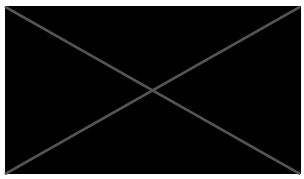
CG项目：层次Z-buffer算法

姓名： 

学号：

邮箱：

1. 编程环境

本实验基于以下平台进行：

- 硬件平台：
  - CPU： intel i7-13700KF
- 软件平台:
  - Windows平台， VS 2019
- 使用工具：
  - 图形界面显示使用 openCV，并使用 vcpkg 管理 VS 中的库

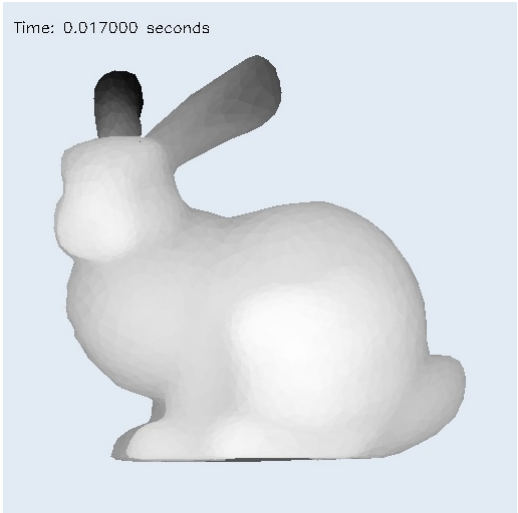
2. 用户界面说明

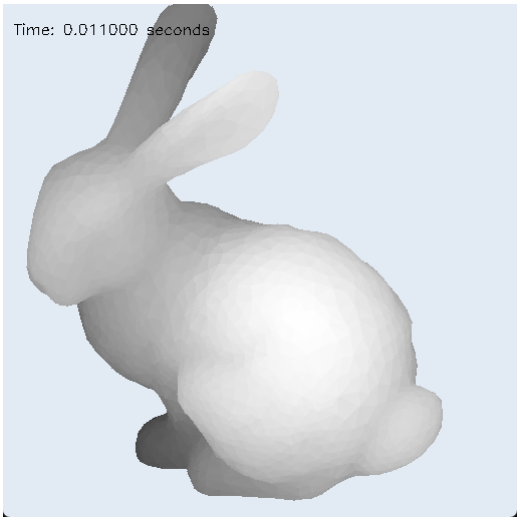
本项目设置了交互形式如下：

- 用户界面支持显示显示模型，按照深度对模型进行着色（由浅至深：由白到黑）；
- 显示一帧的渲染时间；
- 支持拖拽鼠标移动旋转模型，每次旋转会重新渲染 + 计时；

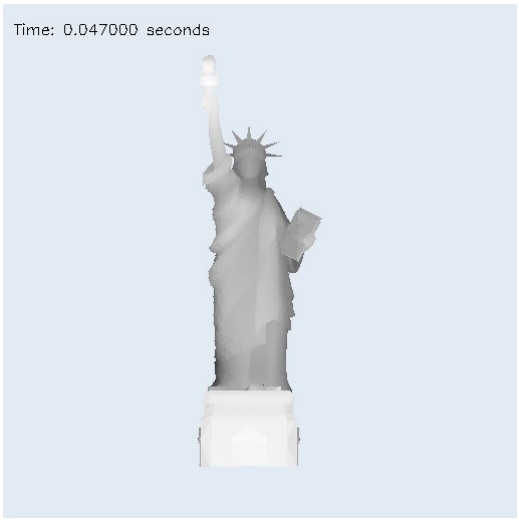
下面是本项目的一些渲染结果图，本项目选择了不同面片数的模型，其渲染结果展示如下：

