计算机图形学课程项目报告

项目名称 基于光线追踪技术的立体恒星系统

组　　长 冯舜

组　　员 孟令琛 刘毅韬 李福平 高坤阳 欧阳小伟

指导老师 赵君峤

一、项目题目

基于光线追踪技术的立体恒星系统（Star system rendered with raytracing technique）

二、小组成员

冯舜（组长） 孟令琛 刘毅韬 李福平 高坤阳 欧阳小伟

三、项目摘要

用光线追踪技术实现一个立体的恒星系统。

四、项目背景

光线追踪（Ray Tracing）算法是一种计算机三维图形渲染算法，其基本出发点就是追踪光线，模拟真实的光路和成像过程。相比于其他大部分渲染算法，光线追踪算法的优势是可以提供更为真实的光影效果，劣势是计算量巨大。

我们作为初学者决定挑战光线追踪技术。一些天文模拟器给予了我们灵感，此外，考虑到球的模型较为容易建立，我们决定尝试运用光线追踪技术尝试建立一个立体的恒星系统。

五、项目目的

1、 基本目标：用光线追踪技术实现一个立体的恒星系统，每个行星有不同的材质，并且有复杂的光照效果。
2、 子目标：
    1) 实时渲染，加入自转和公转运动。
    2) 用户自定义相机参数，从不同角度观察恒星系。

六、项目设定

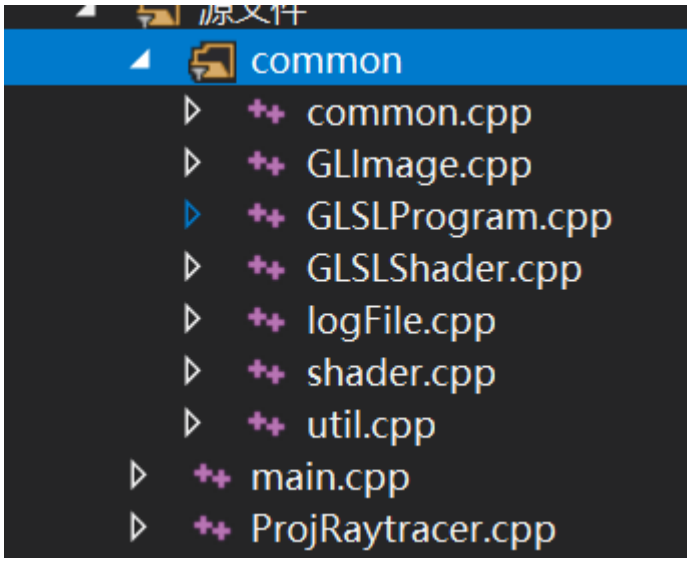
1、 项目计划使用太阳系的行星贴图；
2、 假设所有行星都是光滑的球型，只有反射，没有折射，忽略光线追踪中的折射；
3、 假设行星的自转和公转方向相同，角速度方向为z轴方向。

七、相关技术

1、 主要使用了光线追踪的技术：和三角形渲染方式不同，光线追踪对每个像素进行计算，根据光的反射算出每个像素的颜色来源，求出对应点的颜色。
2、 计算对应点的颜色需要使用纹理贴图：由于不是使用三角形方式渲染，光线追踪计算出球面上对应点的坐标，通过对应点的球面坐标转换成平面坐标，找到图片上对应像素的颜色。
3、 使用GLSL计算点的颜色，GLSL的代码在GPU上运行，可以很好地做到并行计算。

八、项目内容

主要实现raytracing文件为ProjRaytracer.cpp和ProjRaytracer.comp。
cpp文件包括main, ProjRaytracer和一些工具类的文件。工程当中还有对应的.h文件和需要用到的贴图文件。



九、项目实现

具体的实现流程:

1、 找到光线的路径

项目当中实现了逆向的光线追踪，即找到最终照到显示屏上某个像素的光是什么颜色的。那么可以假设有一束光从某个像素发出，沿着观察的路径找到碰到物体的颜色。

计算光线碰到的物体通过计算几何方法，求射线和球面(行星)/平面(地板)的交点。

球的半径定义如下:

```
// 行星球标准半径
const float shrunk = 2.5;
//float sphereRadius[8] = {3.4*shrunk, 6.0*shrunk, 6.3*shrunk, 3.3*shrunk,15.4*shrunk,60.2*shrunk,25.5*shrunk,24.7*shrunk};
float sphereRadius[8] = {3.4*shrunk, 6.0*shrunk, 6.3*shrunk, 3.3*shrunk,15.4*shrunk, 860,25.5*shrunk,24.7*shrunk};
```

