

# hw2: 多边形填充与扫描线算法实现

康艺潇 518431910002

## 1. 扫描线算法

### 算法描述

思路：扫描线沿着y轴从下向上，确定各像素上的场景可见点并计算其显示的光亮度。

由于多边形千变万化，要想填充多边形内部的所有像素，需要找到一种合适的规则，能够沿着一个方向，一个像素不漏地把多边形内部填满，同时不污染多边形外部。于是我们发明了一条水平方向的扫描线，它从 $y=0$ 开始，判断与多边形的交点，这些交点把扫描线分成了若干段，我们需要判断哪些段在多边形内部，哪些段在多边形外部，然后把内部的部分着色，完成后，令 $y=y+1$ ，即扫描线上移一格，重复之前的操作，直到扫描线不再与多边形的任何部分相交。

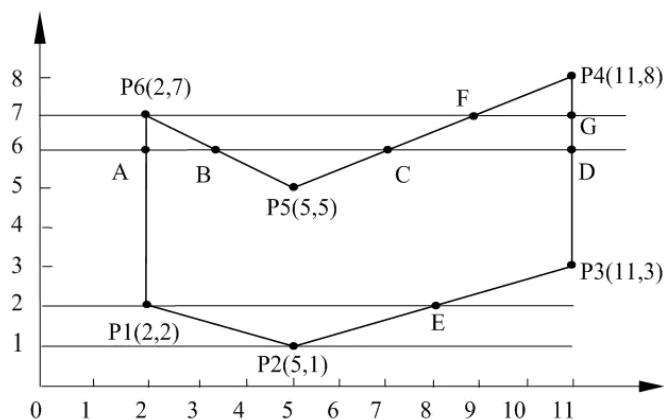
### 举例

下图所示为多边形P1P2P3P4P5P6，而且同时画出了扫描线扫描过程中经过的四个位置 $y=1$ 、 $y=2$ 、 $y=6$ 和 $y=7$ 。

扫描线算法的难点在于如何判断扫描线被多边形分割后哪些部分在多边形内部，哪些部分在多边形外部。

让我们仔细观察上图，答案就在图中。对于未经过顶点的扫描线，如 $y=6$ ，总是与多边形有偶数个交点，而且位于多边形内部的片段和位于多边形外部的片段交替存在。对于经过了顶点的扫描线，如 $y=1$ 、 $y=2$ 和 $y=7$ ，与多边形的交点既可能是偶数，也可能是奇数。但是如果我们将进一步划分，这些经过了顶点的扫描线有些经过了极值顶点，如 $y=1$ 和 $y=7$ ，它们的交点个数是奇数；而有些经过了非极值顶点，如 $y=2$ ，它们的交点个数是偶数。这样的话，不如做一个特殊处理，把所有极值顶点当成两个点，就可以保证扫描线与多边形的交点总是偶数。

当然，把交点个数凑成偶数是有意义的，凑成偶数后就可以把这些交点从左到右两两配对，配对成功的两个点之间的像素全部着色。例如，上图的扫描线 $y=6$ 与多边形的交点序列是ABCD，从左到右两两配对为AB和CD，然后将AB之间着色，CD之间着色。



