# 太空漫游者

# 系统设计说明文档

负责人：张驰

项目成员：李明哲 姚昕 周子孟

2016 年 1 月 8 日

# 目录


1 引言 . . . . . 3

1.1 文档编制的目的 . . . . . 3
1.2 背景 . . . . . 3
1.3 其他 . . . . . 3

2 设计概述 . . . . . 3

2.1 任务和目标 . . . . . 3
2.2 开发与运行环境 . . . . . 4
2.2 项目开发工具 . . . . . 4
2.3 条件与限制 . . . . . 4

3 系统详细设计 . . . . . 4

3.1 系统结构设计及子系统划分 . . . . . 4
3.2 系统功能模块设计详细 . . . . . 5
*3.2.1 globals 文件 . . . . . 5*
*3.2.2 TGA 类 . . . . . 5*
*3.2.3 Moon 类 . . . . . 6*
*3.2.4 Planet 类 . . . . . 7*
*3.2.5 Wormhole 类 . . . . . 8*
*3.2.6 SolarSystem 类 . . . . . 8*
*3.2.7 Camera 类 . . . . . 9*
3.3 MAIN 主程序控制 . . . . . 11

4 项目难点及解决方案 . . . . . 11

4.1 程序截图 . . . . . 11
4.2 纹理映射 . . . . . 13
4.3 碰撞检测与虫洞效果 . . . . . 14

5 程序使用方法 . . . . . 15

6 致谢 . . . . . 15

7 参考资料 . . . . . 15

# 1 引言

## 1.1 文档编制的目的

本文档旨在记录太空漫游者这款游戏在设计过程中所使用的技术以及克服的困难，为后期的代码批注、审阅、修改等工作提供一个初步的参考。

此外，本文档作为本次项目的一个记录，也同时能为其它相关项目提供技术支持，让后来者少走弯路。

## 1.2 背景

太空漫游者这款游戏专门为 2015-2016 秋冬学期浙江大学计算机图形学课程所设计。通过使用 OpenGL 绘制的太空漫游空间，综合展示 OpenGL 的强大绘制功能，并将本学期课程所学揉合起来，说明项目组成员对计算机图形学的深入理解以及对 OpenGL 的熟练掌握。

## 1.3 其他

在本项目中，张驰作为项目负责人，负责系统构架设计、太阳系模型编写和报告撰写；李明哲完成了碰撞检测、星系切换和算法设计功能；姚昕主要负责界面设计和游戏贴图；周子孟负责视场切换部分及程序总测试。

我们的项目完全由 OpenGL 编写，没有其他依赖，并已在 GitHub 上开源，项目地址为

https://github.com/WellyZhang/SpaceWanderMan。

# 2 设计概述

## 2.1 任务和目标

太空漫游者作为一个展示课程所学及自身能力的项目，实现如下功能与效果：

- 构建多个贴图及光照效果不同的太阳系
- 以第一人称视角控制一艘在太空中飞行的飞船
- 通过键盘和鼠标分别控制飞船的运动和视角的调整
- 在临近实体星球时给用户发出提示以避免飞船与星球碰撞
- 与星球碰撞时弹出切换场景并将本轮游戏结束
- 设计虫洞——进入虫洞后飞船被随机送至其他太阳系中

- 为用户提供存档读档功能和截屏功能

即能够实现项目要求中的：

- 基本要求：
  1. OpenGL 的基本体素绘制与建模
  2. 模型的导入导出
  3. 纹理材质编辑与贴图
  4. 基本几何变换
  5. 光照模型的使用与编辑
  6. 场景漫游的实现
  7. 屏幕截取
- 进阶要求：
  1. 碰撞检测
  2. 具有可玩性
  3. 具有一定窗口效果的表达能力

## 2.2 开发与运行环境

本程序基于 Visual Studio 2013 版本的 C++语言开发，理论上能够在任何合理配置 OpenGL 的编程环境上运行。在实际测试中，我们发现，程序能正确地在 Windows 7 及 Windows 10 两个平台上运行。由于程序依赖 Windows.h 库文件，因此 Linux 各发行版本及 Mac OS 系统都无法使用本程序。

