计算机图形学小组项目报告

小组成员：

1452252 冷典典

1452332 高四军

1452309 黄一凡

1454048 赵嘉伟

1452676 虞鸿飞

1452335 吴议

一、涉及的相关技术

1. 多重纹理：

   多重纹理就是在渲染一个多边形的时候可以用到多张纹理图.把多张纹理图进行一些颜色的操作,可以达到一些效果。本次项目中使用该技术来模仿光照和阴影效果；

2. 纹理对象：

   在分配了纹理对象编号后，使用 glBindTexture 函数来指定"当前所使用的纹理对象"。然后就可以使用 glTexImage*系列函数来指定纹理像素、使用 glTexParameter*系列函数来指定纹理参数、使用 glTexCoord*系列函数来指定纹理坐标了。使用多个纹理对象，就可以使 OpenGL 同时保存多个纹理。在使用时只需要调用 glBindTexture 函数，在不同纹理之间进行切换，而不需要反复载入纹理，因此动画的绘制速度会有非常明显的提升。

3. 几何及坐标变换：

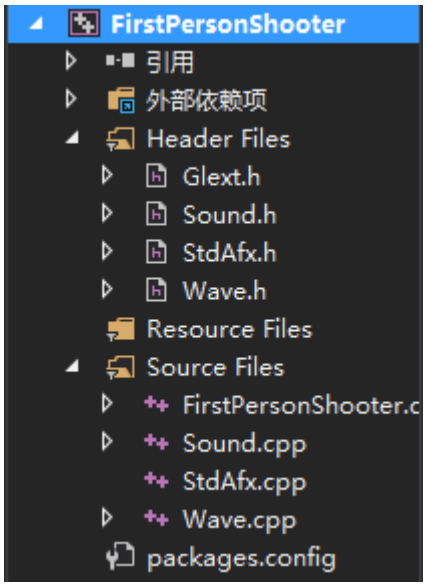
   调用 OpenGL 的 glTranslated、glRotated 等函数，来实现镜头、"子弹"等的位移与变化。

4. 显示列表：

   OpenGL 显示列表（Display List）是由一组预先存储起来的留待以后调用的 OpenGL 函数语句组成的，当调用这张显示列表时就依次执行表中所列出的函数语句。采用显示列表方式绘图一般要比瞬时方式快，因为一旦显示列表被处理成适合于图形硬件的格式，则不同的 OpenGL 实现对命令的优化程度也不同。例如旋转矩阵函数 glRotate*()，若将它置于显示列表中，则可大大提高性能。因为旋转矩阵的计算并不简单，包含有平方、三角函数等复杂运

算，而在显示列表中，它只被存储为最终的旋转矩阵，于是执行起来如同硬件执行函数 glMultMatrix()一样快。一般来说，显示列表能将许多相邻的矩阵变换结合成单个的矩阵乘法，从而加快速度。

二、工程内容及部分源代码展示



Glext.h:OpenGL 的拓展头文件。其余 OpenGL 的支持依赖 Viual Studio 的 NuGet 包中的 NupenGL 包。

Sound.h

Wave.h

Sound.cpp

Wave.h

读取 wav 文件以实现音效。

StdAfx.h StdAfx.cpp: 预编译头文件

FirstPersonShooter.cpp：程序的主要代码。

此外，在项目目录的 Texture 文件夹下保存了程序需要的所有 bmp，用于纹理。

Implementation

1. 窗口及指令接收：

   使用 MFC 的方式实现。

   鼠标调整视角的部分代码

```
    if (ABS(mouse_x - LOWORD(lParam)) > 300)
        {
            if (mouse_x > 360 / 2)
                mouse_3d_x += ((mouse_x - 360) - LOWORD(lParam)) /
sensitivity;//鼠标移到屏幕边缘时进行重新调整
            else if (LOWORD(lParam) > 360 / 2)
                mouse_3d_x += (mouse_x - (LOWORD(lParam) - 360)) /
sensitivity;
        }
        else
        {
            mouse_3d_x += (mouse_x - LOWORD(lParam)) / sensitivity;
        }
```

