**游**

**戏**

**设**

**计**

**文**

**档**

目录

[一． 功能设计 2](#_Toc21366_WPSOffice_Level1)

[1. 功能概述 2](#_Toc1560_WPSOffice_Level2)

[2. 设计说明 2](#_Toc24391_WPSOffice_Level2)

[2.1 碰撞检测 2](#_Toc1560_WPSOffice_Level3)

[2.2 运动控制 4](#_Toc24391_WPSOffice_Level3)

[二． 外观设计 7](#_Toc1560_WPSOffice_Level1)

[1. 外观概述 7](#_Toc4931_WPSOffice_Level2)

[2. 设计说明 8](#_Toc2017_WPSOffice_Level2)

[2.1天空盒 8](#_Toc4931_WPSOffice_Level3)

[2.2小车 10](#_Toc2017_WPSOffice_Level3)

[2.3道路 13](#_Toc19137_WPSOffice_Level3)

[2.4 起点终点 17](#_Toc26787_WPSOffice_Level3)

[2.5 bmp格式图片导入 19](#_Toc16748_WPSOffice_Level3)

[2.6 光照 22](#_Toc11765_WPSOffice_Level3)

# 功能设计

## 功能概述

我们组设计的这款游戏原型是赛车游戏，两款NPC小车做了细微的速度控制，玩家需要做的就是控制黑色小车进行竞赛，争取在规避碰撞之后第一个到达终点，达到获胜的成就。

## 设计说明

### 碰撞检测

车都用AABB盒包裹，实时维护AABB盒在x、y、z轴的最大值，并判断其他盒子是否有点落入。

#### 关键函数说明

int ImpactCheck(Car \*npc)

if (mycar.car\_CBV.OverlapTest(npc->car\_CBV) == true)

{

std::swap(mycar.speed, npc->speed);

//如果角度太大就不交换运动方向了，如果角度小就是完全弹性碰撞

float delta = (mycar.Edir.h - npc->Edir.h);

while (delta > 360) delta -= 360;

while (delta < 0) delta += 360;

if (delta > 180) delta = 180 - delta;

delta = fabs(delta);

bool flag = 0; //表示是否交换方向了

if (delta < 75)

{

std::swap(mycar.dir, npc->dir);

std::swap(mycar.Edir, npc->Edir);

flag = 1;

}

//npc->AutoRun();

if (mycar.speed - npc->speed >= EPS)

{

if (flag == 1)mycar.pos = mycar.pos + mycar.dir\*0.05;

else mycar.pos = mycar.pos + npc->dir\*(0.05);

npc->pos = npc->pos + npc->dir\*(-0.05);

npc->speed -= min(npc->speed, 0.01);

}

else

{

mycar.pos = mycar.pos + mycar.dir\*(-0.05);

if (flag == 1) npc->pos = npc->pos + npc->dir\*0.05;

else npc->pos = npc->pos + mycar.dir\*0.05;

mycar.speed -= min(mycar.speed, 0.01);

}

npc->car\_CBV.setpos(npc->pos, npc->dir);

mycar.car\_CBV.setpos(mycar.pos, mycar.dir);

return 1;

}

return 0;

#### 设计思路

1. 碰撞检测设计原理：

碰撞策略：如果两辆小车运动方向夹角较小，就使用完全弹性碰撞，两车交换速度和运动方向。碰撞完成后被撞小车延被撞方向速度增加，撞车的小车沿运动方向速度减小。如果夹角较大（视觉上70度最佳）两个小车将不再交换速度和方向，被撞小车将被挤压着向被撞方向移动。

1. 道路边缘检测：将道路切割成多段条形带。存下条形带的边界、入口和出口值。小车运动时检测是否超越边界。如果超越将将其弹回到道路中。如果到达入口，进入下一个条形带，到达出口，进入上一个条形带。

### 运动控制

#### 关键函数说明

Car::Run()：小车前进状态更新，根据小车的速度计算更新小车的位置。

float length = fabs(speed);

int mdir = 1;

if (speed > 0)

{

mdir = 1;

}

else

{

mdir = -1;

}

if (mdir == 1)

pos = pos + dir \* length;

else

pos = pos + dir \* (-length);

dir.Normalize();

car\_CBV.setpos(pos, dir);

return 1;

Car::AutoRun()：NPC小车自动前进，更新方向、位置信息。

if (isLerp)

{

if (isFirst)

