浙江大学

题目：实现层次 Z-Buffer 算法实验结果文档

姓　　名：	於其樊
学　　号：	12121049
指导老师：	冯结青
专业：	2021 级计算机科学与技术
学院：	计算机学院

1 任务描述：

实现层次 z-buffer 算法：

1. 完整模式(层次 z-buffer+场景八叉树)；
2. 分别对比简单模式和完整模式与扫描线 z-buffer 算法的加速比。

2 数据文件介绍

在 model 文件夹中存放了用于进行算法比较的模型，它们所含的顶点数和面元数如下表所示。（顶点数是读取到的 vertex 数；面元数是读取到的 face 数，且顶点重复的只记录一次）

表 2.1 obj 文件信息介绍

模型文件	顶点数	面片数	加载时间
Cross.obj	16	8	4ms
Dolphins.obj	855	1692	9ms
teapot.obj	530	992	7ms
Bunny.obj	34834	69451	147ms
african_head.obj	1258	2492	17ms
armadillo.obj	106289	212574	645ms

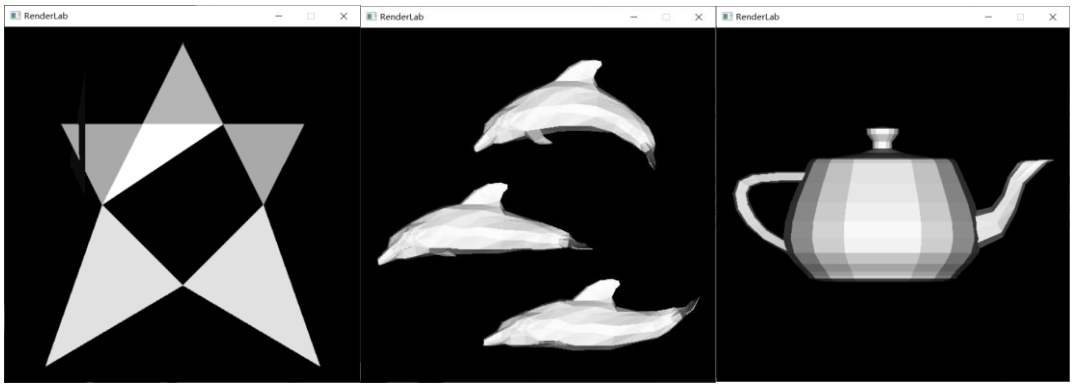
3 算法原理

- 普通 Z-Buffer 算法：直接光栅化所有多边形，并根据光栅化的结果更新 Z 缓存器和帧缓存器。
- 扫描线 Z-Buffer 算法：从上到下扫描每一行像素，在这一过程中维护与扫描线相交的多边形和边，即更新活化多边形表和活化边表，并根据相交线段更新 Z 缓存器和帧缓存器。
- 普通层次 Z-Buffer 算法：使用四叉树维护 Z 缓存器，将所有多边形依次在四叉树中进行查询。查询时找到包含这一多边形的深度最大的四叉树节点，如果通过了深度测试，就将多边形光栅化，并将光栅化得到的像素在这一子树中继续进行查询。
- 带场景八叉树的层次 Z-Buffer 算法：在普通层次 Z-Buffer 算法的基础上，把所有的多边形使用场景八叉树维护，然后将场景八叉树在四叉树中进行查询。一开始两棵树都位于根节点中，每次先将八叉树中属于当前节点的多边形进行光栅化，然后在四叉树中进行查询。之后再对八叉树的子节点进行递归，递归的同时要在四叉树中移动到相对应的节点。