   键盘控制移动的部分代码：

```
    if (keys[VK_UP])
                {
                    XP -= (GLdouble)sin(heading*piover180) * 5.0f;
                    ZP -= (GLdouble)cos(heading*piover180) * 5.0f;
                }//XP 和 ZP 为在各自方向上的增量
```

```
            else if (keys['W'])
            {
                XP -= (GLdouble)sin(heading*piover180) * 5.0f;
                ZP -= (GLdouble)cos(heading*piover180) * 5.0f;
            }
```

2. 读取位图文件并生成纹理对象部分代码：

```
    if (TextureImage[0] = LoadBMP("texture/floor.bmp"))
    //LoadBMP 函数中使用 auxDIBImageLoad 函数来读取位图文件
    {
        Status = TRUE;
        glGenTextures(1, &texture[1]);

        glBindTexture(GL_TEXTURE_2D, texture[1]);
        glTexParameteri(GL_TEXTURE_2D,       GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D,       GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_NEAREST);
        gluBuild2DMipmaps(GL_TEXTURE_2D,                    GL_RGB8,
TextureImage[0]->sizeX,      TextureImage[0]->sizeY,      GL_RGB,
GL_UNSIGNED_BYTE, TextureImage[0]->data);
    }
```

3. 显示列表、场景绘制以及多重纹理贴图部分代码：

```
        ROOM = glGenLists(1);                                   // 生成显示
列表
        glNewList(ROOM, GL_COMPILE);

        // TEXTURE-UNIT #0
        glActiveTextureARB(GL_TEXTURE0_ARB);
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, texture[1]);//第一个纹理
        // TEXTURE-UNIT #1:
        glActiveTextureARB(GL_TEXTURE1_ARB);
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, texture[4]);//第二个纹理
        //两个纹理一个为地板本身的纹理，一个为黑色的色块，将两个纹理混合绘制，可
以模仿出地面上阴影的效果

        glColor4d(1, 1, 1, 1.f);
```

```
        // texture[1] and texture[4] // 地板材质
        glBegin(GL_QUADS);
        glMultiTexCoord2fARB(GL_TEXTURE0_ARB, 10, 10);
        glMultiTexCoord2fARB(GL_TEXTURE1_ARB, 1, 1);
        glVertex3d(MAX_Cells, 0, MAX_Cells);//将两个纹理坐标严格对应于所要绘制的矩形的点，其余三个点相同

        glMultiTexCoord2fARB(GL_TEXTURE0_ARB, 10, 0);
        glMultiTexCoord2fARB(GL_TEXTURE1_ARB, 1, 0);
        glVertex3d(MAX_Cells, 0, 0);

        glMultiTexCoord2fARB(GL_TEXTURE0_ARB, 0, 0);
        glMultiTexCoord2fARB(GL_TEXTURE1_ARB, 0, 0);
        glVertex3d(0, 0, 0);

        glMultiTexCoord2fARB(GL_TEXTURE0_ARB, 0, 10);
        glMultiTexCoord2fARB(GL_TEXTURE1_ARB, 0, 1);
        glVertex3d(0, 0, MAX_Cells);
        glEnd();
        …
         glEndList();//所有场景内容绘制完成之后结束显示列表，之后就可以直接进行调用，提高运行效率
```

