北京电子科技学院

《计算机图形学》实验报告

题　　目：天安门广场 3D 建模

班　　级：1721、1722

姓　　名：20172201　黄济森

20172206　乔善芃

20172233　杜闫琛

20172118　谢豪臣

完成日期：2019.12.14

一、问题描述

“我爱北京天安门，天安门上太阳升。”

我们到底有多爱北京天安门呢，爱到我们为他用 OpenGL 做了一个 3D 虚拟场景模型。下面是实验题目概况。

设计天安门广场的 3D 景物模型，建立形体的参数模型。采用图形的几何变换方法将所定义物体的投影图绘制出来，再使用光照和纹理等技术，增加其真实感效果。

景物模型中可以设计有多种元素，尽量使设计出的场景与真实模型匹配，以丰富实验效果。本实验中设计有如下模型：

建筑模型：天安门、人民英雄纪念碑、毛主席纪念堂、中国国家博物馆、人民大会堂、正阳门。

移动模型：太阳、月亮、小人游客和雪花。

装饰模型：花坛、树木和告示牌。

其他模型：马路和旗杆。

在具有真实感的场景中，除了应该设计模型以外，还应该对场景和模型加以处理，以增强它的真实感，赋予它生命。本实验中用到了如下技术完善场景效果：

光照模型，材质属性，纹理贴图，天空盒。

由于实现的场景比较大，仅在不变的窗口中是不能显示完整的场景的，因此为了使用户和编程者能够观察整个场景，需要给程序添加场景漫游功能——可以通过键盘或鼠标控制程序，任意地移动视角。

在本篇报告的概要设计和详细设计中会具体介绍每一模块的功能与实现，在测试分析中可以看到程序的运行结果，在实验小结中可以了解本程序的优点与不足。

二、需求分析

1. 程序功能：

(1) 显示

程序可以显示出构建的天安门广场的 3D 场景，包括模型的显示，模型的移动，贴图和光照的显示。

(2) 鼠标

通过按住鼠标左键并在屏幕上拖动，用户在场景中的视线方向（视角范围）可以随之移动，进行接近 360 度的任意调整。

通过按住鼠标右键并在屏幕上拖动，用户在场景中的视线方向和观察位置都会随之移动，进行前后左右的位置变化。即视角范围不变化，变化的是观察位置。

(3) 键盘

通过按下鼠标的’q’，’e’，’3’，’4’键，可以实现鼠标的第一个功能，进行视角左右上下的接近 360 度的调整。

通过按下键盘的’w’，’a’，’s’，’d’，’1’，’2’键，可以实现鼠标的第二个功能，且按下’1’，’2’键还可以上下移动观察位置。

通过按下键盘的’z’键，可以控制是否开启雪花功能。

通过按下键盘的’space’（空格）键，可以控制视角回到初始设定的位置。

2. 程序性能：

(1) 显示

程序可以显示出天安门广场的 3D 场景。

静态模型：显示出设计的固定位置的模型。建筑和花草模型，它们的显示和贴图应该有良好效果。

动态模型：动态模型能够在场景中移动。日月模型，雪花和小人游客模型，他们均可自动地在场景中移动。

光照：在光源移动，和光照强度变化的过程中，可以观察到模型表面有明显的亮度变化。

贴图：纹理贴图的效果应该有良好效果，天空背景的贴图切换流畅不会出现问题，打开雪花功能后也不会因为贴图数量过多致使程序出现卡顿现象。

(2) 提示信息

命令提示窗口中显示有键盘按键提示。每当有小人游客走出一定范围时，窗口会有提示。

(3) 键盘鼠标

键盘鼠标可以流畅舒适的控制观察位置和视角范围，和某些功能的开关。

三、概要设计

本实验所用语言为 C 语言外加 OpenGL 和 glut 工具（这点回头再写）。即代码大部分语法为 C 语言语法，文件后缀名为.cpp 格式，即用了一些 C++语言的小语法。在编写程序中，为了避免在一个文件中编写大量的代码，影响编程效率和调试效率。我们将各个重要模块分离开，形成多个.h 文件，然后在主文件中#include “xxx.h”，即可将其他模块导进带有主函数的文件中，使程序在运行时可以找到这些模块。

因此在概要设计的介绍中，我主要也是以划分的模块进行介绍的。概要设计主要大体的说明组成程序模块的功能和工作，简要说明每个模块所编写的函数的功能，不具体说明 OpenGL 函数用法和具体设计细节。

❖ 主模块 main.cpp

进行一些初始化设置，比如：OpenGL 窗口、光照模型、颜色跟踪、模型位置、读取贴图文件、透视投影空间，视角。

调用 OpenGL 回调函数，开启循环检测。

void reshape(int width, int height)

函数功能：窗口刷新函数。调整 OpenGL 窗口大小，设置 OpenGL 场景的显示模式，初始化观察位置和视角范围。

void idle()

函数功能：闲置函数。这个函数会被不断调用，直到有窗口事件发生。在函数里面对设置的表示时间的变量进行改变，以作为绘制动态模型时的角度参考。

void setMaterial()

函数功能：设置材质。设置当前材质的属性，材质对光线的吸收效果，设置的参数有：环境光、漫反射、镜面反射、高光和发射光。

void initial()
函数功能：初始化函数。生成随机种子，初始化雪花和小人的位置。初始化一个点光源，开启颜色跟踪，深度缓冲。

int main(int argc, char* argv[])
函数功能：主函数。各种回调函数的调用，读取贴图文件，开启 OpenGL 循环检测。

❖ 场景漫游 keyboard.h

通过鼠标和键盘，控制对 OpenGL 场景可观察的视角范围。有两种变换方式：1.相机和视线同时移动。2.仅视线移动。

另外的设置：相机移动的同时天空与雪花也会随之移动。复位键：回到程序运行时最初的视角位置。

void orient_LeftRight(float ang)
函数功能：旋转相机，绕 y 轴旋转。

void orient_UpDown(float ang)
函数功能：旋转相机，上下转动，绕 x 轴旋转。

void move_Front_Back(int direction)
函数功能：前后移动相机。

void move_Left_Right(int direction)
函数功能：左右移动相机。

void move_High_Low(int direction)
函数功能：上下移动相机。

void reInit()
函数功能：按空格键回到初始原位置，复位所有的变量。

void OnKeyboard(unsigned char key, int x, int y)
函数功能：键盘响应函数。

void mouse(int button, int state, int x, int y)
函数功能：检测鼠标按下函数。

void motion(int x, int y)
函数功能：检测鼠标移动函数。

❖ 小人模型

用结构体定义的一个小人游客模型，通过结构体定义一个数组，程序可以控制多个这样的小人在场景中移动。小人会随着时间的变化或多或少的出现，比如早上和晚上的人少，下午的时候人开始多起来。

结构体：

```
const int peopleNum = 50;
typedef struct people{
    int peopleTime;     //控制小人移动的时间
    float peopleAngle;
    float peopleX;
    float peopleY;
    float peopleZ;
}p;
p people[peopleNum];
```

void generatePeople()

函数功能：小人绘制函数。通过 OpenGL 语句画小人模型。

void initPeople()

函数功能：初始化小人函数。初始化小人结构体，生成一个随机的初始位置。

void peopleMove(int i)

函数功能：小人移动函数。将小人移动到指定位置，调用小人绘制函数，可以在当前位置生成一个小人模型。小人移动时，实时更新每个小人的信息。

void drawPeople()

函数功能：按照时间变量的变化，调用小人移动函数绘制指定数量的小人。

❖ 粒子系统：雪花效果

用结构体定义一个雪花粒子，通过结构体定义一个数组，程序可以控制大量的雪花在场景中飘散，模拟下雪的效果。原理是将雪花图片贴在正方形上并设置半透明效果，在场景上方的同一平面内随机位置生成雪花，然后让他们随机向下移动。检测到雪花落到地上，或是生存时间到了（避免速度太小，落不下来）会消失和自动复位。

结构体

```
typedef struct Particle
{
    float x,y,z;                //粒子的位置
    unsigned int    r,g,b;      //粒子的颜色
    float vx,vy,vz;             //粒子的速度(x,y,z 方向)
    float ax,ay,az;             //粒子在 x，y，z 上的加速度
```

```
    float lifetime;          //粒子生命值
    float size;              //粒子尺寸
    float dec;               //粒子消失的速度
};
const int snowNum = 8000;
s snow[snowNum];
```

void setSnow(int i)
函数功能：调用此函数可以将当前的全局变量的值赋给某个粒子作为它的属性。

int getSnow(……)
函数功能：调用此函数可以获取某个粒子的所有属性。

void initSnow()
函数功能：初始化雪花粒子，利用随机数产生函数给雪花赋初值。

void drawParticle()
函数功能：绘制粒子。在当前位置绘制一个正方形，然后将雪花贴图映射上去。

void updateSnow(int i)
函数功能：更新粒子。粒子向下做加速运动，每次运动到新的位置，调用绘制函数在当前位置绘制一片雪花，然后更新雪花的属性。

❖ 天空贴图

给场景加上天空贴图，营造一个被天空包围的场景。原理是先绘制一个球体，再将贴图贴到球面上。每当相机移动时，同时移动天空球体的位置，可以产生一个天空位置永远不会移动，永远在最远处的一个效果。

void createSkyBox()
函数功能：进行天空的创建和贴图。

❖ 场景绘制

我们实验的模型没有借助其他的 3D 建模软件，而是使用 OpenGL 绘制出来的，OpenGL 可以绘制立方体、圆、空心圆柱、三角形和球体，我们的模型都是由这些基础图形组合而成。根据实际的天安门广场大小，我们按比例缩放设计自己的广场模型，计算好坐标后，在指定的位置绘制图形，完成每个模型的建立。再利用 glTranslatef()函数进行模型变换，将建筑移动到指定位置。还可以加入纹理贴图使模型显得更加真实。

◆ 基础图形构建 Construct.h

基础图形构建：立方体、三棱柱、四棱台、圆柱。建筑的元素大都是由这些基础图形拼接起来的，且每个建筑在绘制的时候坐标是不一样的。因此将这些基础图形的绘制写成一个参数，可以进行多次复用。

void constract(double x, double y, double z, double x1, double y1, double z1)
函数功能：根据 6 个参数写出立方体的 8 个坐标。

void build()
函数功能：根据上面函数写出的 8 个坐标，分别绘制立方体的 6 个面。

void consTriPrism(double x, double y, double z,
double x1, double y1, double z1, double length)
函数功能：根据函数给出的参数，得到三棱柱的 6 个点的坐标。

void buileTriPrism()
函数功能：根据上面函数写出的 6 个坐标，绘制两个三角形和三个长方形。

void consFourPrism(double x1, double y1, double z1, double x2, double y2, double z2,
double x3, double y3, double z3, double x4, double y4, double z4)
函数功能：根据函数给出的参数，得到四棱台的 8 个坐标。

void buildFourPrism()
函数功能：根据上面函数写出的 8 个坐标，绘制棱台的上下两面和侧面 6 面。

GLvoid DrawCircleArea(float cx, float cy, float cz, float r, int num_segments)
函数功能：画圆。

void mySolidCylinder(GLUquadric* quad,GLdouble base,
GLdouble top, GLdouble height,GLint slices,GLint stacks)
函数功能：通过 OpenGL 自带的函数画空心圆柱，再调用上面的函数补全圆柱的两个圆面。

◆ 建筑模型 tiananmen.h

天安门、人民英雄纪念碑、毛主席纪念堂、中国国家博物馆、人民大会堂和正阳门的模型建立。

void drawPillar(float zhux, float zhuy, float zhuz,
float radius, float height, GLUquadricObj * objCylinder)
函数功能：画柱子。参数为圆柱的位置坐标，半径、高度和圆柱对象。

void drawWindow(float wx, float wy, float wz)
函数功能：画窗户。参数为每一扇窗户的左下角坐标。

void drawSmallFlag(float flx, float fly, float flz)
函数功能：画小红旗。参数为旗杆的圆心位置。

void drawTianAnMen()

函数功能：画天安门。调用多个自定义的函数完成天安门的绘制。

void drawHero()
函数功能：简单绘制人民英雄纪念碑。

void drawRemember()
函数功能：简单绘制毛主席纪念堂。

void drawHall()
函数功能：简单绘制人民大会堂。

void drawMuseum()
函数功能：简单绘制中国国家博物馆。

void drawZhengYangMen()
函数功能：简单绘制正阳门。

- 日月模型 draw.h
  太阳和月亮的模型建立。

  void drawSun()
  函数功能：动态绘制太阳和月亮。让他们昼夜交替显示。

- 装饰模型 draw.h
  花坛、树木和告示牌模型的建立。

  void generateParterre(float x, float z)
  函数功能：绘制一个花坛。由坛沿和草地组成。

  void drawParterre()
  函数功能：绘制多个花坛。即调用多次 generateParterre()函数。

  void generateTree()
  函数功能：绘制一颗树。由树干和树叶组成。

  void drawTree()
  函数功能：绘制多棵树。多次调用 generateTree()函数在指定位置绘制树木。

  void drawBillboard(float x, float z, GLuint tex)
  函数功能：绘制告示牌。

◆ 其他模型：马路和旗杆。

void drawFlag()