## 2.2 项目开发工具

本程序在系统构架上完全依赖于 OpenGL 的 C++版本，并未使用任何其他的程序设计库。同时，为了给模型贴上合适的纹理，我们使用 Adobe Photoshop CC 为程序设计贴图样式。

## 2.3 条件与限制

本项目以 MIT 开源协议开源，项目使用者可以免费地对程序进行任意的修改、删除、分发等操作，而无需与本项目组达成商业协议。

# 3 系统详细设计

## 3.1 系统结构设计及子系统划分

本程序总共被分为八大部分，它们分别是：

- globals 文件——记录程序中所用到的全局变量
- tga 类——负责纹理映射的处理

- moon 类——负责实现星球的卫星模型
- planet 类——负责星球建模
- wormhole 类——特殊设计的虫洞星球以实现穿越效果
- solarsystem 类——负责太阳系建模
- camera 类——负责视角的调整及控制进动
- main 文件——系统主文件

其中 main 文件是程序的主文件，用于程序的整体控制。其余各类相互平行，没有继承关系。

## 3.2 系统功能模块设计详细

### 3.2.1 globals 文件

globals 文件包含两个全局变量：

- const float distanceScale
  功能：记录模型距离与真实距离之间的比例
- extern float planetSizeScale
  功能：记录模型星球大小与真实星球大小之间的比例

### 3.2.2 TGA 类

TGA 类负责纹理贴图的处理。

调用方法：

TGA *varName = new TGA(“filepath”);

库依赖：

- Windows.h
- glut.h

类成员：

- private:
  - GLuint textureHandler
- public:
  - TGA(char *imagePath)
    输入：tga 文件地址
    功能：将 tga 文件读入并存储在相应的结构体中；初始化 textureHandler
  - GLuint getTextureHandler(void)
    功能：返回 textureHandler 为主程序提供纹理映射

### 3.2.3 Moon 类

Moon 类能够建模行星的卫星。

调用方法：

Moon moon = Moon(distanceFromPlanet, orbitTime, rotationTime, radius, textureHandle);

库依赖：

- Windows.h
- glut.h
- globals.h
- cmath

类成员：

- private:
  - float distanceFromPlanet
    功能：记录卫星与行星距离
  - float orbitTime
    功能：记录公转时间
  - float rotationTime
    功能：记录自转时间
  - float radius
    功能：记录半径
  - GLuint textureHandle
    功能：记录其纹理
  - float position[3]
    功能：记录卫星位置
  - float rotation
    功能：记录离转轴偏离的角度
- public:
  - Moon(distanceFromPlanet, orbitTime, rotationTime, radius, textureHandle)
    输入：离行星距离，公转时间，自转时间，半径以及纹理
    功能：初始化卫星数据
  - void calculatePosition(float time)
    输入：时间
    功能：根据输入时间计算星球位置
  - void render(void)
    功能：绘制卫星
  - void renderOrbit(void)
    功能：绘制公转轨道
  - void getPosition(float* vec)
    输入：准备好的容器
    功能：获得卫星位置

- float getRadius(void)
  功能：获得半径

### 3.2.4 Planet 类

Planet 类负责行星的建模。

调用方法：

Planet planetName = Planet(distanceFromSun, orbitTime, rotationTime, radius, textureHandle);

库依赖：

- Windows.h
- glut.h
- globals.h
- moon.h
- cmath
- vector

类成员：

- private:
  - float distanceFromSum
    功能：记录行星与恒星距离
  - float orbitTime
    功能：记录公转时间
  - float rotationTime
    功能：记录自转时间
  - float radius
    功能：记录半径
  - GLuint textureHandle
    功能：记录其纹理
  - float position[3]
    功能：记录卫星位置
  - float rotation
    功能：记录离转轴偏离的角度
  - std::vector<Moon> moons
    功能：记录行星的卫星
- public:
  - Planet(float distanceFromSun, float orbitTime, float rotationTime, float radius, GLuint textureHandle)
    输入：离恒星距离，公转时间，自转时间，半径以及纹理
    功能：初始化行星数据
  - void calculatePosition(float time)
    输入：时间
    功能：根据输入时间计算星球位置
  - void render(void)

