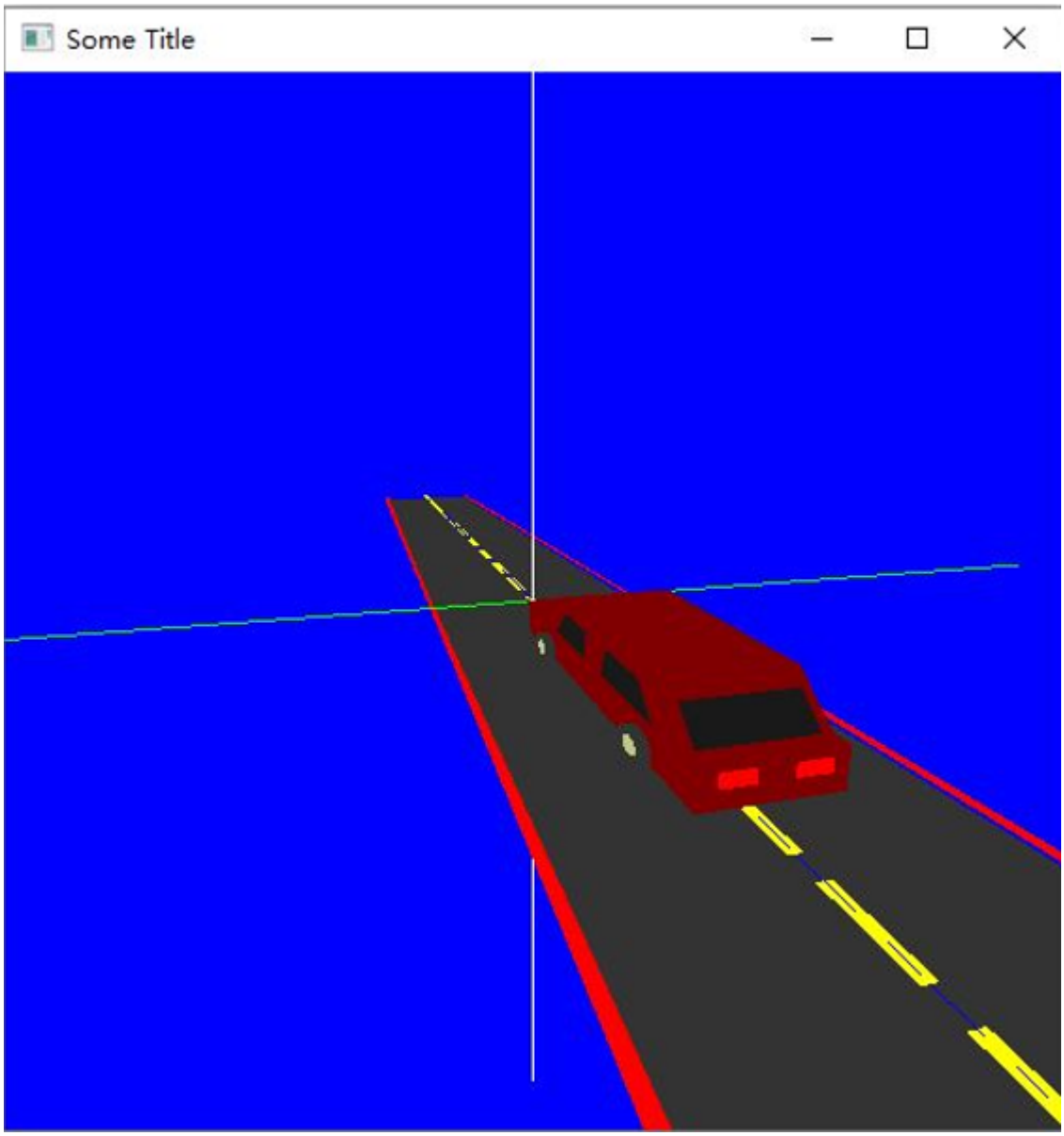
Lab2:A Racing Car

1. 需求

- 创建并渲染一个3D汽车模型和跑道，汽车要有车体和轮子。
- 实现利用键盘控制汽车前进、后退、转弯、加速、减速
- 支持两种视图，用”t”键切换
    1. 坐在车内从驾驶座位向前看的视角
    2. 车外一个固定视点

效果图：



2. 实验环境

```
CPU Intel(R) Core(TM) i5-8250U CPU @1.60GHZ 8核
IDE VS2022
包管理工具 Nuget
依赖库 nupengl.core 0.1.0.1
```

虽然nupengl已经停止维护了，但是历史版本还是可以用的

3. 实验步骤

将分为2个部分：

1. **三维建模**
    a. 绘制小车
    b. 绘制跑道
2. **动画制作**
    a. 实体控制
    b. 视点切换

3.1. 三维建模

这个Lab中的实体只涉及到两个，一个是小车，一个是跑道。先来看小车怎么画。

3.1.1. 小车建模

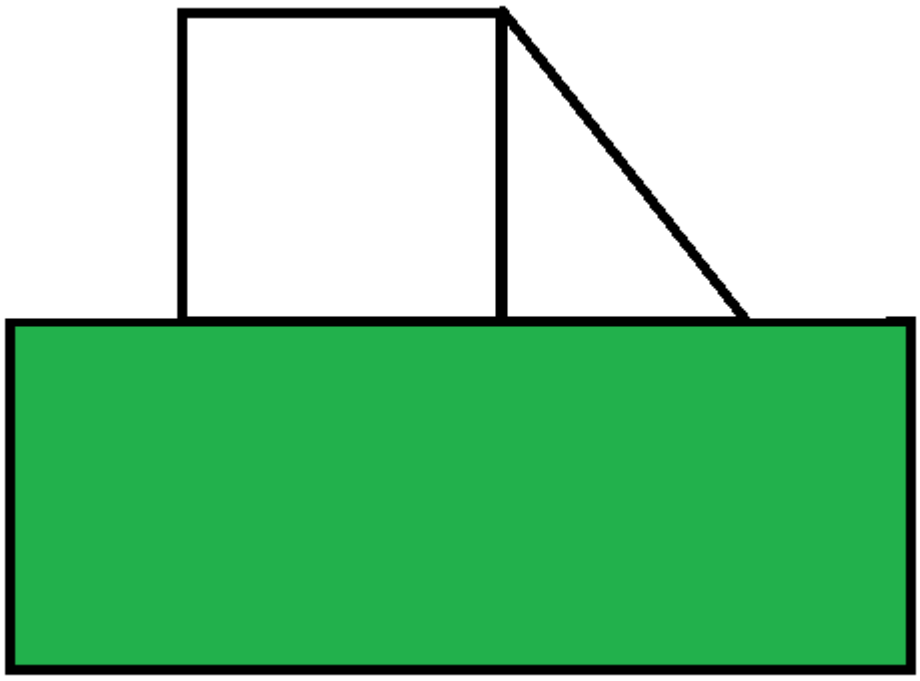
我们可以把小车拆解开，对于车表面的不同位置用不同形状的面片绘制，最后拼接在一起，即可得到一辆完整的车。

很自然地，可以分为以下几个部分：

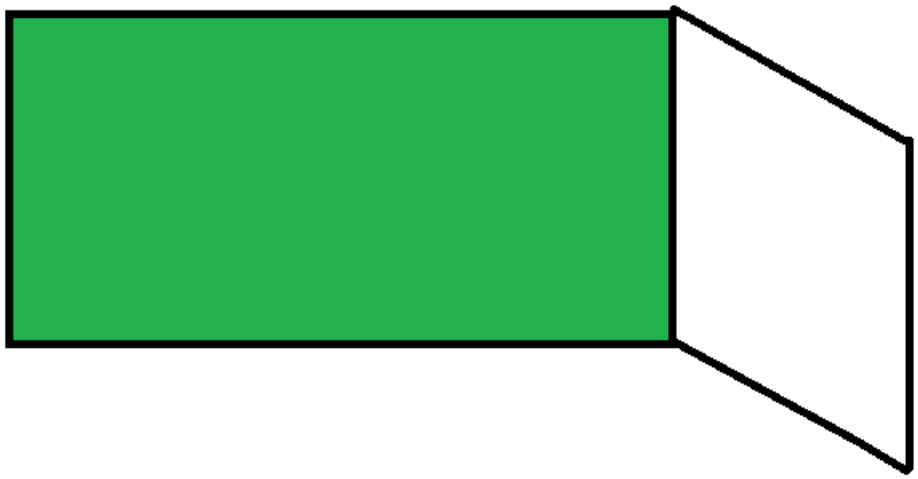
1. 车底
2. 车头
3. 车尾
4. 左侧面（`不含车窗`）
5. 右侧面（`不含车窗`）
6. 车顶（`含前后车窗`）
7. 左车窗
8. 右车窗
9. 轮子