函数功能：绘制国旗杆。包括围杆，基座，草地和旗杆。

void drawRoad2(float rx, float ry, float rz)

函数功能：绘制两条小马路。

void drawRoad()

函数功能：绘制长安街。

四、详细设计

■ 光照系统

openGL 在处理光照时把光照系统分为三部分，分别是光源、材质和光照模型。下面会分别介绍我们程序在光照系统的设计。

(1) 光照模型

指定全局环境光：

glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambient_intensity);

OpenGL 中的光可以分解成 RGB 三个分量，环境光的颜色为：{ 0.5, 0.5, 0.5, 1.0 };

(2) 光源

太阳光应该设定成平行光源，因为太阳的位置离我们很远。但是在本程序中由于只是小范围的场景，太阳离我们场景并不远，所以将它设置成了点光源。

用以下两个函数设置一个点光源。光源中加入了漫反射光，它的效果是：物体的某一部分越是正对着光源，它就会越亮，因此漫反射在光源的效果中最明显。

glLightfv(GL_LIGHT2, GL_POSITION, light2_position);

glLightfv(GL_LIGHT2, GL_DIFFUSE, sunshine_mat);

漫反射光的颜色为：{255.0/255.0, 210.0/255.0, 166.0/255.0，1.0}。

(3) 材质

OpenGL 用材料对光的 RGB 的反射率来近似定义材料的颜色。材质颜色也分为环境、漫反射和镜面反射成分，颜色参数表示对该种色光的反射程度，1 表示完全反射，0 表示完全吸收。若 OpenGL 的光源颜色为（LR、LG、LB），材质颜色为（MR、MG、MB），那么，在忽略所有其他反射效果的情况下，最终到达眼睛的光的颜色为（LR*MR、LG*MG、LB*MB）。可以使用以下函数对材质进行设置。

glMaterialfv(GL_FRONT, GL_AMBIENT, ambient_intensity);

glMaterialfv(GL_FRONT, GL_DIFFUSE, sunshine_mat);

glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);

glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);

glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);

在本程序中虽然编写了设置材质的函数，但是并没有启用，原因是目前光照还没有设置好，因此材质也就还没有设置。所以对于本程序的场景效果而言，每一个模型的材质是一样的，因此他们在光照下的效果是一样的。

(4) 颜色跟踪

由于设置材质时，材质会被保存在状态里面（就像设置颜色一样），所以如果不改变材质的话，默认会用最近用过的材质。而一旦开启了材质，就无法关闭，对于某些不需要设置的地方就会影响它（模型）原本的颜色，比较麻烦。颜色跟踪可以解决这一问题：

glEnable(GL_COLOR_MATERIAL);

glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

启动颜色跟踪之后，我们就可以像以前一样，使用 glColor 函数来指定图元的颜色了。

(5) 光源移动

既然是由日月作为点光源发出光线，那日月在移动的同时，也应该设置光源的位置随之移动。光源动态移动之后，可以很清楚地看到场景表面光照效果的一个变化。

```
light2_position[0] = sunRadius * cos(timer);
light2_position[1] = sunRadius * sin(timer);
glLightfv(GL_LIGHT2, GL_POSITION, light2_position);
```

除此之外，在实际生活中，光强度（光照颜色）也是在变化的。比如早上要暗一些，中午达到最亮，下午的时候甚至可能会偏红。所以在程序中我们不仅动态移动了光源，还动态地改变了光线颜色（仅漫反射）。

```
if(timer <= pi){
    rr = sunshine_mat[0] - cos(timer);
    gg = sunshine_mat[1] - cos(timer);
    bb = sunshine_mat[2] - cos(timer);
    change_light[0] = rr;
    change_light[1] = gg;
    change_light[2] = bb;
}
glLightfv(GL_LIGHT2, GL_DIFFUSE, change_light);
```

■ 纹理

一般开启纹理，都是使用纹理贴图将图片映射到指定的图形上，以显示更好的模型效果，下面是本程序在进行纹理贴图时的步骤：

1.启用 2D 纹理：glEnable(GL_TEXTURE_2D);

2.加载纹理图像（即导入图片）：GLuint texName = load_texture();（自定义的函数）

3.创建纹理对象：glGenTextures(1, &texName);

4.绑定纹理对象：glBindTexture(GL_TEXTURE_2D, texName);

5.设定纹理过滤的参数：glTexParameteri();

6.为纹理对象指定纹理图像数据：glTexImage2D();

7.纹理映射，绘制纹理图像：glTexCoord2f();

因为纹理贴图时所用的代码和函数较多，特别是在设定纹理过滤时，需要设置很多参数，其实我们也没有把每个函数和参数都研究的特别明白，仅仅是了解了纹理贴图的过程并得到了看着还不错的效果，纹理还有更多有用和复杂的应用，等待着我们以后去学习和使用。

■ 场景漫游

在讲解场景漫游前，需要先介绍一下 OpenGL 的矩阵模型变换。用 glMatrixMode 函数来设置当前操作的矩阵的模式。

(1) 投影变换：glMatrixMode(GL_PROJECTION);

投影变换就是定义一个可视空间，可视空间以外的物体不会被绘制到屏幕上。即将 3D 变换成 2D 的一种变换模式。它有两种类型的投影变换，分别是透视投影和正投影。

正投影相当于在无限远处观察得到的结果，它只是一种理想状态，缺乏立体透视效果，本程序没有使用到。

透视投影是视点在有限距离处的投影模式，它所产生的结果类似于照片，有近大远小的效果。

在 OpenGL 中用 gluPerspective 函数来设置在窗口中所观察到的内容。这个函数一般设置一次就不再改变了。

(2) 模型变换：glMatrixMode(GL_MODELVIEW);

在这种模式下，可以改变观察点的位置与方向和改变物体本身的位置与方向。

改变物体的位置和方向主要涉及以下三个函数：glTranslate()，glRotate()，glScale()。第一个函数在本程序中多次使用，它可以改变物体的位置。

改变观察点的位置主要使用：gluLookAt()，他可以设置观察点（相机）的位置和观察目标的位置。

接下来说一下我对场景漫游的理解：场景漫游就是通过鼠标或键盘的操作，改变视角范围，调整在 OpenGL 窗口中显示图像的内容，以使观察者可以获取到一个 3D 场景的全部内容。在投影变换中说到，用 gluPerspective 函数设置我们可以看到的一个“近大远小”的场景，通过调整参数可以设置出一个合理的观察视口。这个设置是不改变的，即在这一视角下能观察到的内容是不全面的，因此可以转到模型变化下，使用 gluLookAt()调整观察点的位置以观察到更多的图像内容。

gluLookAt()有 9 个参数，分别是 3 对坐标，第一对是人眼（相机）位置。第二对表示眼睛“看”的那个点的坐标，可以理解为视线方向。最后一对表示观察者的方向，设置为正向观察即可。在程序中我们是这样设定的：

```
float cx = 0.0f, cy = 15.0f, cz = 80.0f;//相机位置
float lx = 0.0f, ly = 0.0f, lz = -20.0f;//视线方向，初始设为沿着 Z 轴负方向
gluLookAt(cx, cy, cz, cx + lx, cy + ly, cz + lz, 0.0f, 1.0f, 0.0f);
```

我们将视线方向设置为向屏幕内看去，知道了改变观察点位置的函数，我们就可以写出响应键盘和鼠标的函数，根据相应的指令去修改 gluLookAt 函数中的参数即可完成视角范围的变换。

相机和视线同时移动：平移相机改变其位置，视线方向不改变。有六个方向可以移动，分别是前、后、左、右、上、下，对应着改变 cx、cy、cz 的值即可。由于 lx、ly、lz 不改变，因此是只移动相机，不改变视线方向。

仅视线移动：相机位置不变，改变视线方向。可以左右旋转 360 度，也可以上下移动相机角度。按照三角函数计算，对应着改变 lx、ly、lz 即可。由于 cx、cy、cz 不改变，因此是相机位置不变，只转动相机镜头。

(1) 键盘响应

按下不同的键，相应地改变 gluLookAt 函数的参数。由于每次按下是修改一个定值，在视角变化时并不灵活，所以编写了鼠标响应函数。

(2) 鼠标响应

可以检测到鼠标左键还是右键按下了，每当有键按下时就记录当前的鼠标坐标。当鼠标移动时，实时记录鼠标新的位置的坐标，将鼠标按下时的坐标与实时移动时的坐标相减，可以得到在屏幕 x 和 y 轴上的一个差值。根据这个差值去调整 gluLookAt 函数的参数，因为视角的调整是跟随鼠标移动而改变的，因此视角范围变化时会比键盘控制显得更加流畅，灵活自然。

本程序设计了鼠标左键响应和鼠标右键响应。左键点击并移动，控制视线方向。右键点击并移动，控制相机位置。

■ 矩阵的入栈和出栈

一般有两种方法对模型的位置进行处理，一是在建模时就指定好坐标；二是在原点画出模型，再通过移动函数将模型移动到指定位置。

对于第二种思路，移动之后需要注意，矩阵的上一次的变换结果对本次变换有影响，上次 modelview 变换后物体在世界坐标系下的位置是本次 modelview 变换的起点。所以如果不进行处理就开始绘图，或是移动位置，它的基点是在上次移动后的位置，而非原点。解决方法是使用 glPushMatrix()和 glPopMatrix()进行矩阵的压栈和弹栈。

当做了一些移动或旋转等变换后，使用 glPushMatrix()，OpenGL 会把这个变换后的位置和角度保存起来。然后你再随便做第二次移动或旋转变换，再用 glPopMatrix()，OpenGL 就把刚刚保存的那个位置和角度恢复（比如原点位置）。

即本次需要执行的缩放、平移等操作放在 glPushMatrix 和 glPopMatrix 之间，glPushMatrix()和 glPopMatrix()的配对使用可以消除上一次的变换对本次变换的影响。

在程序设计中，经常会调整模型的位置，以达到更合适的布局和效果，所以每次要进行 modelview 变换时，都需要在 glPushMatrix 和 glPopMatrix 之间。除此之外，在本程序中的雪花模型与小人模型中会更多次的用到，因为程序需要控制多个这样的动态模型，在每一时间 t 内，模型都需要移动，因此必须计算好每次移动后的坐标，在 glPushMatrix 和 glPopMatrix 之间进行模型的移动，这样后续的模型才是在原点位置操作，才能移动到正确的位置。

■ 雪花模型

结构体：位置，颜色，大小，速度，加速度，存在时间，消失速度。

原理：在正方形上贴雪花图，设置混合因子获得半透明效果。

```
glBlendFunc(GL_SRC_ALPHA,GL_ONE);
glEnable(GL_BLEND);
```

生成位置：场景上方同一高度 y，随机 x、z 坐标初始化。

```
sx = pow(-1, (rand()%2))*(rand() % groundSize);
```

```
sy = 200;
sz = pow(-1, (rand()%2))*(rand() % groundSize);
```

移动：给定 xyz 轴上的速度，雪花按照此速度在每一时间 t 内向三个方向移动。y 轴向下加速移动（每一时间 t 内进行一次加速），x 和 z 轴上随机方向移动，即雪花按照一个角度倾斜移动。

```
sx += (vx * 5);
sz -= (vz * 5);
sy -= vy;          vy += ay;
glTranslatef(sx,sy,sz);
```

消失并复位：检测到雪花落到地上，即 y 轴坐标小于或等于 0。或是生存时间到了（避免速度太小，落不下来），就重新初始化雪花的位置，即回到初始高度。

```
if (sy <= 0 || lifetime <= 0)
    initSnow();
```