{

CQuaternion QNow, QNext;

float lerpPara[105];

float para = 0;

for (int p = 0; p <= 50; p++)

{

lerpPara[p] = para;

para += 0.02;

}

QNow = dir.ToEuler().ToQuaternion();

QNext = nextdir.ToEuler().ToQuaternion();

QNow.Slerp(QNext, 51, lerpPara, turnLerp);

isFirst = false;

}

if (lerpCnt == 50)

{

isLerp = false;

isFirst = true;

lerpCnt = 0;

}

Edir.h = -turnLerp[lerpCnt].ToEuler().h;

lerpCnt++;

}

else

{

dir = myroad[0].target - pos;

if ((myroad[0].target - pos).len() <= 1)

{

if (myroad.size() > 1)

myroad.erase(myroad.begin());

nextdir = myroad[0].target - pos;

nextdir.Normalize();

dir.Normalize();

isLerp = true;

}

//Car::slerp(Car::dir, nextdir);

dir.Normalize();

Edir = dir.ToEuler();

Run();

}

if (fabs(speed) < EPS) speed = max\_speed;

else if (speed - max\_speed < EPS) speed += 0.0001;

else if (speed - max\_speed > EPS) speed -= 0.0001;

Car::Turnleft()：通过欧拉角计算，让小车向左转一个小角度。

CEuler temp;

CVector updir;

updir.Set(0, 1, 0);

temp.Set(0.8, 0, 0);

Car::Edir.h = Car::Edir.h + temp.h;

Car::dir = Car::Edir.ToVector3(&updir);

return 1;

Car::Turnright()：通过欧拉角计算，让小车向右转一个小角度。

CEuler temp;

CVector updir;

updir.Set(0, 1, 0);

temp.Set(-0.8, 0, 0);

Car::Edir.h = Car::Edir.h + temp.h;

Car::dir = Car::Edir.ToVector3(&updir);

return 1;

#### 设计思路

#### （1）玩家小车的移动

玩家小车由玩家通过键盘操作控制，可以通过”w”、”s”、”a”、”d”或小键盘的”up”、”down”、”left”、”right”控制小车前后左右移动，按下”w”\”up”时小车速度向前增加，按下”s”\”down”时小车速度向后增加，按”a”\”left”小车向左转，”d”\”right”向右转。

#### （2）NPC小车的移动

NPC小车根据地图上的坐标点自动前进。用vector数组存下一些坐标点，NPC小车根据坐标点向前进。

#### （3）多键盘响应

设置与键盘按键相对应的变量，并记录变量的状态，按下的时候为一个状态，松开的时候为另一个状态。

# 外观设计

## 外观概述

从天空盒到赛道，以及开始结束时的广告牌，表示我们组整体的设计师偏向城市风格的。虽然并不能体验“飙车”的快感，但作为一款赛车游戏，这种简单的设计在用户体验上来说，也算是良好。游戏中设计了两辆NPC小车，在速度上有一些区别，为了照顾新手玩家，只要认真规避碰撞，达到游戏胜利即第一个到达的条件也是比较简单的。