运动轨迹通过起始点、经过时间和公转周期决定。

光线和所有球面求交点，使用直线的参数方程计算。直线的参数方程从观测点出发，能看到的点参数t为正，否则t为负。计算方法如下:

假设参数方程为$\mathbf{p} = t\mathbf{d} + \mathbf{p_0}$，其中t是参数，$\mathbf{d}$是观测方向(归一化)，$\mathbf{p_0}$是观测点;

假设球的半径为r，球心为$\mathbf{O}$;

可以通过几何算出:

$$t = b \pm \sqrt{r^2 - a^2 + b^2}$$

其中

$$a = \|\boldsymbol{O} - \boldsymbol{p_0}\|, b = (\boldsymbol{O} - \boldsymbol{p_0}) \cdot \boldsymbol{d}$$

在所有平面和球面的交点当中，取一个最接近观测点的t，找到对应的球和对应的坐标。

2、根据行星贴图找到颜色

算出空间当中交点后，还要求出球面上这个点本身的颜色。需要转化成平面坐标，求出颜色。

假设交点为$\mathbf{p}$，圆心为$\mathbf{O}$，半径为r;

假设经过的时间为time，自转角速度为v;

映射到平面上的方法:

Y坐标：直接线性映射：

$$y = \left(\frac{(\boldsymbol{p}-\boldsymbol{O})\cdot(0,0,1)}{r}+1\right)/4$$

X坐标：

$$x = \left(\frac{\arctan\big((\boldsymbol{p}-\boldsymbol{O})\cdot(0,1,0)-(\boldsymbol{p}-\boldsymbol{O})\cdot(1,0,0)\big)}{\pi}+1}{2}+\text{time}*\text{v}\right)\text{的小数部分}$$

求出的坐标需要按照一个比例映射到二维的图像上，找到对应的像素。

3、光线反射

假设光线和球的交点为$\boldsymbol{p}$，观测点为$\boldsymbol{p_0}$，

可以求出光线的反射方向：

显然反射光线的观测点为$\boldsymbol{p}$；

反射光线的方向：

$$\boldsymbol{d'} = \frac{(\boldsymbol{p}-\boldsymbol{O})}{\|\boldsymbol{p}-\boldsymbol{O}\|}\times\left((\boldsymbol{p}-\boldsymbol{O})\cdot\frac{\boldsymbol{p}-\boldsymbol{p_0}}{\|\boldsymbol{p}-\boldsymbol{p_0}\|}\right)-\frac{\boldsymbol{p}-\boldsymbol{p_0}}{\|\boldsymbol{p}-\boldsymbol{p_0}\|}$$