■ 小人模型

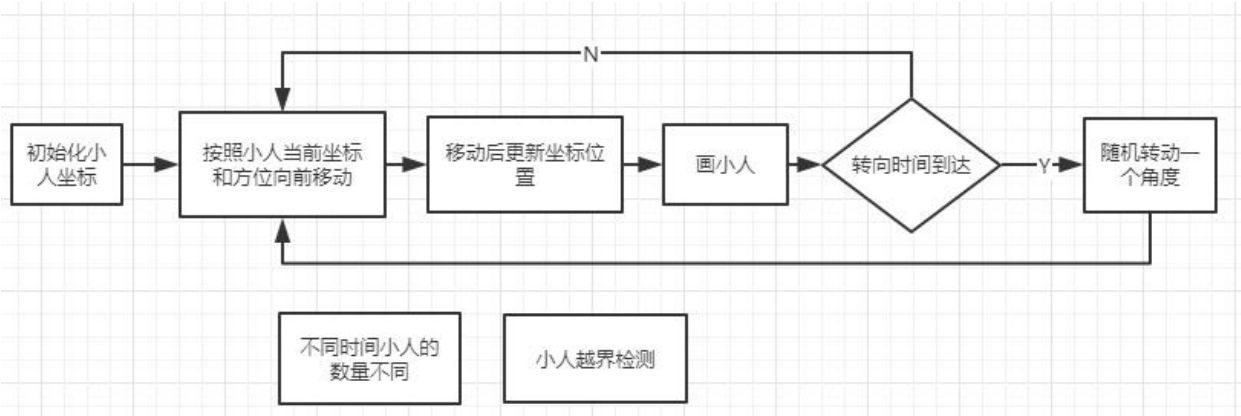
结构体：移动时间，xyz 轴上的坐标，旋转角度。

原理：此模型由 6 个立方体块构成，分别是头，身体，腿和胳膊。当小人移动到指定位置时，调用画小人的函数，将小人模型绘制在此位置。

生成位置：y 轴在场景的地面上，随机的 x、z 坐标。

移动：按照当前的方向，移动一段距离，当到达移动时间后，给小人一个随机角度，这样小人下次就会按照新的方向移动，即实现了小人的转向。

下面是小人模型的逻辑图。



■ 天空贴图

问题：天空盒是一个立方体贴图，但如果图片处理的不好，在立方体每个面的交接处会有一道缝隙。除此之外，为了让立方体的六个面显示的图像能够连接在一起，而不是 6 张一样的图显示在 6 个面上，需要对原贴图进行特别的处理。这两个问题导致我们无法构建一个自然、效果不错的天空盒。

思路改进：我们将立方体改为了一个球体，因为球体贴图时，不需要进行指定坐标位置的纹理映射。所以原本的一张图片映射到球面上，只是将图片上的元素拉伸了，并不像 6 张图片组合在一起那样分割了图像元素。且在处理图像的时候比较容易，只需要将图像的底色设置为同一颜色，不会出现图像边界在相接时的缝隙问题。

下面是创建一个球体并贴图的代码：

```
skySphere = gluNewQuadric();//申请二次曲面空间
glBindTexture(GL_TEXTURE_2D, texBeiJing);
gluQuadricTexture(skySphere, GL_TRUE);//纹理函数
gluSphere(skySphere, groundSize*2, 80, 80);
```

效果改进 1：在场景漫游中需要移动相机位置，在移动时会离天空球体或近或远，且一直向一个方向移动会移出天空球体外。在实际场景中天空应该离我们很遥远，因此在相机移动时，不应该出现与天空的距离拉近或拉远的效果出现。即天空应该一直在视线的最远处，不会随着相机的移动而与我们产生距离的变化。因此在场景漫游中，需要加入这样的功能：在移动相机时，相应地移动天空球体的位置，使天空离我们的距离永远是不变的。

效果改进 2：由于光照对纹理贴图的影响很小，在昼夜交替时，紧靠光照无法实现我们想要的整体环境颜色的明暗变化（针对天空）。因此我们用 PS 软件处理了多张天空贴图，做出不同的明暗效果来，在时间函数的变化中，不断改变贴图内容，去模拟天空明暗变化的效果。

■ 基础图形绘制

在本程序的各种模型建立中，立方体块画的次数是最多的，画一个立方体需要画 6 个面，每个面都需要根据 4 个点的坐标连线生成。所以如果每次画立方体都要写一遍代码会非常麻烦，因此我们写了一个画立方体块的函数，它的代码和思路是这样的：

```
void constract(double x, double y, double z, double x1, double y1, double z1)   //长方体  {
    fang[0][0] = x;       fang[0][1] = y;       fang[0][2] = z;             //   第 0 个点

    fang[1][0] = x;       fang[1][1] = y;       fang[1][2] = z + z1;        //   第一个点

    fang[2][0] = x + x1;      fang[2][1] = y;       fang[2][2] = z + z1;        //   第二个点

    fang[3][0] = x + x1;      fang[3][1] = y;       fang[3][2] = z;             //   第三个点
    for (int i = 0; i<4; i++)     // for()循环来完成其余的四个点
    {
        for (int j = 0; j<3; j++){
            if (j == 1)       fang[i + 4][j] = fang[i][j] + y1;
            else          fang[i + 4][j] = fang[i][j];}}
}

void build()    //和 constract 一起用，画长方体
{
    glBegin(GL_POLYGON);
    glNormal3f(0.0, -1.0, 0.0);
    glVertex3f(fang[0][0], fang[0][1], fang[0][2]);
```

```
        glVertex3f(fang[1][0], fang[1][1], fang[1][2]);
        glVertex3f(fang[2][0], fang[2][1], fang[2][2]);
        glVertex3f(fang[3][0], fang[3][1], fang[3][2]);
        glEnd();    //        下底

        ……
    }
```

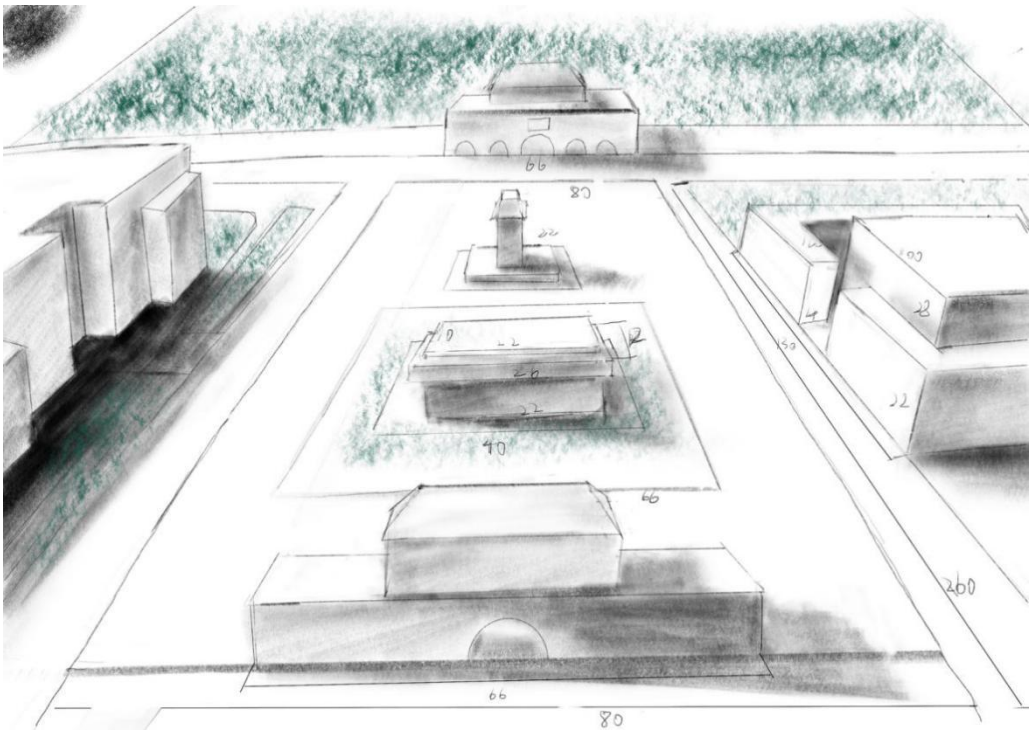
在画立方体时，需要给出两对坐标，第一个是立方体的前一面的左下角的点的坐标，第二个是他到后一面的右上角的点的偏移量。即只需要知道两个点的坐标，就可以得到立方体的长宽高，并写出其余 6 个点的坐标。

获取立方体的 8 个顶点坐标后，就可以分别画出 6 个面了。画四边形时可以用 glBegin()，通过指定参数告诉接下来想要画什么图形，使用 glVertex()来表示多边形的一个坐标。当给出所有的点后，OpenGL 会根据参数里的图形把点连接起来形成一个图形。

