浙江大学

本科实验报告

课程名称：计算机图形学

姓　　名：王晨宇

学　　院：计算机科学与技术学院

系：

专　　业：计算机科学与技术

学　　号：3180104919

指导教师：唐敏

目录

浙江大学实验报告

课程名称：	计算机图形学	实验类型：	综合		
实验项目名称：	OpenGL 实例化				
学生姓名：	王晨宇	专业：	计算机科学与技术	学号：	3180104919
同组学生姓名：	None	指导老师：	唐敏		
实验地点：	紫金港机房	实验时间：	2021-05-23		

一、 实验目的和要求

学习绘制模型的大量实例的高效方法。

二、 实验环境

- Visual Studio C++ 2019
- GLFW、GLAD、GLM

三、 实验内容和原理

3.1 单个实例绘制的局限性

当绘制模型的大量实例 (Instance) 时，简单通过绑定顶点数组对象并调用 glDrawElement() 方法会在绘制顶点数据之前的做很多准备工作——例如传递缓冲读取数据地址和顶点属性等——会消耗大量性能。这些都是在相对缓慢的 CPU 到 GPU 总线 (CPU to GPU Bus) 上进行的）。所以，即便渲染顶点非常快，命令 GPU 去渲染却不是这样

3.2 实例化

实例化的主要思想是一次性向 gpu 发送大量顶点数据并通过一次调用方法完成大量实例的绘制。

当渲染的实例规模超过了 shader 里 uniform 数据大小上限，可以通过实例化数组——一个顶点属性（能够让我们储存更多的数据），仅在顶点着色器渲染一个新的实例时才会更新。

使用顶点属性时，顶点着色器的每次运行都会让 GLSL 获取新一组适用于当前顶点的属性。而当我们将顶点属性定义为一个实例化数组时，顶点着色器就只需要对每个实例，而

不是每个顶点，更新顶点属性的内容了。这允许我们对逐顶点的数据使用普通的顶点属性，而对逐实例的数据使用实例化数组。

四、 代码实现

```
1     _asternoidInstancedShader->use();
2     _asternoidShader->setMat4("view", view);
3     _asternoidShader->setMat4("projection", projection);
4     glBindVertexArray(_asternoid->_vao);
5
6     unsigned int instanceVBO;
7     glGenBuffers(1, &instanceVBO);
8     glBindBuffer(GL_ARRAY_BUFFER, instanceVBO);
9     glBufferData(GL_ARRAY_BUFFER, sizeof(glm::mat4) * _amount, &_modelMatrices
          [0], GL_STATIC_DRAW);
10    // glBindBuffer(GL_ARRAY_BUFFER, 0);
11
12    // 拆分变换矩阵为4个列向量
13    GLsizei vec4Size = sizeof(glm::vec4);
14    printf("%d\n", vec4Size);
15    glEnableVertexAttribArray(3);
16    glVertexAttribPointer(3, 4, GL_FLOAT, GL_FALSE, 4 * vec4Size, (void*)0);
17    glEnableVertexAttribArray(4);
18    glVertexAttribPointer(4, 4, GL_FLOAT, GL_FALSE, 4 * vec4Size, (void*)(
          vec4Size));
19    glEnableVertexAttribArray(5);
20    glVertexAttribPointer(5, 4, GL_FLOAT, GL_FALSE, 4 * vec4Size, (void*)(2 *
          vec4Size));
21    glEnableVertexAttribArray(6);
22    glVertexAttribPointer(6, 4, GL_FLOAT, GL_FALSE, 4 * vec4Size, (void*)(3 *
          vec4Size));
23    glVertexAttribDivisor(3, 1);
24    glVertexAttribDivisor(4, 1);
25    glVertexAttribDivisor(5, 1);
26    glVertexAttribDivisor(6, 1);
27
28    glBindVertexArray(0);
29    _asternoid->instancedDraw(_amount);
```

首先我们绑定陨石模型的顶点数组对象，为此我们将 Model 的 _vao 属性设置成了 public 方便调用。然后将所有陨石的 modelMatrices 内容存在顶点缓存对象（instanced-VBO）中，再设置顶点属性。

因为顶点属性最大允许的数据大小等于一个 vec4，而一个矩阵有 4 个向量，所以需要预留 4 个顶点属性。位置值从 3 开始，3、4、5、6 分别对应每一个顶点属性包含了 4 个 float，间隔的步长为 4 * vec4Size 也就是一个矩阵的大小，起始的位置是一个一个向量。