## 设计说明

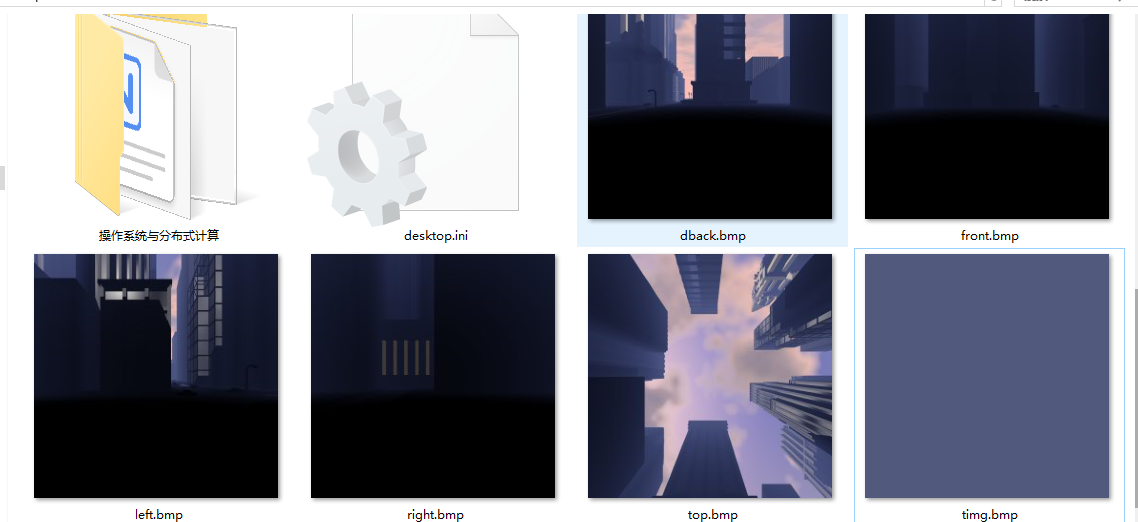
### 2.1天空盒

#### 2.1.1关键函数说明

## （1）天空盒初始化

Skybox::Init()

载入6张bmp格式的环境图片，绑定6个纹理对象，设置天空盒纹理的基本参数，准备构造天空盒。



关键代码：

glBindTexture(GL\_TEXTURE\_2D, m\_texture[i].ID); /\*\*< 创建纹理对象 \*/

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR\_MIPMAP\_NEAREST);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_WRAP\_S,GL\_CLAMP\_TO\_EDGE);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_WRAP\_T,GL\_CLAMP\_TO\_EDGE);

/\*\* 创建纹理 \*/

gluBuild2DMipmaps(GL\_TEXTURE\_2D, GL\_RGB, m\_texture[i].imageWidth,

m\_texture[i].imageHeight, GL\_RGB, GL\_UNSIGNED\_BYTE,

m\_texture[i].image);

其中GL\_TEXTURE\_WRAP\_S，GL\_TEXTURE\_WRAP\_T通常可设置为GL\_REPEAT或GL\_CLAMP两种方式。当待填充的多边形大于纹理的时候，GL\_REPEAT表示多余的部分用重复的方式填充；GL\_CLAMP表示多余的部分用相连边缘的相邻像素填充。在实际绘制中，我们一般采用GL\_CLAMP\_EDGE来处理，这就消除了接缝处的细线，增强了天空盒的真实感。

## （2）构造天空盒

void Skybox::CreateSkyBox(float x, float y, float z,float box\_width, float box\_height,float box\_length)

生成一个方形的天空盒，将6张环境图贴在天空盒的6个面，营造出空间效果。

关键代码：

/\*\* 获得场景中光照状态 \*/

GLboolean lp;

glGetBooleanv(GL\_LIGHTING,&lp);

/\*\* 计算天空盒长 宽 高 \*/

float width = MAP \* box\_width/8;

float height = MAP \* box\_height/8;

float length = MAP \* box\_length/8;

/\*\* 计算天空盒中心位置 \*/

x = x+ MAP/8 - width / 2;

y = y+ MAP/24 - height / 2;

z = z+ MAP/8 - length / 2;

glDisable(GL\_LIGHTING); /\*\*< 关闭光照 \*/

/\*\* 开始绘制 \*/

glPushMatrix();

glTranslatef(-x,-y,-z);

/\*\* 绘制背面 \*/

glBindTexture(GL\_TEXTURE\_2D, m\_texture[0].ID);

glBegin(GL\_QUADS);

/\*\* 指定纹理坐标和顶点坐标 \*/

glTexCoord2f(1.0f, 0.0f); glVertex3f(x + width, y, z);

glTexCoord2f(1.0f, 1.0f); glVertex3f(x + width, y + height, z);

glTexCoord2f(0.0f, 1.0f); glVertex3f(x, y + height, z);

glTexCoord2f(0.0f, 0.0f); glVertex3f(x, y, z);

glEnd();

获取场景中的光照，计算天空盒长宽高，计算天空盒位置，然后在天空盒六个面上绑定纹理，构造天空盒。

#### 2.1.2设计思路

天空盒其实就是一个覆盖场景四周的长方体，但它的各个面上贴有表示天空的纹理图片，即四周的4面纹理的边与顶面纹理的边相连，同时四面纹理前后相连，纹理大小要是2的N次方(32,64,128,…)。天空盒是天空建模中十分常用的方法，将纹理贴在长方体的6个面上就能很简单的模拟出空间效果。

### 2.2小车

#### 关键函数说明

/\*\* 设置材质属性 \*/

GLfloat mat\_ambient[] = { 0.8f, 0.8f, 0.8f, 1.0f };

GLfloat mat\_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };

glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_ambient);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);

//车身

glPushMatrix();

glTranslatef(pos.x, pos.y, pos.z);

