

# 浙江大学



题目：实现层次 Z-Buffer 算法说明文档

姓 名：於其燮

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：

1. 实现 z-buffer 算法和扫描线 z-buffer 算法
2. 实现层次 z-buffer 完整模式(层次 z-buffer+场景八叉树)
3. 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 编程环境：

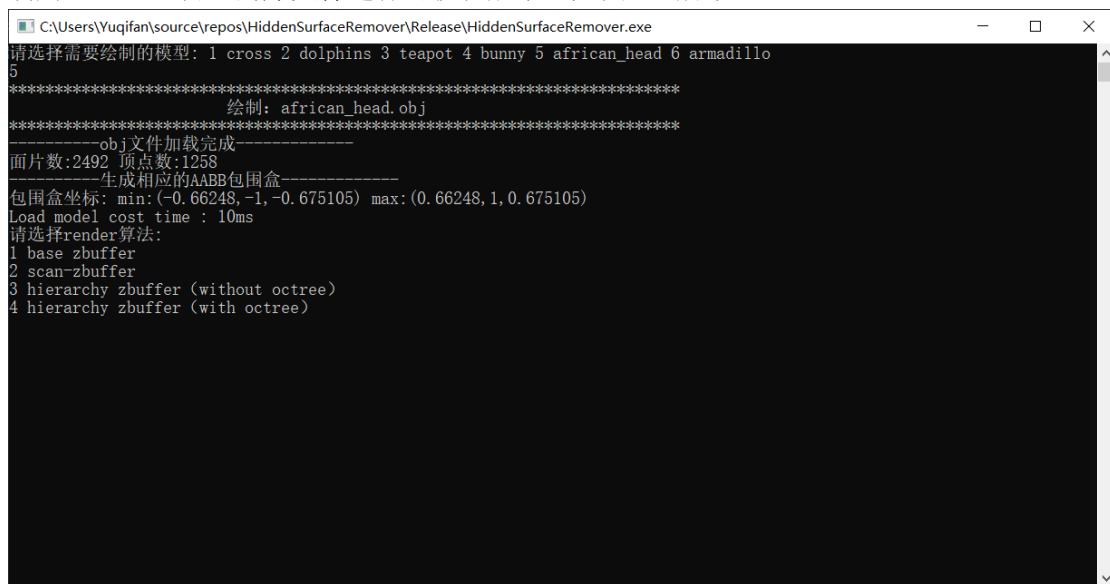
硬件设备：Intel(R) Xeon(R) W-2223 CPU @ 3.60GHz, RAM 24G

编程环境：Visual Studio 2017 (C++)

操作系统：Windows 10 专业版 64 位

# 3 用户界面使用说明：

程序采用 cmd 命令行面板的方式进行用户交互，在这里我主要设计了 7 种不同面片大小的模型验证实验效果，用户通过数字选择模型，通过对 obj 文件的加载，并显示相应的面片数和顶点数。同时将为对象建立相应的包围盒 (AABB)，并对所有坐标进行归一化处理，之后用户可以选择 4 种渲染算法进行绘制 (base zbuffer、scanline-zbuffer、简单的层次 zbuffer 和完整的层次 zbuffer (octree))，整体界面如图 3.1 所示。可以根据加载后的模型选择合适的算法进行模型的绘制，这里我统一采用灰度的形式进行表示，从而更好地展示模型的轮廓，利用 windows 窗口的库文件进行可视化展示，如图 3.2 所示



```
C:\Users\Yuqifan\source\repos\HiddenSurfaceRemover\Release\HiddenSurfaceRemover.exe
请选择需要绘制的模型: 1 cross 2 dolphins 3 teapot 4 bunny 5 african_head 6 armadillo
5
*****
*****          绘制: african_head.obj          *****
*****
*****obj文件加载完成*****
面片数:2492 顶点数:1258
*****生成相应的AABB包围盒*****
包围盒坐标: min:(-0.66248,-1,-0.675105) max:(0.66248,1,0.675105)
Load model cost time : 10ms
请选择render算法:
1 base zbuffer
2 scan-zbuffer
3 hierarchy zbuffer (without octree)
4 hierarchy zbuffer (with octree)
```

