# 04-基于 OGRE 的分布式渲染系统-VR

王鹤翔，江月虎

摘　要: 随着人们对模型精细度和真实性的要求不断提高，单块显卡难以满足渲染需求，因此分布式云渲染系统成为了一个可行的解决方案。在这种技术中，视锥分割与渲染结果合并是一个很重要的模块，同时也是一个技术难点。我们组针对这个问题，在视锥分割方面提出了一种根据深度进行分割的解决方案，即按照距离摄像机的距离在视线方向轴上对视锥进行切片，再提取出每个切片中的模型数据传输到各个渲染结点进行渲染；在最后的渲染结果合并方面，由于我们采用了与众不同的视锥分割方式，结果合并也就相对复杂，我们同样设计了基于深度测试的逐像素的合并算法，对于各个渲染画面先比较深度再决定每一个像素的取值。我们的创新点在于提出了和传统的基于渲染点数或者在投影平面进行分割的方式完全不同的基于深度进行视锥分割的分割算法，并且提出了相应的画面合成算法，这对于接口较为传统的 OGRE 引擎来说大大减少了视锥分割的工作量，降低了视锥分割的复杂度。

**Title Distributed Rendering System Based on OGRE**

Wang Hexiang, Jiang Yuehu

**Abstract**: As requirements for the precision and authenticity of models continue to increase, it is difficult for a single GPU to meet rendering needs, thus a distributed cloud rendering system has become a viable solution. In this system, frustum segmentation and merging of rendering results is an important module and a technical difficulty. Aiming at solving this problem, our group proposed a solution for segmentation according to depth of objects, that is slice the frustum along the sight direction axis according to the distance from the camera, then extract the model data in each slice and transfer them to each rendering node; As for the final merging, we design a specific algorithm based on depth testing in line with our special segmenting method. We compare the depth for each pixel in each rendering result before determining its right color. The innovation of our work is to propose a new depth-based algorithm which is completely different from traditional methods based on the number of rendering points or the projection plane, and we proposed the corresponding merging algorithm in the same time. For engine like OGRE with traditional interfaces, this technic dramatically reduce the workload and the complexity of frustum segmentation.

**Key word：Frustum Segmentation, Distributed System, OGRE, Merging Rendering Results**

# 1 简介与意义/Introduction

## 1.1 项目意义和依据/Significance

随着人们对模型精细度和真实性的要求越来越高，单块显卡可能无法满足渲染需求。比如大飞机模型、扫描场景模型，我们希望可以将模型数字化，让设计制造人员实现在线的多人交互碰撞。因此我们引入分布式云渲染这样一个技术去解决这个问题。分布式云渲染是一种 3D 可视多人交互的在线渲染场景，同一个大规模模型既可以多用户从不同视角进行观察，又可以进行一个总的预览。在这项技术中，一个非常重要也是非常困难的模块就是视锥的分割以及渲染结果的合并，这也是我们小组工作的重心所在。这两部分内容一部分是分布式渲染开展的数据基础，另一部分是综合得到分布式渲染结果的必需过程，重要性之高不言而喻。研究如何高质量高效率的完成这两部分工作意义重大。

现有方法通常是基于点云中各部分点的数量进行分割，或者是在投影平面内将场景进行分割；这些想法比较直观，但是在 OGRE 中实现却有很大的问题，就是由于 OGRE 所提供的操作摄像机和视锥体的接口方法比较有限，而且都是一些调整近切面、远切面、视场角、长宽比的方法，我们很难根据这些方法在投影面内对视锥进行横向的切割，很容易造成各个子视锥体的相互重合，或者部分模型被遗漏的问题。并且，这种切割方式很难对于不确定个数的分布式渲染结点进行自适应，当结点个数发生变化，算法就很容易出现问题。

