


计算机学院实验报告

实验题目： 反走样算法		学号： 202000130143
日期： 11.4	班级： 计科 20.1	姓名： 郑凯饶
Email: 1076802156@qq.com		
实验目的： 了解反走样的算法思想，使用反走样技术处理简易模型边缘的锯齿感		
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： 通过区域采样和加权区域采样方法实现反走样，减少用离散量表示连续量引起的失真现象，如边缘呈锯齿状、图形细节失真： 设一个显示像素点的颜色值为 c ，最顶层图形的为 c_{top} ，背景色为 c_{bg} ，图形覆盖像素点的面积占像素点面积的比例为 k ，基于区域采样方法可以通过一下公式确定显示颜色值： $c = k * c_{top} + (1 - k)c_{bg}$ ，即为图形色和背景色的线性组合。 然而，以上未考虑相交区域在像素内的位置，仍然会导致锯齿效应，故提出加权区域采样，“权”指的是区域 S 与像素中心的距离 d ，公式为 $c = \int f(d)dS * c_{top}$ 为了方便计算，选用离散化方法：将屏幕像素分为 $n * n$ 的子像素，计算落在图形内的子像素个数 k ，最终像素显示颜色为 $\frac{1}{n} \sum c_i * W_i * bool_{in}$ ，其中 W_i 为子像素对应权重。 <div style="display: flex; align-items: center; justify-content: center;">  .* <div style="border: 1px solid gray; padding: 10px; text-align: center;"> $\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ </div> → <div style="border: 1px solid blue; padding: 20px; text-align: center; width: 80px; height: 80px;"> ? </div> </div> 实现时取 $n = 3$, 加权表为 $\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$.		

实验步骤：

1. 将实验 3 的扫描线 Z-buffering 算法稍微做修改，将帧缓存变为二维作为反走样算法的输入；

2. 封装颜色对象，重载相关运算符：

```
struct tuple {
    int a, b, c;
    //tuple (int a, int b, int c) : a(a), b(b), c(c) {}
    tuple operator + (const tuple& x) { // 初次尝试
        return { a + x.a, b + x.b, c + x.c };
    }
    tuple operator * (const int x) {
        return { a * x, b * x, c * x };
    }
    tuple operator / (const int x) {
        return { a / x, b / x, c / x };
    }
    bool operator == (const tuple x) {
        return ((a == x.a) && (b == x.b) && (c == x.c));
    }
};
```

3. 定义矩阵点乘，通过子像素和像素的对应关系进行离散变换：

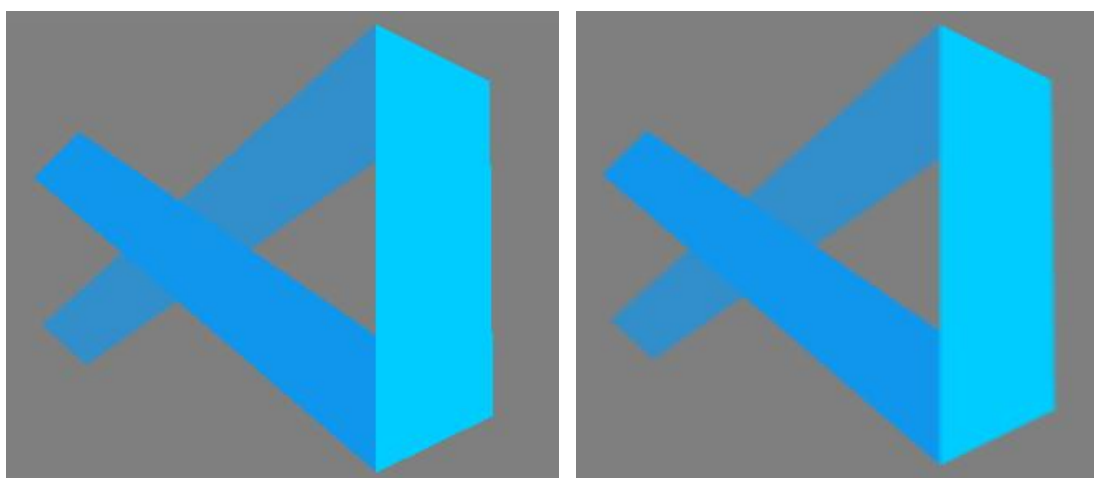
```
// 反走样
for (int i = 1; i <= window_height / 3; i++) {
    for (int j = 1; j <= window_width / 3; j++) {
        // 定义颜色三元组运算
        CB_smoothed[i][j] = matpow(&CB[3 * i - 1][3 * j - 1], window_width);
    }
}
```

实验结果展示及分析：

书接上回，在消隐结果的基础上进行反走样优化，效果对比如下，可以看见边缘锯齿现象明显减弱，图形更加大方美观，贴近官方的图标设计。

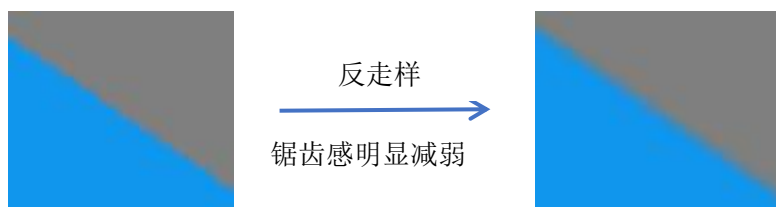


官方原画



反走样前

反走样后



实验中存在的问题及解决:

重定义运算符后, 使用时遇到问题:

error C2677: 二进制 “*” : 没有找到接受 “tuple” 类型的全局运算符(或没有可接受的转换)

原因是调用时写成 `int*tuple` 的形式, 改为 `tuple*int` 通过编译, 重定义后双目运算符参数的输入顺序不可变!