//glRotated(Edir.h, 0, 1, 0);

glScalef(size, 0.4\*size, 2.5\*size);

/\*\* 选择纹理 \*/

glBindTexture(GL\_TEXTURE\_2D, tm->m\_Texture[5].ID);

/\*\* 开始绘制车身 \*/

glBegin(GL\_QUADS);

/// 左侧面

glNormal3f(-1.0f, 0.0f, 0.0f); /\*\*< 指定法线朝左 \*/

glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);

glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);

glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);

glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);

/// 右侧面

glNormal3f(1.0f, 0.0f, 0.0f); /\*\*< 指定法线朝右 \*/

glTexCoord2f(1.0f, 0.0f); glVertex3f(1.0f, -1.0f, -1.0f);

glTexCoord2f(1.0f, 1.0f); glVertex3f(1.0f, 1.0f, -1.0f);

glTexCoord2f(0.0f, 1.0f); glVertex3f(1.0f, 1.0f, 1.0f);

glTexCoord2f(0.0f, 0.0f); glVertex3f(1.0f, -1.0f, 1.0f);

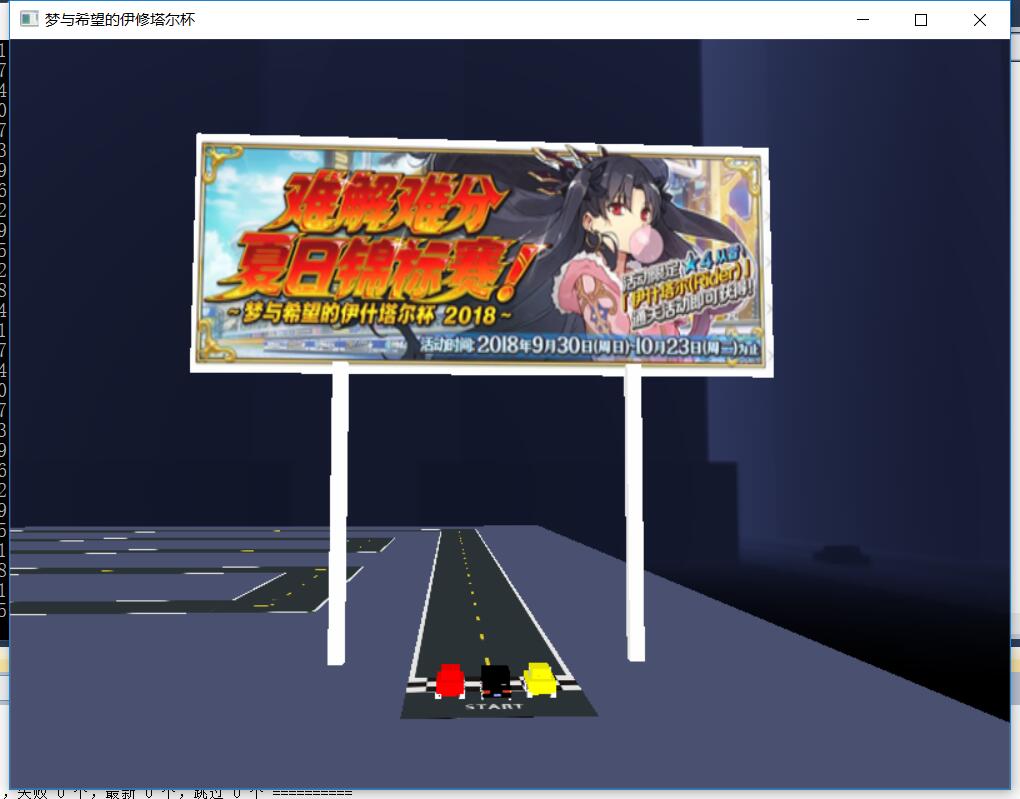
glEnd();

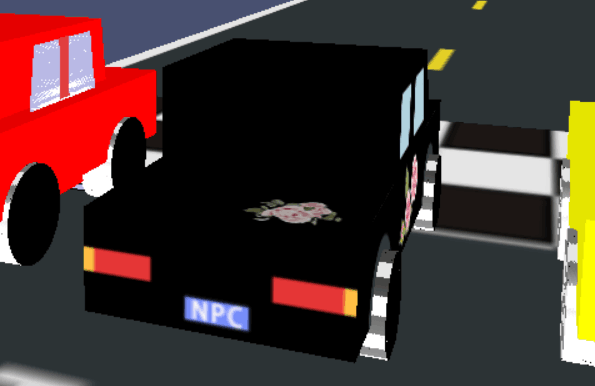
glBindTexture(GL\_TEXTURE\_2D, tm->m\_Texture[7].ID);

