部件
            else if (szoneLine[0] == '#')
            {
                //流读取一行数据
                stringstream ssOneLine(szoneLine);
                //接收#
                ssOneLine >> temp;
```

```
            string numtemp;
            // 以下为循环，读取各部件的名称
            while (ssOneLine)
            {
                ssOneLine >> temp;
                // 如果该部件是肚子
                if(temp=="Chest_Belly")
                {
                    tempModel->partName=last_part_name;
                    tempModel->coordnum = cnum-tempModel->coordnum;
                    tempModel->vertnum = vnum-tempModel->vertnum;
                    tempModel->nornum = nnum-tempModel->nornum;
                    PushBack(tempModel);
                    last_part_name="Chest_Belly";

                    tempModel = new objModel;
                    break;
                }
```

4.2 obj 文件的绘制：

```
// 绑定纹理
            glBindTexture( GL_TEXTURE_2D, image.ID);
            // 绘制每一个面
            for (unsigned int i = 0; i < face.size(); ++i)
            {
                glBegin(GL_POLYGON);
                //第一个点的法线，纹理，位置信息
                glNormal3fv(normal[face[i].vertex[0][2] - 1 - nornum].Data);
                glTexCoord3fv(texcoord[face[i].vertex[0][1] - 1 - coordnum].Data);
                glVertex3fv(position[face[i].vertex[0][0] - 1 - vertnum].Data);
                //第二个点的法线，纹理，位置信息
                glNormal3fv(normal[face[i].vertex[1][2] - 1 - nornum].Data);
                glTexCoord3fv(texcoord[face[i].vertex[1][1] - 1 - coordnum].Data);
                glVertex3fv(position[face[i].vertex[1][0] - 1 - vertnum].Data);
                //第三个点的法线，纹理，位置信息
                glNormal3fv(normal[face[i].vertex[2][2] - 1 - nornum].Data);
                glTexCoord3fv(texcoord[face[i].vertex[2][1] - 1 - coordnum].Data);
                glVertex3fv(position[face[i].vertex[2][0] - 1 - vertnum].Data);
                //第四个点的法线，纹理，位置信息
                glNormal3fv(normal[face[i].vertex[3][2] - 1 - nornum].Data);
                glTexCoord3fv(texcoord[face[i].vertex[3][1] - 1 - coordnum].Data);
                glVertex3fv(position[face[i].vertex[3][0] - 1 - vertnum].Data);
                glEnd();
```

```
        }
```

5. 设置纹理

1. 载入图片文件到内存，可以用一个数组存储

```
/*#######################################################
##  函数: Loadbitmap
##  函数描述:  加载 file 所指向的纹理图片到内存中
##  参数描述:
##  fiLe: 想要加载的纹理图片的路径
#######################################################*/
bool CBMPLoader::Loadbitmap(const char *file)
{
    // 文件指针
    FILE *pFile;
    // 创建位图文件信息和位图文件头结构
    BITMAPINFOHEADER bitmapInfoHeader;
    BITMAPFILEHEADER header;
    // 用于将图像颜色从 BGR 变换到 RGB
    unsigned char textureColors = 0;
    // 打开文件,并检查错误
    errno_t err = fopen_s(&pFile, file, "r");
    if( err != 0)
    {
        return false;
    }
    // 读入位图文件头信息
    fread(&header, sizeof(BITMAPFILEHEADER), 1, pFile);
    // 检查该文件是否为位图文件
    if(header.bfType != BITMAP_ID)
    {
        // 若不是位图文件,则关闭文件并返回
        fclose(pFile);
        return false;
    }
    // 读入位图文件信息
    fread(&bitmapInfoHeader, sizeof(BITMAPINFOHEADER), 1, pFile);
    // 保存图像的宽度和高度
    imageWidth = bitmapInfoHeader.biWidth;
    imageHeight = bitmapInfoHeader.biHeight;
    // 确保读取数据的大小
    if(bitmapInfoHeader.biSizeImage == 0)
```