功能：绘制行星

- void renderOrbit(void)
  功能：绘制公转轨道
- void getPosition(float* vec)
  输入：准备好的容器
  功能：获得行星位置
- float getRadius(void)
  功能：获得行星半径
- void addMoon(float distanceFromPlanet, float orbitTime, float rotationTime, float radius, GLuint textureHandle)
  功能：为行星添加卫星

### 3.2.5 Wormhole 类

Wormhole 类负责虫洞功能的实现，它的接口与 Planet 类一致，但为了与 Planet 类以示区别，我们还是新建了一个类。它对成员函数进行了改写。

### 3.2.6 SolarSystem 类

SolarSystem 负责一个太阳星系的建模。

调用方法：

SolarSystem *galaxy = new SolarSystem();

库依赖：

- Windows.h
- glut.h
- planet.h
- wormhole.h
- camera.h
- tga.h
- cmath
- vector

类成员：

- private:
  - std::vector<Planet> planets
    功能：记录星系中所有的行星
  - std::vector<Wormhole> wormholes
    功能：记录星系中所有的虫洞
- public:
  - SolarSystem()
    功能：为该星系初始化并赋予初始值和纹理映射
  - SolarSystem(int mode)
    输入：模式代码

功能：若模式代码不为零，根据时间随机生成一个太阳星系

- void calculatePositions(float time)
  输入：时间
  功能：根据输入时间测算星系中行星及虫洞的位置
- void addPlanet(float distanceFromSun, float orbitTime, float rotationTime, float radius, GLuint textureHandle)
  输入：离恒星距离，公转时间，自转时间，半径以及纹理
  功能：为星系添加行星
- void addWormhole(float distanceFromSun, float orbitTime, float rotationTime, float radius, GLuint textureHandle)
  输入：离恒星距离，公转时间，自转时间，半径以及纹理
  功能：为星系添加虫洞
- void addMoon(int planetIndex, float distanceFromPlanet, float orbitTime, float rotationTime, float radius, GLuint textureHandle)
  输入：行星下标，离行星距离，公转时间，自转时间，半径和纹理
  功能：为指定下标的行星添加卫星
- void render()
  功能：绘制太阳系
- void renderOrbits()
  功能：绘制所有轨道
- void getPlanetPosition(int index, float* vec)
  输入：行星下标，准备好的容器
  功能：获得指定行星的位置
- float getRadiusOfPlanet(int index)
  输入：行星下标
  功能：获得行星半径
- float testDistancewithPlanet(Camera camera)
  输入：一个镜头实例
  功能：获得镜头与所有星球的最小位置
- float testDistancewithWormhole(Camera camera)
  输入：一个镜头实例
  功能：获得镜头与所有虫洞的最小位置
- bool hasPlanet(unsigned char index)
  输入：一个行星下标
  功能：判断这个太阳系中是否有这个行星

### 3.2.7 Camera 类

Camera 类负责对程序镜头的控制和截屏。

调用方法：

Camera camera;

库依赖：

- Windows.h

- glut.h
- cmath

类成员：

- private:
  - float forwardVec[3]
    功能：记录飞船正对方向的向量
  - float rightVec[3]
    功能：记录飞船右侧方向的向量
  - float upVec[3]
    功能：记录飞船上侧方向的向量
  - float position[3]
    功能：记录镜头的位置
  - float cameraSpeed
    功能：记录镜头前后移动的速度
  - float cameraTurnSpeed
    功能：记录镜头左右移动的速度
  - float mouseUpDown
    功能：记录鼠标上下移动对镜头变化的速度
  - float mouseLeftRight
    功能：记录鼠标左右移动对镜头变化的速度
- public:
  - Camera(void)
    功能：初始化镜头参数
  - void reset(void)
    功能：重置镜头参数
  - void transformOrientation(void)
    功能：通过鼠标位置获得视场旋转变换参数并改变此时镜头关注点
  - void transformTranslation(void)
    功能：通过平移变换模仿相机位置的改变
  - void pointAt(float* targetVec)
    输入：目标方向向量
    功能：将镜头对准给定的向量方向
  - void speedUp(void)
    功能：增加相机移动速度
  - void slowDown(void)
    功能：降低相机移动速度
  - void forward(void)
    功能：相机前移
  - void backward(void)
    功能：相机后移
  - void left(void)
    功能：视场左移
  - void right(void)
    功能：视场右移