4、计算颜色

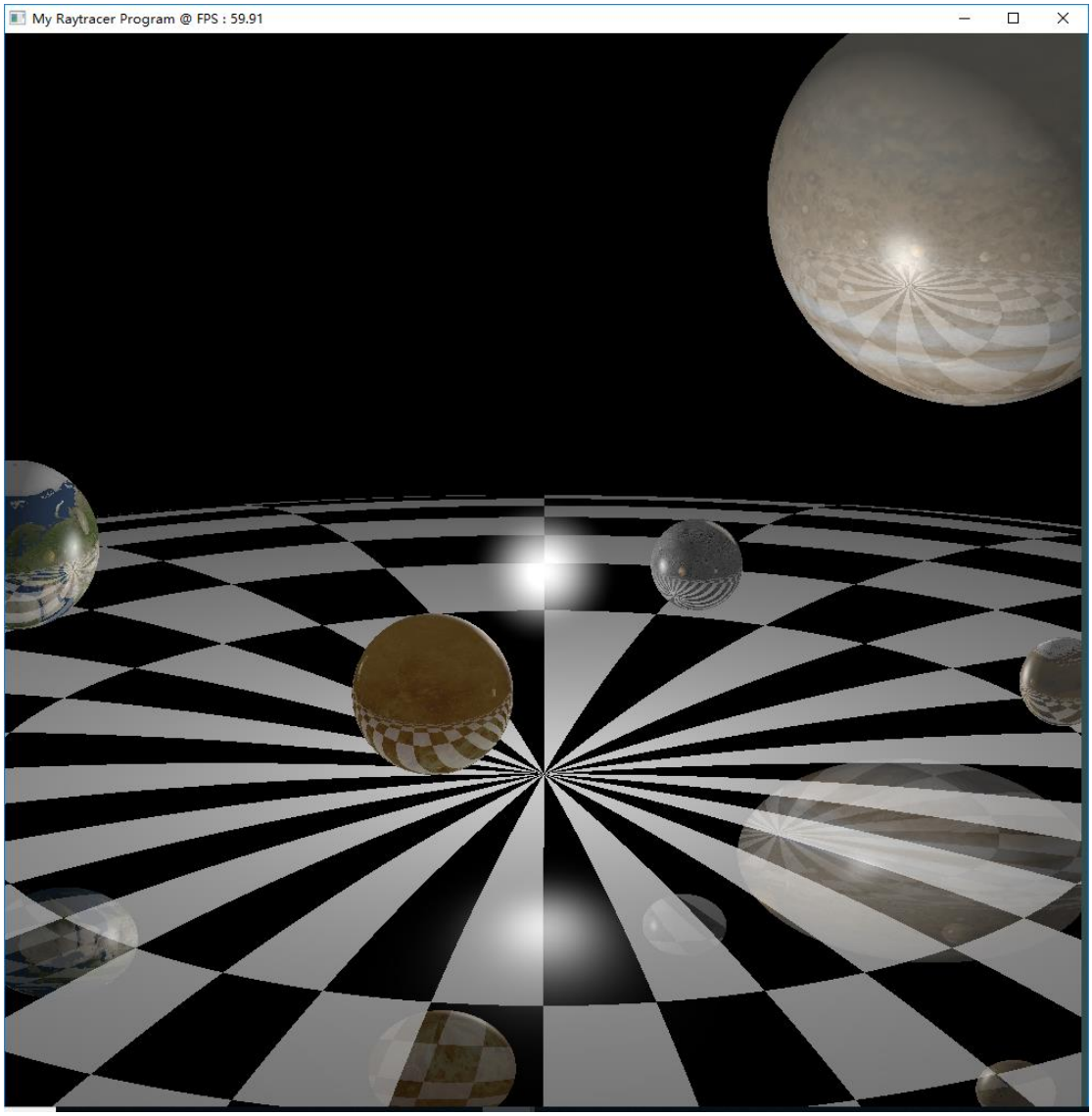
追踪到某一个点$\boldsymbol{p}$，返回的颜色是两个部分的和：

球本身的颜色 + 以$\boldsymbol{p}$为观点，$\boldsymbol{d'}$为追踪方向，再进行一次光线追踪观察到的颜色×一个反射系数。

以上所有的运算全部以rgb的方式计算，三种颜色每一种分别处理。

十、项目成果

FPS在GTX-1070TI上可以达到设置的最大值60，在性能较差的GPU上可以通过缩小分辨率达到流畅运行。



十一、小组分工

李福平、孟令琛：挑选算法并理解、解释，Compute Shader代码等待初步实现；
欧阳小伟、刘毅韬：写Compute Shader代码；
刘毅韬、高坤阳：C++项目的维护、修改；
高坤阳：项目文档撰写；
冯舜：安排工作，机动分工

参考文献

[1] opengl-tutorial [DB/OL].
http://www.opengl-tutorial.org
[2] learnopengl-cn [DB/OL].
https://learnopengl-cn.github.io/
[3] Introduction to Ray Tracing: a Simple Method for Creating 3D Images [DB/OL]
http://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing/ray-tracing-practical-example
[4] Ray tracing with OpenGL Compute Shaders [EB/OL].
https://github.com/LWJGL/lwjgl3-wiki/wiki/2.6.1.-Ray-tracing-with-OpenGL-Compute-Shaders-%28Part-I%29
[5] 3D C/C++ tutorials - Ray tracing [DB/OL].
http://www.3dcpptutorials.sk/index.php?id=16
[6] It's More Fun to Compute: An Introduction to Compute Shaders [DB/OL].
http://antongerdelan.net/opengl/compute.html