Nova Documentation

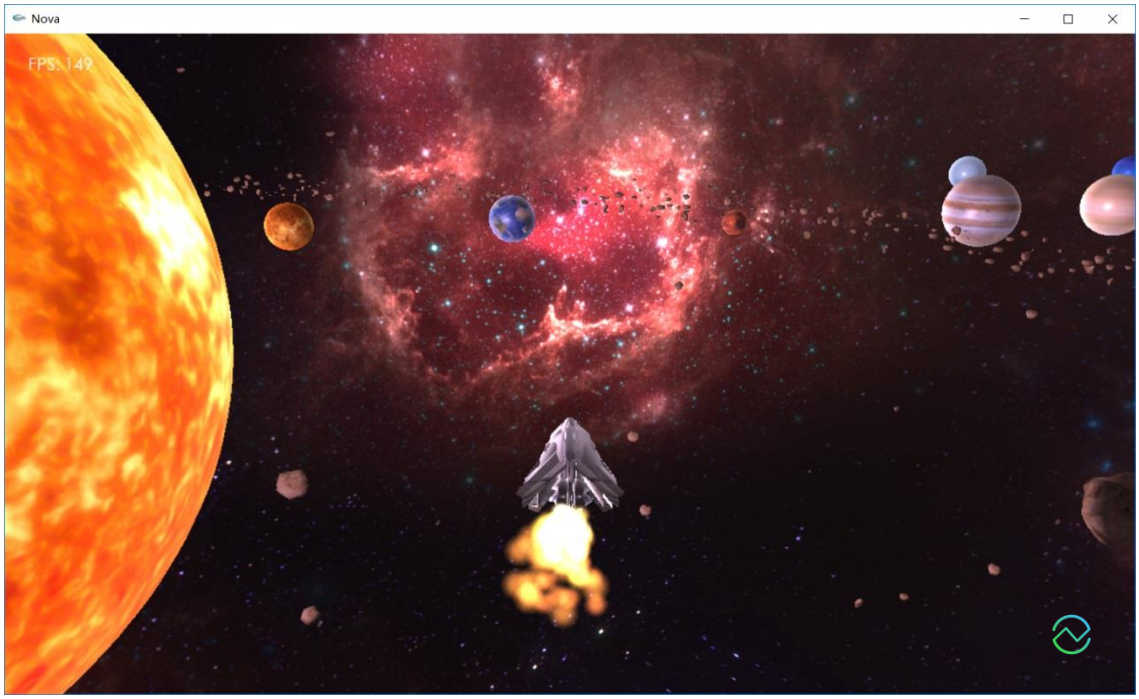
CG Final: A solar system made with OpenGL

作者：陈文韬 学号：515030910323

一、基本介绍

Nova 是一个基于 OpenGL 的太阳系模拟系统。场景中内容主要有：宇宙天空盒，太阳(光源)及八大行星，小行星带(粒子系统)，空中飘散的陨石(粒子系统)，飞船及飞船的引擎火焰(粒子系统)。

软件整体使用 OpenGL 的核心管线模式，使用 VBO, shader，实例化等技术以获得更极致的性能。使用 Assimp 库进行模型读取，Stb_image 进行纹理图片读取，Freetype 进行文字的渲染显示。



场景复杂度

星球 4000 个顶点，陨石 100 个顶点，飞船 4000 个顶点，火焰粒子 4 个顶点

场景在 30000 颗陨石共计 300 万顶点数且在进行公转自转等运动时，可以保持 FPS 在 40 左右。减少模型运动后可以流畅运行 500 万顶点数场景。

正式场景中为获得更真实的效果，仅投入了 1400 颗陨石。

基本操作

W 键向前移动，S 键向移动，A 键向左移动，D 键向右移动。
Q 键向上移动，E 键向下移动。
1 键可以切换显示网格模型或面模型。
2 键可以切换地球的着色器（普通着色器和优化后的着色器）。
3 键可以切换粒子模型。
Z 键提升太阳亮度，X 键减少太阳亮度。

