CAMBIE HERO

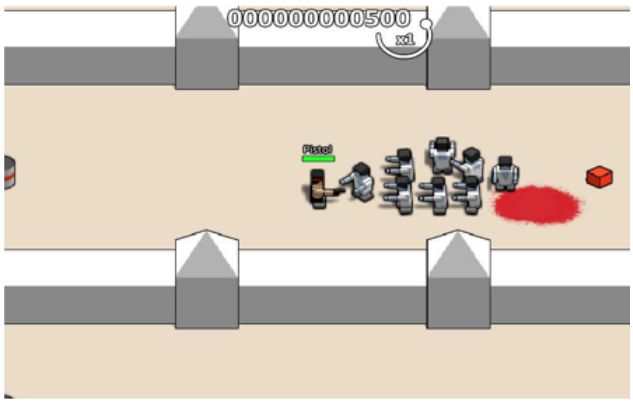
DESIGN SPECIFICATION

Yunhao, Jiao
Dunbang, He
Xinyi, Lin

June, 2016

1 Inspiration

Most people enjoy the mini flash game named “Boxhead”, which is a shooting game displayed on flash. Although it is enchanting with a lot of characters, short of some functions like rendering and lighting, it can only display 2D scenes which may not be fascinating. And the fatal issue for the old shooting game is that the aiming accuracy is far from enough. So our work in this project is to rebuild the scene in 3D version based on the traditional game mode and add some fancy ideas popular with youngers with the help of OpenGL Library.



2 Introduction

2.1 Tasks

Your role setting is a hero trapped in a deserted town, where there are many evils. So your mission is to kill those evils and gain as many points as possible. Of course in the progress, you could get some help from different props and weapons.

2.2 Operations

Keyboard can be used to control the gamer. The keys `w`, `s`, `a` and `d` correspond to up, down, left and right movement respectively. The key `p` is firing, while `l` is changing weapons. The keys `m`, `r` and `q` correspond to pause, restart and quit, respectively. The key `F1` could help you zoom the characters in and out.

2.3 Models

Due to the game mode, the characters are divided into two camps, which are heroes' and evils'. There are two appearances in heroes' camp. At first, it is the elementary version shown below. It beats the evils by pistols which have many different types. After the player defeats a few evils, there will randomly appear one special yellow cube prop. After obtaining it twice, the player will evolve into the advanced version with an imperial crown and wings to move faster and a scepter to beat evils stronger.
There are also two appearances in evils' camp, little zombies and evils. Little zombies, equipped with claws, move half as fast as hero. Evils, with tridents and cannonballs, are even more powerful than little zombies but only 30% as fast as hero in movement.

2.4 Props

2.4.1 Pistols

There are two kinds of pistols. One is Gatling Gun, equipped in the left hand of normal version hero. It fires blue bullets, each of which damages one fifth lives of little zombies and one twenties of evils.
The other one is Laser Gun, which can be gained after obtaining a special yellow cube. It fires red lasers; whose power is twice stronger than Gatling Gun.

2.4.2 Scepter

Scepter is a special equipment on advanced version. It launches the holy light, the most powerful weapon in the game, which could kill a little zombie once and an evil in four launches.

2.4.3 Cubes

There are two kinds of cubes, yellow power cube and green live cube.
Obtaining one yellow power cube, the hero will get Laser Gun immediately. And with two, he can evolve into the advanced version.
Green live cube would help you recover 20% lives.
These cubes appear at random place in a certain time.

2.4.4 Claws

Claws are little zombies and evils' attacking weapons. They can catch heroes and damage its live, belonging to based weapon.

2.4.5 Trident

Tridents belong to evils, which make them more invasive.