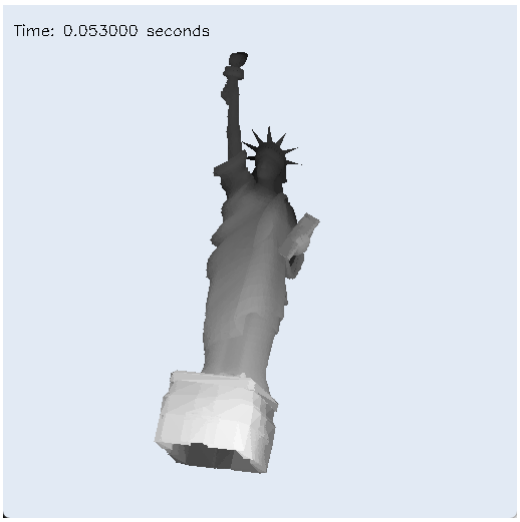
1. bunny 兔子模型：正常和旋转后视图



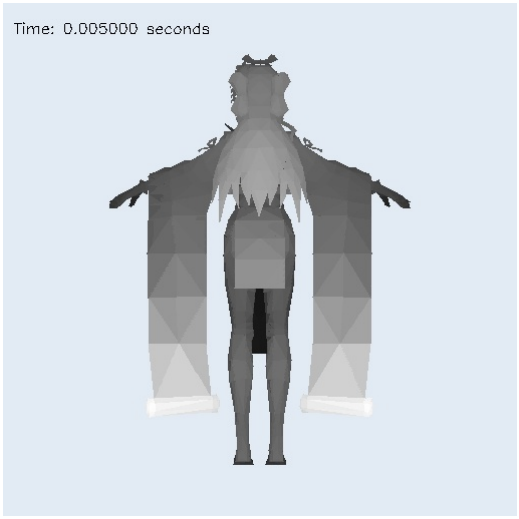


2. 雕像模型：正常和旋转后视图

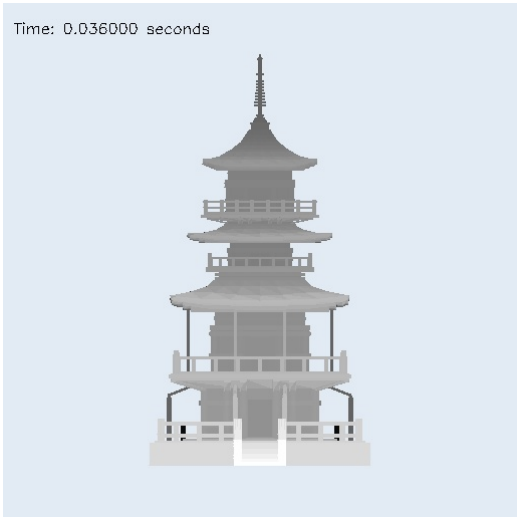




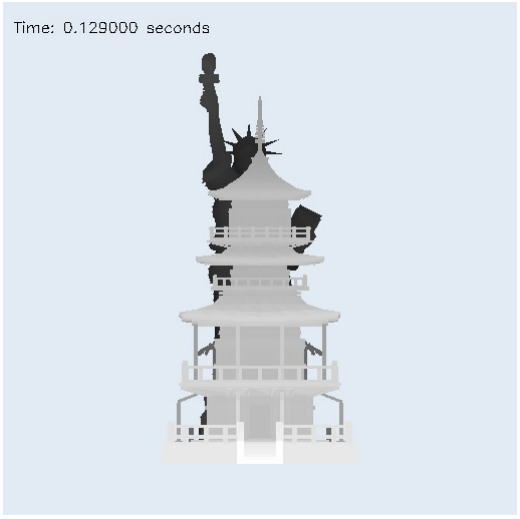
3. 游戏角色模型：



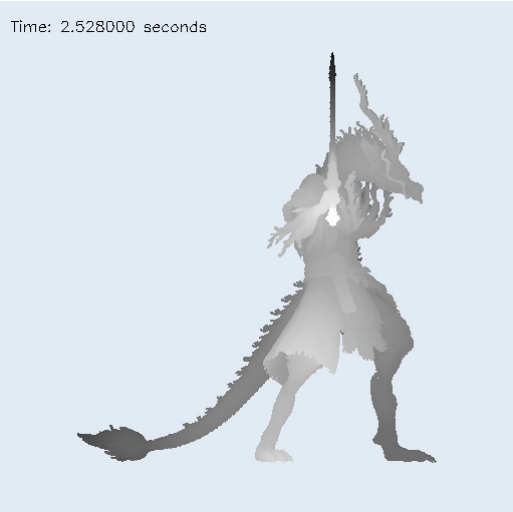
4. 庙宇模型：



5. 复杂遮挡（凭借了前面几个模型）：



6. 《黑神话·悟空》青背龙模型：



3. 项目实现说明

第三方代码声明

本文在读入 obj 文件部分，参考了github仓库的实现：https://github.com/Bly7/OBJ-Loader

由于本文涉及到存在超过 4 条边的面存在的 obj 文件读取，该作者实现并不支持，因此进行了一定的修改，将其数组限制放宽。

其余部分代码独立完成，该部分第三方代码已被放置于 `thirdparty` 文件夹下。

项目结构

```
CG_Project1
|
├── CG_project1: 包含扫描线算法的实现
|
├── hierarchy: 实现层次Z缓冲算法
|
├── hierarchyBVH: 结合BVH和层次Z缓冲算法的实现
|
├── include: 存放头文件
|
├── models: 存放模型的OBJ文件
|
├── output: 存放输出图像
|
└── thirdparty: 参考代码
```