为什么左侧面和右侧面尽可能不要包含车窗呢？主要还是因为车窗的颜色的车体不太一样。

但为什么车顶可以含车窗呢？我们不妨看下效果图，车顶的车窗是可以完全和车体拼接起来的。但车侧面的车窗无法完全和车体拼接起来。简单来说就是这个意思：

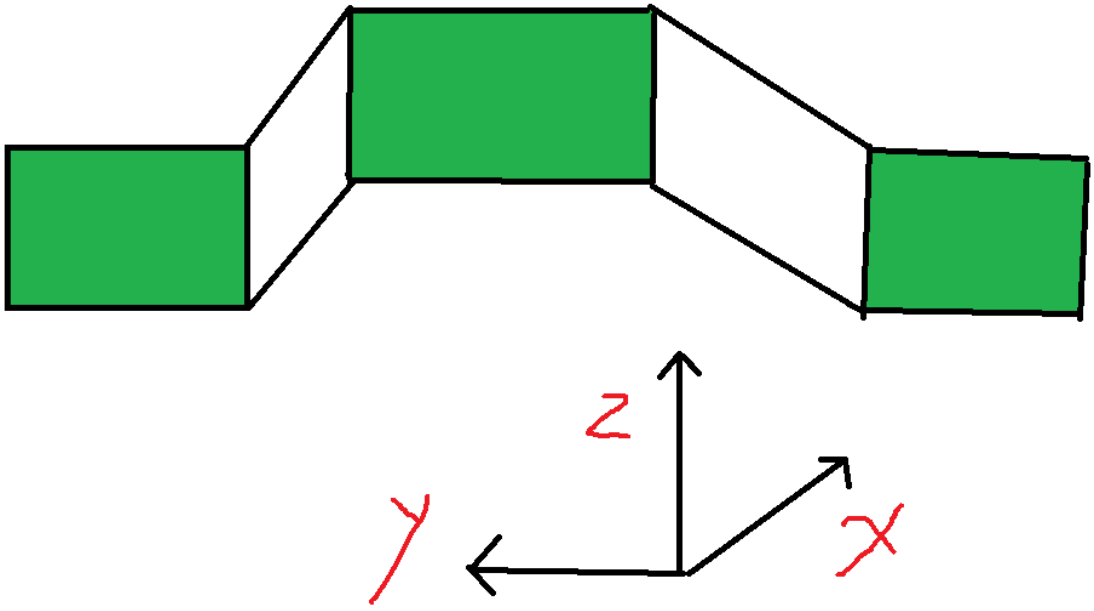


可以看到车体侧面是要长于车侧窗的（侧视图）。因此在绘制时拆开来画比较方便。但是对于车顶以及前后车窗，就是这样的：



这是一个简易的俯视图，可以看到车前窗和车顶是可以“完全”拼接在一起的。因此绘制的时候一起画比较方便。

ok，那么就按照以上的拆解，把车画出来好了。这里就以车顶为例子（最难画的一部分）：



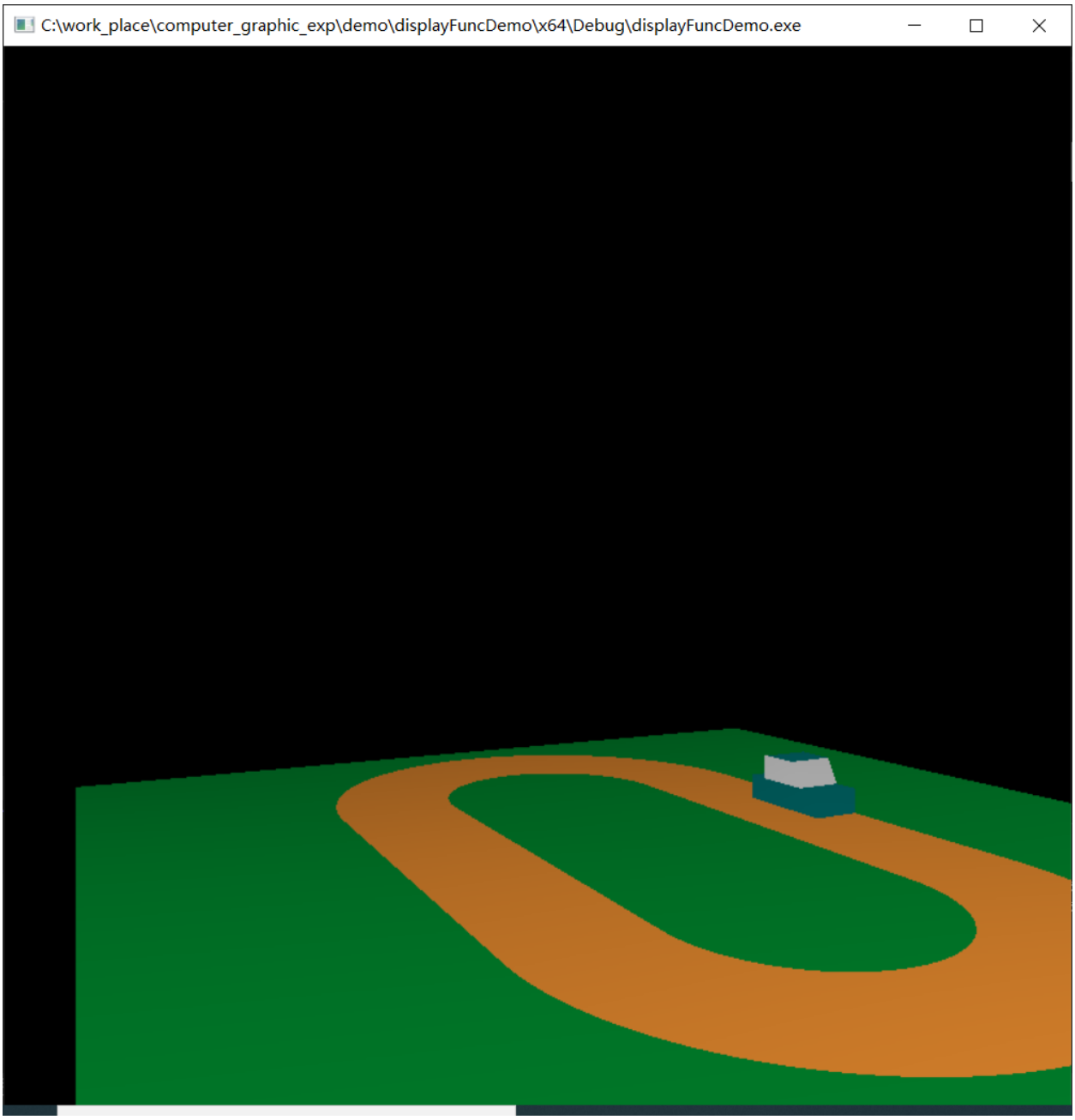
车顶即五个多边形拼起来的，很显然最顶上以及前后车窗形成了一个坡，因此这三个地方的z要高于剩下两个地方的z。

画车顶的代码如下：

```
glBegin(GL_POLYGON);
glColor3f(R, G, B);
glVertex3f(1.0f, 2.0f, 1.5f);
glVertex3f(-1.0f, 2.0f, 1.5f);
glVertex3f(-1.0f, 1.0f, 1.5f);
glVertex3f(1.0f, 1.0f, 1.5f);
glEnd();

glBegin(GL_POLYGON);
glColor3f(1, 1, 1);
glVertex3f(-1.0f, 1.0f, 1.5f);
glVertex3f(-1.0f, 0.5f, 2.5f);
glVertex3f(1.0f, 0.5f, 2.5f);
glVertex3f(1.0f, 1.0f, 1.5f);
glEnd();

glBegin(GL_POLYGON);
glColor3f(R, G, B);
glVertex3f(1.0f, 0.5f, 2.5f);
glVertex3f(-1.0f, 0.5f, 2.5f);
glVertex3f(-1.0f, -1.0f, 2.5f);
glVertex3f(1.0f, -1.0f, 2.5f);
glEnd();

glBegin(GL_POLYGON);
glColor3f(1, 1, 1);
glVertex3f(1.0f, -1.0f, 2.5f);
glVertex3f(-1.0f, -1.0f, 2.5f);
glVertex3f(-1.0f, -1.2f, 1.5f);
glVertex3f(1.0f, -1.2f, 1.5f);
glEnd();

glBegin(GL_POLYGON);
glColor3f(R, G, B);
glVertex3f(1.0f, -1.2f, 1.5f);
glVertex3f(-1.0f, -1.2f, 1.5f);
glVertex3f(-1.0f, -2.0f, 1.5f);
glVertex3f(1.0f, -2.0f, 1.5f);
glEnd();
```

可以看到，中间代码段的z值是最大的，然后z值向两侧递减。

这里因为车轮是圆形的，跟其他部位不一样，所以之后单独列出来。先看看把车表面拼起来之后长啥样。



车位于（0，0，0）处，我找了一个看起来清楚点的视点坐标，大概是（20，30，10）。

```
gluLookAt(20,30,10,lookat[0], lookat[1], lookat[2],
    up[0], up[1], up[2]);
```

最后是车的轮子部分。

OpenGL提供了一个专门用来绘制圆（环）的函数：

glSolidTorus，绘制一个实心圆环。

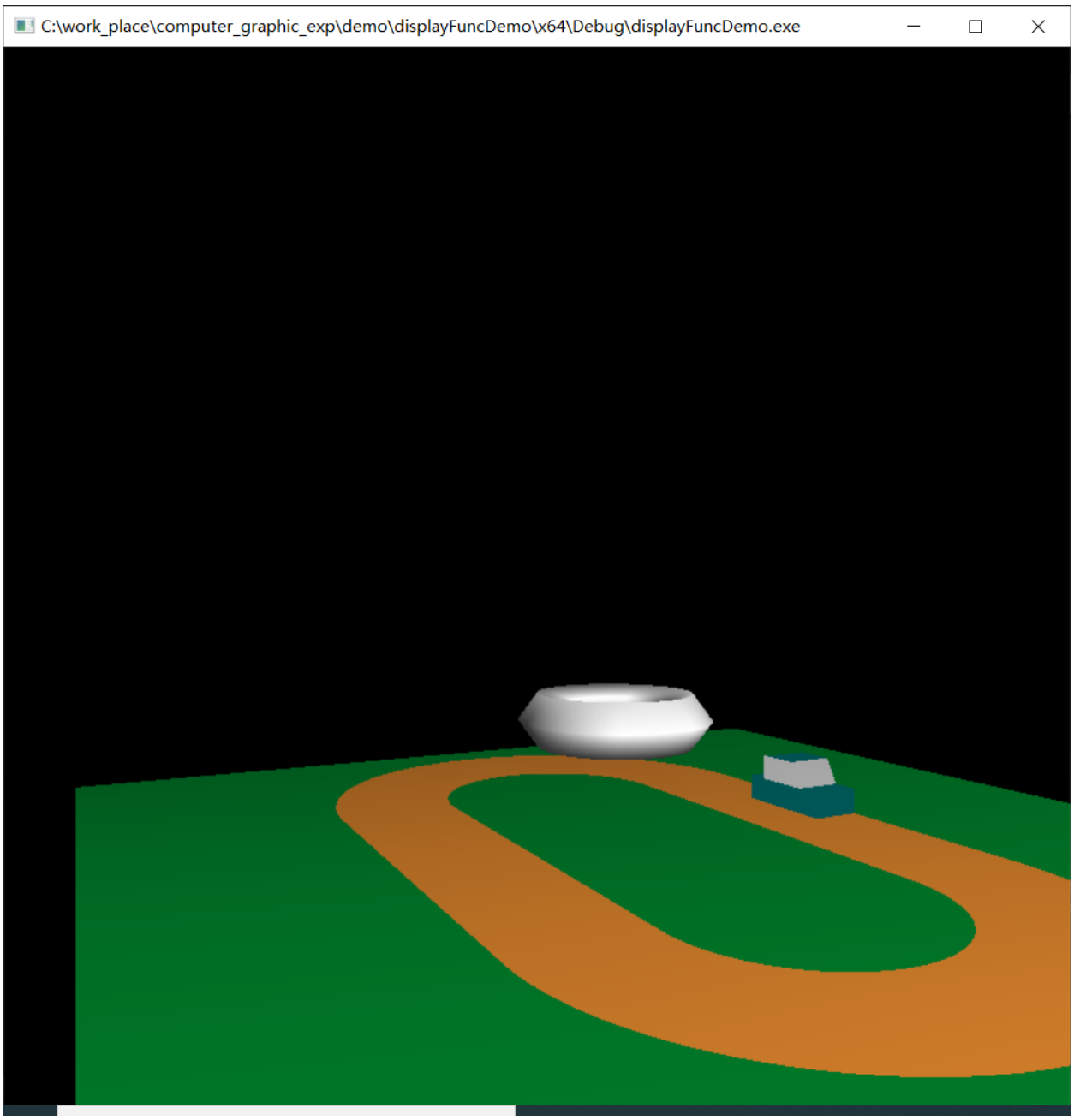
```
void glSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings)
```