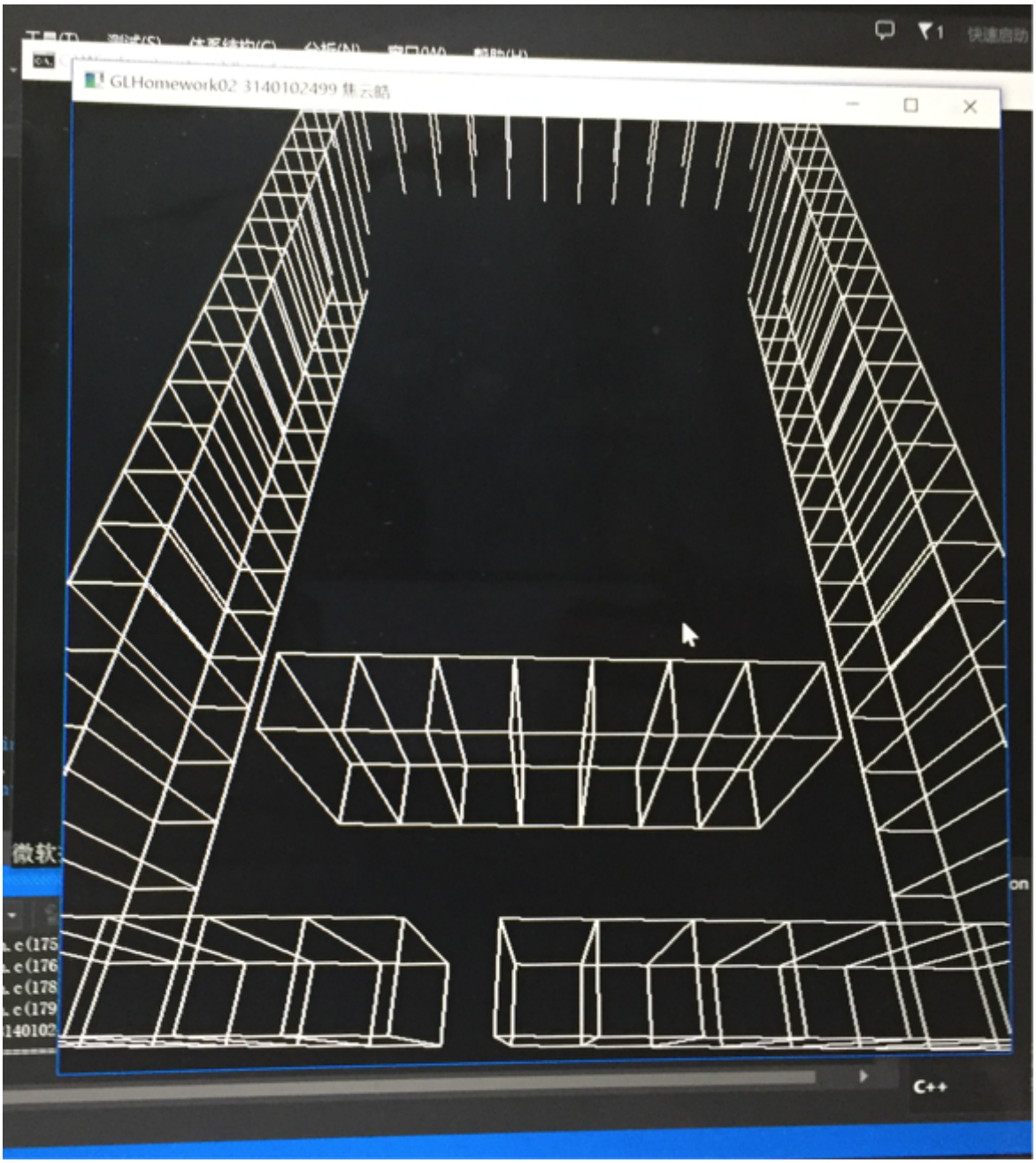
2.4.6 Cannonball

Cannonballs are fired by evils. They can damage 10% lives of the hero and are infinitely valid as long as evils are alive.