伪代码

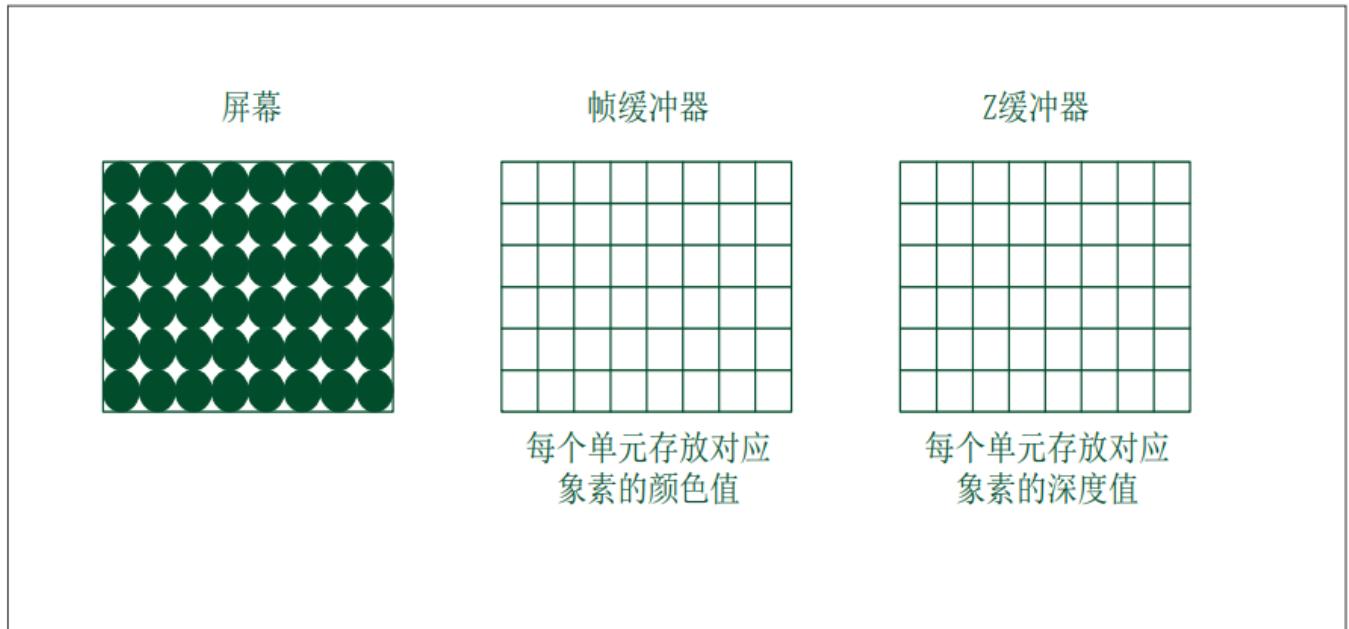
```

for(每条扫描线) {
    将扫描线帧缓存fbuffer中的值置为背景色;
    将扫描线zbuffer的值置为最小值;
    for(每个多边形) {
        求出该多边形与当前扫描线相交区间;
        for(区间中的各个像素) {
            if(多边形在该点处的Z值大于zbuffer的值) {
                zbuffer在该处的值=多边形在该处的Z值;
                fbuffer在该处的值=多边形在该处的亮度值;
            }
        }
    }
}

用fbuffer的内容显示当前扫描线;
}

```

## Z-Buffer



算法思想：先将Z缓冲器中各单元的初始值置为最小值。当要改变某个像素的颜色值时，首先检查当前多边形的深度值是否大于该像素原来的深度值（保存在该像素所对应的Z缓冲器的单元中）如果大于原来的z值，说明当前多边形更靠近观察点，用它的颜色替换像素原来的颜色。

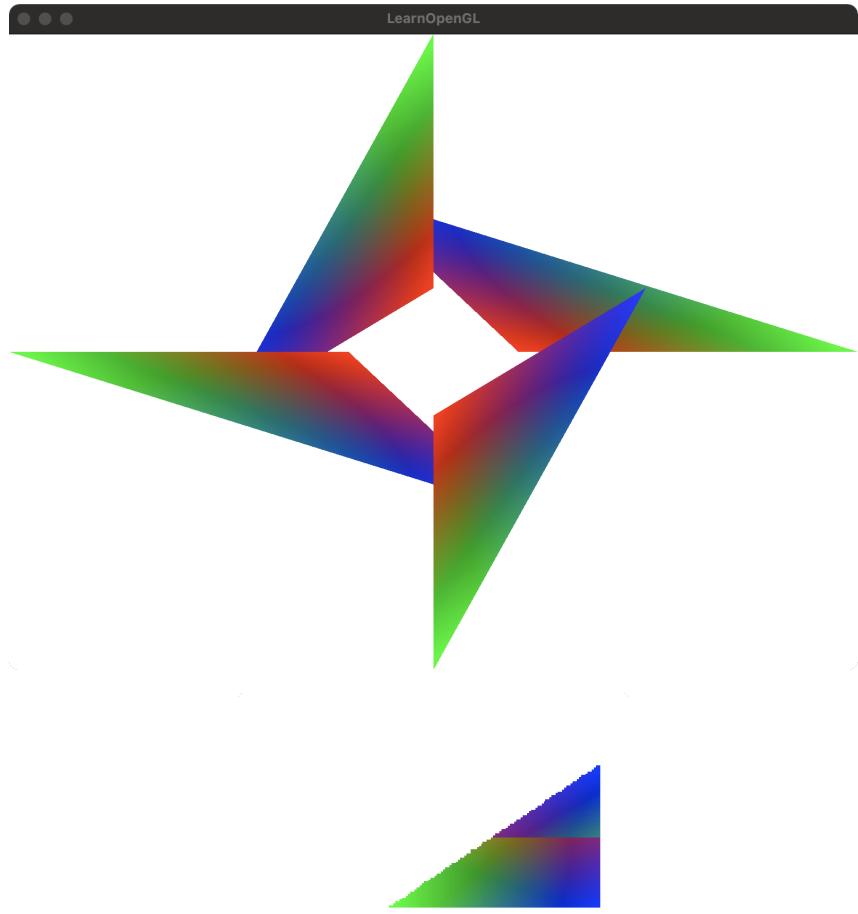
本项目中设置了CBuffer(保存颜色信息) ZBuffer(保存深度信息)

