一、设计思路

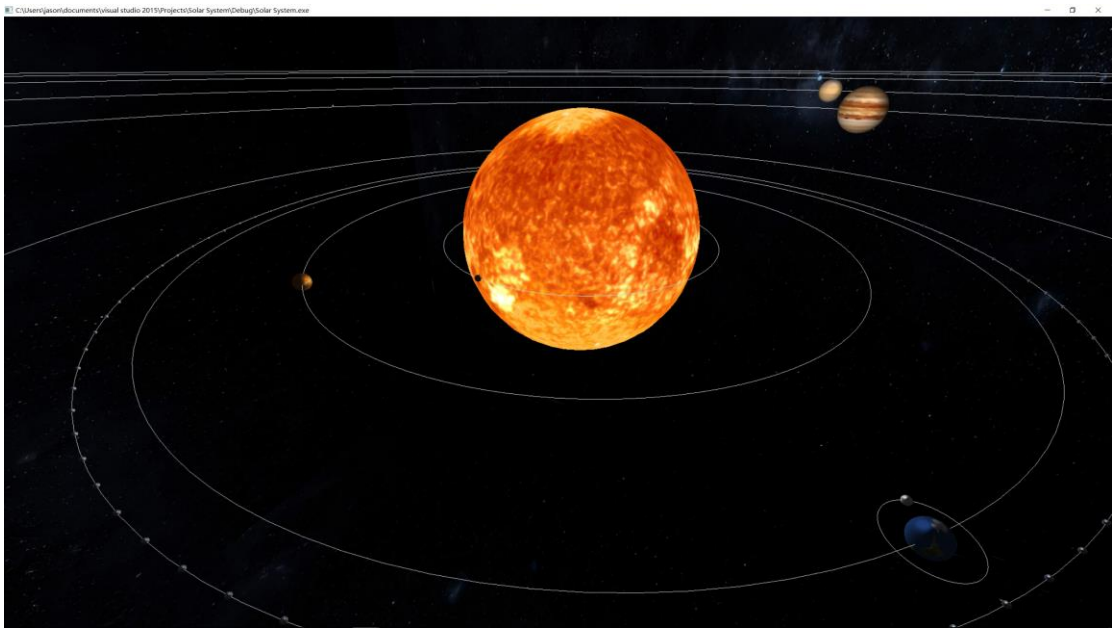
本次计算机图形学大作业的要求是设计一个交互控制的动态粒子系统。粒子系统是指计算机图形学中模拟特定现象的技术，它在模仿自然现象、物理现象及空间扭曲上具备得天独厚的优势，为我们实现一些真实自然而又带有随机性的特效（如爆炸、烟花、水流）提供了方便选择。

刚开始还没有想好具体的设计思路，直到有一天看电影时看到了环球影城 Universal Studio 的开场画面才确定此次计算机图形学大作业的题目为“太阳系”

项目起步时先做了需求分析，太阳系模型整体来说不难，既然是基于面向对象编程，先找对象，太阳系中的八大行星和轨道，所以对象有两个。处理这样一个主题，我们其实并不需要很多的物理知识，被推进的物质并不会以极高的速度运动，且重力条件不会因此产生很大的改变，所以在这边牛顿万有引力定律就够用了。重点是$\vec{F} = d\vec{p}/dt$或者更被熟知的 $\vec{F} = m\vec{a}$。显然，万有引力法则也是很重要的：$\vec{F}_{12} = -G\frac{M_1 M_2}{r_{12}^2}\hat{r}_{12}$，因此在物体 i 上作用的力为：

$$\vec{F}_i = \sum_{j=1, i\neq j}^{N} \vec{F}_{ij} = -GM_i \sum_{j=1, i\neq j}^{N} \frac{M_j}{r_{ij}^2}\hat{r}_{ij} = M_i \sum_{j=1, i\neq j}^{N} \vec{g}_j$$

这是我们简单的$m\vec{g}$（或者你可以用，$m\vec{a}$）。加速度不是根据一个单个的物体，而是来自其他所有的物体。所以，不仅仅地球在拖拉你，而是更诗意的来说，整个宇宙都在拖拉你。



二、难点问题

1. 项目刚开始时我其实对粒子系统是没有很深刻的了解的，后来透过网上查询资料得出粒子系统生成画面的基本步骤：
   (1) 产生新的粒子；
   (2) 赋予每一新粒子一定的属性；
   (3) 删去那些已经超过生存期的粒子；
   (4) 根据粒子的动态属性对粒子进行移动和变换；
   (5) 显示由有生命的粒子组成的图像。