参数：

1. innerR:内半径
2. outerR:外半径
3. sides:环面细分度
4. rings:圆环细分度

后面两个基本不用管，最主要就是前面的内外半径。

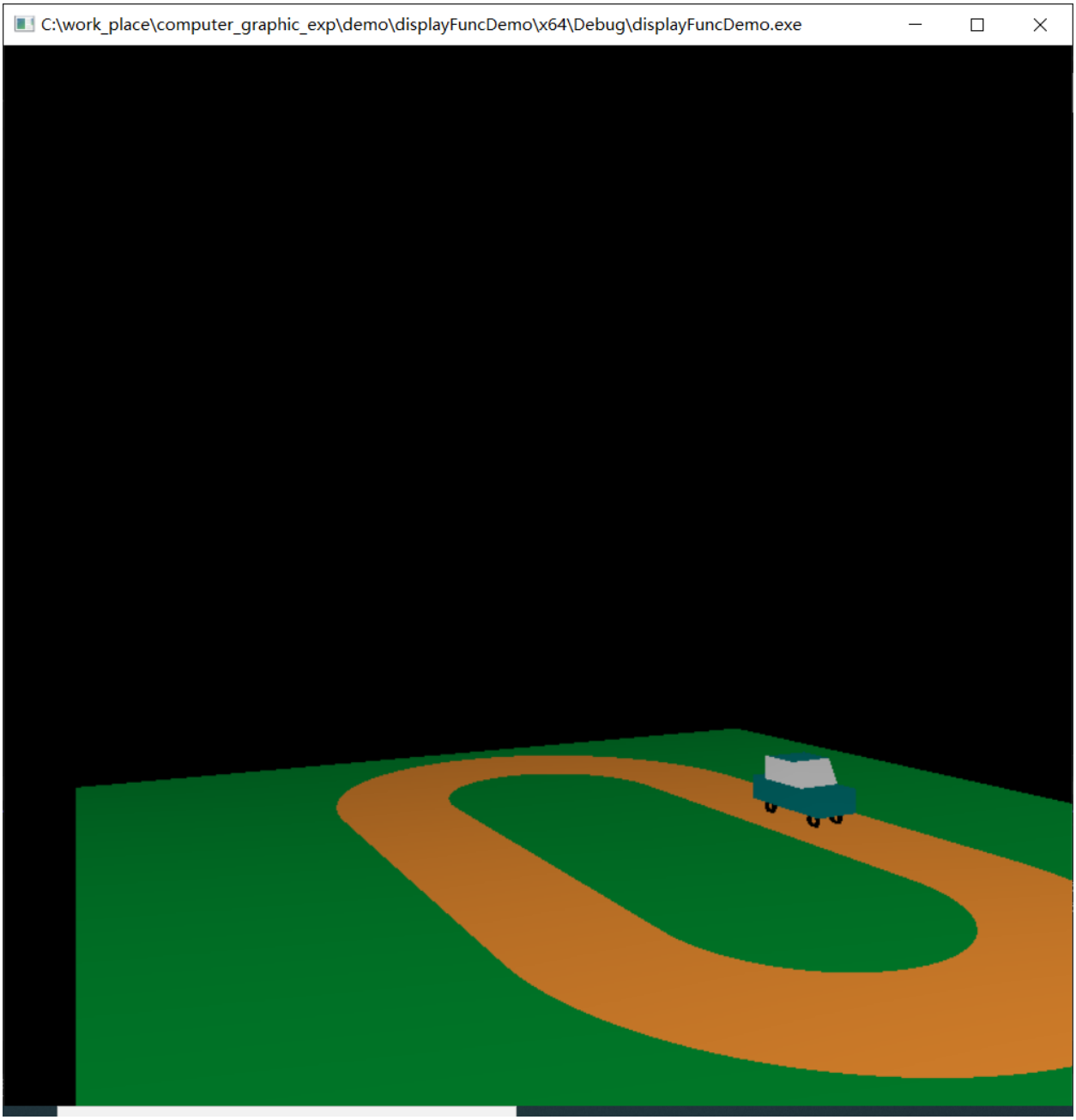
所以轮子就用这个画了。先不管位置，看看画出来长啥样。



长这样。所以我们要做的就是把他平移到车底下，转到竖着的位置，然后大小改一改，颜色改一改，就可以了。

```
glColor3f(0, 0, 0);
 //为了让报告篇幅不那么长，这里就展示一个轮子，但实际上这段代码重复了四次
glTranslated(0.6f, 1.3f, 0.25);
glRotatef(90, 0, 1, 0);
glutSolidTorus(0.1, 0.25, 5, 100);
```

效果图：



这样小车就画完了。

3.1.2. 轨道建模

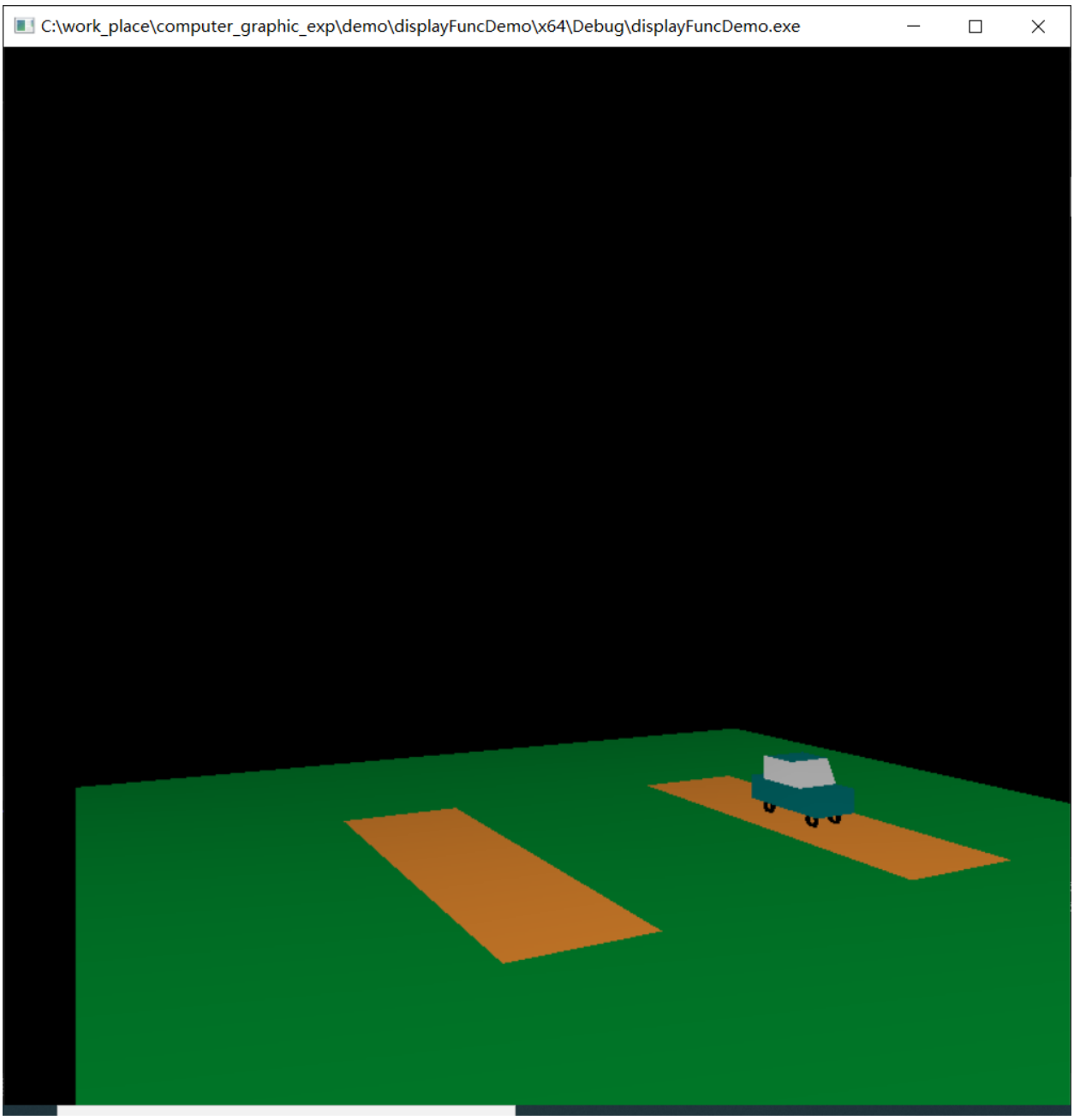
这个比车好画一点，基本上就是一个平面上再扣一个操场跑道出来。

绘制平面时，指定法向量为(0,0,1)，也就是这个平面是和地面平行的（其实就是想画个地面出来）：

```
//绘制面片
void draw_Facets(
  float RED,float GREEN,float BLUE,
  GLenum strategy,
  float cors[][3], int length
  )
{
  glColor3f(RED,GREEN,BLUE);
  glBegin(strategy);
  for (int i = 0; i < length; i++) {
    glVertex3f(cors[i][0], cors[i][1], cors[i][2]);
  }
  glEnd();
}

void draw_track() {
  float height = 0.001f;
  //绘制地面
  glNormal3f(0, 0, 1);
  float corsGround[4][3] = {
    {-20.0f, -20.0f, 0},
    {20.0f, -20.0f, 0},
    {20.0f, 20.0f, 0},
    {-20.0f, 20.0f, 0}
  };
  draw_Facets(0, 0.6, 0.2, GL_QUADS,
      corsGround , 4);
```