图 3.1 用户界面展示

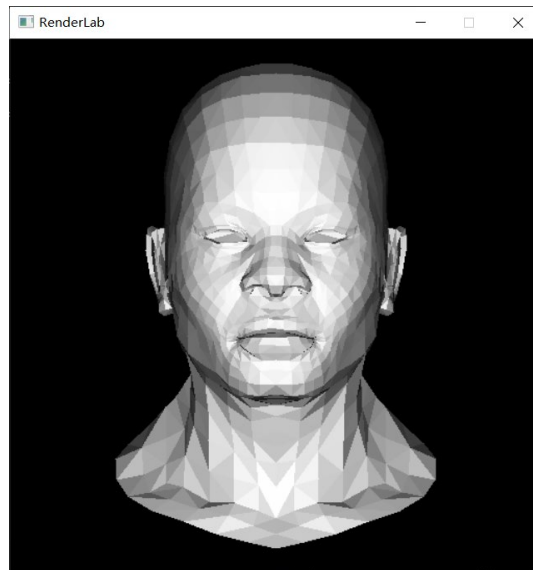


图 3.2 绘制图像窗口

## 4 数据结构说明

### 4.1 主要数据结构：

**Mesh 类：**定义了面片的基本信息（我采用了三角面片进行实现），包括所有的定点信息和所有的三角面片以及模型绘制时用到的包围盒，其中三角片的结构体包括三个顶点的下标序号以及面的序号。除此之外，定义读取 obj 文件的函数，利用文件信息进行存储用于后续的像素绘制；

**Render 类：**绘制类，定义了背景的相关信息以及帧缓冲器，同时完成对前置像素对象的绘制，主要包括背景的宽度高度以及背景颜色，用于填充帧缓冲器。除此之外，定义了绘制的基本函数

```
(void draw_pixel(int x, int y, Color color); // 绘制像素点)
```

```
(void draw_line(int x1, int y1, int x2, int y2, Color color); // 画线算法，绘制一条直线，使用 Bresenham 算法)
```

**Zbuffer 类：**作为 zbuffer 算法的核心类，定义了 z 缓冲器，记录当前的 z 值；同时是 Render 类的继承，从而能够利用其中的帧缓冲器完成对像素点的绘制。重写 drawMesh 函数，只有当前像素点的深度大于 zbuffer 对应的值时，说明没有被遮挡，继而更新 zbuffer 的值以及完成对该位置像素点的绘制

```
class Zbuffer : public Render {
```

```
public:
```

```
    Zbuffer(int w, int h, unsigned char* framebuffer, Color bg_color = Color(0, 0, 0, 255));
```

```
    ~Zbuffer();
```

```
    void drawTriangle(Vec2i* vis, float* Z, Color color);
```

```
    Vec2i world2screen(Vec3f v);
```

```
    virtual void drawMesh(std::vector<Mesh*>& triMeshes) override;
```

```
    virtual void clearZbuffer() override;
```

private:

```
vector<vector<float>> zbuffer; // z缓冲器，记录当前z值(width, height)
// 帧缓冲器在Render.h中定义，颜色绘制利用Render完成
```

};

Scanlinezbuffer 类：是 Zbuffer 类的继承，主要是对图形的绘制不再是针对像素点，而是根据扫描线与多边形的交点，完成多边形内一整条线的判断绘制

```
class ScanlineZBuffer : public Zbuffer {
```

public:

```
ScanlineZBuffer(int width, int height, unsigned char* fb, Color color);
~ScanlineZBuffer();
void clearZbuffer();
void drawScanlineMesh(vector<Mesh*>& triMeshs);
void RenderchangeDET(int y);
void constructPolygonEdge(Vec2i1f* v, Vec3f n, Vec3f* vec3fs, int id, Color color);
vector<float> scanlinezbuffer;
vector<vector<Polygons>> PolygonTable; // 分类的多边形表
vector<vector<Edges>> EdgeTable; // 分类的边表
vector<DynamicEdgeNode> DET; // 活化边表
vector<DynamicPolyNode> DPT; // 活化多边形表
```