3 Design Details

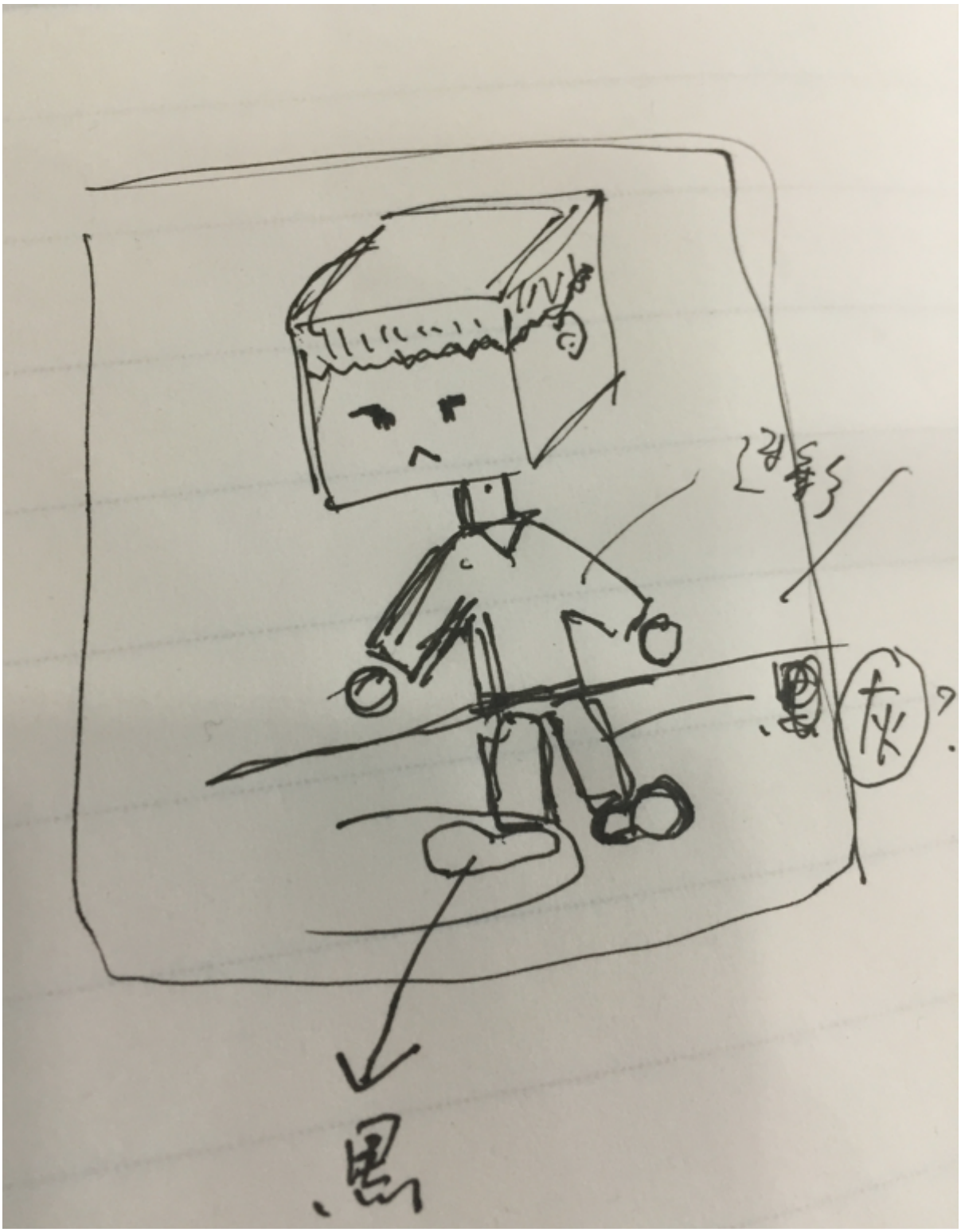
3.1 Buildings

First we need cuboids to outline our scene model and limit the range of movement of characters in this game. Then, to make the buildings like more lifelike, we use triangular prisms to build the roofs at the required level of visual verisimilitude. To be more artistic, we need draw some textures and put them on the walls of the buildings, which would be introduced in details below.



3.2 Models

Model designing is one of the most important work in our project, because it not only affects the whole style, but plays the eye-catching role in the game. Therefore, to design more amazing works, we draw them on the paper and revise them once and once again.