按照此方法，还可以封装三棱柱，四棱台和圆柱体的绘制过程。

■ 建筑模型

编写完基础图形的绘制，设计模型的样子、大小、尺寸和在场景中的位置，按照坐标绘制基础图形图形，完成模型的组合。



(1) 天安门

底座：立方体+门洞贴图+毛主席头像贴图+文字贴图。

主层：立方体+门窗贴图。

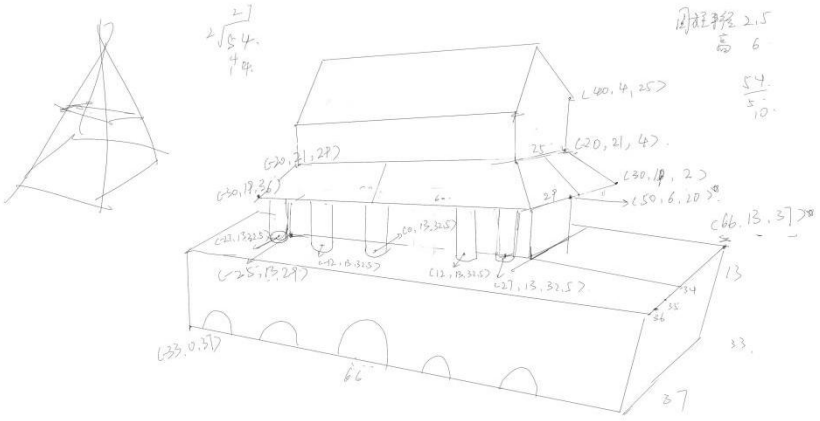
棱台屋檐：四棱台。

立方体屋檐：立方体+国徽贴图。

三棱柱屋檐：三棱柱。

柱子：10 个圆柱。

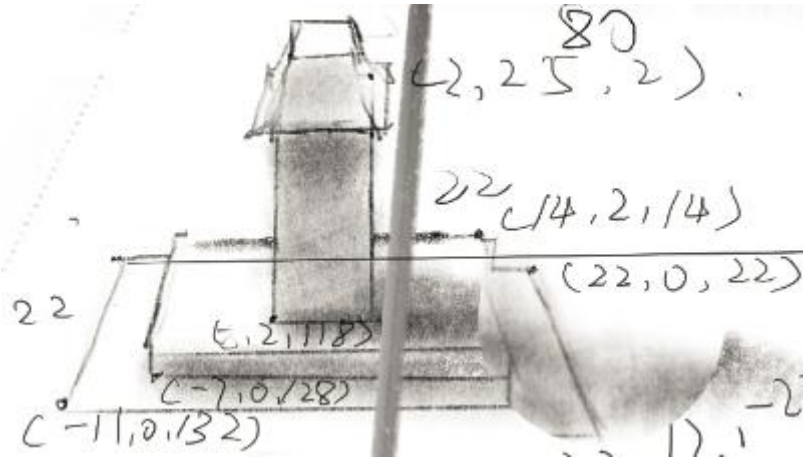
小红旗：6 个小红旗。圆柱旗杆+红旗贴图。



(2) 人民英雄纪念碑

底座：立方体

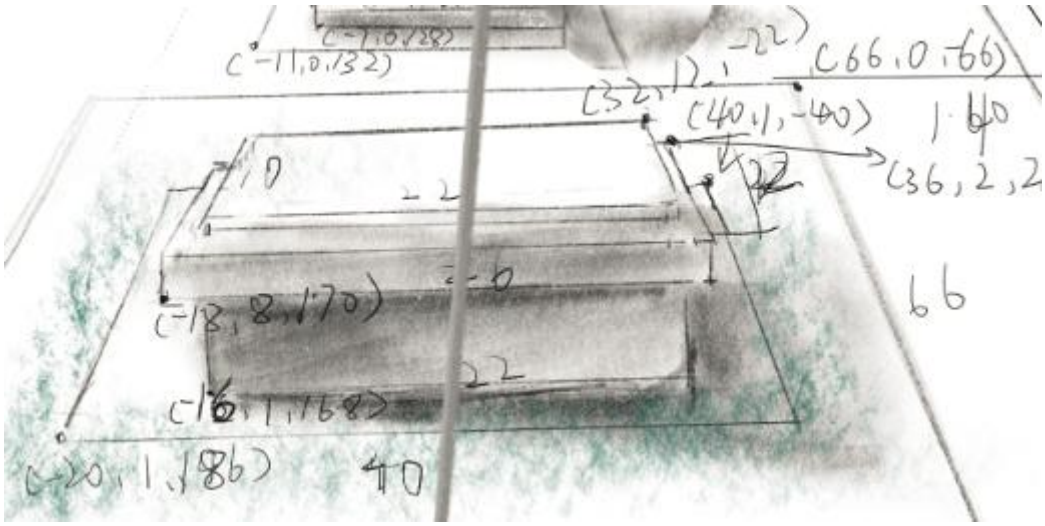
碑：立方体



(3) 毛主席纪念堂

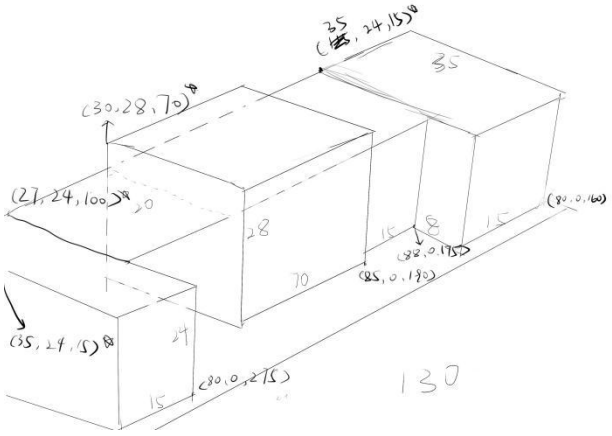
主层：立方体。

屋檐：立方体。



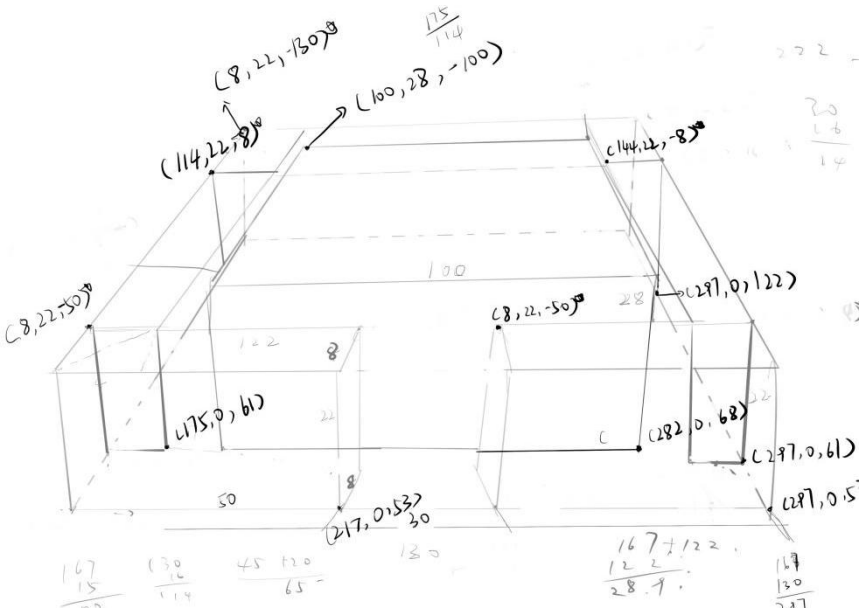
(4) 人民大会堂

由立方体块组合而成。



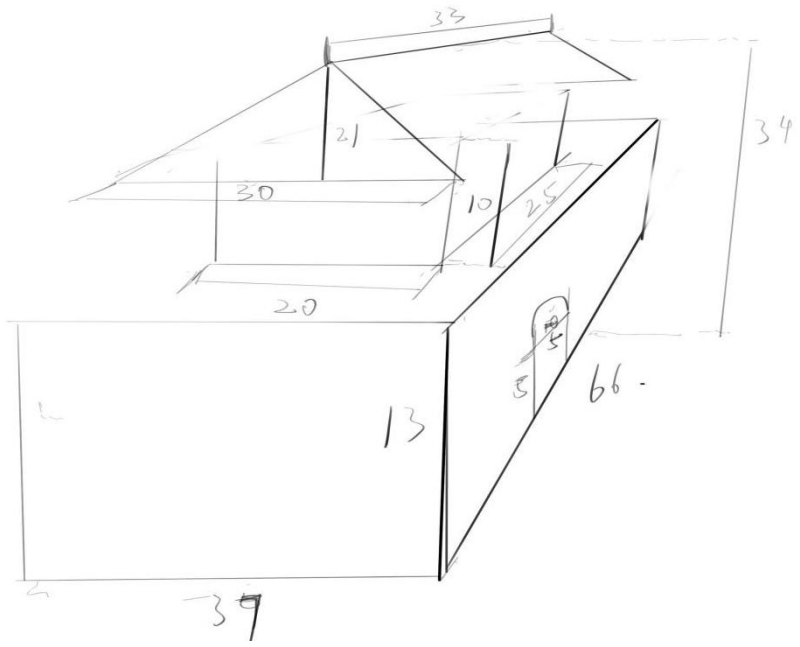
(5) 中国国家博物馆

由立方体块组合而成。



(6) 正阳门

由两个立方体和一个三棱柱构成。



■ 日月模型

日月模型与天空贴图的建立类似，都是画一个球体，再进行纹理贴图，只是天空球体是把整个场景包围起来的。日月球体比较小，而且是动态移动的，因此需要设计一个起始点和终止点以及移动轨迹。我们参考东升西落的自然规律，简单的设计了日月移动的轨迹。

```
if(timer <= pi){//白天
    glTranslatef(sunRadius * cos(timer), sunRadius * sin(timer), 0);
    glBindTexture(GL_TEXTURE_2D, texSun);}
else{       //晚上
    glTranslatef(sunRadius * cos(timer-pi), sunRadius * sin(timer-pi), 0);
```

glBindTexture(GL_TEXTURE_2D, texMoon);}

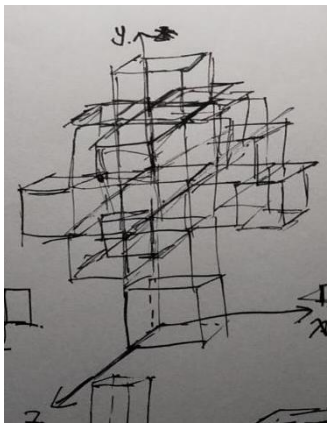
在 glTranslatef 函数中可以看出，z 轴是不变的，即在移动的过程中只改变横向与纵向的位置。通过三角函数将日月移动的轨迹设计为一个半圆形，且根据时间变量的变化，让日月交替的显现，达到一个日月交替的效果。

■ 装饰模型

为了让场景内容更加充实，我们设计了一些装饰模型，他们也是由基础图形组合而成的。

(1) 树木

树木模型是这样设计的，由一个个小立方体组合而成。树根由 5 个立方体搭建而成，第一层树叶由 8 个立方体排列组合而成，第二层由 4 个立方体排列组合而成，最高层有 1 个立方体代表树顶。



(2) 花坛

花坛由坛沿和草地贴图构成，用四个立方体拼成一个方框，在框内贴上草地的贴图。在场景的指定位置画 6 个这样的花坛。

(3) 告示牌

这是个附加的设计，画两个圆柱代表牌子的柱子，再画一个四边形贴上我们组员名字的贴图。在指定的位置分别绘制 4 个告示牌。

■ 其他模型

(1) 马路

由主路和两侧的路沿画成。主路和路沿都是多边形，主路较宽，路沿较窄。在主路的中心还画有中心条，在主路两侧的路沿上还按一定距离间隔地画出了两排树木。

本程序中共绘制了四道马路，两道宽的马路横着放置，路两侧有树木；两道偏窄的马路竖着放置，且路两侧没有树木。

(2) 国旗杆

草地贴图：在指定的位置画出四边形并进行草地贴图的纹理映射。

围栏：由 4 组 8 条围栏构成，分别放置在草地的四角位置。每条围栏由 3 个部分组成，分别是栏柱子（比较大的那个柱子，标志围栏的起始）、栏基座（比较长的长方体条）和栅栏（栏基座上的小柱子）。