Scanline 算法

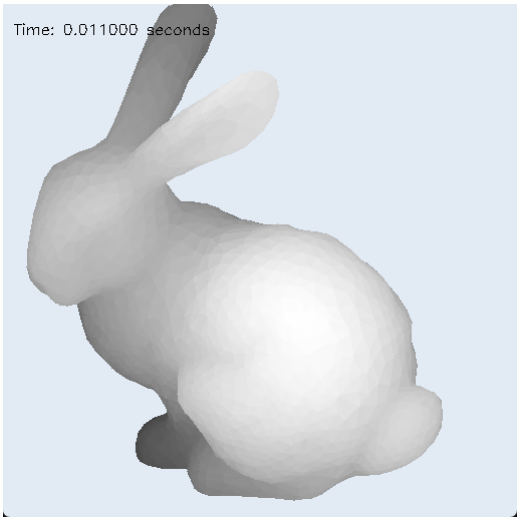
按照课上给出代码，设置了分类多边形表(PT)、分类边表(ET)、活化多边形表(APT)和活化边表(AET)，数据结构如下：

```
struct Polygon
{
    float a, b, c, d; // Polygon plane equation parameters
    int id;    // Polygon identifier
    float dy;     // Polygon height difference
    BoundingBox bbox;
```

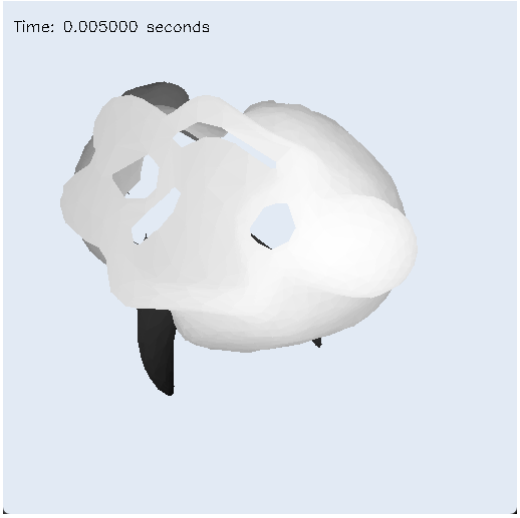
```
    cv::Vec3b color;
    float depth;

    void calculate_params(objl::Vector3 p1, objl::Vector3 p2, objl::Vector3 p3,
objl::Vector3 &n);  // Calculate the parameters of the plane equation
};

struct Edge
{
    float x, dx;
    int dy;
    // Pointer to the polygon containing the edge
    Polygon* id;
};

struct ActiveEdge
{
    float xl, dxl, dyl;
    float xr,dxr,dyr;
    float zl, dzx, dzy;
    Polygon* id;
};

vector<std::vector<Polygon*>> PT;
vector<std::vector<Edge>> ET;
vector<Polygon*> APT, vector<ActiveEdge> AET
```

scanline 算法步骤按照课程 PPT 完成，主要难点在处理超出边界的多边形处理。

本项目在最开始将模型缩放至屏幕空间内，保证模型可以处于渲染结果中。同时，本项目支持模型的旋转，设置旋转矩阵，用于对模型围绕原点进行旋转变换。这一过程采用了 openCV 中的 mouse call 响应机制，来获取鼠标移动距离，这一距离按比例反映在模型的旋转角度。在模型旋转过程中可能存在超出边界的情况，因此本项目在上、下、左右（两边等价）的临界条件上加入了较为细致的判断，经过多次旋转越界测试，本项目在边界情况下具有较好的鲁棒性。



同时，在这一部分，本项目实现简单的面剔除加速支持，通过定义宏定义 `CULL_ENABLE` 控制。主要实现方法是根据法线方向和视角方向，剔除物体背面的网格，但是这个简单的剔除方法会存在的问题是，如果物体不是水密的，在透光处会看到剔除情况。在时间上，启用背面剔除后，对于 bunny 模型，时间从原先的 0.014 s→ 0.006 s，有显著提升。但是因为有瑕疵，且使用后层次 Zbuffer 在单一模型情况下其实不需要剔除任何面片，无法更真切地比较 scanline 算法和其性能，故在后面算法中未采用。