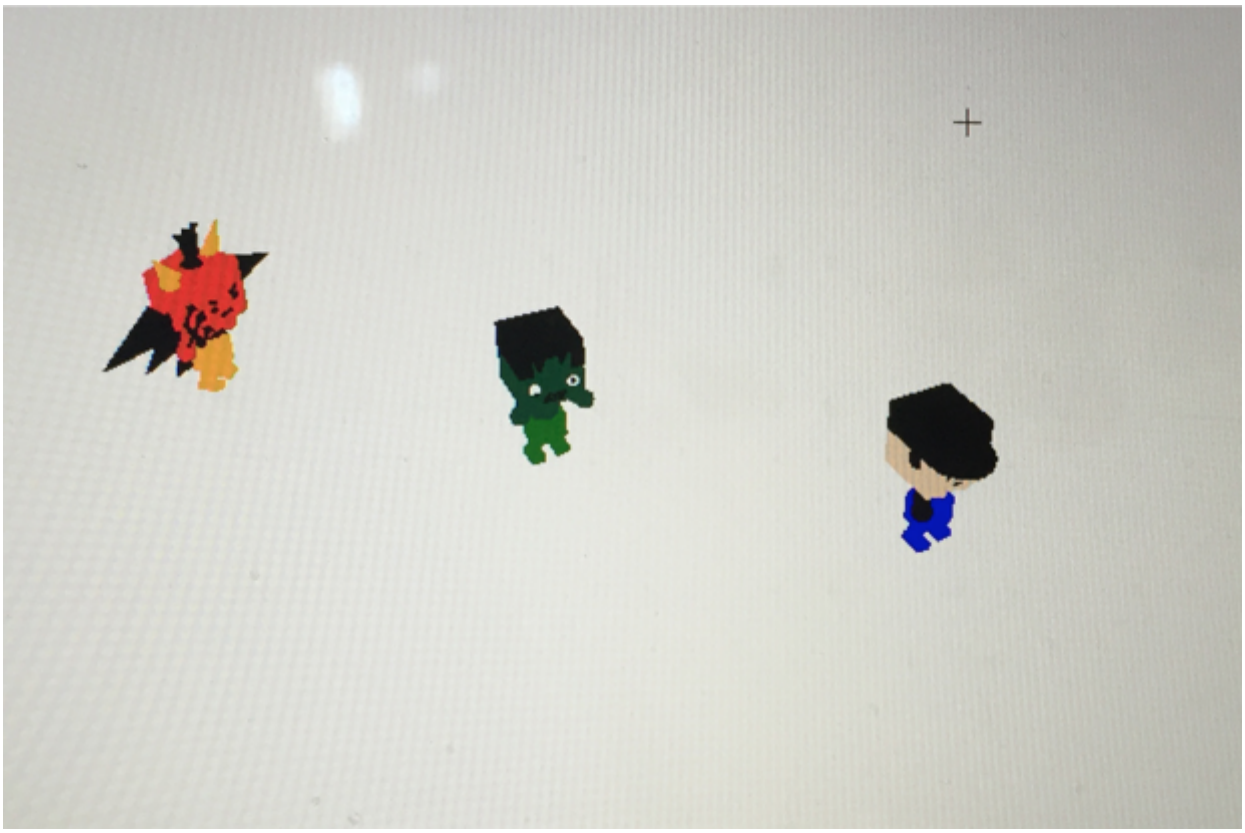
Then, with the help of Microsoft 3D builder, we realize our models in 3D. Then we use Maya to improve them and do some modifications. After these processes, we successfully design our 3D models and save them in .obj format for the following work.

3.3 Loading Models (.obj Files)

We use the function

```
bool loadOBJ(const char * path, std::vector<glm::vec3> & out_vertices, std::vector<glm::vec2> & out_uvs, std::vector<glm::vec3> & out_normals)
```

to load the .obj files into OpenGL. For each vertex of triangle, get the indices of its attributes first. Then, get the attributes thanks to the index. Finally, put the attributes in buffers. As long as we do this job in the time of vertexIndices' size, we would load .obj files successfully.