基座：放置在草地的中心位置。它有两部分组成，分别是底座（一个大立方体）和 4 个小台阶（坐落在底座四周的中心位置）。

旗杆：放置在基座的正中心位置。是一个很细的圆柱。

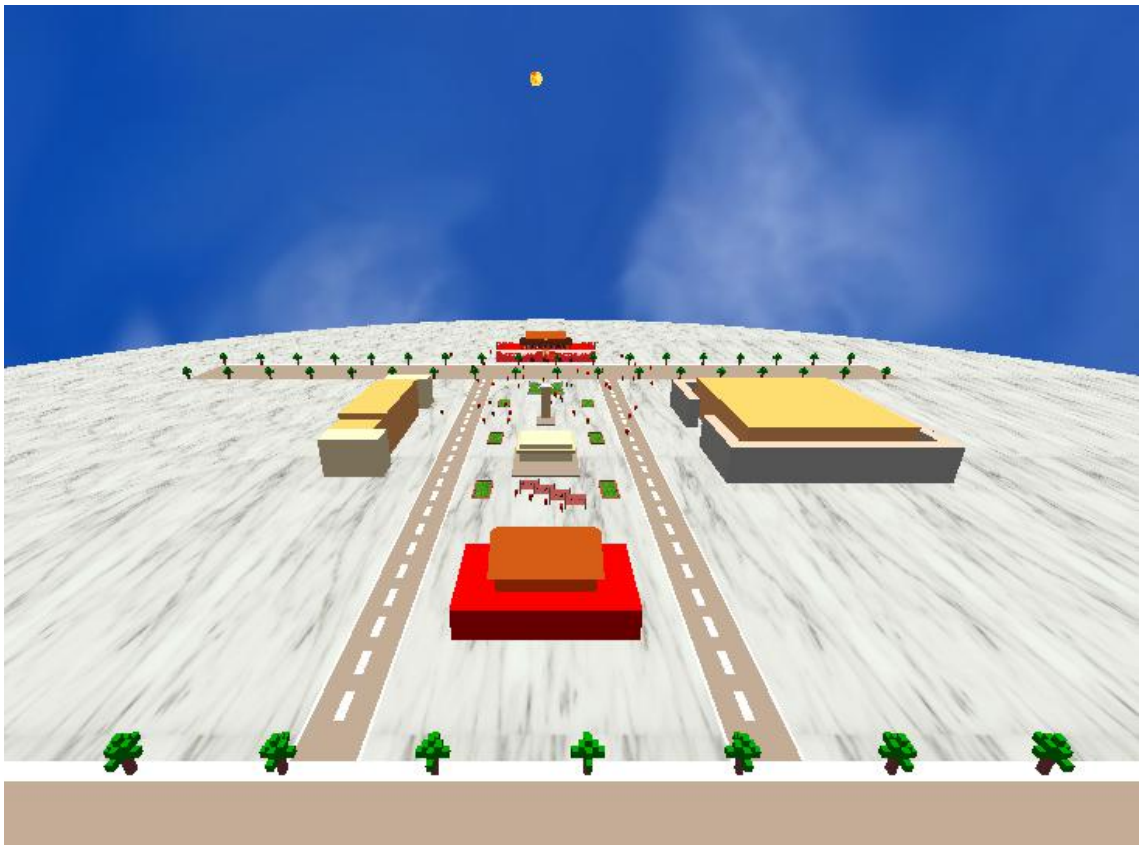
国旗：在旗杆的最上方处，向外画一个四边形，并映射上国旗的贴图。

五、测试分析

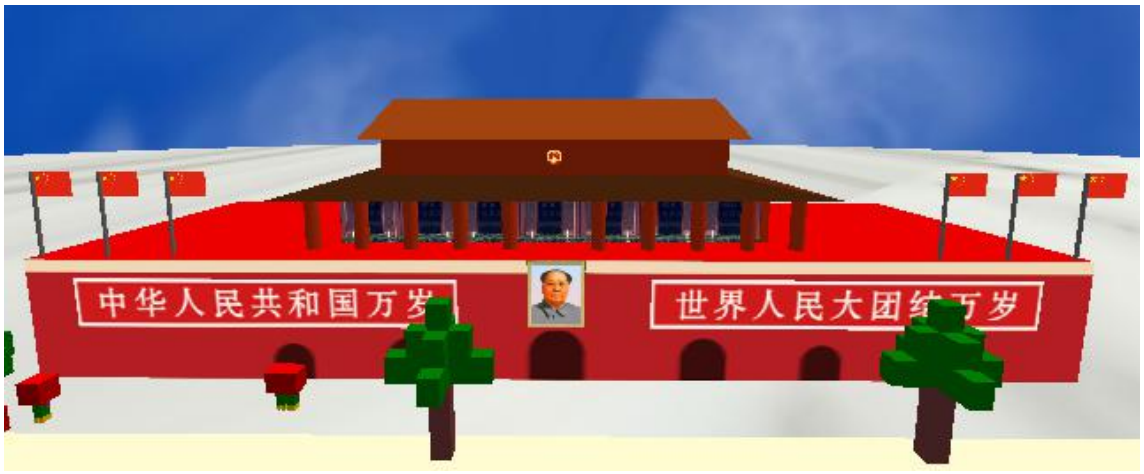
- 测试 1——模型显示检测

  检测场景中的模型是否正常显示，是否与设置的颜色、位置、样式、贴图一致。

  俯瞰全景：

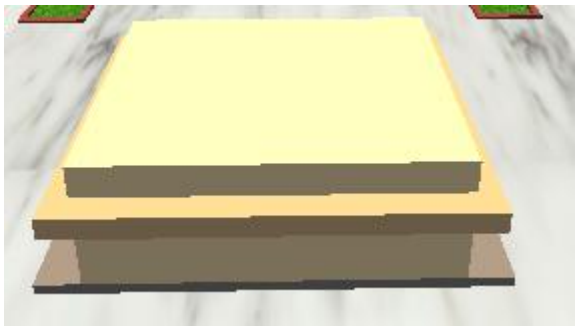


天安门：

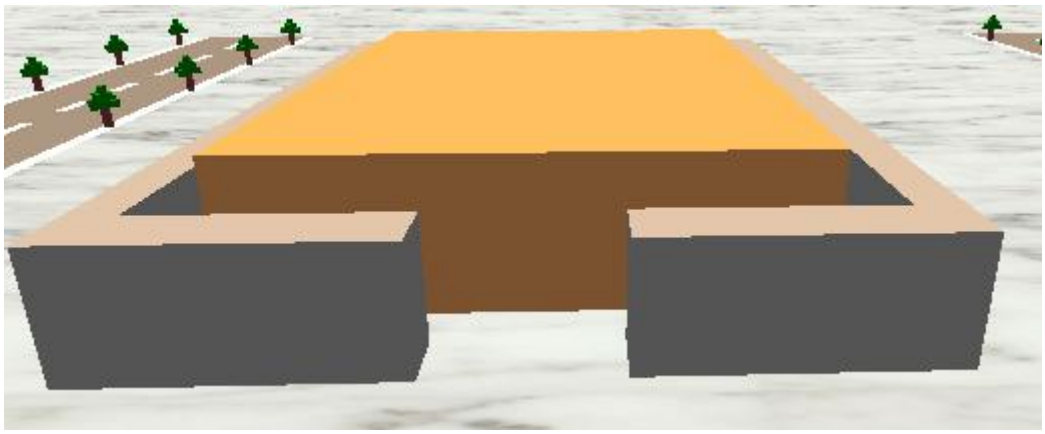


人民英雄纪念碑：

毛主席纪念堂：

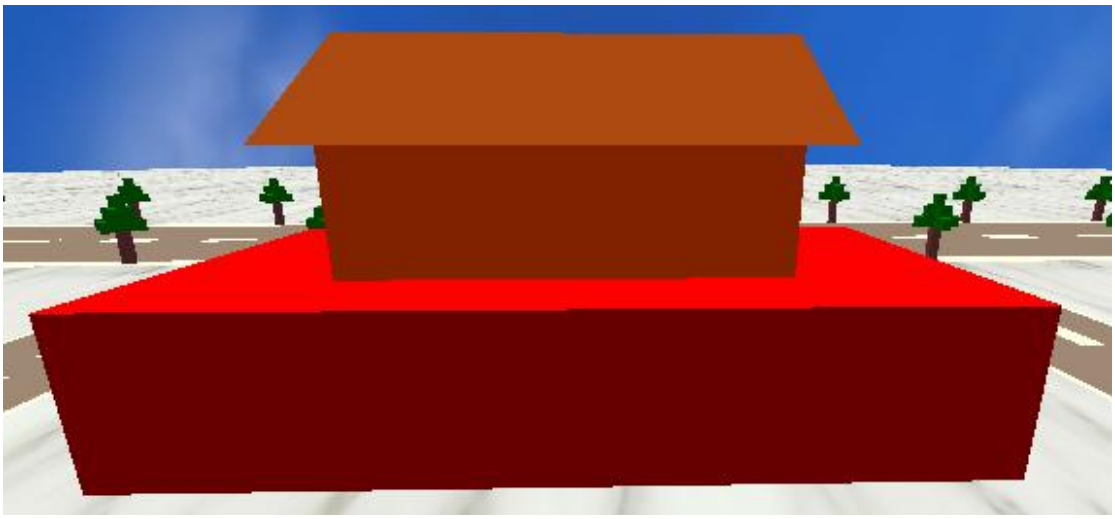


中国国家博物馆：



人民大会堂：

正阳门：

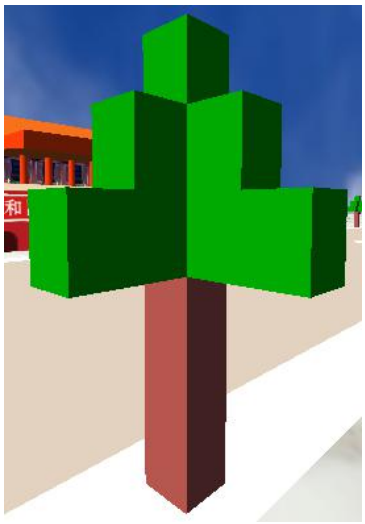


太阳：

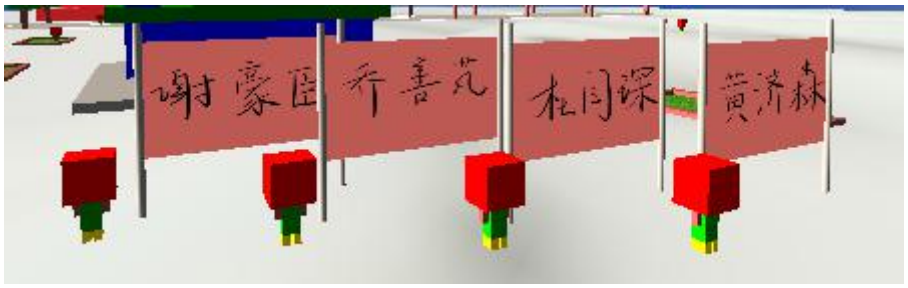
月亮：

花坛：

树木：



告示牌：

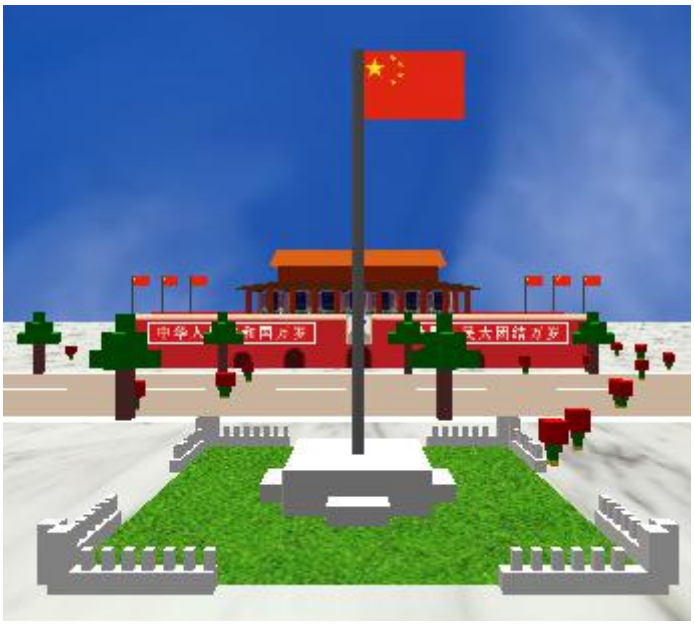


其他

马路（横）：

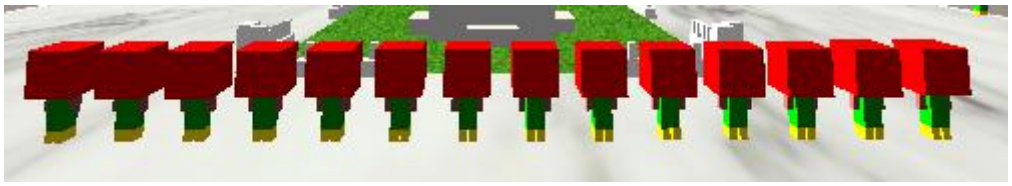
马路（竖）：

国旗杆：



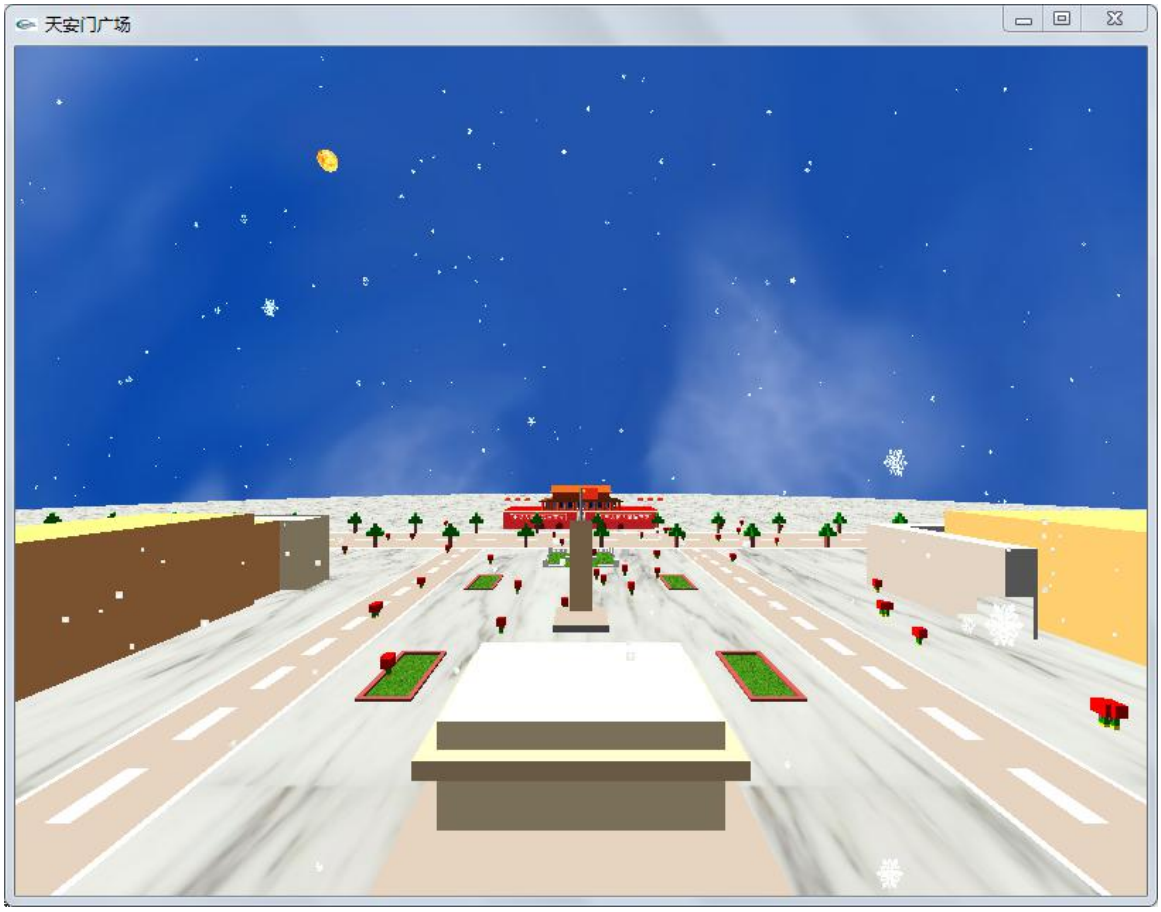
测试结果：以上模型的显示均符合预期效果。

小人：



测试结果：小人能够正常显示，且可以自由移动，不同时间显示的小人数量也不相同，符合预期效果。

雪花：



测试结果：键盘可以控制雪花效果的开启和关闭，且雪花效果符合预期效果。

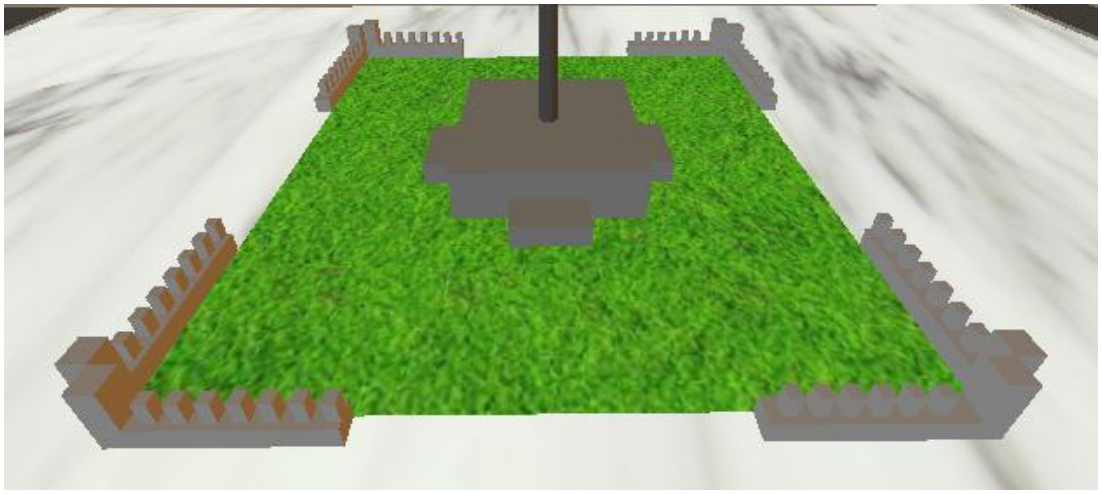
天空变化：

测试结果：天空贴图一共有 10 中变化，这里只展示其中几种。天空背景可以随着时间的变化而变化。

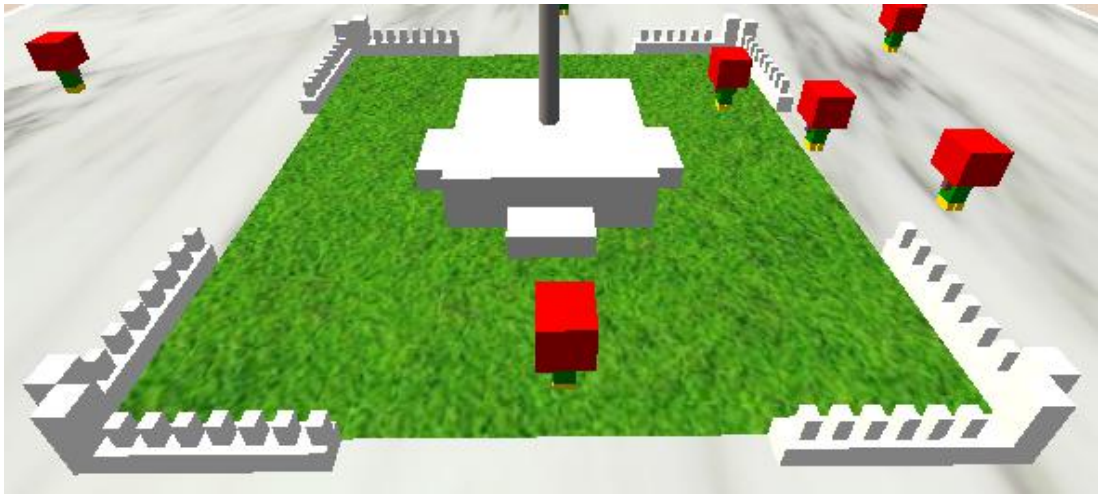
- 测试 2——检测光照

检测光照系统的功能，观察光源在移动和强度变化的过程中，对场景的影响。这里以国旗杆举例说明。