- void yawLeft(void)
  功能：视场左转
- void yawRight(void)
  功能：视场右转
- void setMouse(float x, float y)
  输入：鼠标两个位置参数
  功能：计算鼠标控制视场两个参数
- void transformWithMouse(float theta, float phi, float *forward, float *up, float *right)
  输入：鼠标控制视场的两个参数以及三个方向向量
  功能：依照参数计算并更改镜头方向
- void saveImage(void)
  功能：截图并将其保存在当前目录下

## 3.3 main 主程序控制

main 主程序按照如下逻辑控制项目的运行：

1. 程序开始运行时，首先初始化系统，包括初始化全局光照明，载入贴图，为时间参数设置初始值和初始化镜头控制器。
2. 场景绘制，包括对飞船场景的绘制，背景的绘制，场景的切换效果。在这一步中，程序实现了摄像机镜头的控制，碰撞检测，死亡检测和虫洞的转场效果。
3. 为程序设置鼠标和键盘的监听。程序将会按照已经写好的键盘和鼠标控制模式与用户交互，并根据用户的键鼠输入调整程序的执行逻辑，同时函数通过通知全局变量指示场景绘制函数的行为。

# 4 项目难点及解决方案

## 4.1 程序截图

由于程序的复杂性，截图已经不可能通过既有 API 来实现，需要我们自行动手完成。为此，我们首先读取 OpenGL 窗口中的信息，并试图写入到一个 bmp 文件中。

首先使用 glGetIntegerv 函数获取窗口大小，

```
glGetIntegerv(GL_VIEWPORT, viewport);
int width = viewport[2];

int height = viewport[3];。
```

之后使用 glReadPixels 函数读取 OpenGL 绘制的信息，放入一个 temp 数组（width

* height * 4 的 GLbyte 数组）中，由于 bmp 是 BRG 方式存储像素信息的，所以这里也使用 BRG 方式读取：

```
glReadPixels(0, 0, width, height, GL_BGR_EXT, GL_UNSIGNED_BYTE, temp);。
```

bmp 文件分为三部分，首先是文件头

```
BITMAPFILEHEADER Header;
memset(&Header, 0, sizeof(BITMAPFILEHEADER));
Header.bfType = 0x4D42;
Header.bfReserved1 = 0;
Header.bfReserved2 = 0;
Header.bfOffBits = (DWORD)(sizeof(BITMAPFILEHEADER)+sizeof(BITMAPINFOHEADER));
Header.bfSize = (DWORD)(sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER) + 3 * width * height);。
```

bfType 表明这个文件是一个 bmp 文件，固定为 0x4D42；bfReserved1 和 bfReserved2 为保留值 0，bfOffBits 表示文件头的大小，bfSize 表示文件的大小。

第二部分为信息头

```
BITMAPINFOHEADER HeaderInfo;
memset(&HeaderInfo, 0, sizeof(BITMAPINFOHEADER));
HeaderInfo.biSize = (DWORD)sizeof(BITMAPINFOHEADER);
HeaderInfo.biWidth = width;
HeaderInfo.biHeight = height;
HeaderInfo.biPlanes = 1;
HeaderInfo.biBitCount = 24;
HeaderInfo.biCompression = 0;
HeaderInfo.biSizeImage = 3 * width * height;
HeaderInfo.biXPelsPerMeter = 0;
HeaderInfo.biYPelsPerMeter = 0;
HeaderInfo.biClrUsed = 0;
HeaderInfo.biClrImportant = 0;
```

biSize 为信息头的大小，biWidth，biHeight 表示图片的长宽，biPlanes 默认为 1，biCompressionbia 值为 24 表示 24 位真彩色 bmp 文件，biSizeImage 表示数据文件大小，其他值在本文件中默认为 0。第三部分就是刚刚获取的数据。最后将他