```
        bitmapInfoHeader.biSizeImage = bitmapInfoHeader.biWidth *bitmapInfoHeader.biHeight *
3;
    // 将指针移到数据开始位置
    fseek(pFile, header.bfOffBits, SEEK_SET);
    // 分配内存
    image = new unsigned char[bitmapInfoHeader.biSizeImage];
    // 检查内存分配是否成功
    if(!image)
    {
        delete[] image;
        fclose(pFile);
        return false;
    }
    // 读取图像数据
    fread(image, 1, bitmapInfoHeader.biSizeImage, pFile);
    // 将图像颜色数据格式进行交换,由 BGR 转换为 RGB
    for(int index = 0; index < (int)bitmapInfoHeader.biSizeImage; index+=3)
    {
        textureColors = image[index];
        image[index] = image[index + 2];
        image[index + 2] = textureColors;
    }
    // 关闭文件
    fclose(pFile);
    isLoaded = true;
    // 成功返回
    return true;
}
```

2. 使用*glGenTexture*()生成 ID 以及将图片绑定到 ID 并加载

```
/*#######################################################
##  函数: makeImage
##  函数描述： 绑定纹理 ID
##  参数描述：
##  m_Texture: 自己创建的纹理类，包含纹理 ID，图片加载放置的内存等信息
#######################################################*/
void makeImage(CBMPLoader *m_Texture)
{
    // 设置字节对齐
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    // 生成 1 个纹理 ID
    glGenTextures(1, &m_Texture->ID);
    // 将 TEXTURE_2D 绑定到该 ID 上
    glBindTexture(GL_TEXTURE_2D, m_Texture->ID);
```