因此，我们换用了一种新的思路，既然横向的切割很难进行，那么我们就可以从另一个方向，也就是纵向对视锥体进行分割。这样，我们直接通过调用 OGRE 里设置摄像机的 FarClipDistance 和 NearClipDistance 的函数就完成了对视锥体在沿着摄像机朝向上的分割，分成了若干份切片。这样做的另一个好处就是对渲染结点的数量变化具有很大的灵活性，对于不同数量的渲染结点很容易分割出相应数量的子视锥进行分别渲染。这样视锥分割后，相应的渲染结果合并也需要做出调整，我们的算法采用逐像素比较深度的方法进行画面合成。

## 1.2 本方法/系统框架/Article Structure

我们的系统框架图如图 1所示：

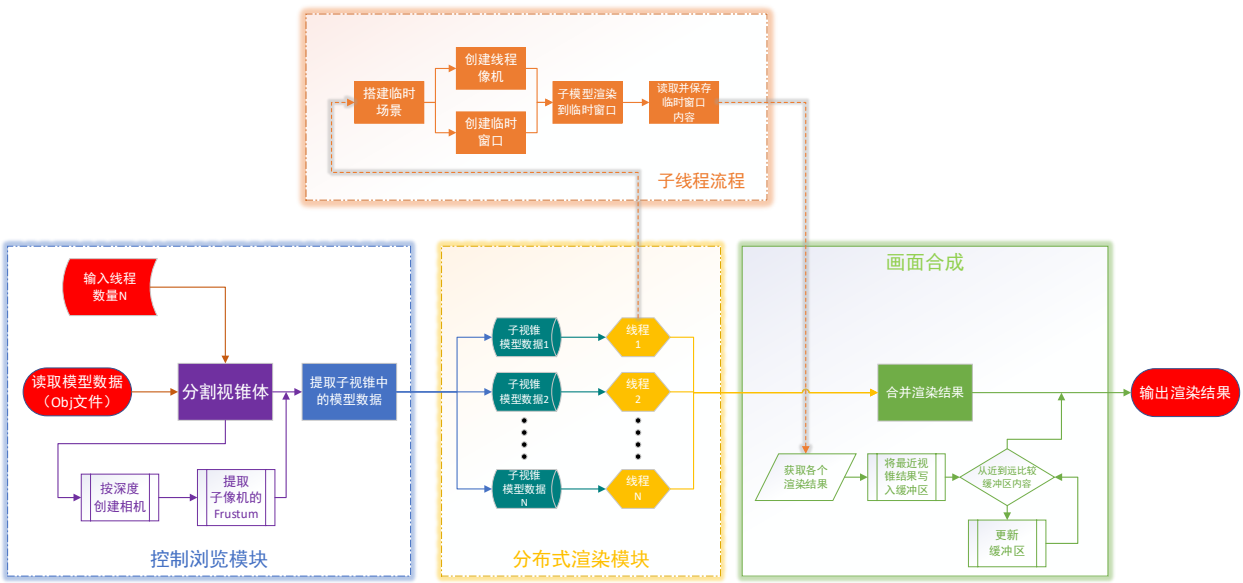


图 1 系统框架图

## 2 相关工作/Related Works

在计算机技术飞速发展的今天，三维电脑图像技术(3D Computer Graphics)得到越来越广泛的应用，给人带来了身临其境的视觉效果。然而，在 3D CG 渲染这一阶段，现有的系统的计算能力成了瓶颈，而分布式渲染就是为了解决这一技术瓶颈而被提出来的。[1] 现有的一些分布式渲染工作中，比较有名的例如 3ds Max 里的 V-Ray 分布式渲染系统，用户使用时可以直接添加额外的计算机作为渲染机到当前渲染任务中，如图 2 所示，并且可以设置本机是否也作为渲染机使用，也可以实时显示执行某一渲染区块的计算机名称与信息；另外还有一些分布式渲染系统的设计是基于开源高性能三维图形渲染工具包 OSG 以及 sort-last 分布渲染模式和 C/S（客户端/服务端）模式实现的。