们写入文件中，文件名用当前的系统时间，

```
   SYSTEMTIME sys;
   GetLocalTime(&sys);
   char filename[64];
   sprintf(filename,
"Snapshot_%4d%02d%02d_%02d%02d%02d.bmp", sys.wYear,
sys.wMonth, sys.wDay, sys.wHour, sys.wMinute,
sys.wSecond);

   FILE *pfile = fopen(filename, "wb+");
   fwrite(&Header, 1, sizeof(BITMAPFILEHEADER), pfile);
   fwrite(&HeaderInfo, 1, sizeof(BITMAPINFOHEADER),
pfile);
   fwrite(pdata, 1, 3 * width * height, pfile);
   fclose(pfile);
```

最后我们在程序中按下 p 键，调用 Camera.saveImage 函数即可实现截屏。

## 4.2 纹理映射

在本项目中，贴图量大且各类贴图繁杂，可以说是项目中的最大难点之一。为此，我们在经过大量调研和实践后打算采用 tga 格式为模型贴图。

tga 文件拥有 bmp 图片的质量同时又具备较小的体积，且支持 alpha 通道的添加。这给我们的窗户效果的设计提供了便利。关于 tga 文件格式的内容，请参看参考文献[8]。

在完成 tga.h 和 tga.cpp 的编写过后，我们即可通过它来进行贴图的设置。

对于图片纹理的载入，我们首先创建新的 TGA 对象，并将文件名作为参数带入构造函数中。此时我们即获得了处理文件的像素信息，然后我们将文件信息与纹理绑定便可在后续的过程中使用该纹理。

在需要使用纹理的时候我们首先调用，

```
   glBindTexture(GL_TEXTURE_2D,name->getTextureHandle());
```

然后调用 TGA 内部的

```
   tga.getTextureHandle();
```

以获得纹理序号。

对于行星纹理的贴图，我们直接使用了系统的贴图方式。但是需要注意的是，我们必须首先将图片进行对称化操作，这样一来图片映射到球体后才不会出现明显的分界。

关于初始太阳系和进入虫洞后被送到的随机太阳系，我们采用如下的处理思路。对于程序开始时所处的太阳系，拥有一套不变的纹理，包括太阳，九大行星，

月球。当进入虫洞以后所进入的星系，所有纹理都是随机从另外 3 种恒星纹理，以及 11 种行星纹理中获取。而虫洞纹理保持恒定不变的。

对于整个星系的背景绘制方法，则是通过

```
drawCube();
```

绘制一个正方体，然后贴上星空的贴图，最后将整个星系置于正方体内来构建的。

最后，由于图片透明度使用结果不理想，对于飞船的视图部分，我们采用了将图片通过拆分，用若干个不规则的四边形拼接组成的方法，来实现窗口，控制台，警报显示的效果。并且，考虑到飞船纹理需要不受到视角变化的影响，我们将其写在所有视角矩阵变化之外，单独置于绘制函数中。

## 4.3 碰撞检测与虫洞效果

由于所有星体都是一个正球体，碰撞检测在实现起来难度稍微有一些降低。为此，我们考虑检测飞船与所有星体的距离。由于星系中存在多个星球，而飞船只存在一架，故将该函数

```
testDistancewithPlanets();
```

设置为星系类 SolarSystem 的成员函数。距离监测的方法很简单，即实时监测各个星球与飞船的三维坐标，计算出飞船（即 camera）与最近的星球表面的距离。若距离较近，则切换飞船纹理，以示警报。若距离过近，则判定飞船坠毁在星球上，程序暂停并输出指示飞船撞毁的信息与图像。