对于跑道，其实也可以拆解以下，很明显它有两个直道和两个弯道。先来看直道。

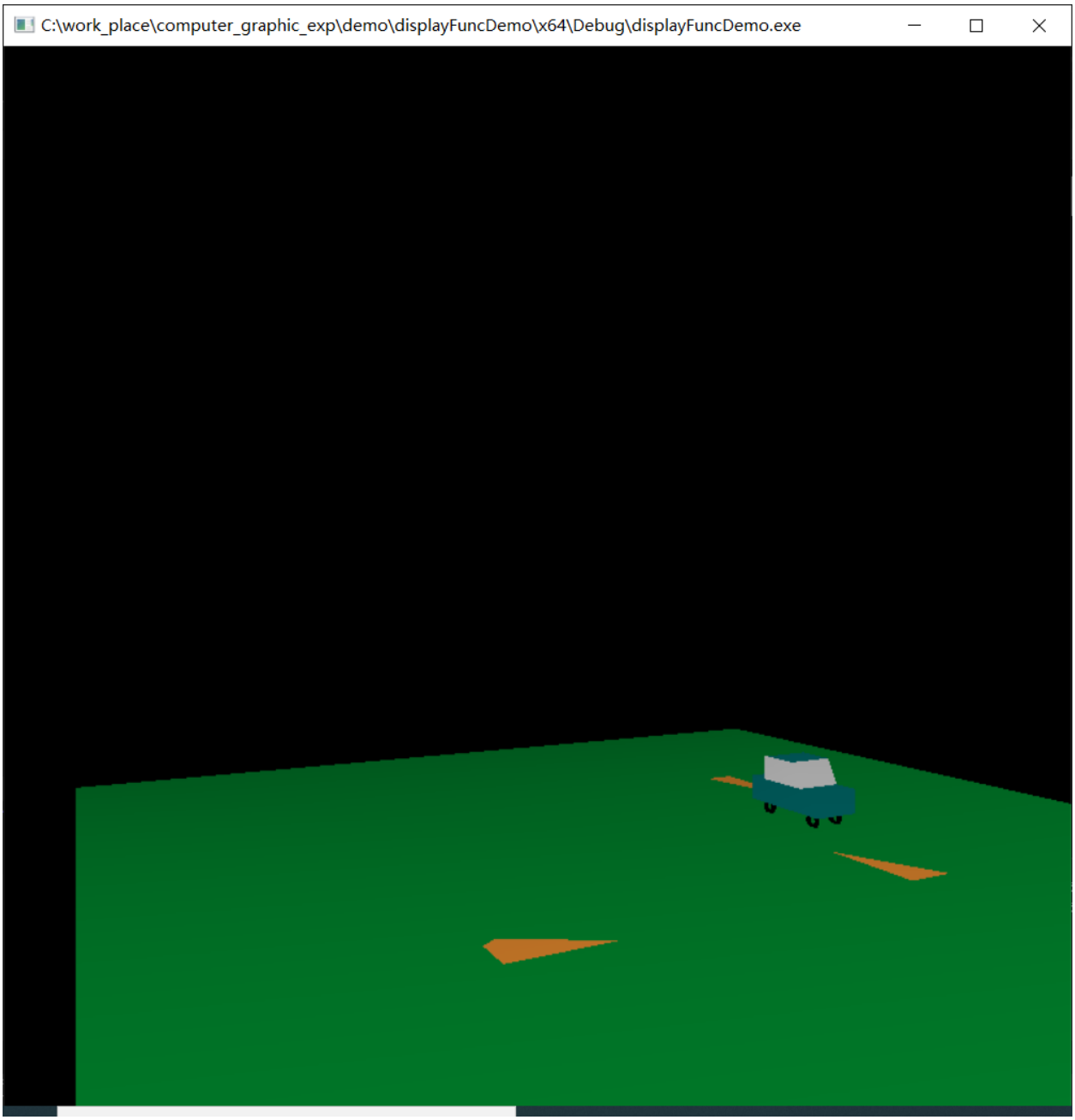
这个也很好画，指定一个矩形区域就行。




```
//跑道的直线部分
//float height = 0.001f;
float corsLeftRide[4][3] = {
  {-10.0f, -8.0f, height},
  {-5.0f, -8.0f, height},
  {-5.0f, 8.0f, height},
  {-10.0f, 8.0f, height}
};
```

```
draw_Facets(1, 0.6, 0.2, GL_QUADS,
  corsLeftRide, 4);
```

注意这里跑道要覆盖地面，也就是给跑道加上一个一点点的高度height，我设置的是0.001f。如果不加这个高度，就成这样了：

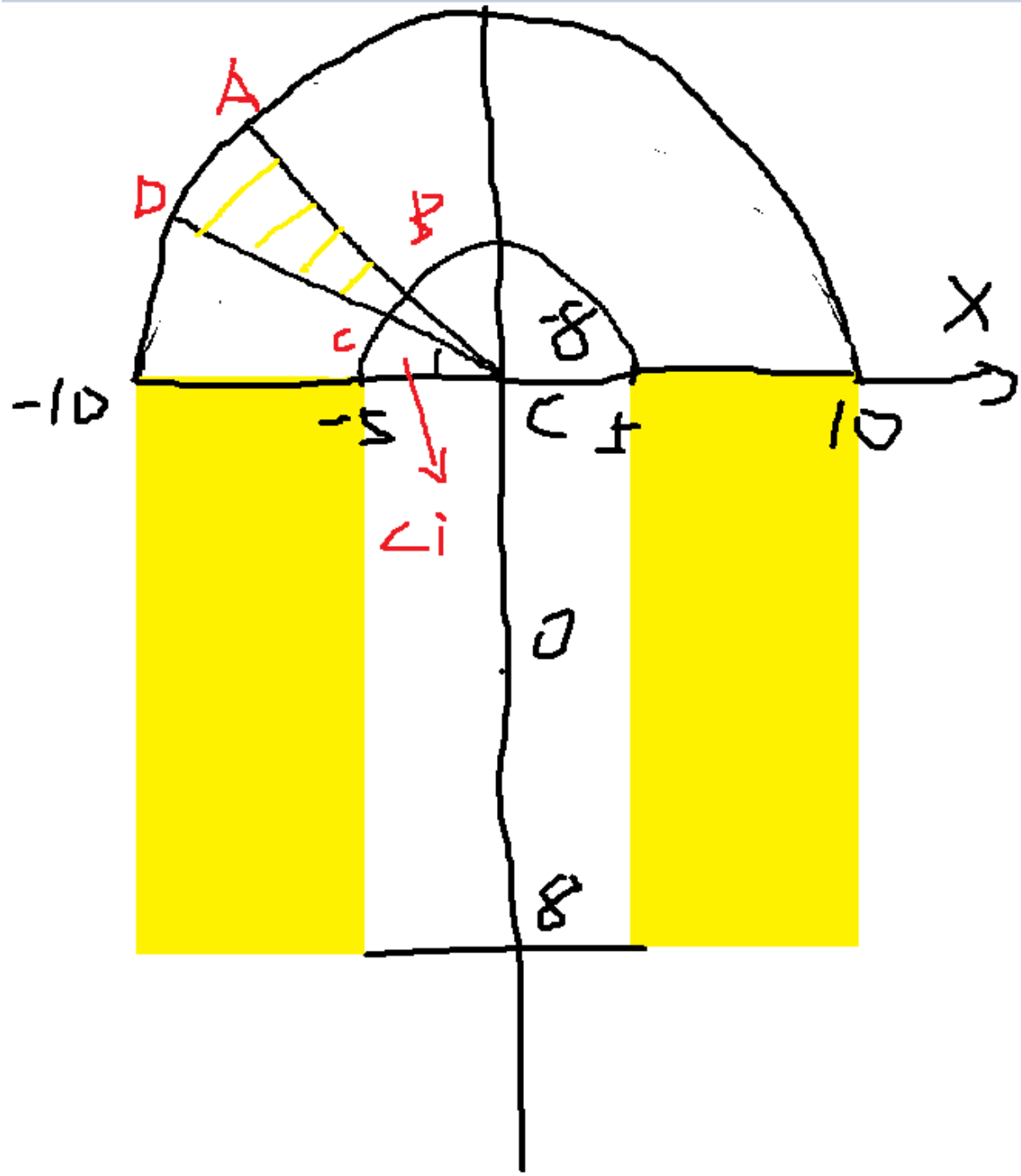


这个“一点点的高度”就是为了让跑道“盖住”地面的。

随后是圆环部分。不妨以上面那个圆环为例（由于遵循右手坐标系，y轴正方向是下面，所以上面的圆环y值是负的）。里面的小半园和外面的大半圆之间的部分是我们要上色的。

我们不妨把半圆划分成一个个扇形区域（微分思想），然后去渲染这一个个的扇形区域，最终就形成了一个（半）圆。如果我们想渲染半圆环，那就要去掉里面的小曲边三角形BCc（c为圆心）。

对于其中的面片四(曲)边形ABCD（要渲染的部分）来说，我们只需要考虑x,y坐标即可，因为z坐标是固定的（跑道与地面平行）。设当前CD与x轴夹角为i，角AcD=δ，那么ABCD四个点坐标为：



点	x坐标	y坐标
A	-10*cos(i+δ)	-8-10*sin(i+δ)
B	-5*cos(i+δ)	-8-5*sin(i+δ)
C	-5*cos(i)	-8-5*sin(i)

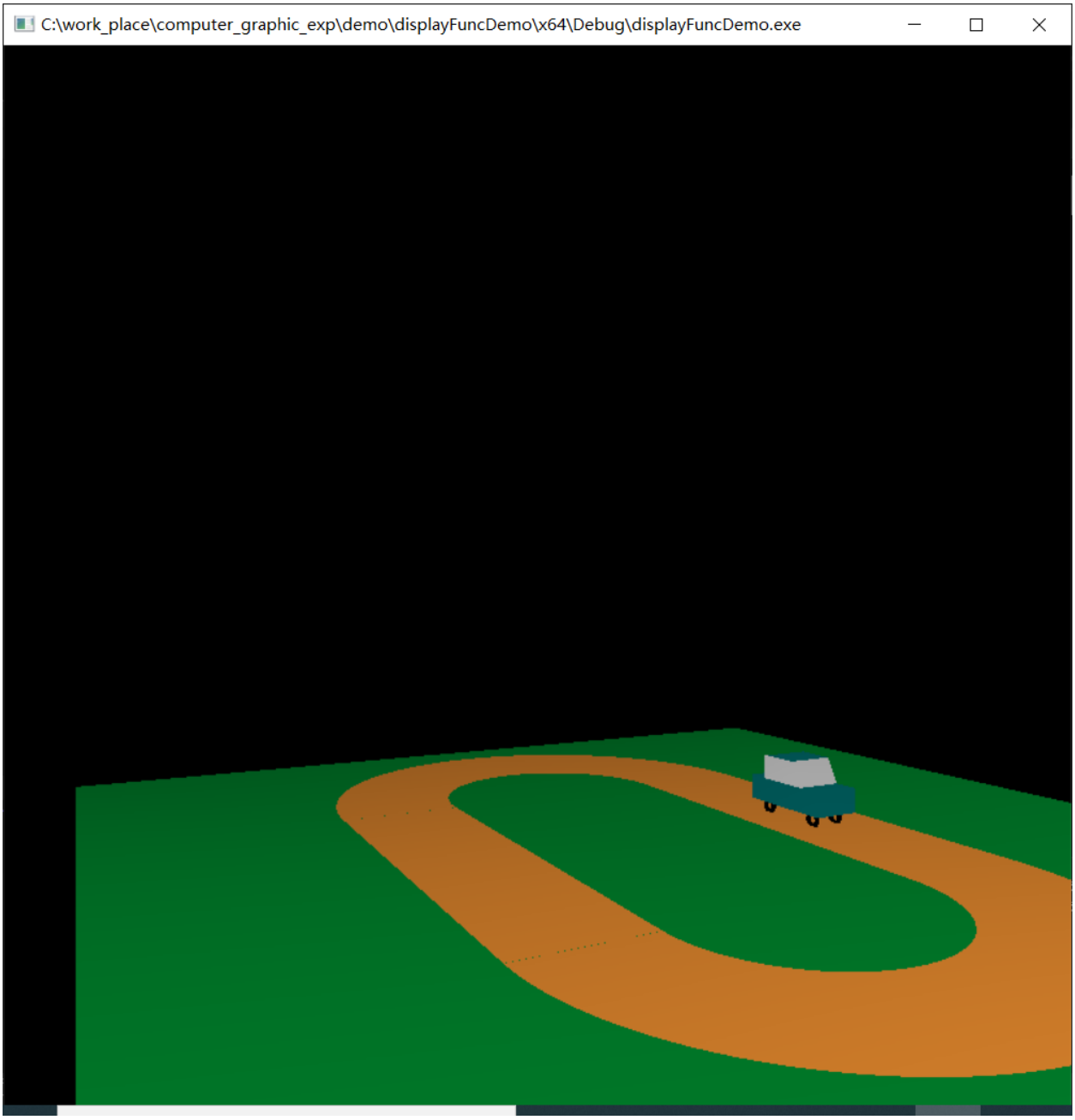
D	-10*cos(i)	-8-10*sin(i)

