计算机学院实验报告

实验题目： Z-buffering 算法		学号：202000130143
日期：10.16	班级：计科 20.1	姓名：郑凯饶
Email：1076802156@qq.com		

实验目的：
了解消隐的算法思想

实验环境介绍：
Dell Latitude 5411
Intel(R) Core(TM) i5-10400H CPU @ 2.60GHz(8GPUs),~2.6GHz
Windows 10 家庭中文版 64 位（10.0，版本 18363）
Visual Studio 2022

解决问题的主要思路：
在实验 2 中多边形扫描转换的扫描线算法的基础上，实现了扫描线 Z-Buffer 算法：
1. 定义重要数据结构：
（a）多边形，每个多边形自身对应一个 AET 和 NET，不同于书本的算法，仅使用一个活化边对表（AET），通过牺牲一定的空间减少边配对后再更新的复杂讨论（可能要存储额外信息得不偿失）

```
struct poly {
    int a, b, c, d;   // 多边形所在平面方程
    std::vector<point> p; // 顶点投影坐标（x，y）
    int Ymin, Ymax;
    std::list<edge> AET;                    // 活化边表
    std::vector<edge> NET[window_height + 5]; // 新边表
    tuple color; // 填充颜色

    void get_NET();
} polygon[15];
```

（b）多边形 Y 表

```
std::vector<poly> Y_table[window_height + 5];    // 多边形 Y 表
```

（c）活化多边形表

```
std::list<poly> APT;  // 活化多边形表
```

2. 算法伪代码
其实只要在扫描线算法上，增加对 APT 的相关操作即可以得到目标算法。

```
Z_Buffer() {
    建立多边形 Y 表，初始化 APT
```

```
    for（自下而上扫描，对于每条扫描线 i）{
        初始化帧缓存以及深度缓存
        将对应扫描线 i 的，多边形 Y 表中的多边形加入到 APT 中
        for（对每个多边形）{
            1.依据 NET 更新其 AET
            2.对（单个多边形的边与扫描线的）交点集合排序并进行配对，存储至 to_fill 之中（vector）
            3.根据 to_fill 更新帧缓存以及 Z-Buffer，根据平面方程 ax+by+cz+d=0（默认 c=1）计算深度多边形在该点处的深度 z，若 z>ZB[j]则更新
            4.更新当前多边形的 AET，删除 y_max=Y 的活化边
        }
        输出帧缓存
        更新 APT，删除 y_max=Y 的活化多边形
    }
}
```

时空复杂度分析：
Z-Buffer 算法：对每个多边形进行扫描转化，假设通过扫描线算法实现，获得 NET 为 $O(E)$，更新 AET 获取其点阵表示为 $O(S)$，S 为多边形的面积，$O(S) = O(WH)$，其中 W，H 为屏幕宽高参数，设有 n 个多边形对象，则扫描转化耗时为 $O\big(n(E + WH)\big)$. 扫描同时代入平面方程计算对应像素点的深度 Z 更新 Z-Buffer，用时为 $O(1)$，因此总的时间复杂度为 $O(\sum E + nWH)$。但是空间上要存储多边形的点阵表示以及 2d 的 Z-Buffer 为 $O((n+1) * WH)$.
扫描线 Z-Buffer 算法：将 Z-Buffer 设置为一维数组，同时免去对各个多边形的点阵表示进行存储相应空间变为 $O(W)$，时间在大 0 表示上没有优化，但是读写的对象减小后势必带来时间性能的提升。

实验步骤：

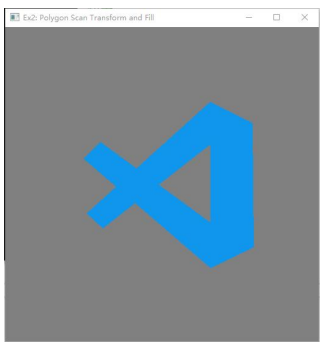
1. 通过函数 set_polygon() 设置多边形的平面方程、顶点坐标、填充颜色等等信息，以测试 Z-Buffer 算法的消隐效果；

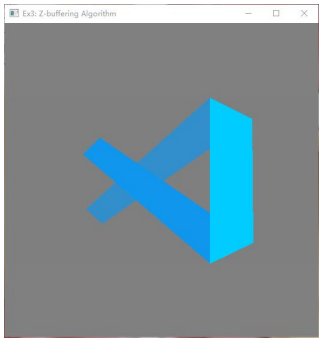
2. 将实验 2 的 get_NET()方法封装至多边形对象 poly 中；

3. 在实验 2 的 polygon_fill 方法的基础上增加对 APT 的相关操作，对每个多边形对象都进行 get_NET()和 AET 的相关操作更新帧缓存以及深度缓存，之后输出帧缓存，完成消隐效果绘制。

实验结果展示及分析：

书接上回，在原有多边形的基础上增加消隐，绘制 vscode 软件的图标：





消隐前　　消隐后　　官方原画

可以看出，消隐后各个多边形之间由于深度不同相互遮挡，视觉上更加立体感。

实验中存在的问题及解决：

```
for (poly &mpoly : APT) { // 取引用，对 mpoly 存在修改
对全局结构体变量 mpoly 进行修改
}
```

这里使用了 C++11 的特性，一开始我并没有加引用，程序运行会发生异常。因为这样写相当于创建一个临时变量，所有修改都是对这个临时变量进行的，循环结束变量会自动销毁，并没有影响全局变量 APT。正确写法应该是取引用，所有操作对 APT 进行。