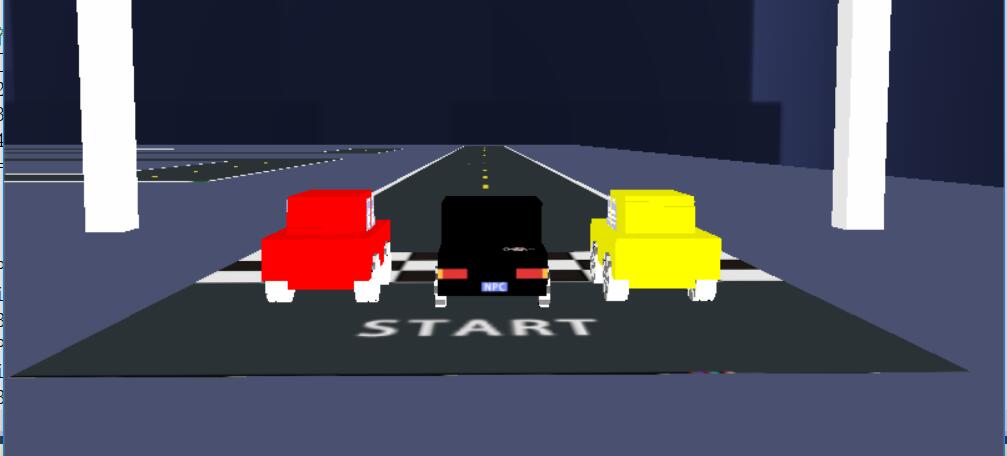
glBegin(GL\_QUADS);

#### 设计思路

一共设计了三辆小车，其中两辆红色和黄色的小车作为NPC小车，黑色的小车，添加了一些比如尾部车灯车头灯等纹理贴纸，作为用户使用的参赛小车。







### 2.3道路

#### 2.3.1 关键函数说明

##### （1）设置道路

road\_x = 475;

road\_y = 510;

road\_z = 473;

/\*\*DrawRoad(41,1),x负方向到z正方向的拐角\*\*/

/\*\* 设置材质属性 \*/

GLfloat mat\_ambient[] = { 0.8f, 0.8f, 0.8f, 1.0f };

GLfloat mat\_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };

glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_ambient);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);

glPushMatrix();

glTranslatef(road\_x, road\_y, road\_z);

glScalef(2.0f, 0.01f, 2.0f);

/\*\* 选择纹理 \*/

glBindTexture(GL\_TEXTURE\_2D, m\_Texture[10].ID);

/\*\* 开始绘制四边形 \*/

glBegin(GL\_QUADS);

/// 顶面

glNormal3f(0.0f, 1.0f, 0.0f); /\*\*< 指定法线向上 \*/

glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);

glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, 1.0f, 1.0f);

glTexCoord2f(1.0f, 0.0f); glVertex3f(1.0f, 1.0f, 1.0f);

glTexCoord2f(1.0f, 1.0f); glVertex3f(1.0f, 1.0f, -1.0f);

glEnd();

glPopMatrix();

DrawRoad(1, 19);

DrawRoad(13, 1);

DrawRoad(3, 18);

DrawRoad(32, 1);

DrawRoad(2, 3);

DrawRoad(24, 1);

DrawRoad(4, 16);

DrawRoad(42, 1);

DrawRoad(2, 3);

DrawRoad(23, 1);

DrawRoad(3, 16);

DrawRoad(32, 1);

DrawRoad(2, 3);

DrawRoad(24, 1);

DrawRoad(4, 16);

DrawRoad(42, 1);

DrawRoad(2, 3);

DrawRoad(23, 1);

DrawRoad(3, 17);

glPushMatrix();

glTranslatef(road\_x + 4, road\_y, road\_z);

glScalef(2.0f, 0.01f, 2.0f);

/\*\* 选择纹理 \*/

glBindTexture(GL\_TEXTURE\_2D, m\_Texture[11].ID);

/\*\* 开始绘制四边形 \*/

glBegin(GL\_QUADS);