3.4 Programming Movements (Collisions Detection)

In different levels, we will expose different number of evils and zombies. Their first locations are randomly created with direction all towards the hero (at xd, yd, zd). All enemies move a little step towards the hero each timeslot, thus we can see a chasing movements on the whole.
To rotate all evils' direction towards our heroes, we need to have some mathematical knowledges, and the codes are shown below:

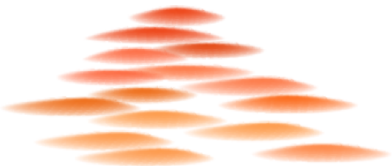
```
void draw_evil(float x,float y,float z,int k)
{
    glTranslatef(x, y, z);
    glRotatef(-90, 0, 1, 0);

    float dx=xd-x,dz=zd-z;
    if (fabs(dx)<0.25&&fabs(dz)<0.25) blood-=1;
    float delta=atan(dx/dz);
    if (dx>0&&dz>0) glRotatef(delta/PI*180.0, 0, 1, 0); else
    if (dx>0&&dz==0) glRotatef(90, 0, 1, 0); else
    if (dx<0&&dz==0) glRotatef(-90, 0, 1, 0); else
    if (dx==0&&dz<0) glRotatef(-180.0, 0, 1, 0); else
    if (dx>0&&dz<0) glRotatef(delta/PI*180.0+180, 0, 1, 0); else
    if (dx<0&&dz>0) glRotatef(delta/PI*180.0, 0, 1, 0); else
    if (dx<0&&dz<0) glRotatef(-180+delta/PI*180.0, 0, 1, 0);
    ......
}
```

Collisions detection is quite simple in our game since all obj characters are designed by ourselves with equal height and width. So we only need to compare each object's center and judge whether it is less than a certain distance.

3.5 Textures Designing

We use Adobe Illustrator and Adobe Photoshop to design the background and the texture for the buildings along the desert road. Since these objects in the scene are created using vector graphics, they can be scaled to any size without a decrease in image quality. First are the blue sky, clouds and sun. Also a cactus (because we are in a desert). On the first layer, draw a square or rectangle that spanning the art board, and fill it with our desired sky color. Our next step is going to be to create the key component of our clouds. We create a symbol which is comprised of 3 ellipses with radial gradient fills. Since we will be using these shapes as the basis for our clouds, we don't want our symbol to be completely symmetric, so we modify our duplicate ellipse a bit. We have reduced the height of my duplicate ellipse to about 80px and rotated it -23°, using the Selection Tool.

To maximize the district that our players can reach, we do a few alterations by removing the clouds and making the sun smaller after importing these elements into Adobe Photoshop for further editing. We also add some ellipse-based shapes to the ground to make our desert more lifelike.