对于另一半圆环，推导过程相似，这里就不赘述了。

既然有了公式，直接把公式套进去就ok了。这里我δ取得0.01（弧度）

```
//圆环部分
//微分圆心角
for (i = 0; i <= PI - 0.01; i += 0.01) {
    float corsLowerRing[4][3] = {
      {-5.0 * cos(i), -8 - 5.0 * sin(i), height},
      {-10.0 * cos(i), -8 - 10.0 * sin(i), height},
      {-10.0 * cos(i + 0.01), -8 - 10.0 * sin(i + 0.01), height},
      {-5.0 * cos(i + 0.01), -8 - 5.0 * sin(i + 0.01), height}
    };
    draw_Facets(1, 0.6, 0.2, GL_QUADS,
      corsLowerRing, 4);
  }
```

这样就能渲染出完整的跑到了，效果图如下：



3.2. 动画制作

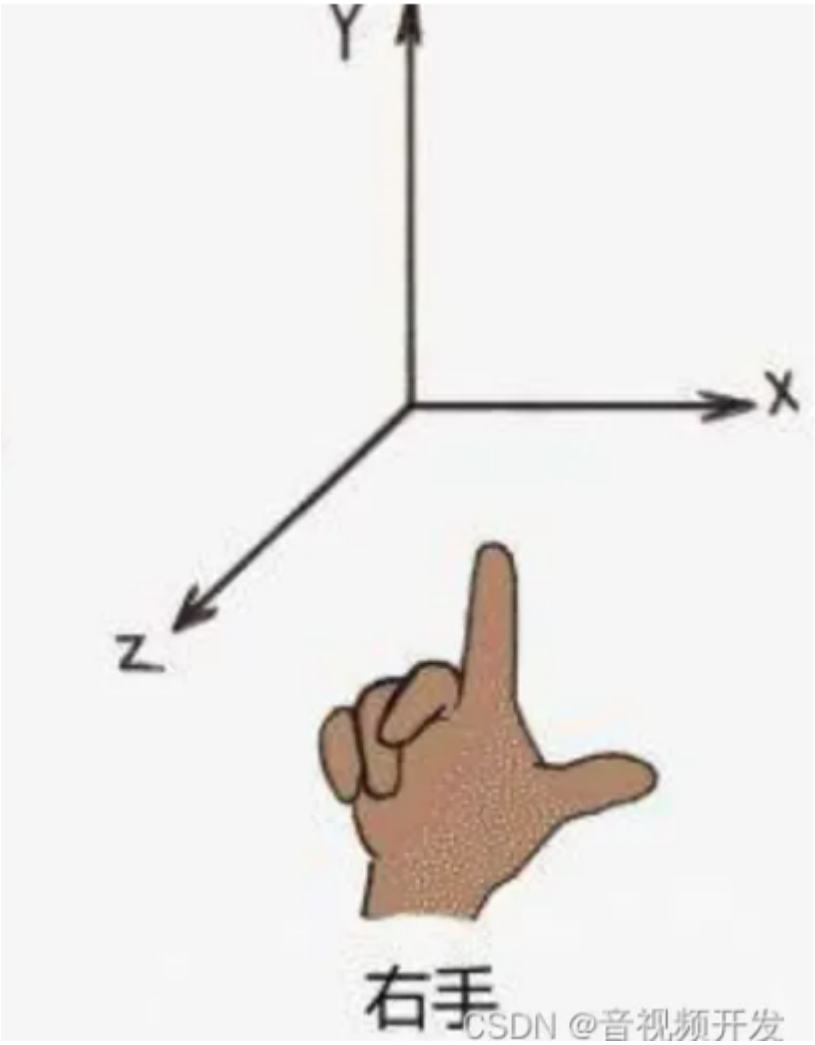
3.2.1 实体控制

如果每一帧都画出车（因为跑道是不动的，所以不用管它）的相对位置。那么在短时间内渲染多帧就可以实现“动画”的效果。

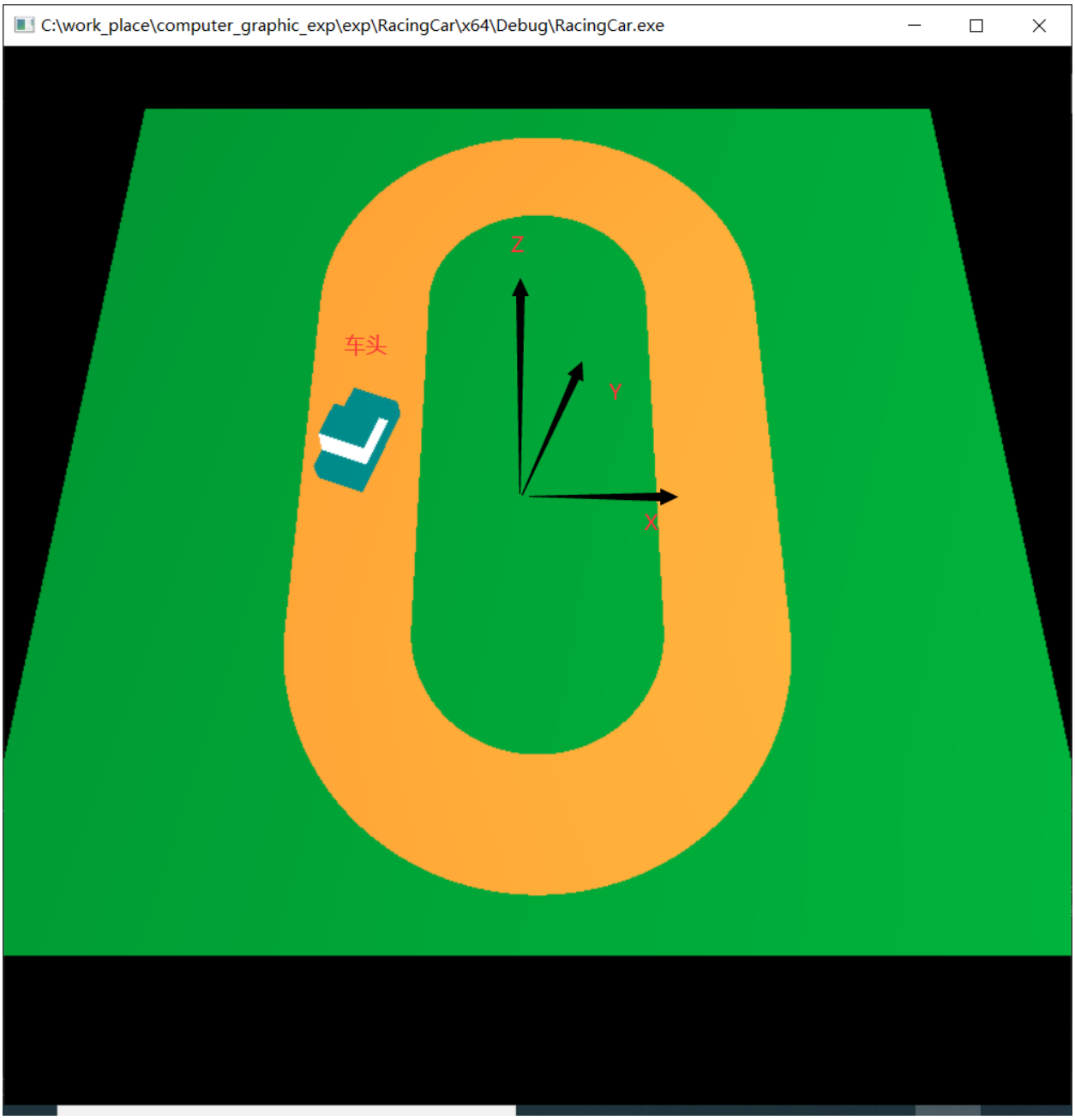
而这个相对位置是指当前车的位置相对于坐标原点的位置。所以我们需要维护一个前缀和，在每一帧都根据速度与方向角更新这个值。这个值包含两个量，一个x分量，一个y

分量（幸好这次实验没有要求上坡，不然还得计算z分量的值）。随后每一帧尝试访问这个前缀和，就能知道车的坐标在哪了。

但一涉及到方向角，就涉及到OpenGL的坐标系了。经过实验，得知OpenGL中的坐标系是右手坐标系，把右手大拇指、食指、中指伸出来，中指面对自己，就是z轴方向，食指指向y轴方向，大拇指指向x轴方向。如下图。



由于之前画车的时候车头是朝向y轴方向，因此在我们的视角下坐标系为：



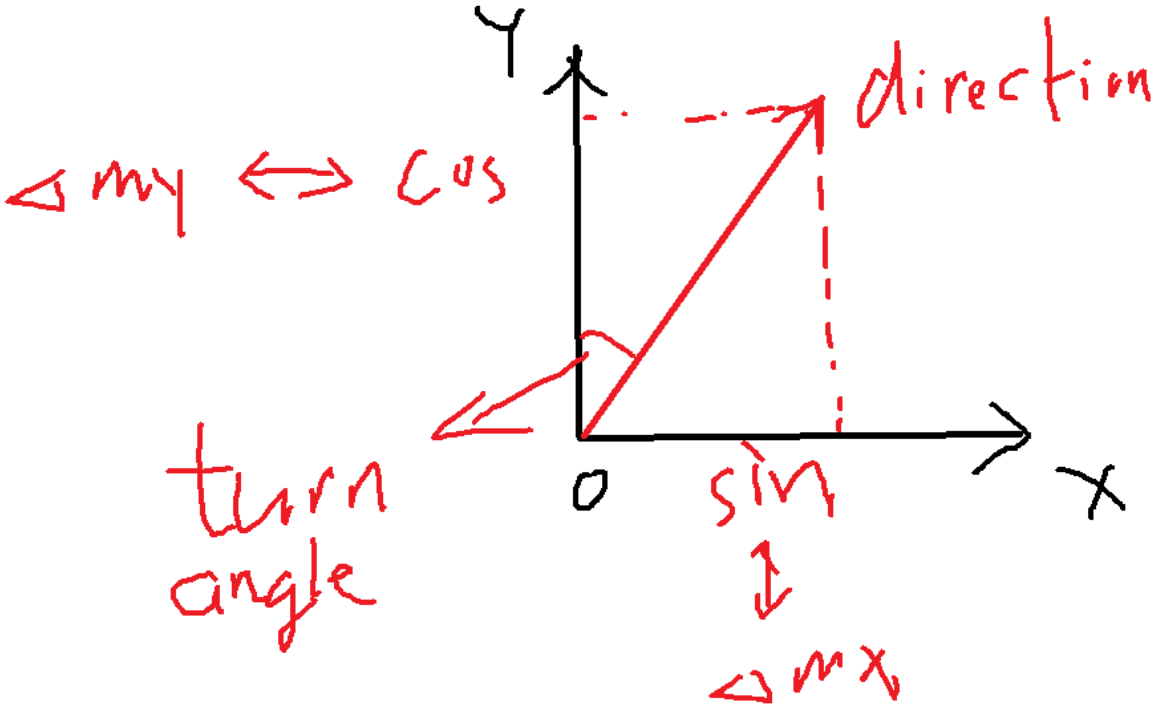
随后我们规定：

1. 方向角小于0，顺时针转
2. 方向角大于0，逆时针转

而OpenGL中提供的旋转矩阵glRotate3f是轴分量为1逆时针，-1顺时针。因此逆时针下一个负的角度其实就是顺时针转一个正的角度，正好符合我们的方向角小于0顺时针转。