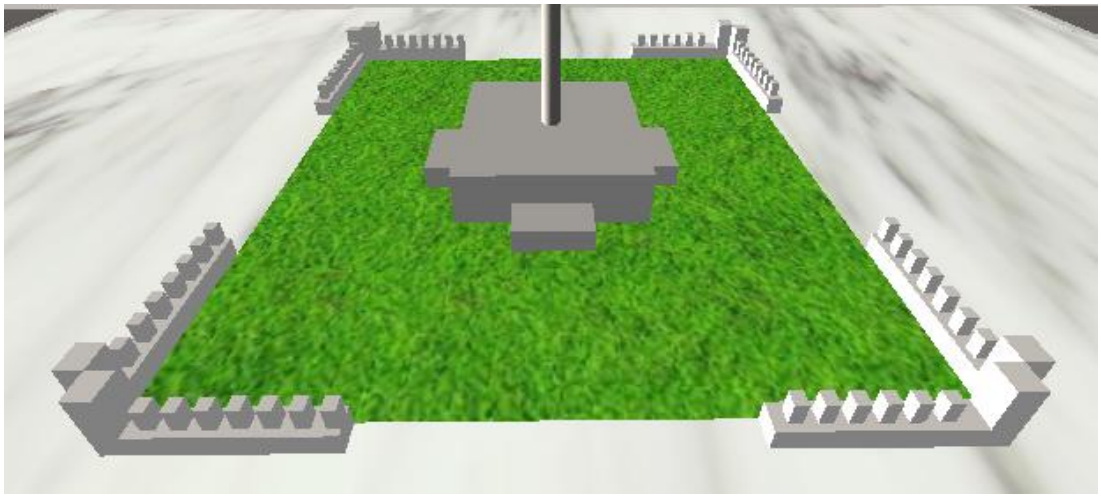
太阳在东侧，刚出现：

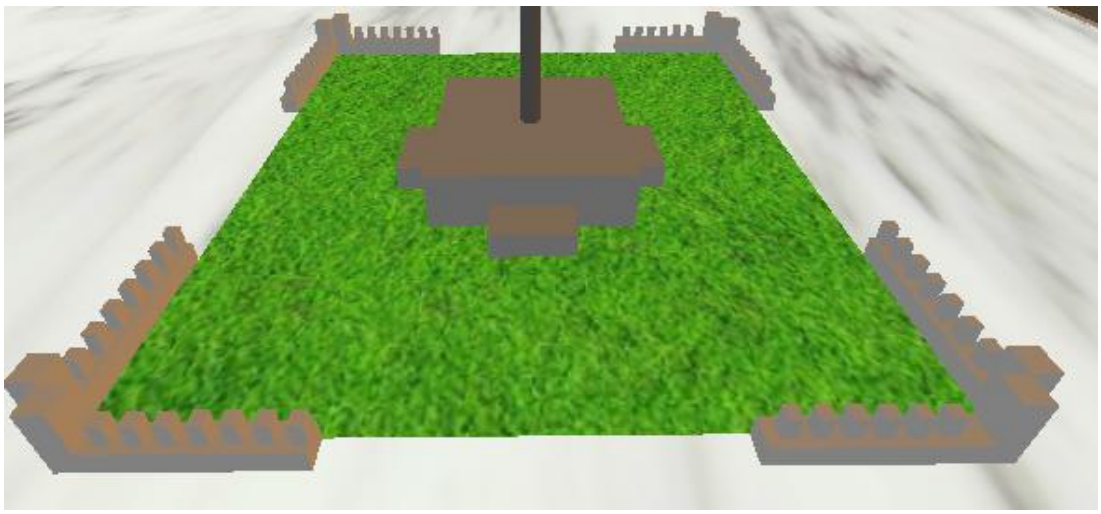


太阳在正上方：

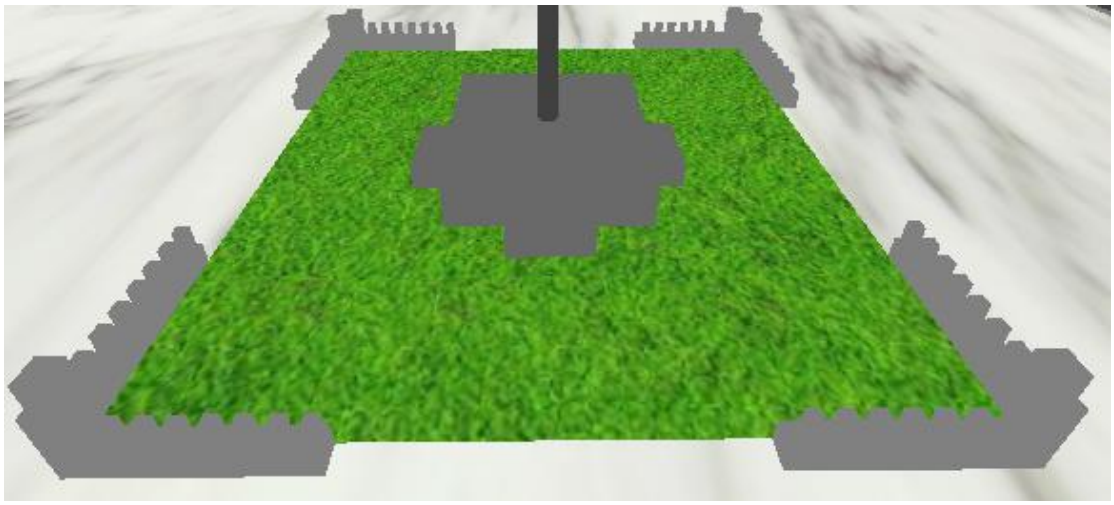


太阳在西侧：





太阳消失，月亮出现：

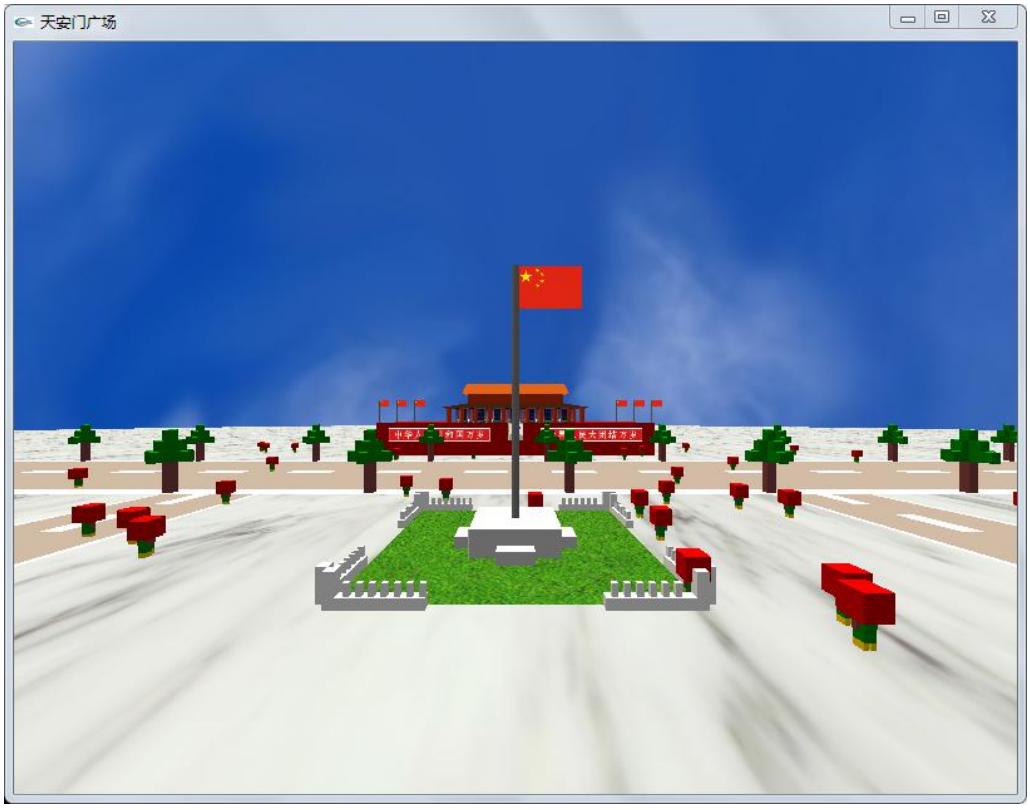


测试结果：光照移动时，可以看到建筑表面光强的移动。光强度变化时，可以看到建筑表面光强的变化。

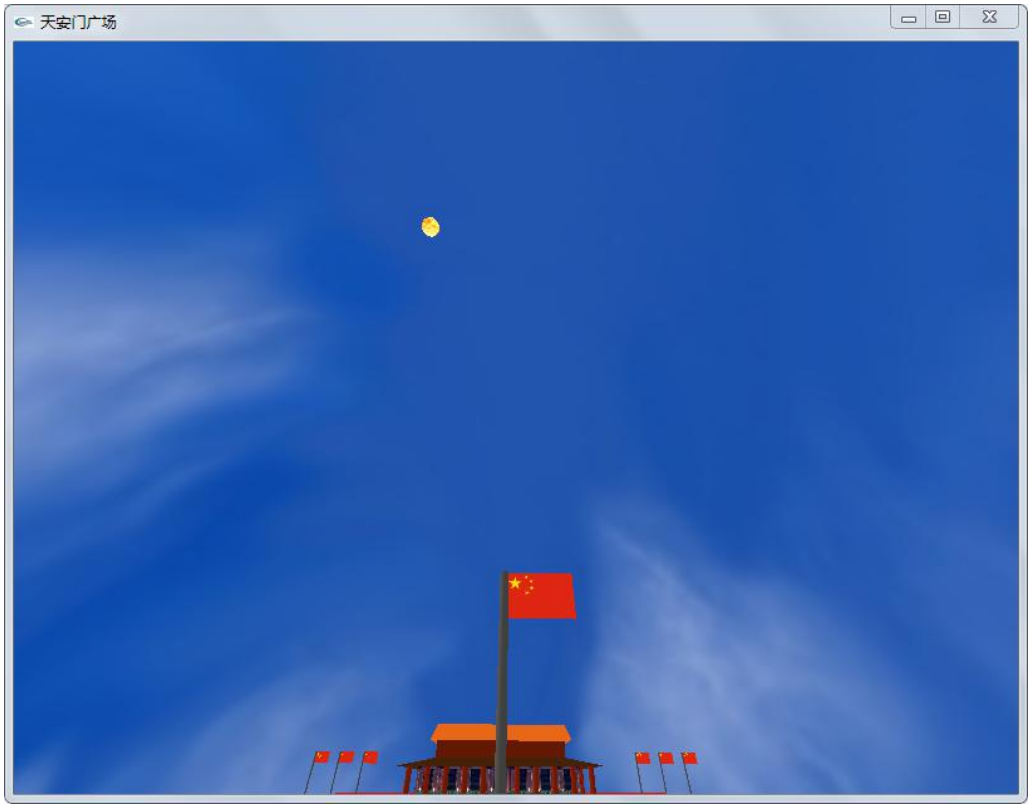
测试 3——鼠标键盘功能检测

检测键盘和鼠标对场景漫游的功能实现效果。

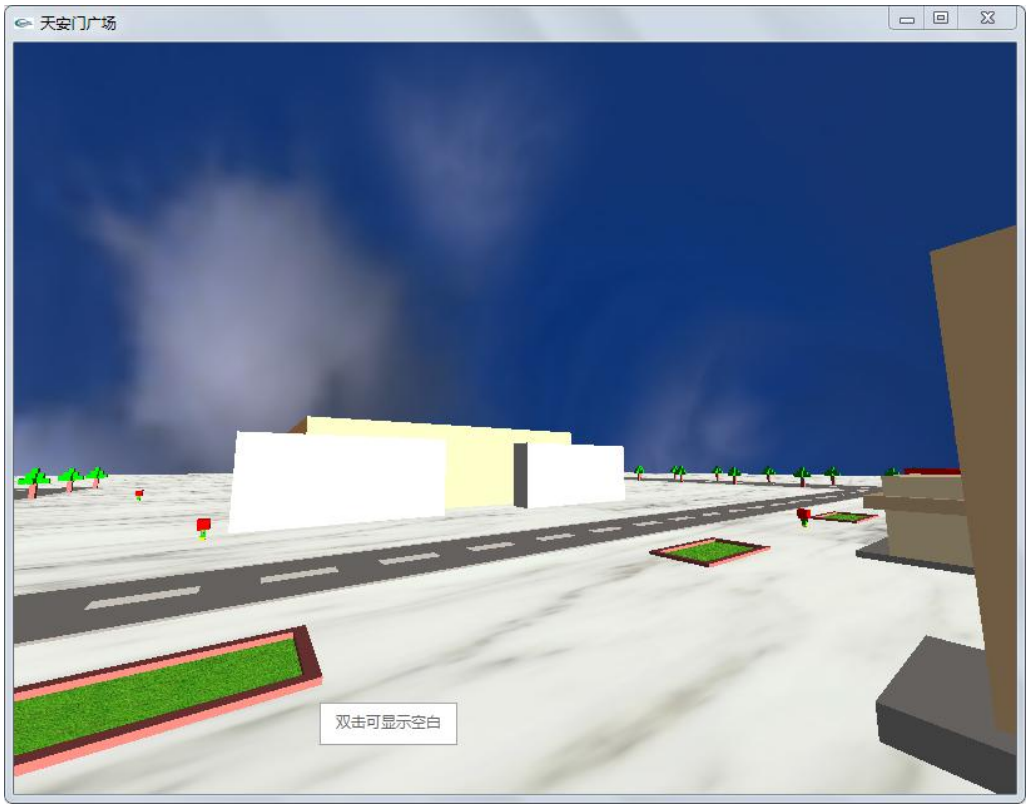
初始视角：



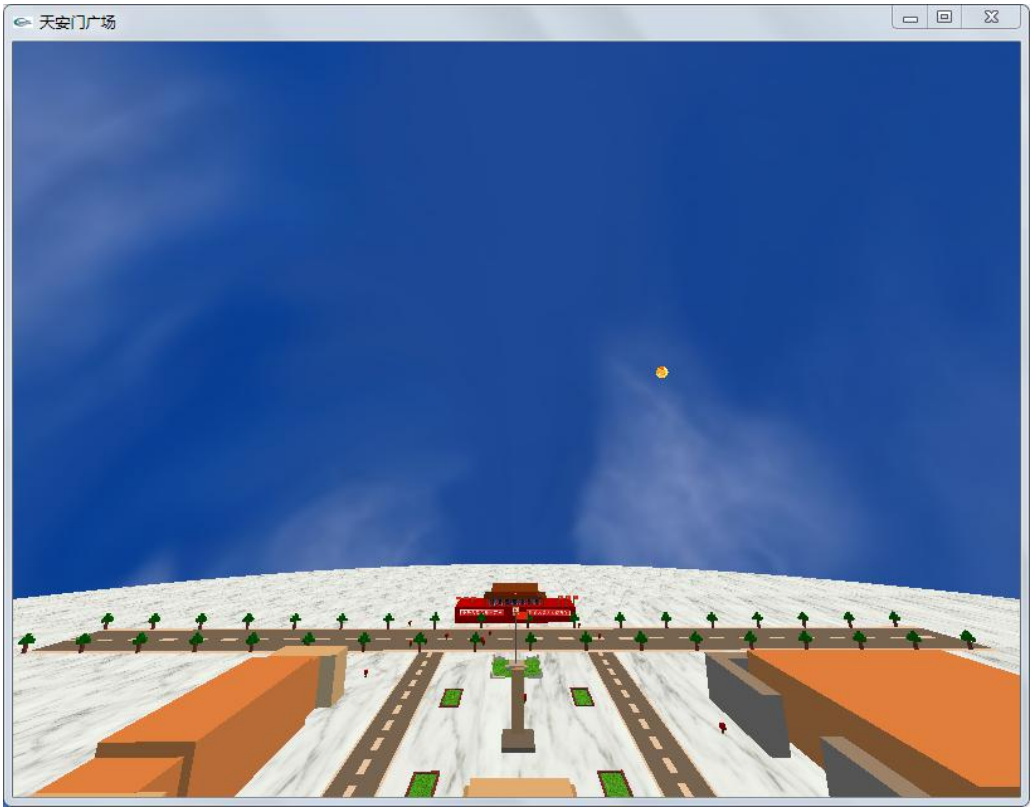
视角上移：



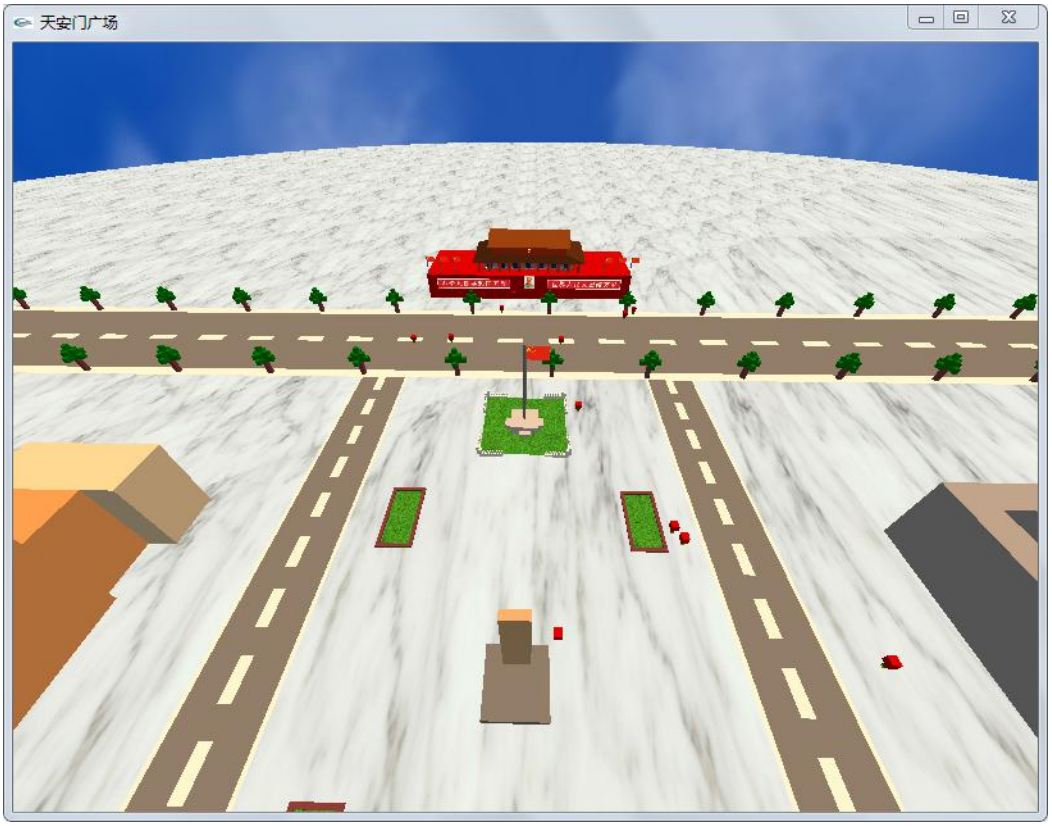
视角左右移：



相机移动：



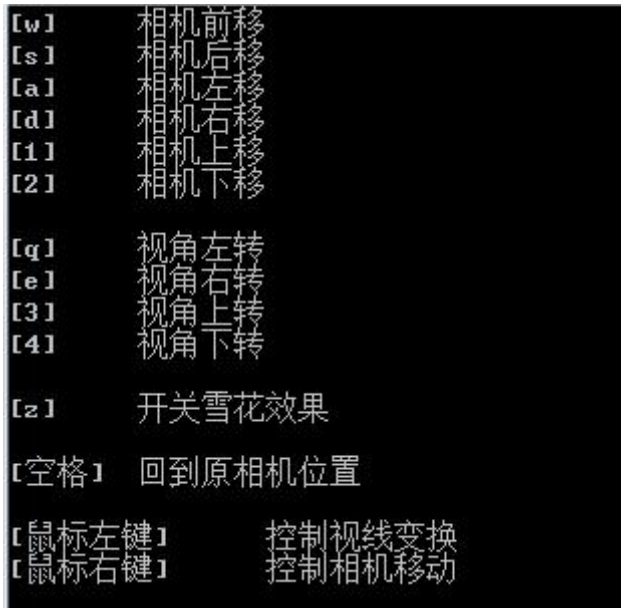
视角下移：



测试结果：键盘和鼠标均可按照预期效果进行相机的移动和视线的改变，可以正常完成场景漫游。

测试 4——输出检测

控制台显示：



测试结果：显示正确。

六、设计小结

1. 优点

- 模块划分

将程序划分成多个模块，每个模块代表一个功能，每个模块相对独立，所以在编程时便于调试当前模块的功能。

- 代码编写较为规范

函数名单个单词首字母小写/大写，单词间第一个单词首字母小写，其他单词首字母大写。

变量名单个单词首字母小写，单词间第一个单词首字母小写，其他单词首字母大写。下划线连接的单词首字母小写（多用在给控件起名）。

定义的常量变量，单词间首字母大写或单词全为大写。

代码有良好的注释，增强代码的可读性。

- 场景构建

场景构建比较还原真实场景的布局结构，建筑比例和坐标都是经过计算得到的。中间夹杂着其他模型的设计和布局，增添了场景中的元素，丰富了场景内容。光源的移动和光线的强弱变化，游客系统在不同时刻的数量变化，天空背景的转变，也都是为了尽量模拟真实场景所设计的功能。