## 2. Part 1 三角形

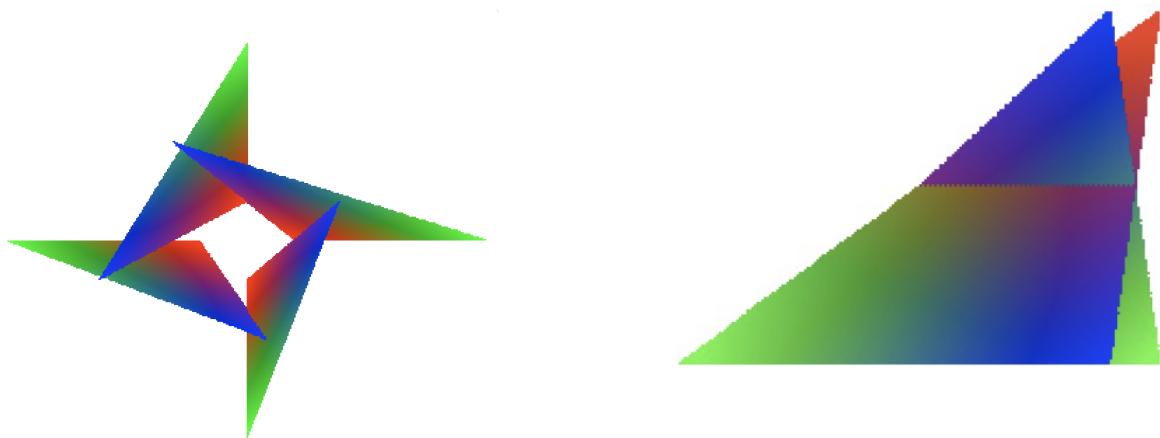
ifstream 从文件读取三角形参数。

构造的类：Vertex, line, Shape, Triangle, Rectangle。其中Triangle 和Rectangle 继承自shape类。

错误的覆盖关系



正确的关系

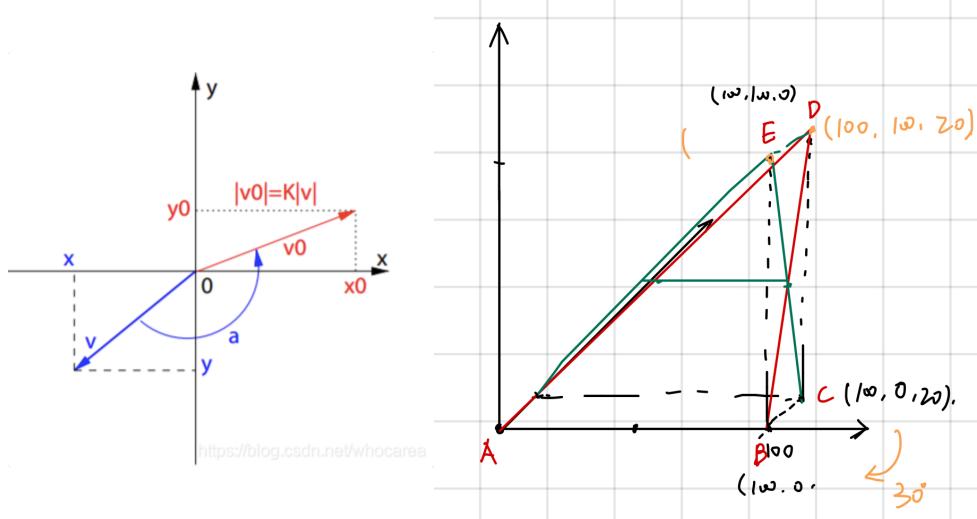


算法：依次输入每个点的颜色和深度，当扫描线y和边有相交时，先算出两个交点的颜色和深度信息，再用插值法得到整个线段上的点的颜色和深度信息。将这些点压入vector 返回。之后与buffer里面的点的深度比较，如果深度更小，就更新buffer。