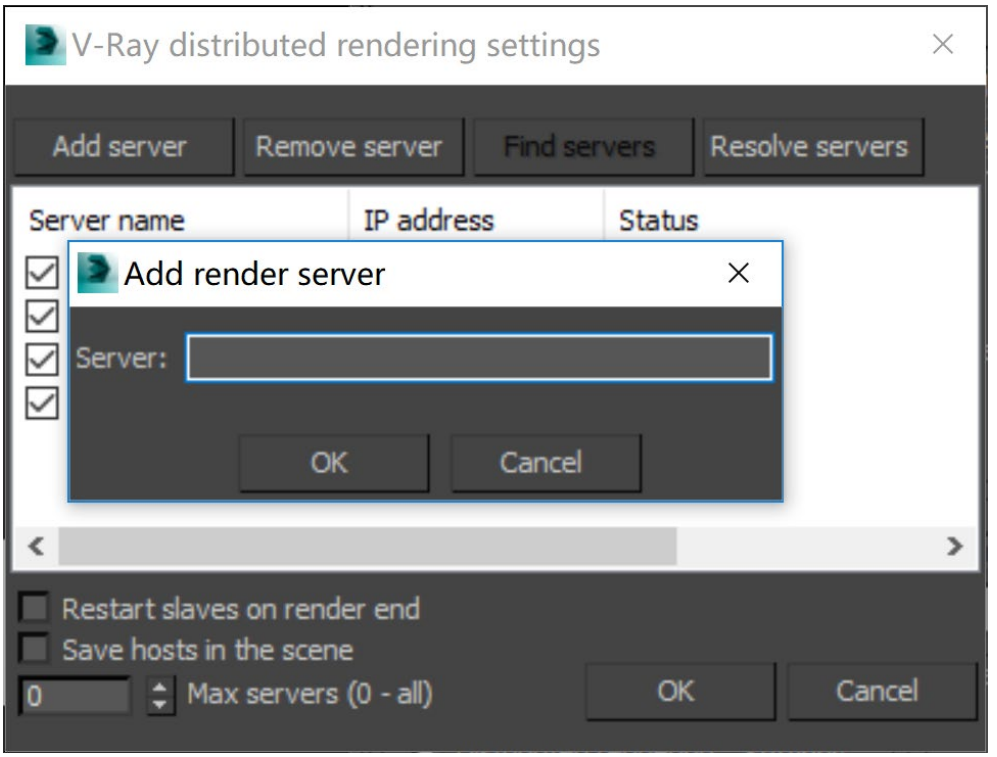


图 2 V-Ray 分布式渲染系统图示

OGRE 是一个开源三维图形渲染引擎，是面向对象的并且高效的，抽象化了不同的 API 和平台，能够以场景为对象来渲染物体，并且支持多种场景。OGRE 已经成功的被应用于诸多三维领域，包括网络游戏和三维仿真项目等等。但是，目前来说，基于 OGRE 的分布式渲染系统的设计和开发尝试十分缺乏，很少有大型分布式系统是基于 OGRE 设计出来的，这在某些程度上也可以解释为什么在模型数据愈加复杂庞大、画质需求越来越高的今天 OGRE 引擎逐渐的走向了没落。因此，我们的本次项目打算尝试基于 OGRE 渲染引擎来模拟开发一个分布式渲染系统，希望能借此提高 OGRE 渲染引擎的时代竞争力。但是，鉴于硬件条件的限制，我们很难找到多台计算机来真正的搭建分布式渲染系统，因此采取 C++多线程的方式模拟进行分布渲染，重在系统逻辑的设计与开发。

另外，由于 OGRE 方面的研究工作确实比较难以搜索，相关资料也很缺乏，我们的项目主要是自己参考 OGRE 官方 API 文档摸索进行的。

## 3 研究内容与方法(或算法)/Contnts and Methods(or Algorithm)

从图 1可知我们的项目主要是由三个模块构成：控制浏览模块、分布式渲染模块、画面合成模块。因此，接下来将围绕这三个模块来介绍我们的算法。

### 3.1 控制浏览模块

#### 3.1.1 模型载入

模型载入部分，我们逐行读取两个 obj 模型文件，根据每行数据的类型存储到三个 Vector3 向量中——顶点、纹理坐标和面的索引号，接着计算每个面的法向量，之后对每个点所在平面进行加权和并单位化得出每个顶点的法向量，保存到对应的向量中。最后将所有读取的数据存储到我们自己定义的 object 结构体中，除了上述 5 个向量，每个 object 结构体还包含对应标记每个顶点是否可见，每个 object 对应的材质，用于后续绘制分场景中的物体。