/// 顶面

glNormal3f(0.0f, 1.0f, 0.0f); /\*\*< 指定法线向上 \*/

glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);

glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, 1.0f, 1.0f);

glTexCoord2f(1.0f, 0.0f); glVertex3f(1.0f, 1.0f, 1.0f);

glTexCoord2f(1.0f, 1.0f); glVertex3f(1.0f, 1.0f, -1.0f);

glEnd();

glPopMatrix();

##### （2）绘制道路

/\*\* 设置材质属性 \*/

GLfloat mat\_ambient[] = { 0.8f, 0.8f, 0.8f, 1.0f };

GLfloat mat\_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };

glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_ambient);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);

/\*\* dir=1,z轴正方向，dir=2,z轴负方向，dir=3,x轴正方向，dir=4,x轴负方向 \*\*/

/\*\*赛道\*\*/

for (int i = 1; i <= length; i++)

{

glPushMatrix();

if (dir == 1 || dir == 13 || dir == 14)

road\_z += 4;

else if (dir == 2 || dir == 23 || dir == 24)

road\_z -= 4;

else if (dir == 3 || dir == 31 || dir == 32)

road\_x += 4;

else if (dir == 4 || dir == 41 || dir == 42)

road\_x -= 4;

glTranslatef(road\_x, road\_y, road\_z);

if (dir == 3 || dir == 4)

glRotatef(90, 0, 1, 0);

if (dir == 14 || dir == 32)

glRotatef(180, 0, 1, 0);

if (dir == 13 || dir == 42)

glRotatef(90, 0, 1, 0);

if (dir == 41 || dir == 23)

glRotatef(0, 0, 1, 0);

if (dir == 31 || dir == 24)

glRotatef(-90, 0, 1, 0);

glScalef(2.0f, 0.01f, 2.0f);

/\*\* 选择纹理 \*/

if (dir <= 4)

glBindTexture(GL\_TEXTURE\_2D, m\_Texture[2].ID);

else

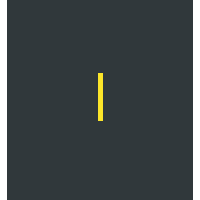
glBindTexture(GL\_TEXTURE\_2D, m\_Texture[3].ID);

/\*\* 开始绘制四边形 \*/

glBegin(GL\_QUADS);

#### 2.3.2 设计思路

组里的美工绘制了两种重复使用的道路元素，在拼合成整个场景地图的时候，采用这两个元素进行拼接。（直线部分和拐弯部分）



拼接效果：



### 2.4 起点终点

#### 2.4.1 关键函数说明

void TextureManager::DrawBeginAndEnd()

/\*\* 设置材质属性 \*/

GLfloat mat\_ambient[] = { 0.8f, 0.8f, 0.8f, 1.0f };

GLfloat mat\_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };

glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_ambient);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);

glPushMatrix();

glTranslatef(475, 521, 477);

glScalef(8.0f, 3.3f, 0.2f);

/\*\* 选择纹理 \*/

glBindTexture(GL\_TEXTURE\_2D, m\_Texture[4].ID);

/\*\* 开始绘制四边形 \*/

glBegin(GL\_QUADS);

/// 后侧面

glNormal3f(0.0f, 0.0f, -1.0f); /\*\*< 指定法线背向观察者 \*/

glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);

glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);

glTexCoord2f(0.0f, 1.0f); glVertex3f(1.0f, 1.0f, -1.0f);

glTexCoord2f(0.0f, 0.0f); glVertex3f(1.0f, -1.0f, -1.0f);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslatef(471, 514, 477);

glScaled(0.4, 8, 0.4);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslatef(479, 514, 477);

glScaled(0.4, 8, 0.4);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslatef(553, 521, 489);

glScalef(0.2f, 3.3f, 8.0f);

/\*\* 选择纹理 \*/

glBindTexture(GL\_TEXTURE\_2D, m\_Texture[4].ID);

/\*\* 开始绘制四边形 \*/

glBegin(GL\_QUADS);

/// 左侧面

glNormal3f(-1.0f, 0.0f, 0.0f); /\*\*< 指定法线朝左 \*/

glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);

glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);

glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);

glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslatef(553, 514, 485);

glScaled(0.4, 8, 0.4);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslatef(553, 514, 493);

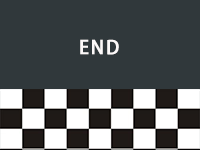
glScaled(0.4, 8, 0.4);