   粒子系统采用随机过程来控制粒子的产生数量，确定新产生粒子的一些初始随机属性，如初始运动方向、初始大小、初始颜色、初始透明度、初始形状以及生存期等，并在粒子的运动和生长过程中随机地改变这些属性。粒子系统的随机性使模拟不规则模糊物体变得十分简便。
   粒子系统应用的关键在于如何描述粒子的运动轨迹，也就是构造粒子的运动函数。函数选择的恰当与否，决定效果的逼真程度。其次，坐标系的选定（即视角）也有一定的关系，视角不同，看到的效果自然不一样了。

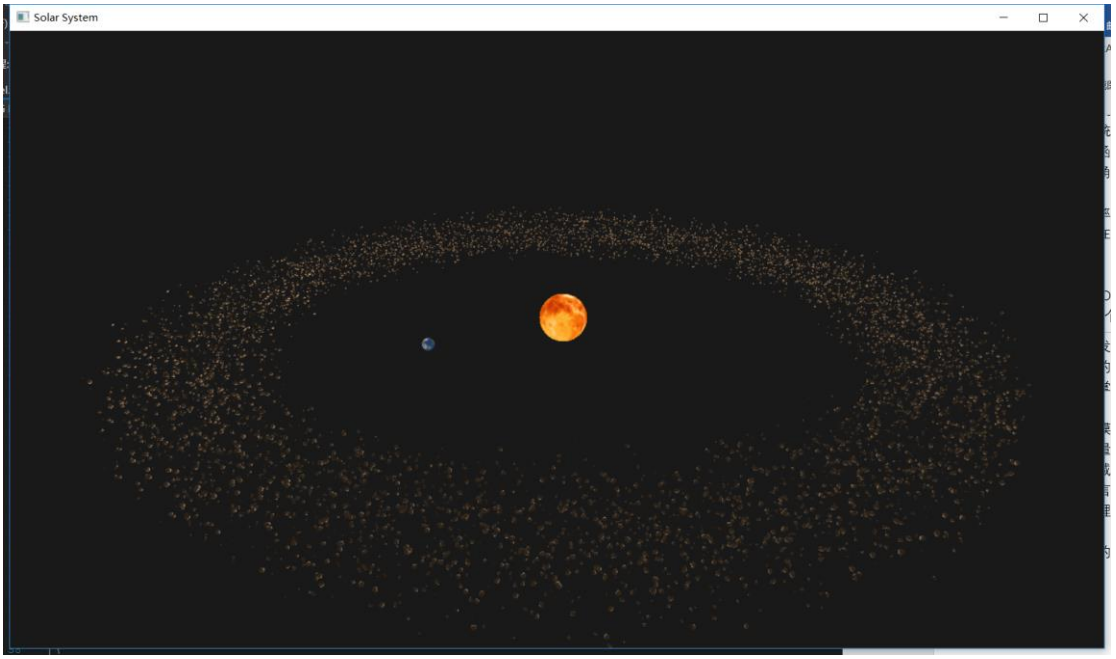
2. 渲染效率非常低。这次大作业写了两个版本，第一个版本是采用旧版的渲染模式，在太阳系中小行星带的数量达到 700 时就已经帧数略低非常不流畅了。

   早期的 OpenGL 使用立即渲染模式（Immediate mode，也就是固定渲染管线），这个模式下绘制图形很方便。OpenGL 的大多数功能都被库隐藏起

来，开发者很少能控制 OpenGL 如何进行计算的自由。而开发者迫切希望能有更多的灵活性。随着时间推移，规范越来越灵活，开发者对绘图细节有了更多的掌控。立即渲染模式确实容易使用和理解，但是效率太低。

原来的模式下，每一次模型绘制都需要与 GPU 进行至少一次的通信，这在绘制大量粒子系统时会带来很高的性能开销。核心管线可以使用更加底层的技术，减少 CPU 与 GPU 之间的通讯，并提供更高的设计灵活性。对于粒子系统而言用核心管线，一次性将所有粒子的信息从 CPU 传送到 GPU，类似于批处理思想，能够极大的提高性能。

优化后的粒子系统在数量达到一万时仍能够保持高帧率



3. 环境的配置
   在第一版本的太阳系当中只用了 GLUT 这个库开发，计算机图形学课程第一次作业：Bouncing ball 也是用的这个库，所以开发时并没有遇到问题。然而，凡事都有个然而，为了增加渲染效率再开发第二版本时就不仅仅只用到了 GLUT 这个库，光是将环境配置好能顺利编译不报错就至少花了一周的时间调试。在开发完第二版本后，却发现第一版本已经不再兼容系统了…怎么编译都会报错…因为处于期末周这个问题暂时没有时间解决了