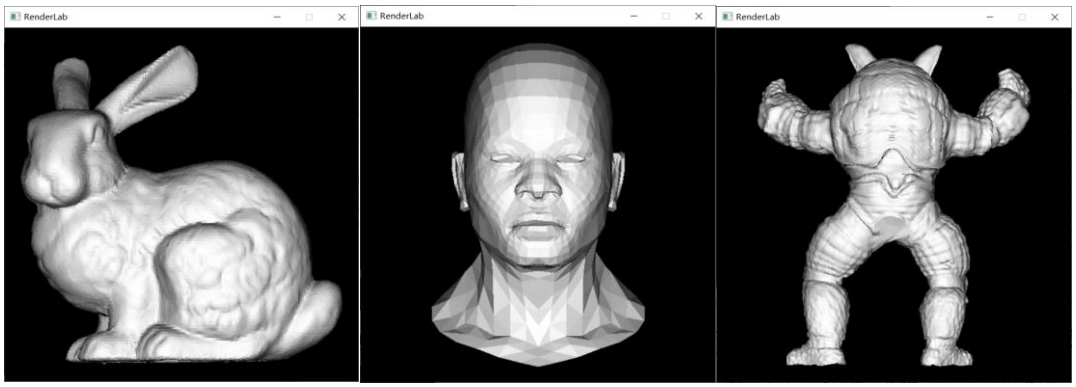
4 绘制结果

分别使用普通的 z-buffer、扫描线 z-buffer、简单模式的 z-buffer 以及完整模式的 z-buffer 对 obj 文件进行了消隐，并显示绘制图如下所示：