4. “迷宫”的墙壁部分代码：

```
   glColor4d(1, 1, 1, 1);

          glBegin(GL_QUADS);
          glTexCoord2d(4, 4);
          glVertex3d(MAX_Cells / 2 + 26, 48, MAX_Cells / 2 + 26);
          glTexCoord2d(4, 0);
          glVertex3d(MAX_Cells / 2 + 26, 48, MAX_Cells / 2 - 26);
          glTexCoord2d(0, 0);
          glVertex3d(MAX_Cells / 2 + 24, 48, MAX_Cells / 2 - 26);
          glTexCoord2d(0, 4);
          glVertex3d(MAX_Cells / 2 + 24, 48, MAX_Cells / 2 + 26);
          glEnd();

          glColor4d(.75f, .75f, .75f, 1);

          glBegin(GL_QUADS);
          glTexCoord2d(4, 4);
          glVertex3d(MAX_Cells / 2 + 26, 0, MAX_Cells / 2 - 26);
```

```
            glTexCoord2d(4, 0);
            glVertex3d(MAX_Cells / 2 + 26, 0, MAX_Cells / 2 + 26);
            glTexCoord2d(0, 0);
            glVertex3d(MAX_Cells / 2 + 26, 48, MAX_Cells / 2 + 26);
            glTexCoord2d(0, 4);
            glVertex3d(MAX_Cells / 2 + 26, 48, MAX_Cells / 2 - 26);
            glEnd();

            glBegin(GL_QUADS);
            glTexCoord2d(0, 0);
            glVertex3d(MAX_Cells / 2 + 24, 0, MAX_Cells / 2 + 26);
            glTexCoord2d(0, 4);
            glVertex3d(MAX_Cells / 2 + 26, 0, MAX_Cells / 2 + 26);
            glTexCoord2d(4, 4);
            glVertex3d(MAX_Cells / 2 + 26, 48, MAX_Cells / 2 + 26);
            glTexCoord2d(4, 0);
            glVertex3d(MAX_Cells / 2 + 24, 48, MAX_Cells / 2 + 26);
            glEnd();

            glBegin(GL_QUADS);
            glTexCoord2d(0, 0);
            glVertex3d(MAX_Cells / 2 + 24, 0, MAX_Cells / 2 - 26);
            glTexCoord2d(0, 4);
            glVertex3d(MAX_Cells / 2 + 26, 0, MAX_Cells / 2 - 26);
            glTexCoord2d(4, 4);
            glVertex3d(MAX_Cells / 2 + 26, 48, MAX_Cells / 2 - 26);
            glTexCoord2d(4, 0);
            glVertex3d(MAX_Cells / 2 + 24, 48, MAX_Cells / 2 - 26);
            glEnd();

            glColor4d(.25f, .25f, .25f, 1);
            glBegin(GL_QUADS);
            glTexCoord2d(4, 4);
            glVertex3d(MAX_Cells / 2 + 24, 0, MAX_Cells / 2 - 26);
            glTexCoord2d(4, 0);
            glVertex3d(MAX_Cells / 2 + 24, 0, MAX_Cells / 2 + 26);
            glTexCoord2d(0, 0);
            glVertex3d(MAX_Cells / 2 + 24, 48, MAX_Cells / 2 + 26);
            glTexCoord2d(0, 4);
            glVertex3d(MAX_Cells / 2 + 24, 48, MAX_Cells / 2 - 26);
            glEnd();
```

每一面墙需要绘制前后左右上共 6 个面，并且不需要使用多重纹理，墙壁内侧的偏暗效果依靠调整颜色的数值来实现。

5. 碰撞检测部分代码：

该部分代码是判断当前位置的 x 坐标是否超出范围

```
//碰撞检测 cx/cy/cz = 位置坐标; cxi/cyi/czi = 位移量; padding = 碰撞体积; bounce = 弹性,用于碎片的效果;
bool DetectCollision(GLdouble &cx, GLdouble &cy, GLdouble &cz, GLdouble &cxi, GLdouble &cyi, GLdouble &czi, GLdouble padding, GLdouble bounce)
{
    bool Status = false;
    if (cx > MAX_Cells - padding)
    {
        if ((cz > MAX_Cells / 2 - 15 && cz < MAX_Cells / 2 + 15) && (cy > 0 && cy < MAX_Cells / 8))//用于传送门的判断
            cx = 0 + padding * 2;
        else
            cx = MAX_Cells - padding;
        cxi = -cxi;
        cxi *= bounce;
        Status = true;
    }
    else if (cx < 0 + padding)
    {
        if (cz > MAX_Cells / 2 - 15 && cz < MAX_Cells / 2 + 15 && cy > 0 && cy < MAX_Cells / 8)//用于传送门的判断
            cx = MAX_Cells - padding * 2;
        else
            cx = 0 + padding;
        cxi = -cxi;
        cxi *= bounce;
        Status = true;
    }
…
}
```