层次 Zbuffer 实现

本项目首先实现了层次 Zbuffer 的简单版本，也就是不对场景中物体进行预排序，仅仅维护一个层次 Zbuffer。由于前面的扫描线算法扫描顺序是从 `ymax` 开始从上往下扫描，它有节省空间的优点，每次扫描只需要维护一行 Zbuffer。但是仅对这一行缓冲进行层次 Zbuffer 实现，效率并不高，不能利用好面片之间的空间局部性。因此本文修改了扫描线算法，按多边形表中面片遍历渲染，同时维护了一个自底向上的层次 Zbuffer，按照 N * N - N/2 * N/2 以 2 为底，自底向上构建，数据结构如下：

```
std::vector<std::vector<float>> z_pyramid;
```

在渲染过程中，自顶向下遍历 zbuffer，直至遇到最小的包含这一多边形面片的层级。如果命中则说明面片被遮挡，不再渲染；若没有命中，则渲染并自底向上更新层次 Zbuffer。

BVH+层次 Zbuffer 实现

本项目随后实现了基于 BVH 树对场景空间物体进行预排序的复杂模式。BVH 建树逻辑为：

1. 随机选择某一个轴（x/y/z），本文设置的选择顺序为 z/x/y，因为优先排序深度(z)有利于剔除；
2. 对空间中物体按照目标轴排序，并在物体总数的一半对应物体处划分空间；
3. 递归划分左右两棵子树；
4. 设定单棵子树最多容纳物体的阈值，低于这一数目（本项目设置为总物体数/500），作为子节点；

BVH节点的数据结构为：

```
struct BVHNode {
    BoundingBox bbox;
    BVHNode* left = nullptr;
    BVHNode* right = nullptr;
    std::vector<Polygon*> polygons; // Leaf nodes contain pointers to polygons

    bool isLeaf() const { return left == nullptr && right == nullptr; }
};
```

4. 项目结果分析

Model	#Polygon	Scanline(s)	层次Zbuffer(s)	层次Zbuffer+BVH(s)
character	3256	0.005	0.014	0.010
bunny	4968	0.014	0.007	0.015
temple	29702	0.036	0.029	0.041
statue	42100	0.075	0.055	0.07
complex	75058	0.129	0.072	0.115
dragon	626504	4.779	2.02	2.389

本项目涉及了 3k ~ 600k的面片数，并进行了三种方法的对比测试。可以看到，在面片数量较少时，scanline算法基本快于层次 Zbuffer 的两种模式（character 快于两种模式，bunny 和 temple 快于复杂模式。随着面片增多，scanline 算法在性能上逐步低于层次 Zbuffer，这一差距在青背龙模型（600k）模型中体现最大。

同时，注意到，层次 Zbuffer 的两种模式在实际测试中区别不大，甚至简单模式会优于复杂模式，这一点笔者一开始有些摸不着头脑，因此笔者设置了被剔除面片的统计，并统计了各个阶段的细节时间：

```
层次Zbuffer + 初始polygon顺序:
Number of Polygons: 626504
Construct Time: 0.245s
cull_cnt: 222203
Scan Time: 1.885s
Time: 2.132s

层次Zbuffer + 刻意从深到浅排序polygon
Number of Polygons: 626504
Construct Time: 0.267s
cull_cnt: 15882
Scan Time: 3.557s
Time: 3.827s

BVH + 层次Zbuffer:
Number of Polygons: 626504
Construct Time: 0.346s
BVH Time: 0.477s
cull_cnt: 227172
Scan Time: 1.672s
Time: 2.499s

BVH + 层次Zbuffer + 刻意排序:
Number of Polygons: 626504
Construct Time: 0.247s
BVH Time: 0.56s
cull_cnt: 227172
Scan Time: 1.698s
Time: 2.508s
```

	初始顺序	刻意排序	BVH + 初始顺序	BVH + 刻意排序
剔除面片数	222203	15882	227172	227172
时间	2.123 s	3.827 s	2.499 s	2.508 s

由此见得，BVH 加速不明显的原因在于建树需要消耗一定时间，而本项目的模型正好在深度排序上利好使用层次 Zbuffer，浅的面片大多在 obj 文件中排于深的面片前，正好产生遮挡。在修改面片按从深到浅的顺序排列后，简单模式的性能显著下降，证明了这一点。在实际应用场景中，涉及多个复杂模型，且模型面片深度没有这样有序的排布时， BVH + 层次 Zbuffer 的复杂模式将优于简单模式，且剔除稳定性更高。