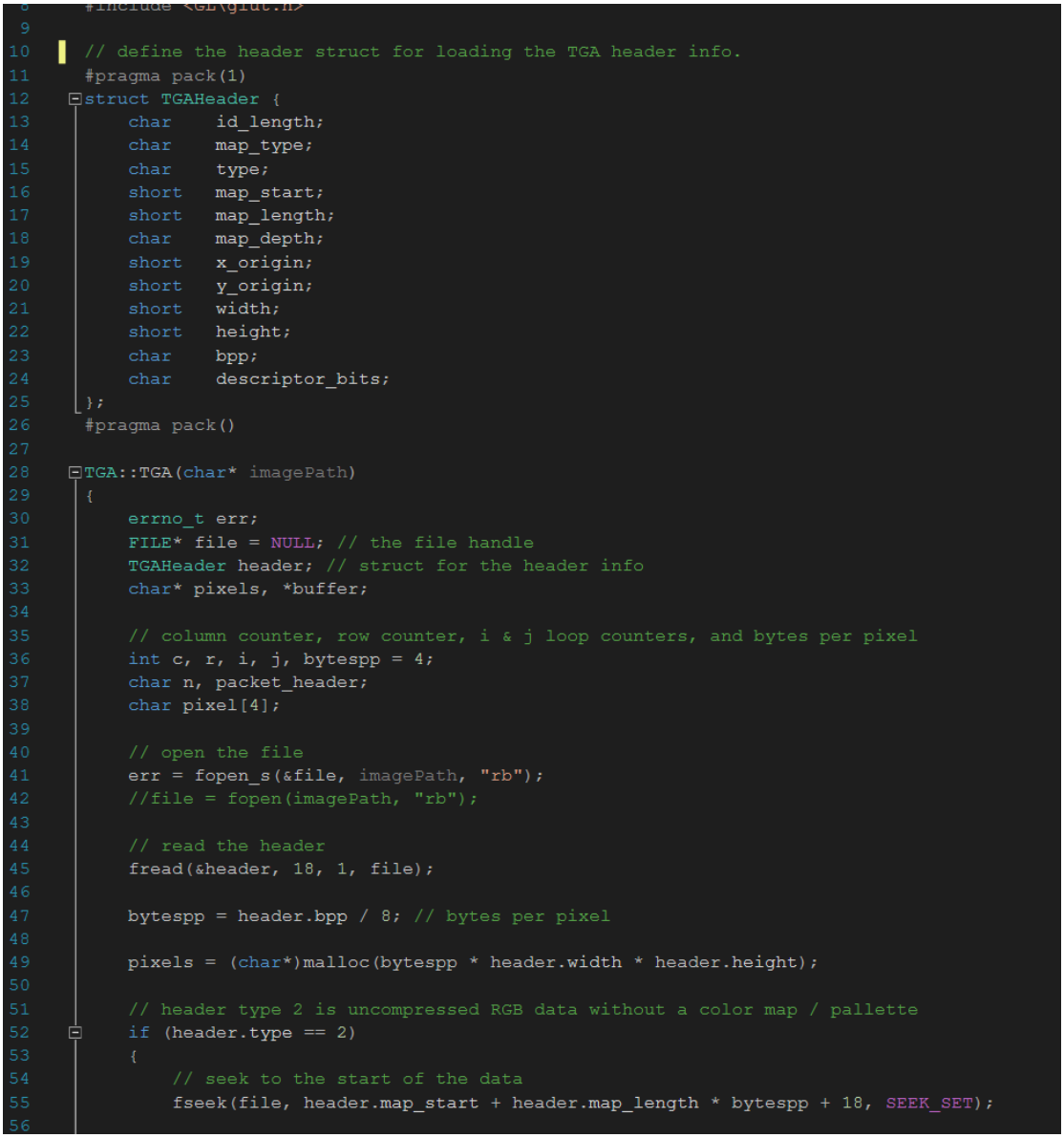
   我相信在未来的学习和实践中能够运用到更高级的库和工具。

三、操作说明

1. 移动视角：本项目可运用鼠标搭配 W 向前移动、S 向后移动、A 向左移动、D 向右移动做到视角的移动
2. 退出：Esc 退出程序

四、技术实现

1. 纹理映射：读取 TGA 文件，根据物体大小创建纹理， 在绘画物体过程中绑定纹理，并在物体上显示出来。 纹理有很多参数需要设置，包括滤波 filter,边界填充 wrap，这两个参数我运用在了天空盒的实现中。一个完全纹理的四个顶点的坐标分别是(0.0f,0.0f)、(0.0f,1.0f)、(1.0f,1.0f)、(1.0f,0.0f)，分别对应 左下、左上、右上、右下角。映射纹理时需要注意这些点的选择，否则会出现前后左右颠倒。



```
#include <GL\glut.h>

// define the header struct for loading the TGA header info.
#pragma pack(1)
struct TGAHeader {
    char    id_length;
    char    map_type;
    char    type;
    short   map_start;
    short   map_length;
    char    map_depth;
    short   x_origin;
    short   y_origin;
    short   width;
    short   height;
    char    bpp;
    char    descriptor_bits;
};
#pragma pack()

TGA::TGA(char* imagePath)
{
    errno_t err;
    FILE* file = NULL; // the file handle
    TGAHeader header; // struct for the header info
    char* pixels, *buffer;

    // column counter, row counter, i & j loop counters, and bytes per pixel
    int c, r, i, j, bytespp = 4;
    char n, packet_header;
    char pixel[4];

    // open the file
    err = fopen_s(&file, imagePath, "rb");
    //file = fopen(imagePath, "rb");

    // read the header
    fread(&header, 18, 1, file);

    bytespp = header.bpp / 8; // bytes per pixel

    pixels = (char*)malloc(bytespp * header.width * header.height);

    // header type 2 is uncompressed RGB data without a color map / pallette
    if (header.type == 2)
    {
        // seek to the start of the data
        fseek(file, header.map_start + header.map_length * bytespp + 18, SEEK_SET);