6. 移动及视角转换的实现：

```
    XP *= 0.8f;//将两个方向上的增量逐步降低，以使得在停止操作以后能够让镜头停下来
    ZP *= 0.8f;
```

```
    xtrans2 += XP / 10;//现在的位置
    ztrans2 += ZP / 10;

    xtrans = -xtrans2;
    ztrans = -ztrans2;

    DetectCollision(xtrans2, ytrans2, ztrans2, XP, BlankNum, ZP, 2.5f,
0);//移动的碰撞检测

    heading = mouse_3d_x;//heading 实际是以 y 轴为中心的旋转角度

    yrot = heading;

    sceneroty = 360.0f - yrot;

    glLoadIdentity();

    glRotated(mouse_3d_y, 1.f, 0, 0);//视角的上下移动
    glRotated(sceneroty, 0, 1.f, 0);//视角的左右移动

    glTranslated(xtrans, -10, ztrans);//视角的移动

    glCallList(ROOM);
```

7. 开枪及命中检测：

```
    // 开枪
    if (isFire)
    {
        Sound1[shots_fired].Play(0);//播放枪声

        shots_fired++;
        if (shots_fired > MAX_SHOTS_FIRED - 1)
            shots_fired = 0;
        isFireComplete = false;
        isFire = false;

        //子弹的位置。将开枪时所站的位置作为初值赋予子弹
        fire_x = -xtrans;
        fire_y = 10;
        fire_z = -ztrans;

        //子弹的位移量。根据开枪时的瞄准方向算出各方向增量
```

```
        fire_xp = -(GLdouble)sin(heading*piover180) / 5;
        fire_yp = -(GLdouble)tan(mouse_3d_y*piover180) / 5;
        fire_zp = -(GLdouble)cos(heading*piover180) / 5;
    }
    int fire_counter = 0;

    // 不断计算子弹的位置，直到检测到碰撞
    while (!isFireComplete)
    {
        fire_counter++;
        fire_x += fire_xp;
        fire_y += fire_yp;
        fire_z += fire_zp;

        if  (DetectCollision(fire_x,  fire_y,  fire_z,  BlankNum,
BlankNum, BlankNum, 0, 0))//子弹的碰撞检测
        {
            isFireComplete = true;//子弹命中
            Ricochet();//对命中效果进行初始化
            isRicochet = true;
        }
        else if (fire_counter > 10000)
        {
            isFireComplete = true;
        }
    }
```