```
    // 设置贴图参数
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    // 对颜色进行组合
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    // 使用 MIP 纹理贴图
    gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, m_Texture->imageWidth, m_Texture->imageHeight,
          GL_RGB, GL_UNSIGNED_BYTE, m_Texture->image);
}
```

3. 在绘制纹理前要$glEnable(GL_TEXTURE_2D)$打开纹理，并且设置相应的纹理坐标

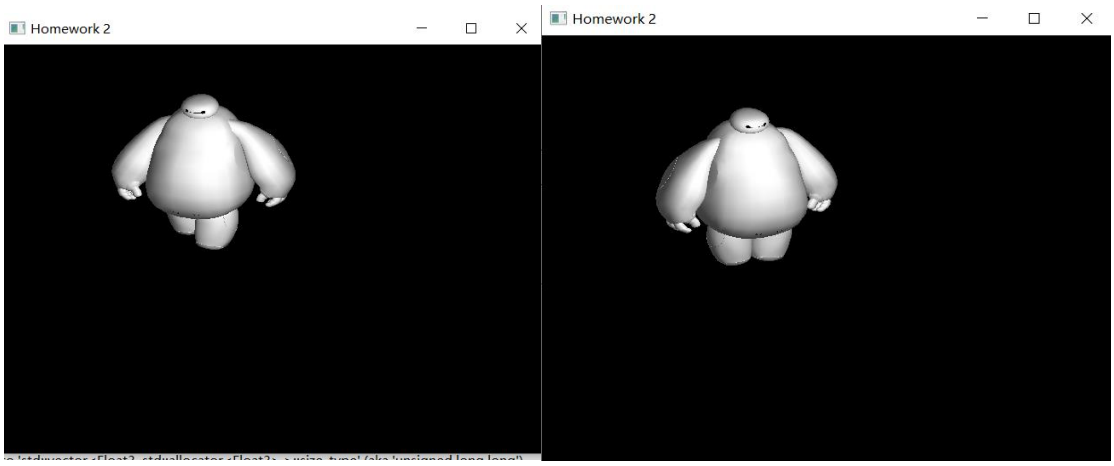
```
glEnable(GL_TEXTURE_2D);
```

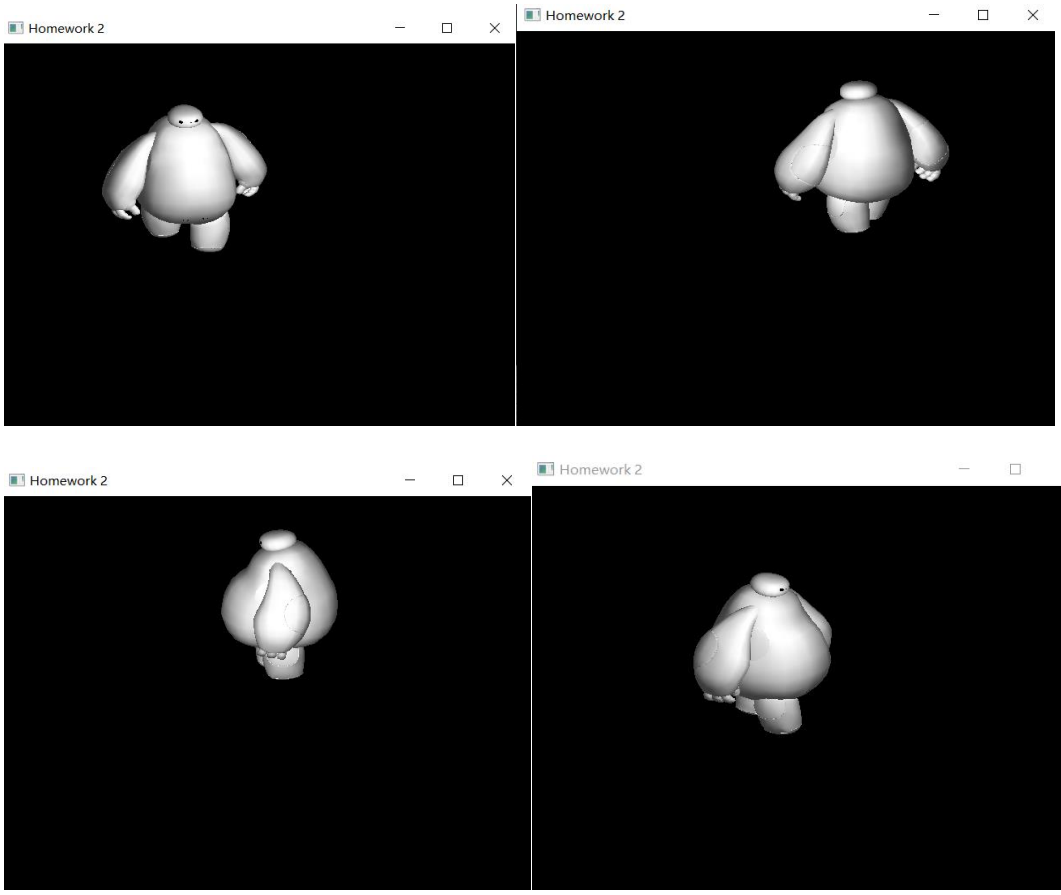
```
//允许平滑着色
        glShadeModel(GL_SMOOTH);
```

```
glTexCoord3fv(texcoord[face[i].vertex[0][1] - 1 - coordnum].Data);
```

6. obj 载入和纹理贴图后的结果截图：

（我载入了机器人-大白）





它的眼睛就是我贴上去的纹理

三、总结

1. 困难：刚开始画机器人的时候，发现怎么画都显示不出来

解决：往往显示不出任何东西的时候都是 glOrtho 和 gluLookAt 的问题，记得要调整 glOrtho 使其能包括所画的物体，同时 gluLookAt 不能超出视景体，否则也看不到。

2. 困难：转动手臂和腿的时候，刚开始转动的很奇怪，是以下方为支点进行转动，而不是以手臂和身体连接处为支点转动

解决：将支点平移到原点进行转动，这样手臂和腿就会以其和身体连接处为支点转动。

3. 困难：mesh 模型载入的时候画面发现有重叠

解决：每一个f都要单独的使用$glBein()$ $glEnd()$来绘制，这样表示一个面

4. 困难：刚开始载入机器人大白的时候身上一块黑一块白的

解决：未设置平滑着色，打开平滑着色模式即可

5. 困难：纹理贴图无法显示

解决：要设置纹理坐标，并且绑定纹理 ID