二、程序结构设计

1. main
   调用 `glut` 库进行显示窗口的基本设置，调用 `Application` 类进行图像的更新及绘制，收集用户的键盘及鼠标输入，并转发到 `Application` 类进行处理。

2. Application
   初始化时调用 `ResourceManager` 进行资源（模型、着色器、纹理）的读取，调用 `EntityManager` 进行 3D 场景中物体（星球、飞船）的创建，调用 `ParticalManager` 进行粒子系统（陨石，火焰）的创建，调用 `Skybox` 绘制宇宙场景天空盒。每一帧对所有物体进行渲染，处理用户输入并转发到 `Camera`, `Entity` 等并更新所有物体的位置、角度等物理信息。

3. ResourceManager:
   使用单例模式构造的资源管理器。初始化时输入文件位置，读取并存储模型、着色器、纹理等资源，供其他模块调用获取。同时也含有一个硬编码正方形模型，用于火焰粒子贴图。

4. EntityManager & Entity:
   创建并管理大量的 `Entity` 类，每帧进行绘制及信息更新。每个 `Entity` 存储有位置、角度、旋转轴等物理信息。`Entity` 按模型功能分为三类，一类为普通星球，绕太阳进行公转及自转；一类为月球，绕地球进行自转及公转；一类为飞船，与 `Camera` 一同随用户输入而进行移动，并且在尾部发动机位置产生火焰粒子效果。

5. Light & Camera:
   `Light` 为环境光源，存储有光的颜色，光的位置，光的各项照明（ambient, diffuse, specular）的权重。`Camera` 为镜头，存储有欧拉角，朝向及位置信息，根据用户输入进行移动旋转，实现场景的自由浏览。`Light` 和 `Camera` 通常都会在物体绘制时当做参数传入，并传入着色器进行后续的计算。

6. Config
   `Config` 存储有各个全局的宏定义，包括物体速度，粒子数量及范围设定，光照设定，窗口设定等信息。

7. TextRenderer

用于显示窗口上的 FPS。读取 TFF 字体文件，创建相应的字体纹理，将传入的字符串转换为多个贴有对应纹理的长方形并绘制在屏幕上，实现字体的显示。

8. Shader & glsl

Shader 读取 glsl 着色器源文件，完成编译并输出相关辅助信息，并返回 ID。GLSL 含有自己设定的各类着色器，包含：用于字体渲染的 2D 着色器，天空盒的最深深度着色器，陨石的实例化着色器，火焰的渐变贴图实例化着色器，地球四贴图的特定着色器，各星球的边缘光着色器

9. ParticalManager

负责对陨石粒子系统及火焰粒子系统的管理。通过 `ResourceManager` 获取模型及着色器，将实例化数组存入 VBO 并进行内容划分，每帧对每个粒子进行运动计算，为火焰粒子按照生存时间更新纹理，并写入到 VBO 数据缓存区进行实例化绘制。

三、难点问题

1. **键盘延迟**

如果直接调用 `glutKeyboardFunc()`对键盘输入进行判断处理，会有奇怪的延迟。就像打字的时候长按 A 键，当第一个 A 键出来之后会停顿大概一秒，之后 A 才会连续不断的打出来。这样的效果不能做到每帧输入处理，用于移动处理效果很差。最终改为了系统调用 `GetKeyState()`。

2. **自制模型**

3D 模型有点难收集。好看的要收费，免费的不好看。于是决定自己做一个模型看看。粒子的模型中，有一个模型就是自己用 Cinema 4D 绘制的 Low-poly 风格的太空机器人。如果从 C4D 中直接导出材质颜色生成 OBJ 并放入程序中，会变回普通的圆滑风格。