8. 命中后的效果：

```
    typedef struct                  // 枪击中物体后的粒子效果
    {
    GLdouble   life;                // 效果的持续时间
    GLdouble   fade;                // 效果小时的速度
    GLdouble   size;                // 烟雾的大小
    GLdouble   triangle_x1;                     // 粒子的坐标
    GLdouble   triangle_x2;
    GLdouble   triangle_x3;
    GLdouble   triangle_y1;
    GLdouble   triangle_y2;
    GLdouble   triangle_y3;
    GLdouble   triangle_z1;
    GLdouble   triangle_z2;
    GLdouble   triangle_z3;
```

```
    GLdouble   triangle_rotate_x;                    // 旋转角度
    GLdouble   triangle_rotate_y;
    GLdouble   triangle_rotate_z;
    GLdouble   triangle_rotate_xi;                   //旋转角度增量
    GLdouble   triangle_rotate_yi;
    GLdouble   triangle_rotate_zi;
    GLdouble   x;                    // 击中位置
    GLdouble   y;
    GLdouble   z;
    GLdouble   xi;                   // 三个方向上的增量
    GLdouble   yi;
    GLdouble   zi;
    }
particles;

    void Ricochet()//击中物体后的效果的初始化
    {
      int i;

      for (i = 0; i < MAX_PARTICLES; i++)
      {
        if (i == 0)
          particle[i][shots_fired].size = 2;
        else
          particle[i][shots_fired].size                              =
GLdouble((rand()%25)+25)/50;

        particle[i][shots_fired].triangle_x1                         =
GLdouble((rand()%50)-25)/75.f;
        particle[i][shots_fired].triangle_x2                         =
GLdouble((rand()%50)-25)/75.f;
        particle[i][shots_fired].triangle_x3                         =
GLdouble((rand()%50)-25)/75.f;

        particle[i][shots_fired].triangle_y1                         =
GLdouble((rand()%50)-25)/75.f;
        particle[i][shots_fired].triangle_y2                         =
GLdouble((rand()%50)-25)/75.f;
        particle[i][shots_fired].triangle_y3                         =
GLdouble((rand()%50)-25)/75.f;

        particle[i][shots_fired].triangle_z1                         =
GLdouble((rand()%50)-25)/75.f;
        particle[i][shots_fired].triangle_z2                         =
```

```
GLdouble((rand()%50)-25)/75.f;
       particle[i][shots_fired].triangle_z3                          =
GLdouble((rand()%50)-25)/75.f;

       particle[i][shots_fired].triangle_rotate_x = rand()%360;
       particle[i][shots_fired].triangle_rotate_y = rand()%360;
       particle[i][shots_fired].triangle_rotate_z = rand()%360;

       particle[i][shots_fired].triangle_rotate_xi                   =
GLdouble((rand()%50)-25)/5;
       particle[i][shots_fired].triangle_rotate_yi                   =
GLdouble((rand()%50)-25)/5;
       particle[i][shots_fired].triangle_rotate_zi                   =
GLdouble((rand()%50)-25)/5;

       particle[i][shots_fired].life = 1;
       particle[i][shots_fired].fade   =   GLdouble(rand()%100)/5000
+ .02f;
       particle[i][shots_fired].x = fire_x;
       particle[i][shots_fired].y = fire_y;
       particle[i][shots_fired].z = fire_z;
       GLdouble angle = rand()%360;
       GLdouble angle2 = rand()%360;
       GLdouble speed = GLdouble(rand()%100) / 500;
       GLdouble Hyp = Hypot(angle,angle2);

       particle[i][shots_fired].xi = sin(Hypot(angle,Hyp))*speed +
fire_xp;
       particle[i][shots_fired].yi = cos(Hypot(angle,Hyp))*speed +
fire_yp;
       particle[i][shots_fired].zi = cos(Hypot(angle2,Hyp))*speed +
fire_zp;
     }
}



//击中效果的显示
   for (int j = 0; j < MAX_SHOTS_FIRED; j++)
       {
           for (i = 8; i < MAX_PARTICLES; i++)
           {
               if (particle[i][j].life > 0)
               {
```

```
                    isRicochetComplete = false;

                    particle[i][j].x += particle[i][j].xi;
                    particle[i][j].y += particle[i][j].yi;
                    particle[i][j].z += particle[i][j].zi;

                    DetectCollision(particle[i][j].x,
particle[i][j].y, particle[i][j].z,
                        particle[i][j].xi,         particle[i][j].yi,
particle[i][j].zi, 0, .75f);

                    particle[i][j].yi -= .01f;  // 模拟重力，让粒子向下掉
                    particle[i][j].xi *= .99;
                    particle[i][j].yi *= .99;
                    particle[i][j].zi *= .99;
                    particle[i][j].life -= particle[i][j].fade;

                    //调整碎片的位置到击中点
                    glLoadIdentity();
                    glRotated(mouse_3d_y, 1.f, 0, 0);
                    glRotated(sceneroty, 0, 1.f, 0);
                    glTranslated(xtrans, -10, ztrans);
                    glTranslated(particle[i][j].x,  particle[i][j].y,
particle[i][j].z);
                    glRotated(-sceneroty, 0, 1.f, 0);
                    glRotated(-mouse_3d_y, 1.f, 0, 0);

                    //将碎片进行随机化的旋转
                    particle[i][j].triangle_rotate_x              +=
particle[i][j].triangle_rotate_xi;
                    particle[i][j].triangle_rotate_y              +=
particle[i][j].triangle_rotate_yi;
                    particle[i][j].triangle_rotate_z              +=
particle[i][j].triangle_rotate_zi;
                    particle[i][j].triangle_rotate_xi             -=
particle[i][j].fade;
                    particle[i][j].triangle_rotate_yi             -=
particle[i][j].fade;
                    particle[i][j].triangle_rotate_zi             -=
particle[i][j].fade;
                    glRotated(particle[i][j].triangle_rotate_x, 1, 0,
0);
                    glRotated(particle[i][j].triangle_rotate_y, 0, 1,
0);
```