- 选题有意义

今年正值祖国 70 周年华诞，我们组完成本次作品以表我们对祖国的热爱，同时祝愿我们亲爱的祖国更加的繁荣昌盛。

2. 缺点及改进

- 建筑细节

可以看到只有天安门加入了贴图和多处细节，其他建筑只是简单的基础图形的组合，再加上颜色。没有贴图的模型无法更好的还原原模型的特点，显得简单单调，看不出建筑特色与风格。因此其他几个建筑都应该像天安门广场那样，除了在基础图形组合上更加细致，在贴图和细节设计上也需要更加完善。

- 粒子系统

在雪花的设计上有一个小 bug，由于 OpenGL 只读取 bmp 位图文件，而 24 色位图文件属于不透明的图片（经尝试 32 色也不是），所以在进行贴图时必定会保留雪花元素以外的背景颜色。经过我们的代码处理，目前可以设置半透明的效果，但是只有当雪花落在天空背景前方才有效果（看不出位图背景）。当雪花落在所有模型附近时，背景会被重新显露出来，可以看到原本的带有背景的位图图片。由于雪花设置的比较小，在开启雪花效果后，总体来说效果还是不错的，但是此问题仍需要我们去解决优化。

- 光照系统

我们的光照系统只设置了两种光源，且光颜色的参数以及动态变化也都没有精细的研究与分析，和更多次的测试去观察效果，虽然程序最后能呈现出光照的效果，但是这个光照效果是否达到了最好还有待测试。

使用其他光照模型：比如 Phong 模型，它给光源加上了镜面反射光，他不管使用哪种模型，都需要进行多次的测试去找到能产生最好效果的颜色参数。由于对颜色的不敏感和对光照模型的理解尚浅，还有对自然界的光照的了解不够，寻找合适颜色参数去尽量模拟真实的光照效果对我们来说有些困难。

材质：在我们的光照系统中是没有开启材质的，这个也是由于光照参数本身设置的还不够完美，而且材质的参数需要针对多种不同的模型，调制起来比单个光源更加困难。想要达到最好的效果，单有光照肯定不够，每个模型都有自己对光线的反射和吸收效果，所以需要分析每类模型被光照射的影响，去尽量模拟真实世界的效果。因此对我们来说也比较困难。

光照对纹理的影响：我们知道光照是可以在纹理贴图上产生效果的，但是我们组确实对这方面没有特别多的学习与研究，因此没有做这方面的处理。所以在程序运行的结果中可以看到，当场景切换到晚上的时候，光线应该是最暗的，但是天安门表面的贴图仍与白天时的颜色一样。这一效果有待我们去完善。