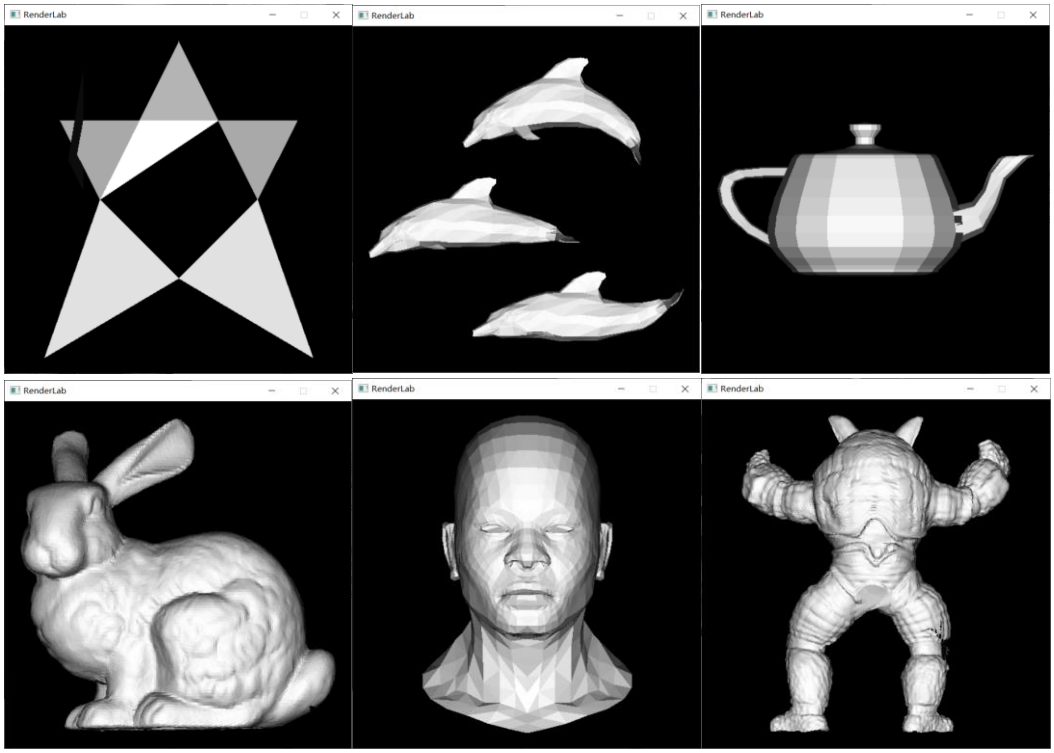
4.1 Naïve z-buffer



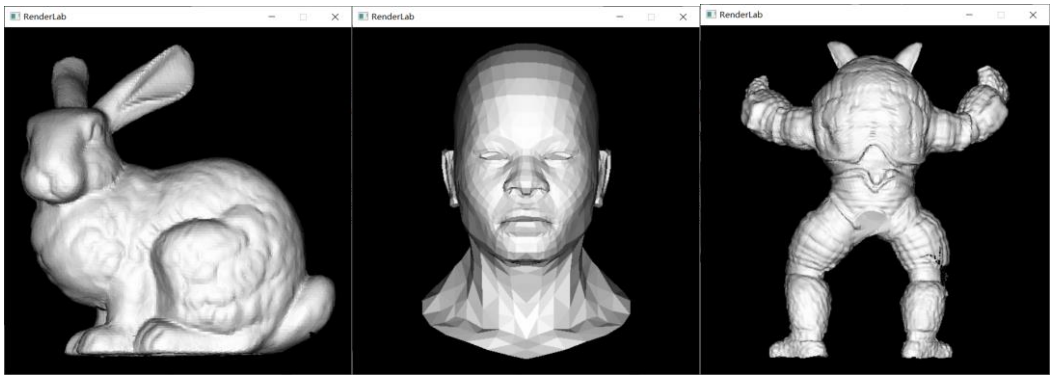
4.2 Scanline z-buffer



4.3 Hierarchy Z-buffer(without octree)



4.4 Hierarchy Z-buffer(with octree)



5 算法比较

各个模型使用各种算法绘制所需的时间如下表所示。

表 5.1 各算法绘制不同 obj 文件所需时间比较

	Cross	Dolphins	teapot	Bunny	African head	armadillo
Naïve Z-Buffer	2ms	23ms	13ms	967ms	29ms	5578ms
Scanline Z-Buffer	1ms	3ms	2ms	361ms	7ms	1095ms
Hierarchy Z-Buffer (without octree)	5ms	26ms	22ms	291ms	45ms	524ms
Hierarchy Z-Buffer (with octree)	5ms	7ms	6ms	58ms	12ms	97ms

可以看到，在模型比较简单时即使是简单的普通 Z-Buffer 时间开销也小于层次 Z-Buffer，考虑到利用空间换时间构造相应的数据结构也需要一定的时间，因此在模型比较简单时时间开销更高，而随着模型面片数和顶点数增加层次 Z-Buffer 的性能逐渐凸显出来。从算法原理来看，层次 Z-Buffer 更适用于多边形多、多边形之间遮挡关系较多时的情形，并且更易于并行化。因此层次 Z-Buffer 对模型的复杂度增加的情况有更好的适配性。

6 总结与心得

6.1 代码部分

模型的绘制部分我参考的网上的相关代码，并自己定义读取函数和存储的数据结构对 obj 文件中的顶点信息和面片信息进行存储。之后自己实现了 naïve Z-Buffer、scanline Z-Buffer、Hierarchy Z-buffer(without octree)和 Hierarchy Z-buffer(with octree)，主要参考了知乎[i]上 OpenGL 的实现过程以及课件 Lesson12 和 Lesson13 中扫描线 Z-Buffer 算法和层次 Z-Buffer 算法主要原理，但相应的数据结构和算法均由自己进行设计，并进行比较以及完成帧缓冲器各个像素的绘制。窗口化代码我主要是参考了 C++中如何绘制相应的窗口完成实现。

6.2 整体收获

由于本科了解到的图形学主要是利用 WebGL 提供的 API 进行人物建模设计，底层的裁剪、绘制、消隐等算法均从未了解，并且我的专业方向主攻 AI 多模态方向，对图形学相关知识涉及较少，学习图形学主要是为了扩充自己的知识，便于日后多模态研究中可能的应用。课程上接收知识后并转化到项目代码的实现，对于我来说是比较大的挑战，但完成后对于我来说也是比较大的收获。我总共花了大概两周时间进行编程，在实现过程中我也遇到了很多的问题，比如扫描线 Z-Buffer 中活化边和活化多边形数据结构保存和指针指代的问题，层次 Z-Buffer 提前剔除的判断问题以及八叉树的实现过程，我通过查阅课外资料和回顾课程 PPT 最终一一解决了问题。该项目的实现过程对于我后续思考能力的提升也有很大的帮助，也让我更加清晰地掌握了消隐算法的实现流程。

---

[i] [500 行代码学懂 OpenGL]之四 z-buffer - 知乎 (zhihu.com)