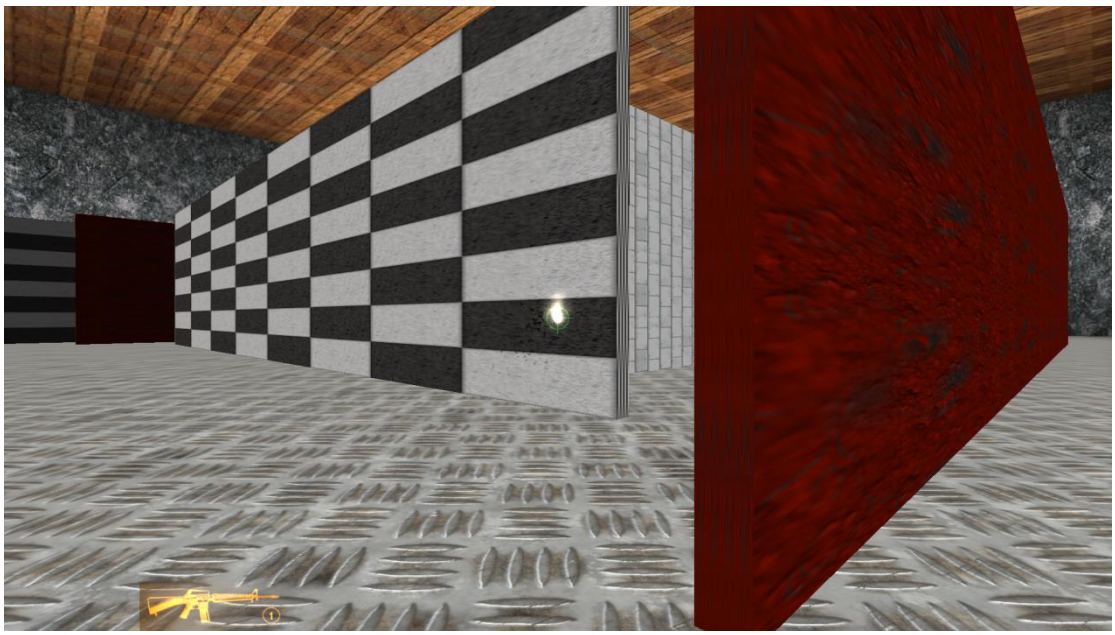
```
                    glRotated(particle[i][j].triangle_rotate_z, 0, 0,
1);

                    //碎片的透明度随时间增加
                    glColor4d(0, 0, 0, particle[i][j].life);

                    //绘制碎片
                    glBegin(GL_TRIANGLES);
                    glVertex3d(particle[i][j].triangle_x1,
particle[i][j].triangle_y1, particle[i][j].triangle_z1);
                    glVertex3d(particle[i][j].triangle_x2,
particle[i][j].triangle_y2, particle[i][j].triangle_z2);
                    glVertex3d(particle[i][j].triangle_x3,
particle[i][j].triangle_y3, particle[i][j].triangle_z3);
                    glEnd();
                }
            }
        }

//烟雾效果的实现类似
```