在第二个三角形画完之后是重叠的，需要绕OZ轴旋转，其实是左乘旋转矩阵，写成坐标描述为：

$$x_1 = x_0 \cos(\alpha) - z_0 \sin(\alpha)$$

$$z_1 = z_0 \cos(\alpha) + x_0 \sin(\alpha)$$



为了得到目标效果，需要将图形绕z轴旋转一定角度，这里顺时针旋转30度，于是就有

$x_1 = x_0 \cos(30^\circ) - z_0 \sin(-30^\circ) \quad z_1 = z_0 \cos(30^\circ) + x_0 \sin(-30^\circ)$

```
void rotate(Vertex &v){
    v.x=v.x*1.732/2-v.z*1/2;
    v.z=v.z*1.732/2+v.x*1/2;
}
```

### 3. Part2 矩形的编织效果填充算法

和三角形的扫描线算法相似，但是有几个不同点：

1. 构造函数：矩形和三角形的构造函数不同，他们继承的shape类里面需要存储每个点和边的信息，三角形只要取两个点连接成边就可以，矩形需要去除对角线，因此在构造的时候，顺时针传入各个点，边的构造是<1,2><2,3><3,4><4,1>，否则会两个对角的三角形。

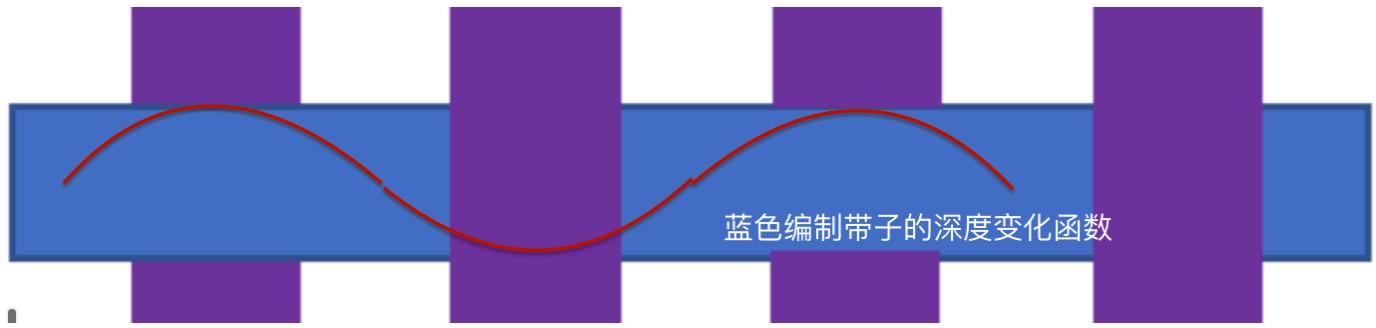


2. 编制效果的实现

1. 要实现交错的编制效果，同时每个是完整的矩形，那整个编制带的深度就是不一样的。同时在交错的地方有规律性的出现深度大小的交叠。因此我选用了三角函数，在一个带子取到深度最高的地方，另一个带子取到深度最低。

其中三角函数的周期  $T = \text{两个同为上面的带子的间距}$

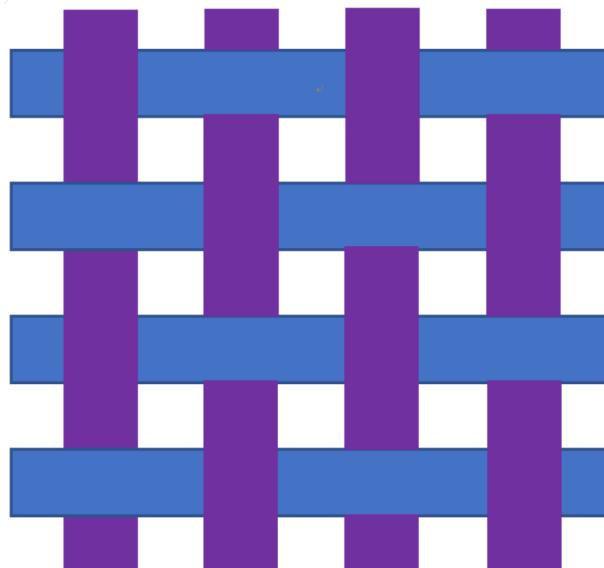
$$depth = A * \sin(2\pi x/T)$$



编制效果

经纬方向的矩形颜色分别为 (112, 48, 160), (68, 114, 196)