};

Hierarchicalzbuffer 类：是 Zbuffer 类的继承，定义了四叉树的结构，即层次的 zbuffer，利用空间换时间，对每一个深度存储一个 zbuffer，当判断当前深度不满足绘制时，则当前层的像素点全部拒绝，能够实现快速拒绝。

```
class HierarchialZBuffer : public Zbuffer {
```

```
// 没有涉及八叉树的层次Z-buffer算法，快速剔除无法看到的面
```

public:

```
HierarchialZBuffer(int w, int h, unsigned char* framebuffer, Color bg_color = Color(0, 0, 0, 255));
void clearZbuffer();
vector<float*> hi_zbuffers; //存储每一层的zbuffer
vector<pair<int, int>> zbuffer_size; // 每一层z-buffer的宽度，zbuffer_len[i]表示第i层zbuffer，表示不同像素点对应的zbuffer
int zbuffer_height{ 0 };
int hidden_num{ 0 };
bool in_single_block(int l, int min_x, int min_y, int max_x, int max_y);
// 剔除的面数
void drawNewTriangle(Vec2i* vis, float* Z, Color color);
void drawHiMesh(vector<Mesh*>& triMeshs);
void updateHiZbuffers(int x, int y, float z);
```

};

Octreehierarchicalzbuffer 类：是 Zbuffer 类的继承，将非完整的层次 zbuffer 扩展到了八叉树完整形式，四叉树层次遍历的方式依旧要对每一层进行遍历判断，而八叉树利用二叉树 log 访问的特性进行快速访问从而剔除不满足的面片像素，进一步提升了时间效率。

## 4.2 辅助数据结构

Utils 类：定义了各类顶点信息的结构，包含了 Vec2i, Vec2ilf, Vec3f, 分别表示平面坐标和立体坐标；定义了像素点的颜色信息（用 rgba 进行表示）；实现了 Axis-Align 包围盒算法，为物体添加包围体的目的是快速的进行碰撞检测或者进行精确的碰撞检测之前进行过滤；定义了分类多边形表结构和分类边表结构以及活化边表结构用于辅助扫描线 z-buffer 算法的实现。

主函数中辅助绘制结果展示的 windows 窗口函数，参考开源代码实现，并自己进行了修改，定义了 windows\_init、windows\_render、windows\_close 三个函数，分别表示初始化窗口（定义窗口大小）、绘制结束后帧缓冲器的显示以及窗口的关闭和保持。

## 5 算法加速说明：

1. 利用逆时针的走向，根据三角面片两条边的点积得出法向量。通过考虑光照方向和法向量大于 90 度，则该面片为后向面，不需要再进行绘制，对向量的点积大于 0 的三角面片进行绘制，如此一来可以将处理的面片大大减少，从而降低时间复杂度和空间复杂度。
2. 构建了包围盒数据结构来进行绘制时 zbuffer 信息的判断，能够快速比较有无遮挡，从而能够快速进行剔除过滤，加速了判断。
3. 利用 vector 来实现基本的数组存储目标，能够实时分配内存空间，节省了空间开销。
4. 利用深度来完成绘制的比较过程从普通的逐像素点比较到扫描线比较再到层次 zbuffer 快速剔除，时间复杂度从  $O(n^2)$  到  $O(n)$  再到  $O(\log n)$ ，加快的时间效率。考虑到构造数据结构本身需要一定的时间，当模型比较简单时可能越简单的模型越快，但是当面片较多、多边形之间遮挡关系较多时加速算法性能表现得越明显，具体比较将在实验结果报告中说明。
5. 采用八叉树结构 Z-Buffer 消隐是对于某一个像素点,不必对在靠近或远离视点的面进行排序,只需在遍历各面后找出该像素的深度最大的点(Z 最大,即离视点最近)(先找深度大的点 draw\_pixel(),如果以后还有深度更大的,则通过设置新的颜色 draw\_pixel()来覆盖以前的点,从而不需要对每个像素都进行深度比较,大大减少了比较次数,加快了时间效率。