#### 3.1.2 视锥分割

我们采用基于深度的视锥分割方法，在摄像机朝向的方向上对视锥进行切片分割，根据渲染节点的数量 N 将主视锥纵向分割成 N 个切片，这一过程通过使用 OGRE 建立多个 Camera 并且分别设置他们的 FarClipDistance 和 NearClipDistance 来实现。我们通过首先遍历主场景中的模型找出距离相机所在平面的最近距离和最远距离，然后将这段距离平分为 N 份，依次设定每个分视锥体的远近平面，以保证每个视锥中都有相当规模的模型数据，每个渲染节点的渲染任务量不会相差太多，避免部分节点性能浪费，部分节点负荷太大。

#### 3.1.3 提取子视锥数据

将上述的分割后的相机信息和读取的场景中的物体信息传递到分割物体函数中。首先获取每个相机对应视锥体近视远视面在世界坐标系中的八个顶点的坐标，对于物体中的每个点，创建一个从相机位置出发到该点的射线，判断该点与近视远视面是否相交，如果相交，计算两点的距离，将该距离与相机位置沿射线方向到两个平面的交点的距离进行比较，如果位于两者之间，则将该点标记为在可见中，将信息同样存储到新的 object 结构体中，返回新的结构体，这样在后续的每个分节点的场景绘制时，对于每个三角面片，如果其中任意一点可见，则绘制该面片，即可达到分割物体的效果。

### 3.2 分布式渲染模块

本模块中，将上一模块中根据视锥分割出的各个子模型数据分别传入各个子渲染节点中，进行模拟分布式的并行渲染，每个子节点分别得到一帧画面，并渲染到临时窗口中以便后续读取使用。

在这该模块中，我们最初是计划采用多线程，但是实际运行时我们发现，OGRE 渲染节点是继承自 singleton<Root>，属于单例模式，只能创建一个渲染节点，因此无法采用多线程的方法，想要采用多线程只能对源代码进行修改，但是这超过了我们能力范围，因此我们只能退而求其次，采用线性渲染的方式，创建多个场景管理器，每个节点对单个场景进行单独的渲染。

#### 3.2.1 数据的储存与传递

我们自己定义了一个结构体类型 Object 用于在主线程和各个子渲染节点之间进行模型数据的传递。其中用一些 Vector3 和 Vector2 类型的容器来保存顶点、法向量、三角片元、序列号等模型信息；用一个 bool 类型的容器储存一组信号量，用来标识该片元数据是否可见，即是否被分割进入了当前子节点中；另外还用一个 string 类型的容易来标识该部分模型数据对应的纹理贴图的名字。

#### 3.2.2 子节点渲染流程

我们在主线程初始化时就根据用户输入的子渲染节点的模拟数量 N 为每个子渲染节点分别分配了一个场景管理器 SceneManager，然后每个子节点需要进行的任务就是：根据传入的部分模型数据，以及主场景的灯光、相机以及 Transform 等信息，调用本节点的 SceneManager 创建临时的 ManualObject，然后搭建一个相机、灯光和矩阵变换都和主场景一致的临时场景，通过 OGRE 引擎将这部分模型渲染到临时窗口中。这样一个渲染流程就和真正的分布式渲染系统中每一个子节点需要执行的渲染任务是一致的，很好的起到了模拟效果。

3.2.3 交互式操作

我们设计的简单交互行为主要有：

- 在主渲染流程开始后，每次按下左 shift 键就会模拟一次分布式渲染操作，并且我们设计了一个同步信号量结合帧监听函数来保证各个子节点的渲染都完成后才进行最后的画面合成操作；
- 通过键盘输入和鼠标滚轮控制主场景中模型的平移与放缩，各个子渲染节点将会实时渲染出相应的画面，并且最后的合成画面也会相应变化；
- 单击空格键输出主窗口和各个子渲染窗口的 FPS、Batch Count、Frame Time 等等数据，进行性能比较。