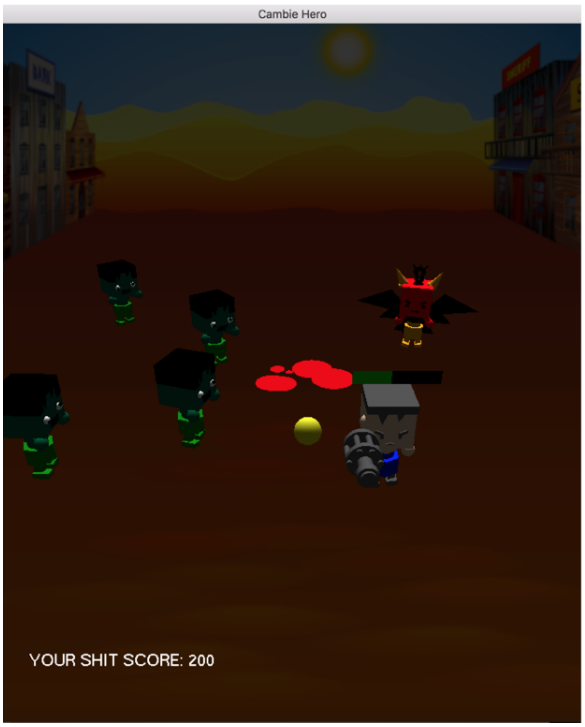


The steps to design textures for the buildings are similar.
Below is how we make out textures.

```
GLuint maketex(const char* tfile, GLint xSize, GLint ySize) {
    GLuint rmesh;
    FILE* file;
    unsigned char* texdata = (unsigned char*)malloc(xSize * ySize * 3);
 //3 is {R,G,B}

    file = fopen(tfile, "rb" );
    fseek(file,54,SEEK_CUR); //54 is bmp header size
    fread(texdata, xSize * ySize * 3, 1, file);
    fclose(file);
    glEnable(GL_TEXTURE_2D);

    char* colorbits = new char[xSize * ySize * 3];

    for(GLint a = 0; a < xSize * ySize * 3; ++a) colorbits[a] = 0xFF;

    glGenTextures(1, &rmesh);
    glBindTexture(GL_TEXTURE_2D, rmesh);

    glTexImage2D(GL_TEXTURE_2D, 0 ,3 , xSize,
                 ySize, 0 , GL_RGB, GL_UNSIGNED_BYTE, colorbits);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    //Save viewport and set up new one
    GLint viewport[4]; //4 is {X,Y,Width,Height}
    glGetIntegerv(GL_VIEWPORT, (GLint*)viewport);
    glViewport(0, 0, xSize, ySize);

    //Clear target and depth buffers
    glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

    glPushMatrix(); //Duplicates MODELVIEW stack top
    glLoadIdentity(); //Replace new top with {1}

    glDrawPixels(xSize, ySize, GL_BGR, GL_UNSIGNED_BYTE, texdata);
    glPopMatrix();
    glCopyTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,
                     0, 0, xSize, ySize, 0);

    //Restore viewport
    glViewport(viewport[0], viewport[1], viewport[2], viewport[3]); //
{X,Y,Width,Height}

    delete[] colorbits;
    free(texdata);
    return rmesh;
}
```

3.6 Lighting & Modify (Norms)

Lighting settings in this program is simple. We only added a center lighting over the map high in the sky. But when we were realizing it, we found that it's important to declare the norms in order to get a good rendering result.
Especially the textures and other images we want to show in the game.



3.7 Camera Position

Camera in this game is fixed at a certain position, alwayts shooting at (0,0,0). However when `F1` key is pressed down, the camera will have a translation to a further location.

3.8 Advanced work

3.8.1 Shooting & Bleeding

When the hero fires the bullets, it looks like a straight line but damages in a certain angle which is like a sector. If the distance between the evils and the hero is no more than a fixed value, the evils would be hit and lose their lives. At the time they are hit, there would appear a red circle, radius of which is one to three randomly, in the place away from them in one to three randomly as well. That would make it look like blood from the dead evils. And because it is stored in a loop array whose size is 1000, after one thousand evils are killed, the initial circle would disappear one by one. That looks like the blood drying.

3.8.2 Functional Cubes

We design two kind of cubes, one yellow and the other green.

```
void draw_cube(float x, float z, int kind) {
    glTranslatef(x, 0.1, z);
    glRotatef(-45, 0, 1, 0);
    glBegin(GL_TRIANGLES);
    if(kind <= 45) {
        material[0] = 1; material[1] = 1; material[2] = 0; material[3] = 1;
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, material);
    } else {
        material[0] = 0; material[1] = 1; material[2] = 0; material[3] = 1;
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, material);
    }
    glutSolidCube(0.25);
    glRotatef(45, 0, 1, 0);
    glTranslatef(-x, -0.1, -z);
}
```

The cube status is initialized as 0 at the beginning. With the function `void Timer(int value)`, it would change the cube status at a certain time. And with the random algorithm, it appears in random places.
After the hero touches it, the status will be turned into 0 again, which makes the cube disappear. And the hero's blood or weapon would be improved.

3.8.3 Change Weapons

Our weapons are equipped in different heroes respectively, and they are a whole. We would introduce this part below.

3.8.4 Character Changing

As different characters for the gamer are designed, the changing process becomes quite easy. We just switch the Obj file here.

3.8.5 Score Counter

We have a score counter `score_int` , which is the mark of your result. It appears in the bottom left of the scene. And after defeating a little monster and an evil, it would increase 100 points and 500 points respectively. What's more, we set it white for distinguish it from the background.

3.8.6 Pause, Restart & Quit

After pressing `m` , the whole progress would pause. And the key `r` would make the game recover to the initial status, letting it restart. And the key corresponds to quitting.

3.8.7 Zoom In & Zoom Out

Pressing F1 can make the camera move to a different location and thus realizing the zoom in&zoom out functions.

4 Credit

- Yunhao, Jiao: Designed the movements of the characters and different functions of the game. Made the trail video.
- Dunbang, He: Loaded Obj File into OpenGL and designed textures.
- Xinyi, Lin: Tested and fixed bugs. Wrote the overall report / Keynote.

Hope you like our game.