于是在 Shader 中在 out_color 类型上加入了 flat 关键字，确保每个三角形只会用一种颜色而不会线性插值。此时模型看上去有 Low-poly 风格的雏形，但是颜色不太正确，例如相邻三角面片尽管朝向不同却因为使用了同一个顶点的颜色而有了相同颜色，而且通过几何着色器显示的法线来看，法线并不是垂直于三角面片的，而大概是周围三角面片法向量的平均值，渲染的结果很奇怪。

所以决定从 C4D 中直接导出纹理贴图，以确保颜色的正确性。不幸的是，又发现 obj 格式简单模型可以正确运行，但是复杂模型会在读取时崩溃。fbx, 3ds, x 等格式尽管可以读取并运行起来，但是纹理映射并不正确，纹理会贴歪。

最终，大量的 cout 使我发现了模型读取模块中的 bug。C4D 生成的部分 obj 格式文件会中，顶点可能会不含有法向量，造成了读取时的崩溃。法向量需要自己根据 Indices 进行 `glm::cross()`进行生成。3ds 等格式因为是二进制格式文件，看不到详细的存储内容，所以暂时没有得到解决。

3. **实例化引起屏幕闪烁**

   在场景中添加了实例化粒子系统后，屏幕会间歇性的产生闪烁。以为是粒子的位置、速度等信息处理不当造成，但几经研究并未发现错误。最终解决方案是在更新缓冲数组时将 `glBufferData()`改为 `glBufferSubData()`。

4. **OpenGL 的抗锯齿不生效**

```
glEnable(GL_MULTISAMPLE_ARB);
glEnable(GL_MULTISAMPLE);
```

   调用了多重采样的抗锯齿函数后，画面并没有任何变化。

```
static GLint buf[1], sbuf[1];
glGetIntegerv(GL_SAMPLE_BUFFERS_ARB, buf);
printf("number of sample buffers is %d\n", buf[0]);
glGetIntegerv(GL_SAMPLES_ARB, sbuf);
printf("number of samples is %d\n", sbuf[0]);
```

   运行上述代码，输出抗锯齿缓冲大小，结果显示缓冲大小为 0，问题暂未得到解决。

四、技术实现

1. **粒子系统**

   粒子系统用于显示宇宙中飞行的陨石，小行星带，以及飞船发动机尾部火焰。

   陨石分为两类，一类是由大量陨石绕着太阳进行公转，构成太阳系中的小行星带，初始化时生成大量陨石后不再删除和创建。另一类则在宇宙中任意飞行，从随机算法的指定边界上随机生成，并朝太阳系中心附近进行匀速直线运动，直到离开指定边界，进行销毁。

   火焰则较为复杂，根据飞船的移动轨迹在飞船尾部生成火焰粒子，并随着生存时间的延长逐渐切换纹理坐标显示动画效果，超过生存期限的粒子将被删除回收。

   粒子的渲染使用了实例化技术，初始化时分配大量空间，每帧更新时一次性将所有粒子的信息通过 `glBufferSubData()`发送到 GPU，并调用 `glDrawElementsInstanced()`一次性完成大量粒子渲染，极大的提高了整体性能。除此之外，还进行了以下优化：

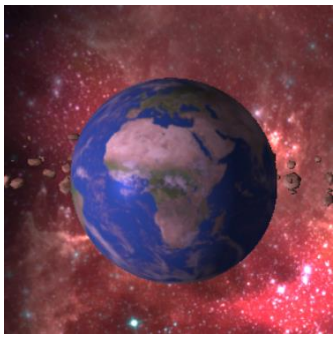
   - 经调试发现，帧率的降低基本上都是因为 `glm::rotate()`带来的高性能开销造成，所以修改了粒子系统实现，使得部分粒子不进行自转，最终结果对整体效果的影响微乎其微，但是帧率提升了 40%左右。
   - 编译时使用 Release 模式，帧率翻倍。
   - 从 Nvidia 文档中找到了使用代码切换到独显的宏定义，帧率翻倍。
   - 在粒子死亡时并不真正进行删除，而是直接更新该粒子数据为一个刚生成的新粒子，减少频繁删除带来的性能开销。