### 3.3 画面合成模块

这一模块主要是获取各个子渲染节点的渲染结果，等到各子渲染流程都完成后对提取到的各帧画面进行合成，得到最终结果帧。

3.3.1 获取子节点渲染结果

在一开始的主进程中创建纹理管理器，根据用户输入的渲染节点的个数动态创建多个纹理指针，设置纹理的大小与各分窗口一致。对于每个渲染节点，从纹理指针中获取渲染渲染对象指针，并且将指针绑定到给定渲染节点的相机上，设置允许每个材质信息能够随着窗口的改变动态变化材质的内容。

在每一帧渲染结束时，将每个分渲染节点的纹理指针转换为 image，传递给后续合成图片函数。

3.3.2 画面合成

由于采用的特殊的视锥分割方式，我们的画面合成算法也比较特别，采用了逐像素的基于深度的合成算法。具体实现为：首先将缓冲区的每个图片设置为深度值最低的图片的像素，接着按照深度从小到大的顺序遍历各个渲染画面，对画面中的每一个像素，如果当前像素值为黑色并且该画面在该像素处的值不为黑色，就将该处像素的值置为当前画面该像素处的值，伪代码如下：

**Algorithm 1**:merge image

```
Input: images[n]
Output: image
    1.image=images[0];
    2.for(i in images) do
    3.    for(j in images[i].pixels) do
    4.        if(image.pixel==default) do
    5.            image.pixel:=images[i].pixel;
    6.return image;
```