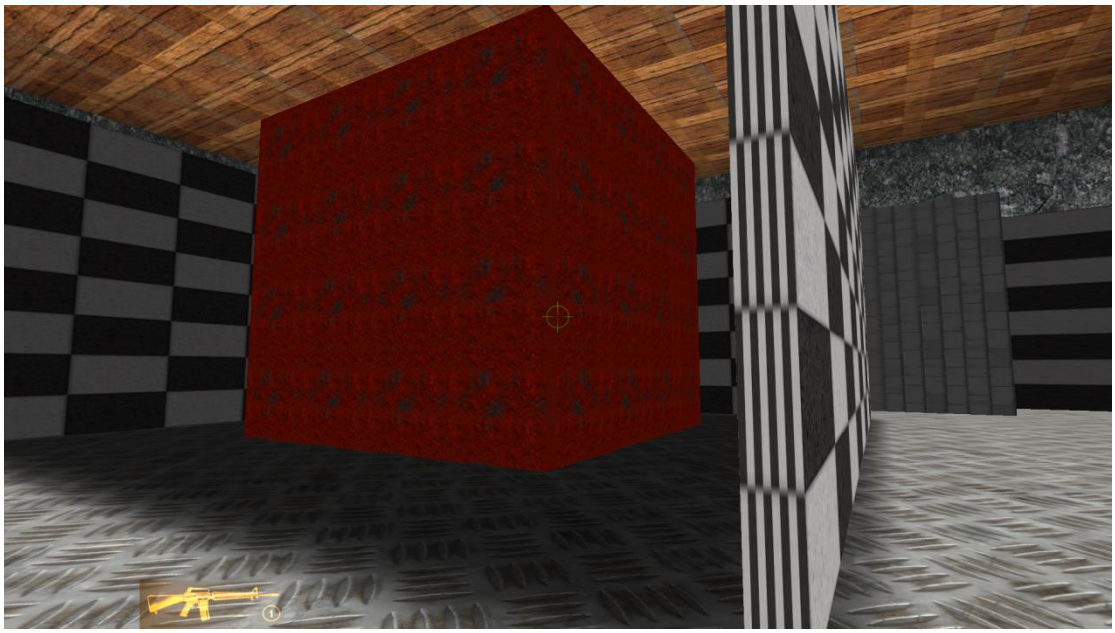
9. 绘制 HUD

```
// 绘制 HUD
    glDisable(GL_DEPTH_TEST);

    glLoadIdentity();
    glTranslated(-765, -567.0f, -1000.0f);

    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    glDisable(GL_TEXTURE_2D);
    glBegin(GL_QUADS);//绘制武器 HUD 的背景
    glColor4d(.4f, .2f, 0, .5f);
    glVertex2f(-10, -10);
    glVertex2f(-10, 71 + 10);
    glColor4d(1, 1, 1, 0);
    glVertex2f(256 + 10, 71 + 10);
    glVertex2f(256 + 10, -10);
    glEnd();
    glEnable(GL_TEXTURE_2D);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE);
```

```
    glBindTexture(GL_TEXTURE_2D, texture[11]);//武器的 HUD 纹理

    glColor4d(1, .5f, 0, 1);
    glBegin(GL_QUADS);//绘制 HUD 图标
    glTexCoord2d(1, 1); glVertex2f(256, 71);
    glTexCoord2d(1, 0); glVertex2f(256, 0);
    glTexCoord2d(0, 0); glVertex2f(0, 0);
    glTexCoord2d(0, 1); glVertex2f(0, 71);
    glEnd();

    glEnable(GL_DEPTH_TEST);

    glDisable(GL_DEPTH_TEST);
    glLoadIdentity();
    glTranslated(0.0f, 0.0f, -35.0f);//十字线与 Camera 的距离

    glBlendFunc(GL_SRC_ALPHA, GL_ONE);

    glColor4f(1, 1, 1, 1);
    glBindTexture(GL_TEXTURE_2D, texture[15]);//十字线的纹理
    glBegin(GL_QUADS);//绘制瞄准十字
    glTexCoord2d(1, 1); glVertex2f(1, 1);
    glTexCoord2d(1, 0); glVertex2f(1, -1);
    glTexCoord2d(0, 0); glVertex2f(-1, -1);
    glTexCoord2d(0, 1); glVertex2f(-1, 1);
    glEnd();
```

三、项目成果展示





详情参考同本文件一起上传的工程压缩包

四、小组工作总结

期初时本小组预定工作时间节点如下：

1、5~7 周	学习相关的编程知识；
2、8~10 周	简单场景创建，定义主视角 camera；
3、第 10 周	静态 3D 场景，进行中期展示；
4、11~12 周	贴图，反思，修正中期出现的问题；

5、13~15 周	可进行交互操作，碰撞判定。（如时间充裕，考虑增加进阶功能）

从实际来看，工作进度基本符合。

不足：中期展示时由于大家比较陌生，协调不到位，导致进度上比老师的预期要慢。工作分配不够平均与细致，导致有些周内容很多有些周则很少。

五、参考资料

1、《C++程序设计》（第二版） 谭浩强著 清华大学出版社

2、《C++游戏与图形编程》（第二版） Gaddis，Tony 著，周靖译 清华大学出版社

3、互联网：主要包括 百度、中国知网 等