现在我们来考虑一个特殊情况，靠这个特殊情况来推导公式，并泛化到一般情况。这个特殊情况就是小车一开始向右偏航。其实就是上面那张图的情况。

这个时候车的x,y值都应该增大，而此时的偏航角是小于0的（因为车顺时针转了），因此我们要保证公式中加上这个偏航角导出的位移分量时x和y确实是增大的。



假设车当前位置的坐标为mx,my。那么公式如下：

```
Δmy = v * cos(-turnAngle) = v * cos(turnAngle)
Δmx = v * sin(-turnAngle) = -v * sin(turnAngle)
```

可以验证一下这个公式，turnAngle<0，那么Δmy和Δmx都为正的。正确。

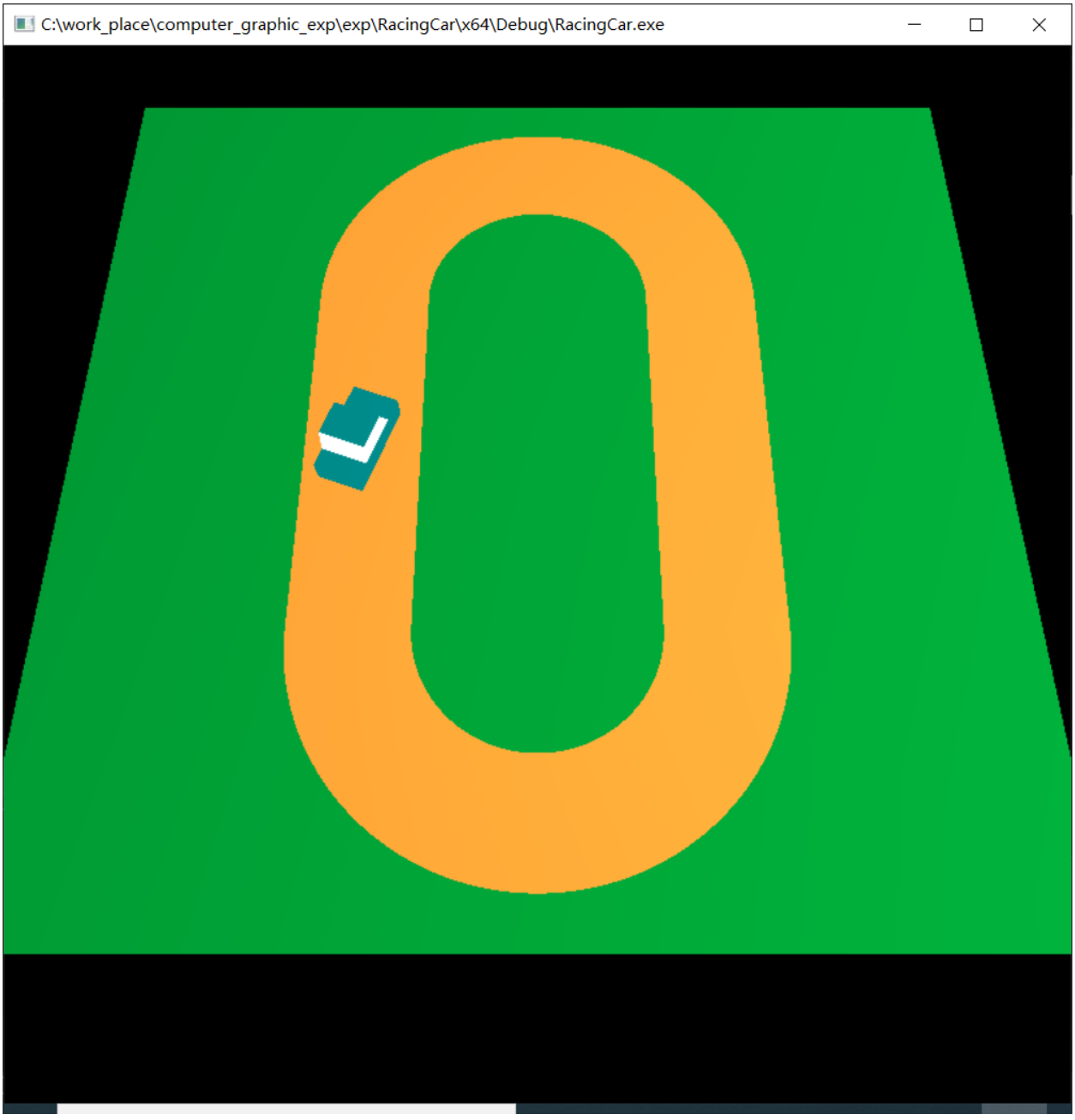
随后每帧渲染的时候把车挪到现在计算出来的(mx,my)坐标处即可。x轴向左边挪7.5是为了让车挪到跑道上。

```
glTranslatef(mx - 7.5, my, 0); //初始位置:(-7.5,0,0) 随mx,my位置变化
```

现在车的位置有了，但是画出来的时候要让用户看出来车头的朝向，因此我们还得把车转一个方向角。

```
glRotatef(turnAngle, 0, 0, 1);
```

这样画出来的车就对了。这里看的视角和之前放的图不一样了，位于(0,-20,34)左右。（我觉得这样目光朝向就是车头朝向，看起来更符合上述分析）



3.2.2. 视点切换

需求中要求我们切换视角，可以切换第一人称视角和第三人称固定视角。

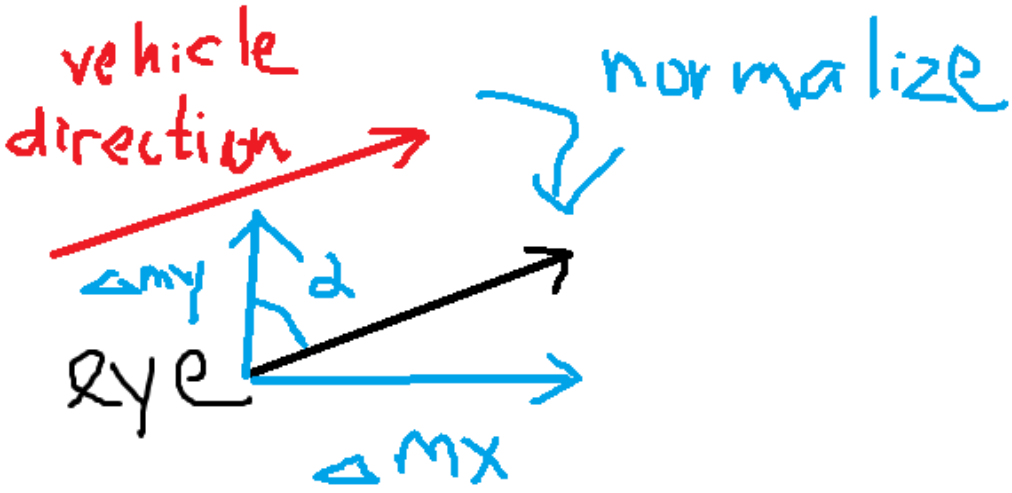
固定视角就比较简单，固定眼睛位置和看向的位置就可以了。

```
else {
    gluLookAt(x, y, z,
      0.0,0.0,0.0,
      up[0], up[1], up[2]);
  }
```

注意gluLookAt要给一个头顶方向（相机法向量），这里就给(0,0,1)了。

比较难的是第一人称视角怎么办。第一人称视角眼睛位置肯定在车里了。上一步我们正好把车的位置算出来了，这里直接用就行了。

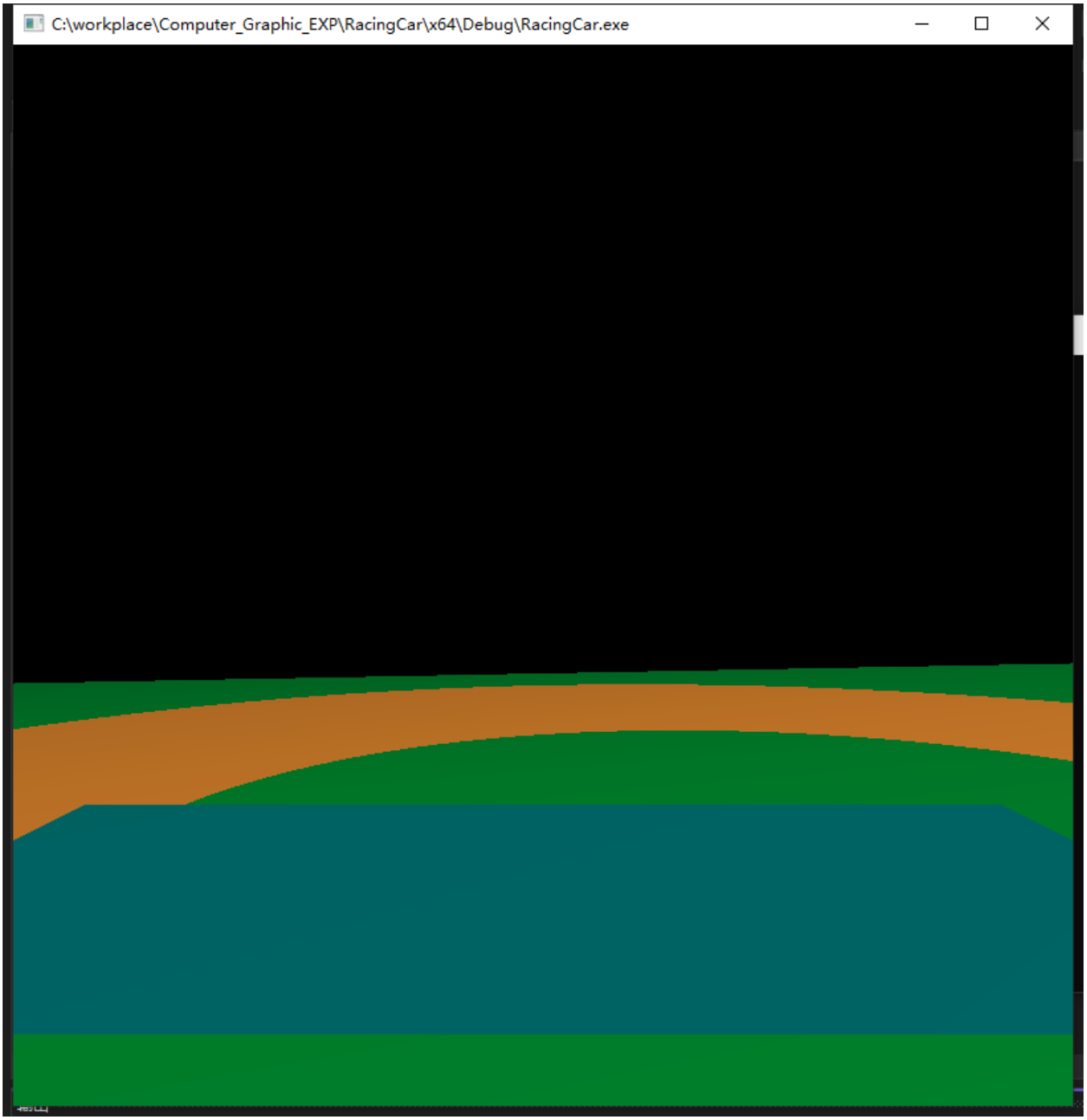
现在还缺看向的位置，既然是基于我们的眼睛看的，那么就给我们的眼睛坐标加上一个偏移量，成为看向的位置就可以了。开车一般目视前方，所以我们加上的偏移量应该和车的方向角一致。