## 4 实验结果与分析/Experiment Results and Analysis

以输入线程数量 4 为例，我们得到的各个子渲染节点的渲染画面如图 3、图 4、图 5和图 6所示，最后合成后的总画面如图 7所示。

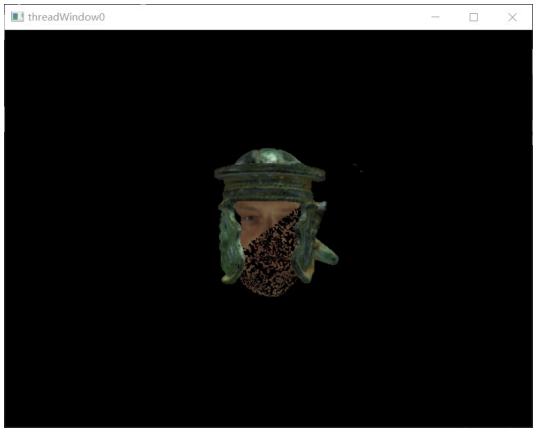


图 3 子节点 1 渲染画面

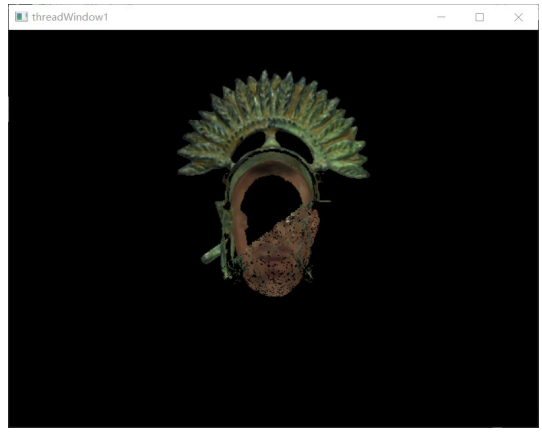


图 4 子节点 2 渲染画面

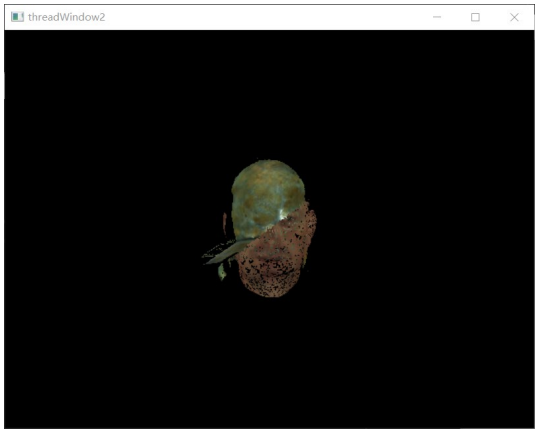


图 5 子节点 3 渲染画面

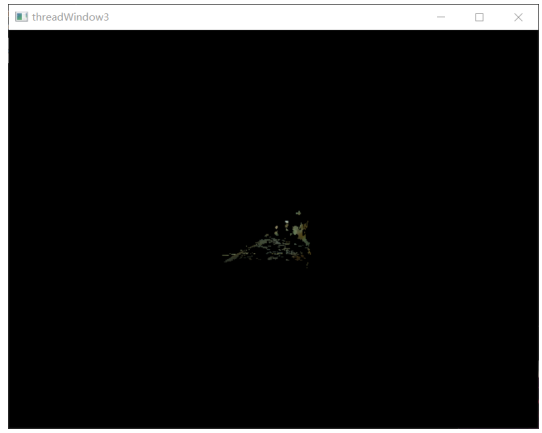


图 6 子节点 4 渲染画面

图 7 分布式渲染结果图示

从结果图中可见，每个子节点分别对模型的一部分进行了渲染，最后合成的画面就是原模型，说明分布式渲染并合成的整个流程成功完成。

然后，我们以线程数量 3 为例再对我们的简单交互操作进行了测试，将模型进行左移和放大操作，得到的各个渲染节点的同步画面如图 8、图 9、图 10所示，得到的合成后的最终画面如图 11所示。