光强度变化：我们的光强度变化做的比较简单，以最亮时候的 RGB 值做基准，利用三角函数对 R、G、B 三个值同时做同样的变化。也就是说我们并没有去分析光强度的更准确和具体的变化，只是将它整体调亮或者调暗，没有说在某一时刻对于某一分量做特殊处理，它的变化是均匀的。

- 场景漫游

场景漫游完成的效果以及很不错了，键盘和鼠标可以同时响应，比如用键盘控制相机（观察点）的移动，同时可以用鼠标左键控制视角，可以达到较好的漫游效果。但是代码中仍然存留有一些小 bug。

鼠标右键可以控制相机的前后左右的移动，最近在运行程序的时候发现了问题：移动相机的方向永远都是针对初始化视线方向（面向天安门的方向）的前后左右移动，当视线移动时，移动方向并没有随之改变。这一 bug 以后还需修改。

当调整视线方向时，左右旋转 360 度没有任何问题，但是当把方向向上或是向下改变时，我们发现可以旋转的角度有一个极值点，当旋转到一定角度时就会停止，甚至再向此方向旋转时，它会反方向的移动。且这个角度的极值点会随着相机高度的变化而变化。我们分析应该是在调整视线方向时设计的公式不正确，这一 bug 以后还需修改。

- 日月运动轨迹

与我们程序的大多问题一样，日月运动轨迹也没有去更真实地模拟真实场景效果，只是简单的在同一 z 轴上做东升西落的半圆变化。实际生活中肯定不是这样的，但是考虑到我们所做的场景比较小，这一部分目前还没有去深究。但是为了尽量模拟真实场景还是需要完善的。

- 天空盒

可以将球体改为画半球体。天空盒的背景图仍需要更好的处理，或者说用我们这种球体的贴图方法本身就不好处理，原图中的元素在映射到球体时都被拉伸了，这是很正常的。因为我们这种解决方法本身有点取巧，也没有找到我们计划效果的图片，对于图片的处理也还不够熟练。对于一个球体贴图，按理说应该设计这样一张图片：先在球体上画好天空的图案，然后把它展开成一个 2 维图片，这样在 2 维贴图转到 3 维球体上才能还原原本的效果。

我们组对天空盒的研究还存在不足，且 glut 工具包确实不好处理天空盒，我们也没有找到适合的图片，最终无法达到网上那些很好的效果。如果还有机会去完善本程序的话，我们希望使用其他的工具包尝试一下。

- 碰撞检测

这块我们已经写出一个简单的框架了，但由于时间关系，暂且将它搁置，转而去完善场景的构建了。在我们目前的场景中，在小人移动时是会穿过模型的，所以才需要到碰撞检测，让小人避开模型的位置，使其向着其他方向移动。

碰撞检测使用 AABB 包围盒，即一个以最小空间就能包围住模型的立方体，此后在描述模型的时候，就可以用立方体来代替。与绘制立方体块的思路一样，只要我们知道了前左下角的点与后右上角的点，我们就能确定这一个立方体的大小和位置。两个物体碰撞的条件是这样的：当两个模型的包围体在 x、y、z 轴上的映射全部重合在一起时，就可以确定这两个模型发生了碰撞。因此在模型移动的过程中，只要一直做这个检测就可以了。此功能还有待完善。

3. 预想计划

- 阴影

作为模拟真实世界的 3D 场景，怎么能少得了阴影呢，但是我们的程序还真就缺少了阴影效果。对于阴影的实现，我们小组有进行学习和研究，甚至尝试编写了代码，但是效果很不理想。我们大概了解了它的原理，但是用 glut 工具包不容易实现代码，且它的过程比较复杂，以我们对于 OpenGL 的认识与了解，理解并编写代码的确有一定的难度。如果还有机会，我们一定会给模型加上阴影效果，赋予它生命，而不是没有影子的“虚假”模型。

- 太阳光晕

光晕就是光线的“可视化”，在我们程序中是看不到光线的，只能看到光线照射到模型上产生的影响。而在真实世界中，人眼对着太阳时，会被它耀眼的光芒闪到，我们要做的就是把这个光芒显示出来。它的大概思路是：用一张面片绘制太阳和光晕，使其始终朝向镜头。

光晕分布在太阳位置和镜头中心的连线上，所以只有镜头对向太阳时才能看到光晕。如果还有机会，我们预计会实现这样的效果，以增强太阳在场景中的真实感。

- 飘起来的红旗

静态的红旗看起来十分的僵硬，因为它不比其他的模型，它的体质是很轻的，这就注定了它应该是一个随风飘扬的形象。我们组在网上找到了一篇关于飘动的红旗实现的文章，大概浏览了一下之后果断将它的优先级排到了我们实验的最末尾（大概思路是将一个多边形分成多个小块块，控制这些小块块的移动以实现红旗摆动的效果）。如果还有机会，我们预计会增加这一功能，让场景显得生动起来。

- 积雪效果

虽说本程序加入了雪花效果，但是雪落到地上迅速就“化掉了”，显然不够真实。程序最好是能让落下来的雪花贴图在地面或是建筑上停留一段时间（设定消失时间），当雪下的大时就能形成积雪的效果，是不是又真实又美妙呢。

同样，既然可以造出雪花，那也能模拟出雨滴，如果还有机会，我们预计会增加下雨的效果，增加雾化的效果，且最好是能在场景中形成小水坑的真实效果。

- 汽车

既然本程序都加上了游客系统，为何不考虑一下长安街上行驶的汽车呢。如果还有机会，我们预计会增加汽车功能，构建一个汽车模型，让他行驶在马路上，最好可以看到轮子转动的效果，并且可以自动转向拐弯。

- 完善场景

我们组最开始做的时候也没有想到能做成现在这样看着还不错的结果，场景中的很多元素都是慢慢添加的，随着对 OpenGL 的慢慢熟悉，随着对模型设计和建立慢慢地得心应手，当你开始认识到自己可以依靠自己掌握的技术做出什么东西时，你就会迫不及地去实现它，并构想更多的你想做的东西，在克服困难慢慢地实现你所构想的东西的过程中，你会发现你学到了很多东西，并会构想更复杂、更有意思的东西。我觉得我们组就是这样的状态，如果还有机会，我们不会满足于只实现当前这么小的场景（当然大致还是那个小范围），只做那么简陋的模拟真实感的各种效果。我们可能会逐渐完善周边的环境，完善建筑的细节（这个前面也介绍了），学习更好的技术以做出更好的效果。

七、使用说明

运行环境：

(1) Windows 操作系统。
(2) Codeblocks 软件。
(3) OpenGL 工具包中的 glut 库。

可以在附录中查看如何在 Codeblocks 中配置 glut 库。

启动方式：

1.在配置了 glut 环境的 Codeblocks 中打开 cbp 项目文件，点击构建和运行。
2.直接在项目文件中的 bin 文件中运行.exe 文件。

注：位图的文件位置设置的是绝对路径："C:\\Users\\user\\Desktop\\Assignment\\bmp***"如果程序位置不一致，运行的结果是不带有贴图的效果。

八、附录

1. Codeblocks 配置 glut 的步骤：

1.首先需要下载以下三个文件：

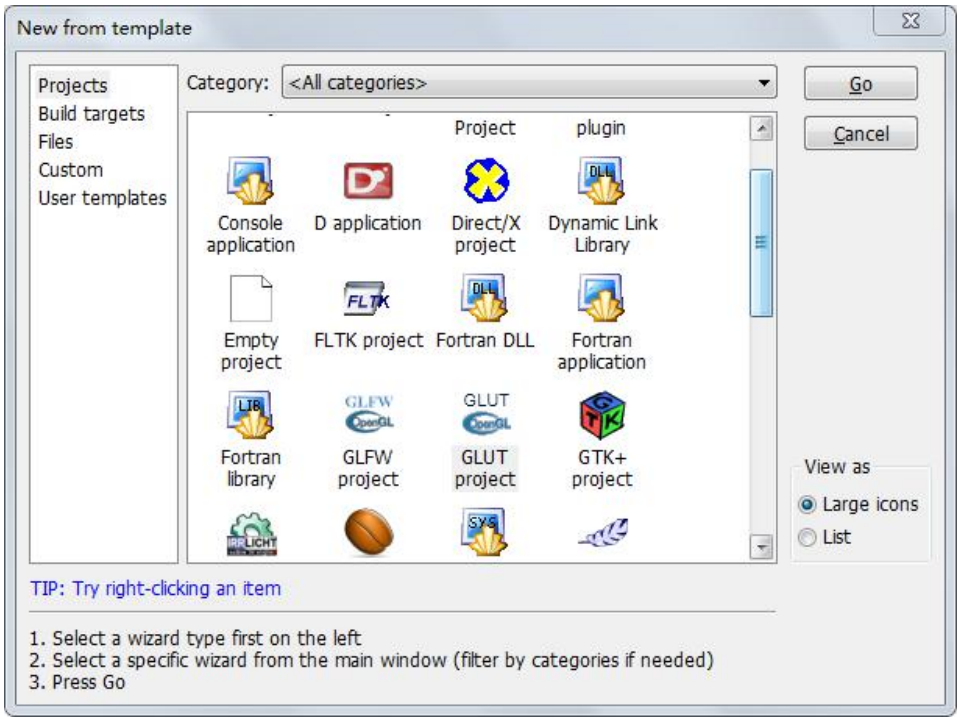
2.将 glut.h 文件放到 Codeblocks\MinGw\include\GL 目录下面。

3.将 glut32.dll 文件放到 C:\windows\system32 目录下面（如果是 64 位操作系统的话，将这个文件放到 C:\Windows\SysWOW64 目录下面）。

4.将 libglut32.a 放到 Codeblocks\MinGw\lib\目录下面。

2. Codeblocks 中创建 glut 项目的步骤：

1.创建 project,选择 GLUT project,点击 Go;



2.点击 next;

3.给 project 起名，并指定项目所在路径；

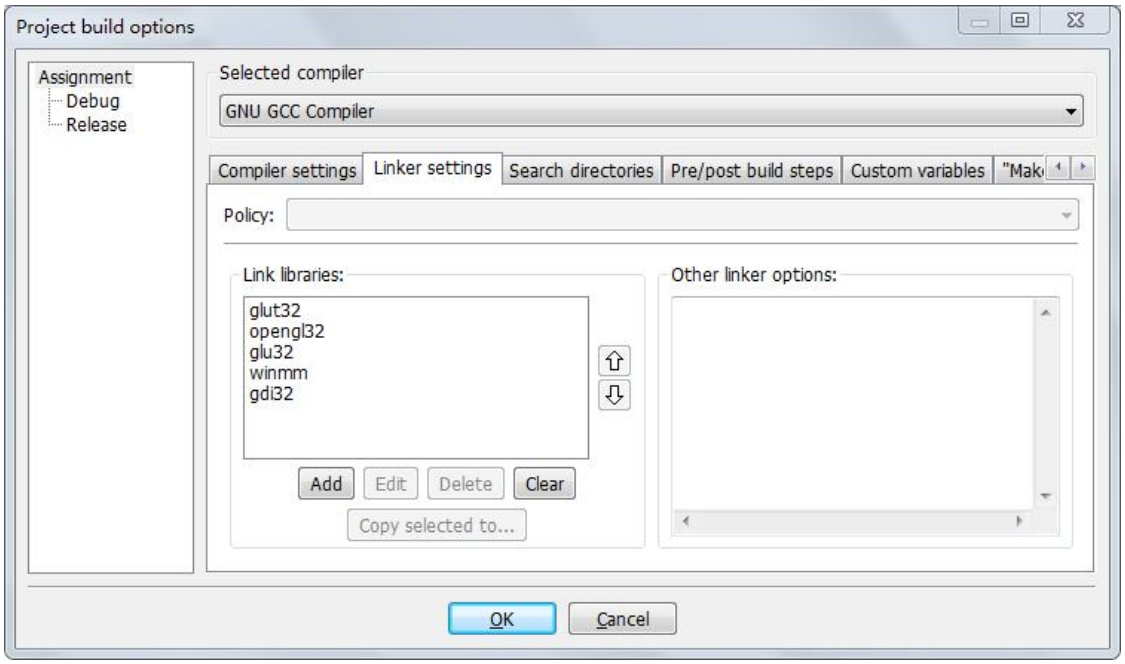
4.给定 MinGW 所在路径；

5.点击 finish.

6.在 CodeBlocks 窗口打开 Sources 下的 main.cpp，必要时可以添加：#include <windows.h>

7.在主菜单 Project 中点击选项 build Options,点击 Linker 页框，点击

Add 按扭，在弹出窗口里选择 libglut32.a 文件（包含所在路径），点击 OK 按扭，再点击 OK 按扭关闭 build Options。



8.此后只要直接新建 glut 项目，加入#include <gl/glut.h>，就可以用 glut 工具编写 OpenGL 代码了。