最后需要设置属性除数。设置为 1 告诉 OpenGL 我们希望在渲染一个新实例的时候更

新顶点属性，即位置 2 的顶点属性是一个实例化数组。

在 Model 中增加了一个实例化绘制的方法：

```
1  void Model::instancedDraw(int n) {
2    glBindVertexArray(_vao);
3    glDrawElementsInstanced(GL_TRIANGLES, _indices.size(), GL_UNSIGNED_INT, 0, n)
         ;
4    glBindVertexArray(0);
5  }
```

glDrawElementsInstanced() 相对于原来的方法需要一个参数 n 来指明所需要绘制的个数。

五、 实验结果

对于 amount=10000 的规模，普通的绘制帧率稳定在 20-40fps 内：

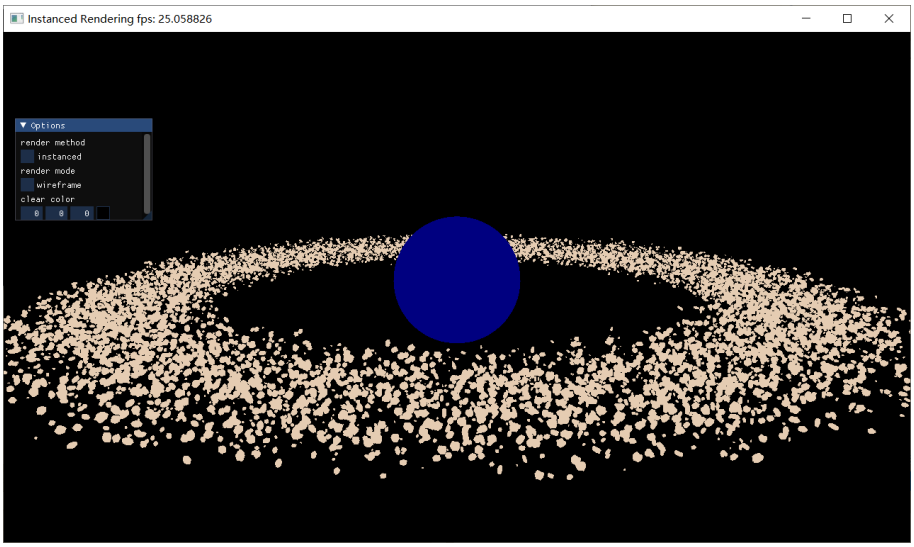

图 1: 普通方法

实例化后的方法帧率在 100-200fps 中（由于设备问题帧率一直都非常不稳定，还不知道原因）