这里`我选择加上一个单位向量，也就是直接加上sin(方向角),cos(方向角)即可`。($\sin^2(\alpha)+\cos^2(\alpha)=1$)

```
if (first_perspective) {
    gluLookAt(mx - 7.5, my, 2.0,
      mx - 7.5 + sin(angle2Rad(-turnAngle)), my + cos(angle2Rad(-turnAngle)) , 2.0,
      up[0], up[1], up[2]);
  }
```

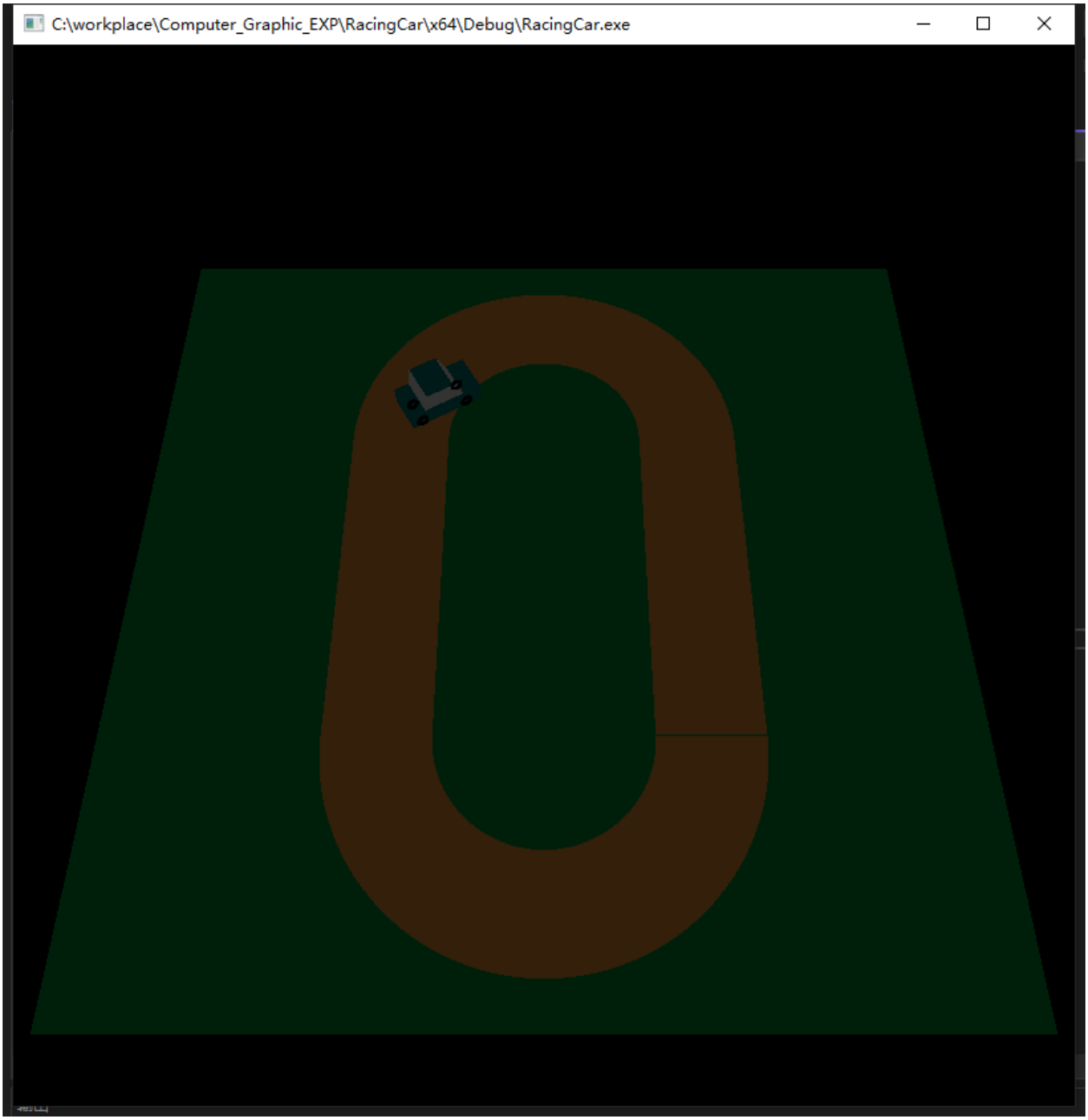
效果图：



3.3. 杂项

3.3.1. 简单光照模型

一开始我画出来的图长这样：



很灰暗。后来我上网查了一下，发现是因为没设置光源导致的。这一块和上课讲过的简单光照模型有关，设计三个部分：

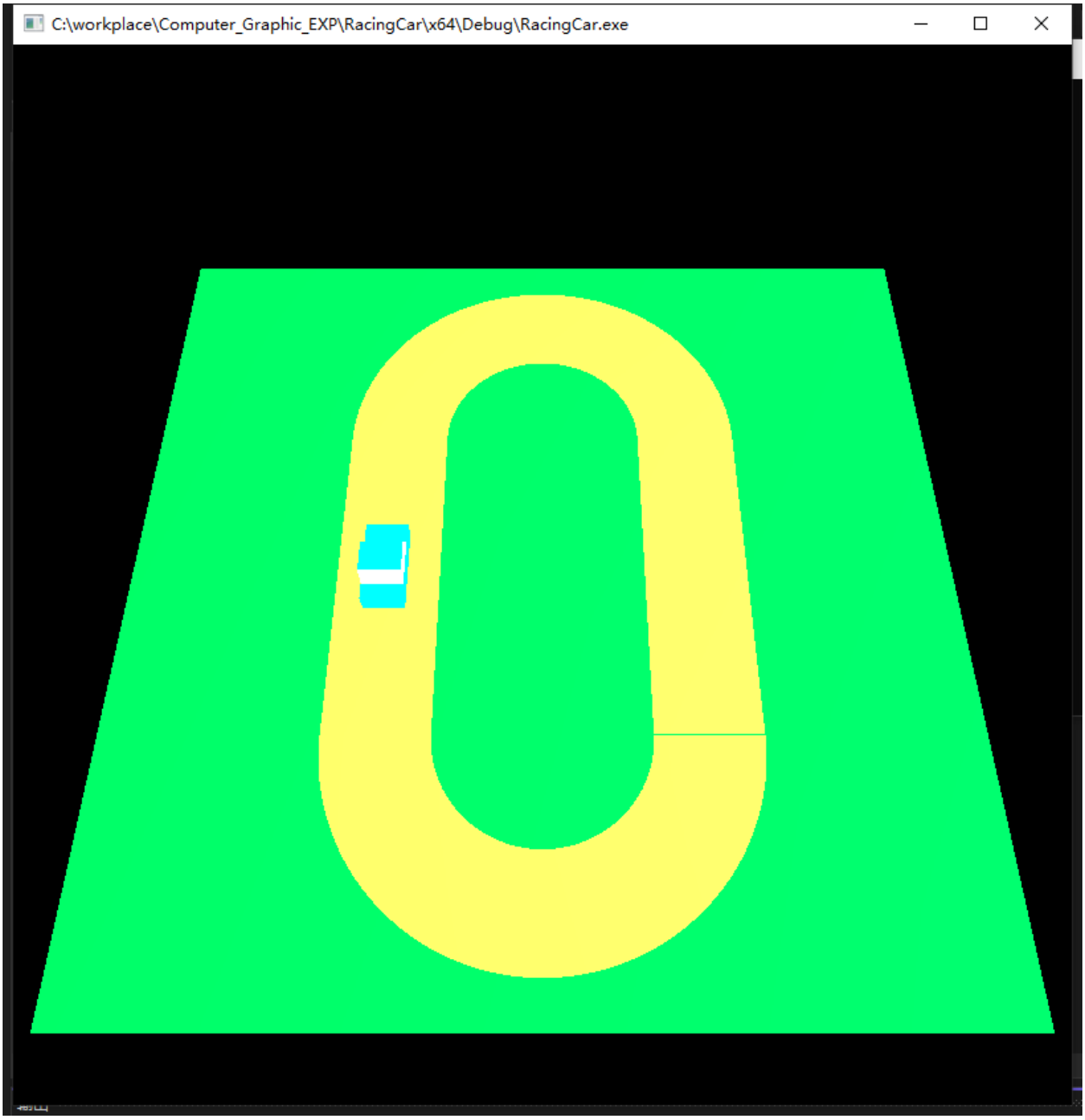
1. 环境光
2. 漫反射光
3. 镜面反射光

这三种光照OpenGL都给了对应的绘制函数。随后我就加上了这三个光照参数。

```
GLfloat  white[] = { 1.0, 1.0, 1.0, 1.0 };
glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
glLightfv(GL_LIGHT0, GL_AMBIENT, white);
glLightfv(GL_LIGHT0, GL_SPECULAR, white);
glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
glEnable(GL_DEPTH_TEST);
glEnable(GL_LIGHT0);
```

其中GL_POSITION那个是设置位置的，我设置到了(25,25,25)上。

然后就亮多了。(后来我感觉颜色太亮了就把环境光那个参数去掉了）



3.3.2. 视口调整

一开始我画出来的窗口全黑。后来发现是视口坐标以及投影矩阵没有设置。后来设置上就可以自适应窗口与用户视角渲染图像了。

```
void reshape(int width, int height) {
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(fovy, (GLdouble)width / height, nearPlane, farPlane);
```

```
  glMatrixMode(GL_MODELVIEW);
}
```