```

2. 模型加载：使用了一个非常流行的模型导入库是 Assimp。Assimp 能够导入很多种不同的模型文件格式，它会将所有的模型数据加载至 Assimp 的通用数据结构中。当 Assimp 加载完模型之后，我们就能够从 Assimp 的数据结构中提取我们所需的所有数据了。由于 Assimp 的数据结构保持不变，不论导入的是什么种类的文件格式，它都能够将我们从这些不同的文件格式中抽象出来，用同一种方式访问我们需要的数据。

```
private:
    /*  Functions   */
    // loads a model with supported ASSIMP extensions from file and stores the resulting meshes in the meshes vector.
    void loadModel(string const &path)
    {
        // read file via ASSIMP
        Assimp::Importer importer;
        const aiScene* scene = importer.ReadFile(path, aiProcess_Triangulate | aiProcess_FlipUVs | aiProcess_CalcTangentSpace);
        // check for errors
        if (!scene || scene->mFlags & AI_SCENE_FLAGS_INCOMPLETE || !scene->mRootNode) // if is Not Zero
        {
            cout << "ERROR::ASSIMP:: " << importer.GetErrorString() << endl;
            return;
        }
        // retrieve the directory path of the filepath
        directory = path.substr(0, path.find_last_of('/'));

        // process ASSIMP's root node recursively
        processNode(scene->mRootNode, scene);
    }

    // processes a node in a recursive fashion. Processes each individual mesh located at the node and repeats this process on its children nodes (if any).
    void processNode(aiNode *node, const aiScene *scene)
    {
        // process each mesh located at the current node
        for (unsigned int i = 0; i < node->mNumMeshes; i++)
        {
            // the node object only contains indices to index the actual objects in the scene.
            // the scene contains all the data, node is just to keep stuff organized (like relations between nodes).
            aiMesh* mesh = scene->mMeshes[node->mMeshes[i]];
            meshes.push_back(processMesh(mesh, scene));
        }
        // after we've processed all of the meshes (if any) we then recursively process each of the children nodes
        for (unsigned int i = 0; i < node->mNumChildren; i++)
        {
            processNode(node->mChildren[i], scene);
        }

    }
```

3. 光照实现：这里用以太阳为中心定义了一个简单的太阳光，用 glLightfv 设置了位置、光照颜色、漫反射和镜面反射等参数，设计的比较简单。

```
// set up lighting
glEnable(GL_LIGHTING);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
GLfloat matSpecular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat matAmbience[] = { 0.3, 0.3, 0.3, 1.0 };
GLfloat matShininess[] = { 20.0 };
glClearColor(0.0, 0.0, 0.0, 0.0);
glShadeModel(GL_SMOOTH);

glMaterialfv(GL_FRONT, GL_SPECULAR, matSpecular);
glMaterialfv(GL_FRONT, GL_SHININESS, matShininess);
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbience);

GLfloat lightAmbient[] = { 0.3, 0.3, 0.3, 1.0 };
GLfloat lightDiffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat lightSpecular[] = { 1.0, 1.0, 1.0, 1.0 };

glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmbient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDiffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, lightSpecular);

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glDisable(GL_LIGHTING);
```

五、小结

总体来说，本次大作业是非常辛苦的，但也很开心，让我学到了许多图形编程相关的知识，虽然因为在外实习没法每一堂课都出席，但也让我明白了来时上课的内容提及的基础内容是非常有用的。学习过程中也遇到了像因为版本关系而需要 debug 非常久的问题，再加上老师课上提及的渲染模式没有 shader 也没有主循环效率较低，这部分需要自行上网查询相关资料。不仅如此，老师和助教提供的库比较老，最新的 opengl 有很多更高效、更有用的库供我们使用。大学三年来说，这门课应该是最有趣也是成就感最大的一门课，相信在了解熟悉掌握这些库的使用后，图形学编程会变得迎刃有余，最后，感谢老师和助教的辛勤教导！