另外，本程序所实现的模拟虫洞跳转星系的功能也是一个难点。为了实现虫洞效果，程序会首先新建一个虫洞类 Wormhole，并在星系类 SolarSystem 中添加虫洞对象 wormholes。同时，在星系类中添加函数

```
testDistancewithWormholes();
```

以监测飞船是否进入虫洞，其中该函数与上文所述的星球距离检测函数相似。

为了实现星系的跳转，程序没有将显示的星系作为一个全局常量对象，而是使用一个指针

```
SolarSystem *Galaxy;
```

去指向不同的星系对象，以便于星系之间的转换。每当进入虫洞要跳转到一个新的星系时，程序便会创建一个新的星系对象。星系对象有不同的构造函数，会随机生成出不同数量，旋转半径和大小的星球，因此我们便可以构造不一样的随机星系。这样一来，存储功能保存的 dat 文件就只需要储存当前星系对象和镜头对象便可以在之后将其恢复。当然，为了完成这个功能，之前的星系对象不能释放空间，否则内存中就会找不到该对象，以至于程序报错。

# 5 程序使用方法

运行该程序后，用户将会首先以第一视角看到在茫茫宇宙星空中一艘由自己驾驶的飞船。若移动鼠标，用户即可调整观察视角。使用键盘上的 WSAD 按键，用户可分别进行前进、后退、左转和右转操作。用户按下 O 键即可控制轨道绘制；按下 P 键能够将当前场景截图保存在目录下；使用 B 键可以保存当前的程序运行状态；使用 N 键能够将已经保存的状态重新载入；使用-/=键可以控制时间流逝的快慢；使用</>能够控制飞船飞行速度的快慢。

同时注意，当飞船逼近星球时，飞船上端的状态显示将由 Safe 转化为 Danger，以提醒用户危险的到来。若用户不慎与星球碰撞，飞船便会撞毁。此时，用户可按下 t 键重新初始化程序。

此外，还需要注意，在星系中，存在一种特殊的星际物质——虫洞。虫洞本身不发光，且极易吸收光，因此通常难以被眼睛捕捉到。若飞船不慎跌入虫洞，将会被虫洞随机地送往另外一个宇宙空间，只得等待运气来临时再次进入虫洞得以返回。

# 6 致谢

在程序设计过程中，我们得到了诸多人员的无私帮助。我们尤其感谢张宏鑫老师对项目成员的亲切指导；感谢吴尚阳助教对 OpenGL 函数框架的解析；感谢室友们对项目组成员工作的理解与支持。此外，我们还要感谢诸多无私的网友，他们或者提供了部分技术支持，或者撰写博客解释了 OpenGL 的应用，或者属于专业技术开发人员，或者在 Google Group 中与大家热烈地进行了讨论。

我们从这些交流中受益匪浅。没有他们的帮助，本项目必然不可能在规定的时间内完成。为此，我们在此对他们的帮助表示衷心的感谢！

# 7 参考资料

[1] NeHe，OpenGL 教程
http://nehe.gamedev.net/
[2] 慢步前行，实验 7 OpenGL 光照
http://www.cnblogs.com/opengl/archive/2012/11/14/2770745.html
[3] Ryan Pridgeon，使用 OpenGL 与 C++编写太阳系
http://ryanpridgeon.net/?cat=5
[4] 08lovely 计算机一班，OpenGL 基本概念入门——纹理贴图
http://blog.sina.com.cn/s/blog_62dfaf550100g2l1.html
[5] 江南烟雨，游戏编程常用 TGA 图像格式详解以及加载纹理编程实现
http://blog.csdn.net/xiajun07061225/article/details/7646058
[6] 从零开始，OpenGL 里关于鼠标响应的函数
http://www.cppblog.com/wicbnu/archive/2010/11/08/132984.html

[7] plusplus7，OpenGL 编程学习实战教程——实现截图功能
http://www.linuxidc.com/Linux/2013-02/78959p10.htm
[8] 宁波搬家，TGA 文件内部格式及程序实现
http://blog.sina.com.cn/s/blog_5ff6097b0100xtvw.html