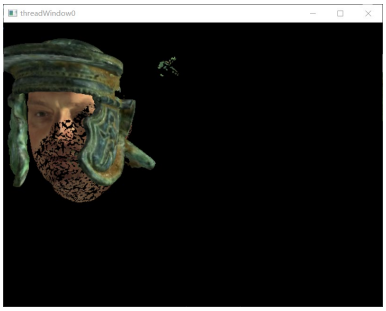


图 8 渲染节点 1-平移放缩

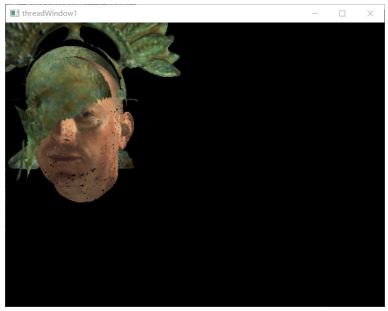


图 9 渲染节点 2-平移放缩

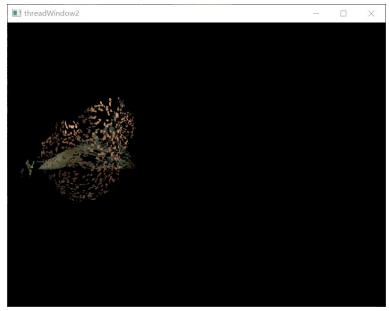


图 10 渲染节点 3-平移放缩

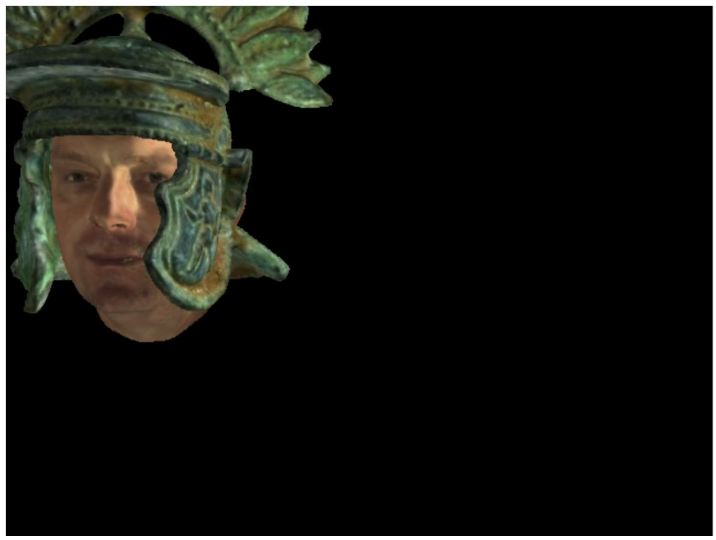

图 11 平移放缩后的结果画面

由上图可知，平移与放缩等基本模型操作也可以在我们的分布式渲染模拟系统中成功实现。

在合成结果后，我们尝试将结果输出到主窗口中，我们采取的办法是将合成画面保存到本地 ogre 的纹理中，然后根据纹理动态创建一个材质。原本主窗口中没有任何物体，现在我们绘制一个和窗口一样大小的平面，然后将刚才动态创建的材质添加到该平面上。这样在主窗口上就能显示合成的信息。

但是在实际的测试中我们发现，在移动物体后，每个分视图的信息会相应改变，同时存储到本地的图片也会发生相应的改变。但是主视图的结果却一直保持不变，经过反复的测试和查阅资料我们发现，ogre 在开始运行时，会提前加载好所有的材质纹理信息到内存中，因此即使在后续将新的图片覆盖原来的纹理，ogre 不会更新已有的纹理，还是会采用原本的纹理进行贴图。目前我们没有找到更好的解决方法，因此只能采用将合成结果保存到本地的办法。

## 5 特色与创新/ Distinctive or Innovation Points （5%篇幅）

本小组比较创新的地方在于视锥体的分割。一开始我们考虑的方向是视锥体的纵向以及横向分割，但是由于 OGRE 的视锥体均是中心对称，因此难以将一个视锥体完美地分割成多个视锥体，因此考虑正交投影下的视锥体分割会简单很多，但是这样可能会对真实感存在一定的影响。后来与助教进行交流时，我们发现了一种按深度分割视锥体的方法，利用距离相机的深度值分割视锥体，获取场景中物体的最远和最近的点信息，将视锥体等分或者按照按点密度分，最后再合并。因此，我们认为视锥体分割是我们的创新点所在。

另外，由于 OGRE 是单例模式的，在同一台电脑同一个主程序中只能有一个渲染根节点生成，因此通过单纯的多线程的方式其实是无法建立多个独立的渲染节点的，所以，我们提出了通过创建多个 SceneManager 的方式来对多渲染节点进行 OGRE 特色的模拟，这也是本小组项目的一个特色点。

最后，要感谢助教秦义明以及助教黎宇航的耐心帮助，没有他们提供的指引与教导本组很难完成这次项目。

**References**:

[1] 梁志远. 分布式渲染系统架构研究与优化[D].

[2] 基于 OSG 的分布式渲染系统的设计与实现 许丹阳; 赵红领; 谭同德; 秦鑫 微计算机信息 2009-03-25 期刊

[3] 分布式渲染系统架构研究与优化 梁志远 华南理工大学 2010-05-24 硕士

[4] 一种场景内容分布的交互式渲染系统 孙昭; 柳有权; 张彩荣; 石剑; 陈彦云 图形学学报 2019-02-15 期刊

[5] 基于 MapReduce 的分布式光线跟踪的设计与实现 郑欣杰; 朱程荣; 熊齐邦 计算机工程 2007-11-20 期刊

# 时间安排与分工统计表

| **组员信息**（含组长） | | | |
|---|---|---|---|
| 学生姓名 | 王鹤翔 | 学　　号 | 517021910819 |
| 项目分工 | 输出渲染机运行状况信息、分割视锥体、设计并实现每个线程 | | |
| 学生姓名 | 江月虎 | 学　　号 | 517021910948 |
| 项目分工 | 模型载入、分配数据到各个渲染节点、设计并实现画面合成算法 | | |