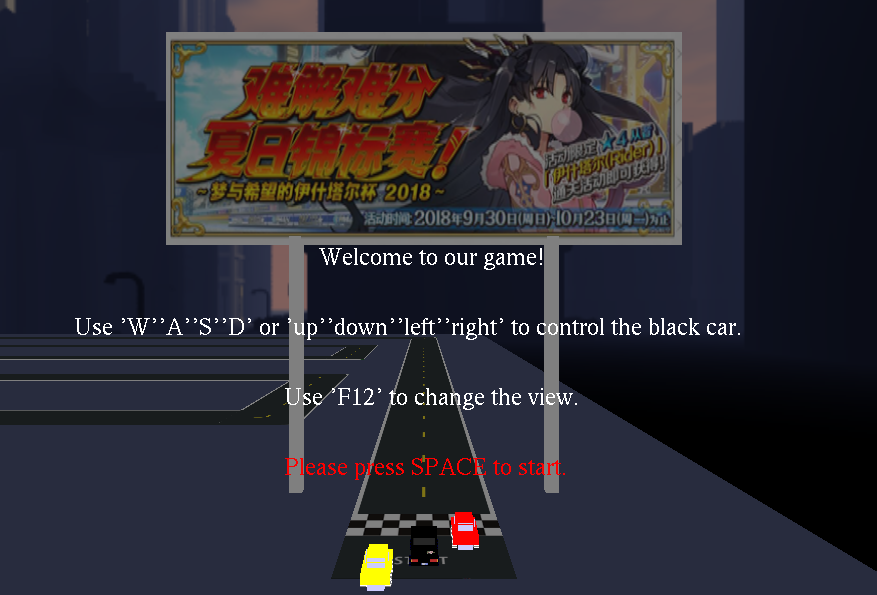
glutSolidCube(1);

glPopMatrix();

#### 2.4.2 设计思路

在终点和起点处均设计大型广告牌和起终点标识，用来指示起点和终点的位置。





### 2.5 bmp格式图片导入

#### 2.5.1 关键函数说明

CBMPLoader::CBMPLoader()

{

/\*\* 初始化成员值为0 \*/

image = 0;

imageWidth = 0;

imageHeight = 0;

}

/\*\* 析构函数 \*/

CBMPLoader::~CBMPLoader()

{

FreeImage(); /\*\*< 释放图像数据占据的内存 \*/

}

/\*\* 装载一个位图文件 \*/

bool CBMPLoader::LoadBitmap(char \*file)

{

FILE \*pFile = 0; /\*\*< 文件指针 \*/

/\*\* 创建位图文件信息和位图文件头结构 \*/

BITMAPINFOHEADER bitmapInfoHeader;

BITMAPFILEHEADER header;

unsigned char textureColors = 0;/\*\*< 用于将图像颜色从BGR变换到RGB \*/

/\*\* 打开文件,并检查错误 \*/

pFile = fopen(file, "rb");

if(pFile == 0)

{

printf("%d\n",2);

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)

{

fclose(pFile); /\*\*< 若不是位图文件,则关闭文件并返回 \*/

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) /\*\*< 若分配内存失败则返回 \*/

{

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;

}

fclose(pFile); /\*\*< 关闭文件 \*/

return true; /\*\*< 成功返回 \*/

}

/\*\* 释放内存 \*/

void CBMPLoader::FreeImage()

{

/\*\* 释放分配的内存 \*/

if(image)

{

delete[] image;

image = 0;