至此我们完成了Lab2。

完整代码如下：

```
#include <cmath>
#include <ctime>
#include <string>
#include <vector>
#include <fstream>
#include <sstream>
#include <iostream>
#include <GL/glew.h>
#include <GL/glut.h>
#include <GLFW\glfw3.h>
using namespace std;

const double PI = 3.14159265359;
const float velocity_acceleration = 0.005f;
const float v_max = 0.05f;// max velocity
const float rad = PI / 180.0f;

float angle2Rad(float angle) {
  return rad * angle;
}

bool first_perspective = false; // user perspective
float translatex = 0;
float translatey = 0;
float cameraDistance = 40.0f;
float cameraAnglex = 0;
float cameraAngley = 120;


float v = 0; //velocity
float mx = 0;
float my = 0;
float turnAngle = 0;

float R = 0, G = 0.5, B = 0.5;

GLfloat lightPos[] = { 25.0, 25.0, 25.0, 1.0 };
GLfloat lookat[] = { 0.0, 0.0, 0.0 };
GLfloat up[] = { 0.0, 0.0, 1.0 };
GLdouble fovy = 60.0;
```

```
GLdouble nearPlane = 1.0;
GLdouble farPlane = 1000.0;

void keyboard(unsigned char key, int x, int y)
{
  switch (key) {
  case 't':
    first_perspective = !first_perspective;
    break;
  case ' ':
    v = 0;
    break;
  case 'w':
    v = v + velocity_acceleration;
    if (v > v_max) v = v_max;
    break;
  case 's':
    v = v - velocity_acceleration;
    if (v < -v_max / 2.0) v = -v_max / 2.0;
    break;
  case 'a':
    turnAngle += 10.0;
    break;
  case 'd':
    turnAngle -= 10.0;
    break;
  case 27:
    exit(0);
    break;
  default: {
    if (v > v_max) {
      v = v_max;
    }
    if (v < -v_max / 5.0) {
      v = -v_max / 5.0;
    }
  }
       break;
  }
  glutPostRedisplay();
}

void draw_Facets(
  float RED,float GREEN,float BLUE,
  GLenum strategy,
  float cors[][3], int length
  )
{
  glColor3f(RED,GREEN,BLUE);
  glBegin(strategy);
  for (int i = 0; i < length; i++) {
    glVertex3f(cors[i][0], cors[i][1], cors[i][2]);
```

```
  }
  glEnd();
}

void draw_track() {
  float height = 0.001f;
  //绘制地面
  glNormal3f(0, 0, 1);
  float corsGround[4][3] = {
    {-20.0f, -20.0f, 0},
    {20.0f, -20.0f, 0},
    {20.0f, 20.0f, 0},
    {-20.0f, 20.0f, 0}
  };
  draw_Facets(0, 0.6, 0.2, GL_QUADS,
      corsGround , 4);

  //跑道的直线部分：
  float corsLeftRide[4][3] = {
    {-10.0f, -8.0f, height},
    {-5.0f, -8.0f, height},
    {-5.0f, 8.0f, height},
    {-10.0f, 8.0f, height}
  };
  draw_Facets(1, 0.6, 0.2, GL_QUADS,
    corsLeftRide, 4);

  float corsRightRide[4][3] = {
    {10.0f, -8.0f, height},
    {10.0f, 8.0f, height},
    {5.0f, 8.0f, height},
    {5.0f, -8.0f, height}
  };
  draw_Facets(1, 0.6, 0.2, GL_QUADS,
    corsRightRide, 4);

  //跑道的圆环部分：
  float i;
  for (i = 0; i <= PI - 0.01; i += 0.01) {
    float corsUpperRing[4][3] = {
      {-5.0 * cos(i), 8 + 5.0 * sin(i), height},
      {-5.0 * cos(i + 0.01), 8 + 5.0 * sin(i + 0.01), height},
      {-10.0 * cos(i + 0.01), 8 + 10.0 * sin(i + 0.01), height},
      {-10.0 * cos(i), 8 + 10.0 * sin(i), height}
    };
    draw_Facets(1, 0.6, 0.2, GL_QUADS,
      corsUpperRing,4);
  }

  for (i = 0; i <= PI - 0.01; i += 0.01) {
    float corsLowerRing[4][3] = {
      {-5.0 * cos(i), -8 - 5.0 * sin(i), height},
```

```
      {-10.0 * cos(i), -8 - 10.0 * sin(i), height},
      {-10.0 * cos(i + 0.01), -8 - 10.0 * sin(i + 0.01), height},
      {-5.0 * cos(i + 0.01), -8 - 5.0 * sin(i + 0.01), height}
    };
    draw_Facets(1, 0.6, 0.2, GL_QUADS,
      corsLowerRing, 4);
  }
}

void draw_car() {
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  //车底面
  glBegin(GL_POLYGON);
  glColor3f(R, G, B);
  glVertex3f(-1.0f, 2.0f, 0.5f);
  glVertex3f(1.0f, 2.0f, 0.5f);
  glVertex3f(1.0f, -2.0f, 0.5f);
  glVertex3f(-1.0f, -2.0f, 0.5f);
  glEnd();
  //车头
  glBegin(GL_POLYGON);
  glColor3f(R, G, B);
  glVertex3f(-1.0f, 2.0f, 0.5f);
  glVertex3f(-1.0f, 2.0f, 1.5f);
  glVertex3f(1.0f, 2.0f, 1.5f);
  glVertex3f(1.0f, 2.0f, 0.5f);
  glEnd();
  //车尾
  glBegin(GL_POLYGON);
  glColor3f(R, G, B);
  glVertex3f(-1.0f, -2.0f, 1.5f);
  glVertex3f(-1.0f, -2.0f, 0.5f);
  glVertex3f(1.0f, -2.0f, 0.5f);
  glVertex3f(1.0f, -2.0f, 1.5f);
  glEnd();
  //左侧面
  glBegin(GL_POLYGON);
  glColor3f(R, G, B);
  glVertex3f(-1.0f, 2.0f, 1.5f);
  glVertex3f(-1.0f, 2.0f, 0.5f);
  glVertex3f(-1.0f, -2.0f, 0.5f);
  glVertex3f(-1.0f, -2.0f, 1.5f);
  glEnd();
  //右侧面
  glBegin(GL_POLYGON);
  glColor3f(R, G, B);
  glVertex3f(1.0f, 2.0f, 0.5f);
  glVertex3f(1.0f, 2.0f, 1.5f);
  glVertex3f(1.0f, -2.0f, 1.5f);
  glVertex3f(1.0f, -2.0f, 0.5f);
  glEnd();
```

```
//车顶(从头向尾绘制，共有五个多边形拼接成)
glBegin(GL_POLYGON);
glColor3f(R, G, B);
glVertex3f(1.0f, 2.0f, 1.5f);
glVertex3f(-1.0f, 2.0f, 1.5f);
glVertex3f(-1.0f, 1.0f, 1.5f);
glVertex3f(1.0f, 1.0f, 1.5f);
glEnd();

glBegin(GL_POLYGON);
glColor3f(1, 1, 1);
glVertex3f(-1.0f, 1.0f, 1.5f);
glVertex3f(-1.0f, 0.5f, 2.5f);
glVertex3f(1.0f, 0.5f, 2.5f);
glVertex3f(1.0f, 1.0f, 1.5f);
glEnd();

glBegin(GL_POLYGON);
glColor3f(R, G, B);
glVertex3f(1.0f, 0.5f, 2.5f);
glVertex3f(-1.0f, 0.5f, 2.5f);
glVertex3f(-1.0f, -1.0f, 2.5f);
glVertex3f(1.0f, -1.0f, 2.5f);
glEnd();

glBegin(GL_POLYGON);
glColor3f(1, 1, 1);
glVertex3f(1.0f, -1.0f, 2.5f);
glVertex3f(-1.0f, -1.0f, 2.5f);
glVertex3f(-1.0f, -1.2f, 1.5f);
glVertex3f(1.0f, -1.2f, 1.5f);
glEnd();

glBegin(GL_POLYGON);
glColor3f(R, G, B);
glVertex3f(1.0f, -1.2f, 1.5f);
glVertex3f(-1.0f, -1.2f, 1.5f);
glVertex3f(-1.0f, -2.0f, 1.5f);
glVertex3f(1.0f, -2.0f, 1.5f);
glEnd();
//左车窗
glBegin(GL_POLYGON);
glColor3f(1, 1, 1);
glVertex3f(-1.0f, 0.5f, 2.5f);
glVertex3f(-1.0f, 1.0f, 1.5f);
glVertex3f(-1.0f, -1.2f, 1.5f);
glVertex3f(-1.0f, -1.2f, 2.5f);
glEnd();
//右车窗
glBegin(GL_POLYGON);
glColor3f(1, 1, 1);
glVertex3f(1.0f, 1.0f, 1.5f);
```

```
  glVertex3f(1.0f, 0.5f, 2.5f);
  glVertex3f(1.0f, -1.2f, 2.5f);
  glVertex3f(1.0f, -1.2f, 1.5f);
  glEnd();

  //车轮
  glColor3f(0, 0, 0);

  glTranslated(0.6f, 1.3f, 0.25);
  glRotatef(90, 0, 1, 0);
  glutSolidTorus(0.1, 0.25, 5, 100);

  glTranslated(-0.6f, 1.3f, 0.25);
  glRotatef(90, 0, 1, 0);
  glutSolidTorus(0.1, 0.25, 5, 100);

  glRotatef(-90, 0, 1, 0);
  glTranslated(0, -2.6f, 0);
  glRotatef(90, 0, 1, 0);
  glutSolidTorus(0.1, 0.25, 5, 100);

  glRotatef(-90, 0, 1, 0);
  glTranslated(1.2, 0, 0);
  glRotatef(90, 0, 1, 0);
  glutSolidTorus(0.1, 0.25, 5, 100);

  glPopMatrix();

}

void drawObjects(GLboolean shadowRender) {
  draw_track();
  glPushMatrix();
  glTranslatef(mx - 7.5, my, 0); //初始位置:(-7.5,0,0) 随mx,my位置变化
  glRotatef(turnAngle, 0, 0, 1);
  draw_car();
}


void display(void) {
  //计算物体的当前位移
  my += v * cos(angle2Rad(-turnAngle));
  mx += v * sin(angle2Rad(-turnAngle));
  //摄像机位置
  GLfloat z = 40.0f;
  GLfloat y = -20.0f;
  GLfloat x = 0.0f;
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glPushMatrix();
  glLoadIdentity();
  if (first_perspective) {
    gluLookAt(mx - 7.5, my, 2.0,
```

```
      mx - 7.5 + sin(angle2Rad(-turnAngle)), my + cos(angle2Rad(-turnAngle)) , 2.0,
      up[0], up[1], up[2]);
  }
  else {
    gluLookAt(x, y, z,
      0,0,0,
      up[0], up[1], up[2]);
  }
  drawObjects(GL_FALSE);
  glPopMatrix();
  glutSwapBuffers();
}


void init(void) {
  GLfloat  white[] = { 1.0, 1.0, 1.0, 1.0 };
  glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
  glLightfv(GL_LIGHT0, GL_SPECULAR, white);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
  glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_LIGHT0);
  glEnable(GL_LIGHTING);
  glEnable(GL_TEXTURE_2D);
  glEnable(GL_COLOR_MATERIAL);
  glEnable(GL_CULL_FACE);
}

void reshape(int width, int height) {
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(fovy, (GLdouble)width / height, nearPlane, farPlane);
  glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char** argv) {
  srand(time(0));
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);
  glutInitWindowSize(800, 800);
  glutInitWindowPosition(0, 0);
  glutCreateWindow(argv[0]);
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutIdleFunc(glutPostRedisplay);// do nothing
  glutMainLoop();
```

```
  return 0;
}
```