2. 星球渲染

起初的星球只使用了普通的漫反射、反射、高光的普通 Phong 光照模型，效果看起来并不理想。于是首先打算实现星球的边缘大气层泛光效果，网上通常的教程方法为先使用 HDR 区分高光区域，并使用帧缓冲进行提取，再进行高斯模糊添加到原渲染画面上。整个流程十分复杂而且性能损耗比较高，故选取了一个折衷的方案。

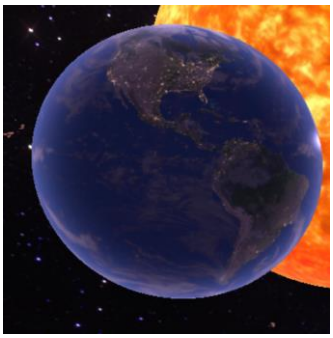
根据法向量和视线之间的夹角，区分圆形的边缘和中间部分。边缘法向量和视线的向量乘积很小，在 shader 中根据这个乘积的大小额外添加光照，实现了边缘光效果。尽管没有大气层反光效果那么惊艳，但是还是使得整体画面有了质的提高。

除此之外，对地球模型进行了额外的优化。地球模型共使用了 4 个贴图，一个普通的漫反射贴图，一个法线贴图以使陆地有微微的突起感，一个高光贴图以使水面的高光反射度远高于陆地，模拟水的物理效果。另外自己突发奇想，添加了一个夜晚贴图，根据光照强度确定是否为该位置夜晚，使得背对太阳的半球上，陆地会逐渐亮起星星点点的灯光。



普通 Phong 光照模型

边缘光+4 贴图效果

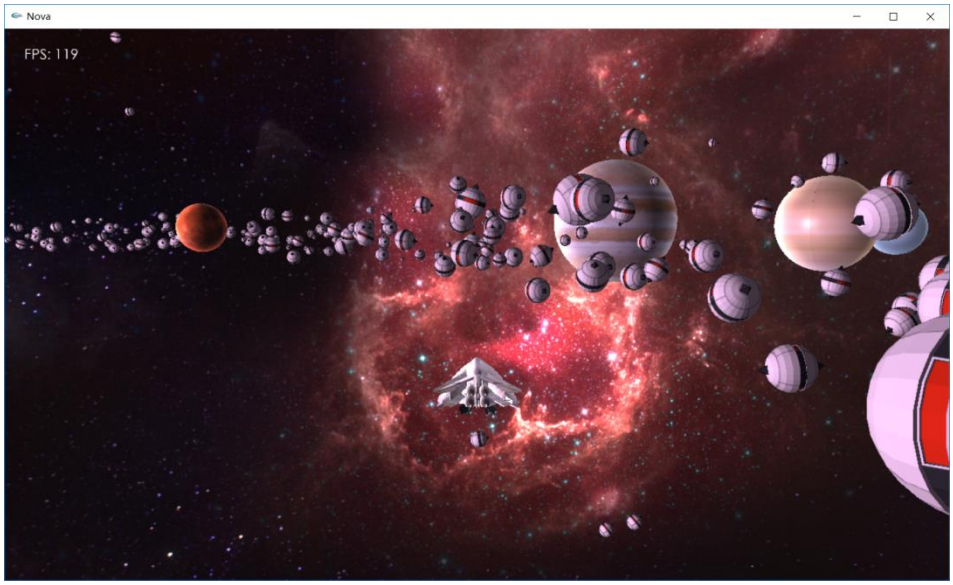


夜晚贴图效果

3. 模型

鉴于宇宙场景中大多数模型都是相同的网格结构，但有不同的贴图，故对模型读取部分进行了修改，使得模型读取时可以**共用模型但使用独特的纹理**，减少了模型的存储空间及读取时间。

陨石模型按 3 键可以切换到自制 Low-poly 模型（丑萌的太空机器人）。



自制 Low-poly 太空机器人