}

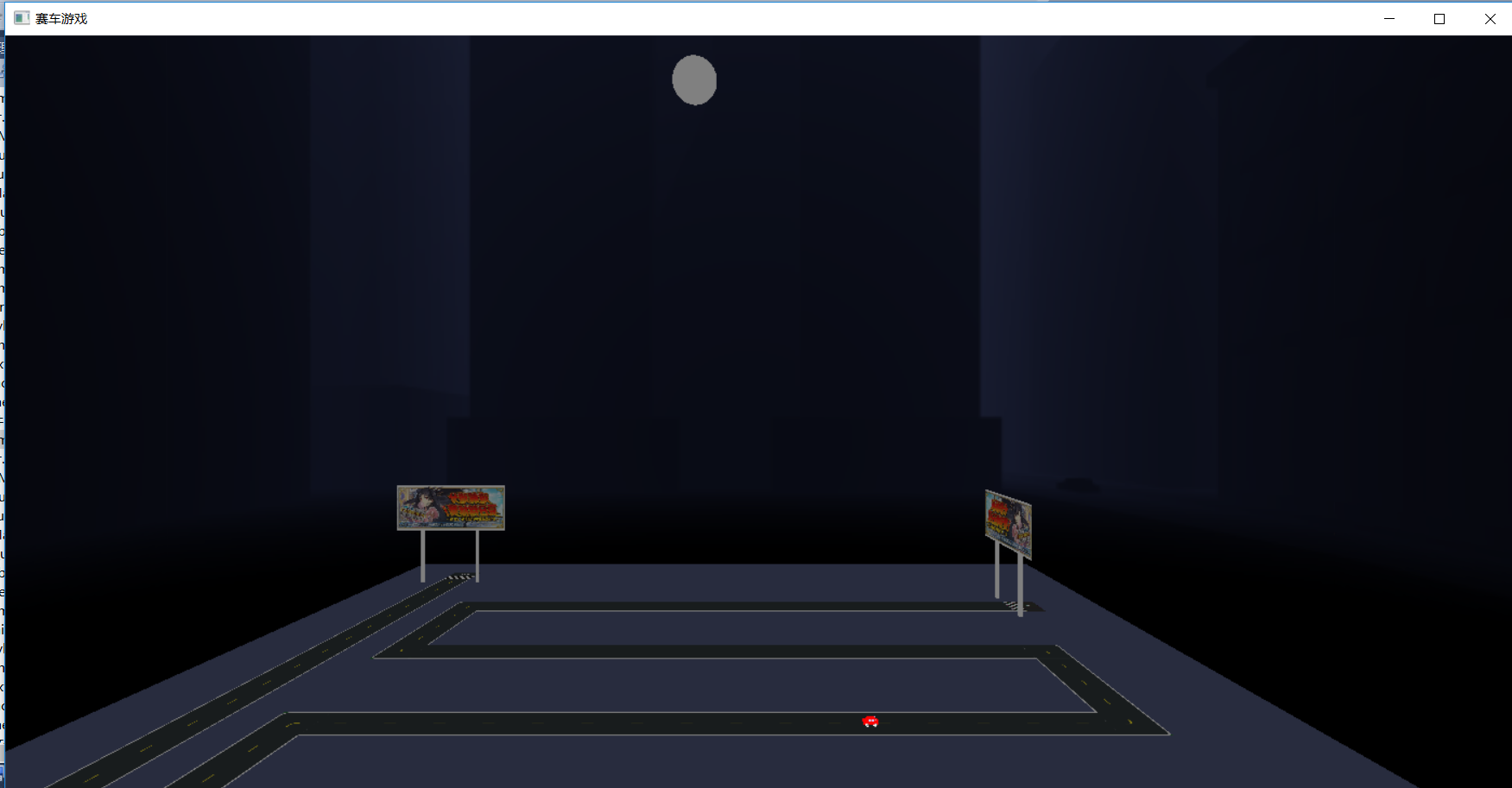
}

#### 2.5.2 设计思路

导入bmp格式 的图片文件，用于给环境进行贴图。

### 2.6 光照

下图所示白色圆球即为光源



#### 2.6.1 关键函数说明

/\*\* 定义光源的属性值 \*/

GLfloat LightAmbient[] = { 1.0f, 1.0f, 1.0f, 1.0f }; /\*\*< 环境光参数 \*/

GLfloat LightDiffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f }; /\*\*< 漫射光参数 \*/

GLfloat LightSpecular[] = { 1.0f, 1.0f, 1.0f, 1.0f }; /\*\*< 镜面光参数 \*/

GLfloat LightPosition[] = { 0.0f, 0.0f, 0.0f, 0.0f }; /\*\*< 光源位置 \*/

GLfloat LightPosition1[] = { 0.0f, 0.0f, 1.0f, 1.0f }; /\*\*< 光源位置2 \*/

glLightfv(GL\_LIGHT0, GL\_POSITION, LightPosition); /\*\*< 设置光源位置 \*/

glLightfv(GL\_LIGHT1, GL\_POSITION, LightPosition1); /\*\*< 设置光源位置 \*/

#### 2.6.2 设计思路

本次游戏设计采用了两种光源，一种是平行光源，一种是点光源。

（1）环境反射：环境光(Ambient Light)

周围环境入射至景物表面并等量地向各个方向反射出去

（2）漫反射：漫射光(Diffuse Light)

特定光源照射在景物表面，向空间各方向均匀反射出去的光

1. 镜面反射：朝一定方向的反射光