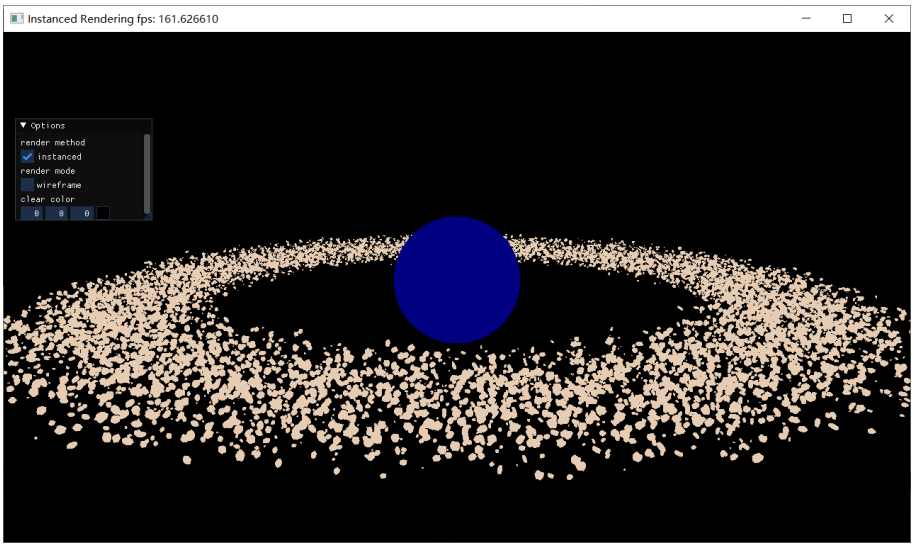

图 2: 实例化

当规模达到 10w 时，普通的绘制帧率就只有 3fps 左右，大概就跟 ppt 差不多：

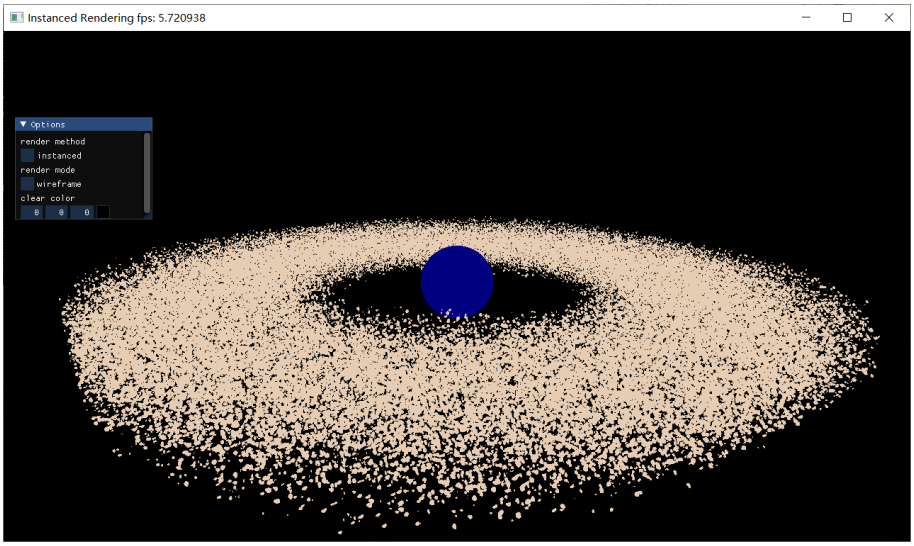

图 3: 普通方法

而实例化能达到 15fps 左右，看起来还是比较流畅，明白了什么是 1 帧能玩，两帧流畅：

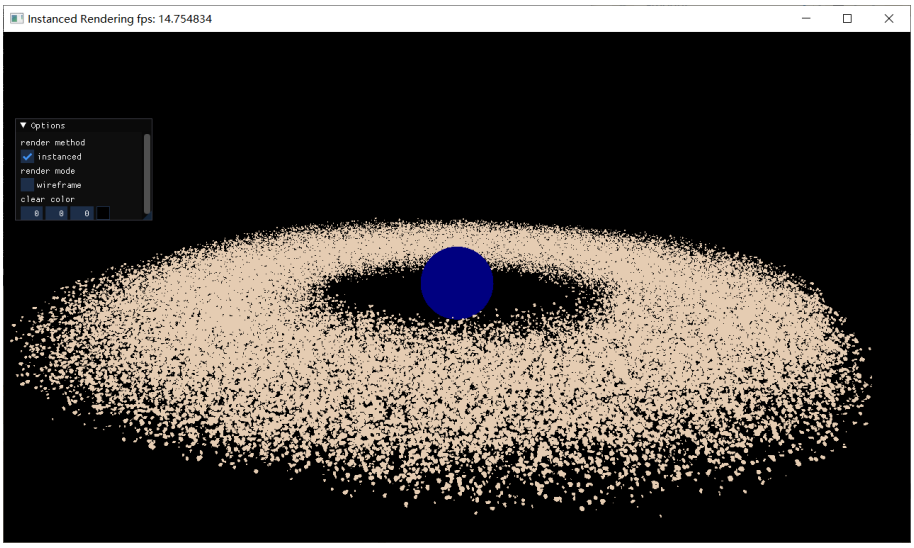

图 4: 实例化

然后又随手添加了一个公转效果。

但是生成可执行文件之后对于规模 1w 的小行星，两者的差别好像又没有 3 倍这么大了（因为自己电脑上帧率实在太不稳定），还不知道是为什么。

六、 讨论和心得

这次的实验总体来说比较简单，教程也讲得比较详细。写完了发现代码里还有一个 InstancedModel 类，但是感觉没有非常大的用处，就直接写在 render 方法里了。

空余时间在学习模型导入和贴图，但是还有一点小 bug 就还没实现出来。希望下次实验可以实现。

突然意识到之前交源代码的时候没有交 external 和 base 之类的库，不知道会不会对报告有影响 QAQ 所以在这次的代码里加上了