经纬矩形间距80 pixel, 背景色为 (255, 255, 255)



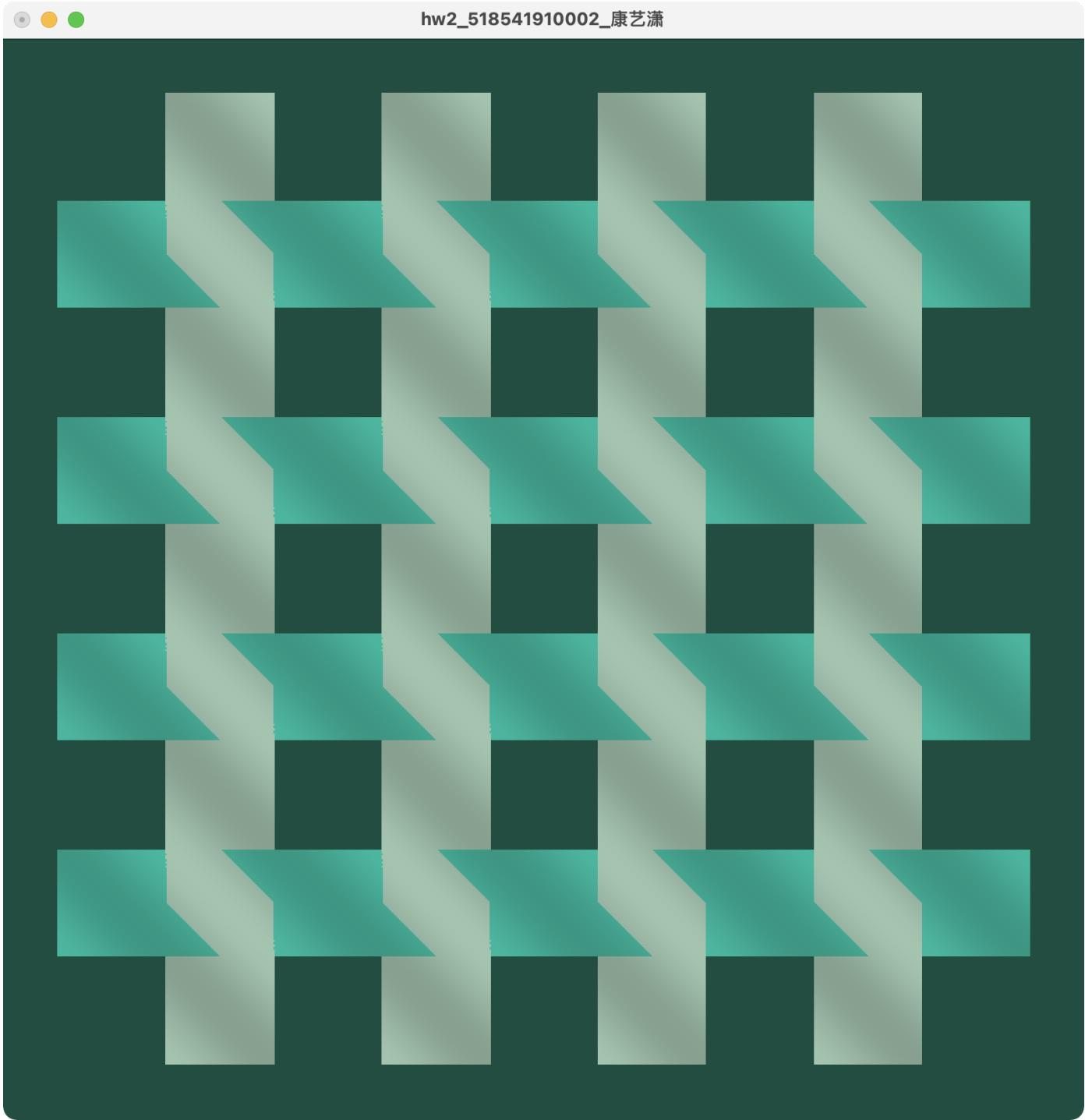
phase3.1

变化版本

1. 可以用亮度表示深度，比如深度深的地方颜色暗一些。因此在计算颜色采用

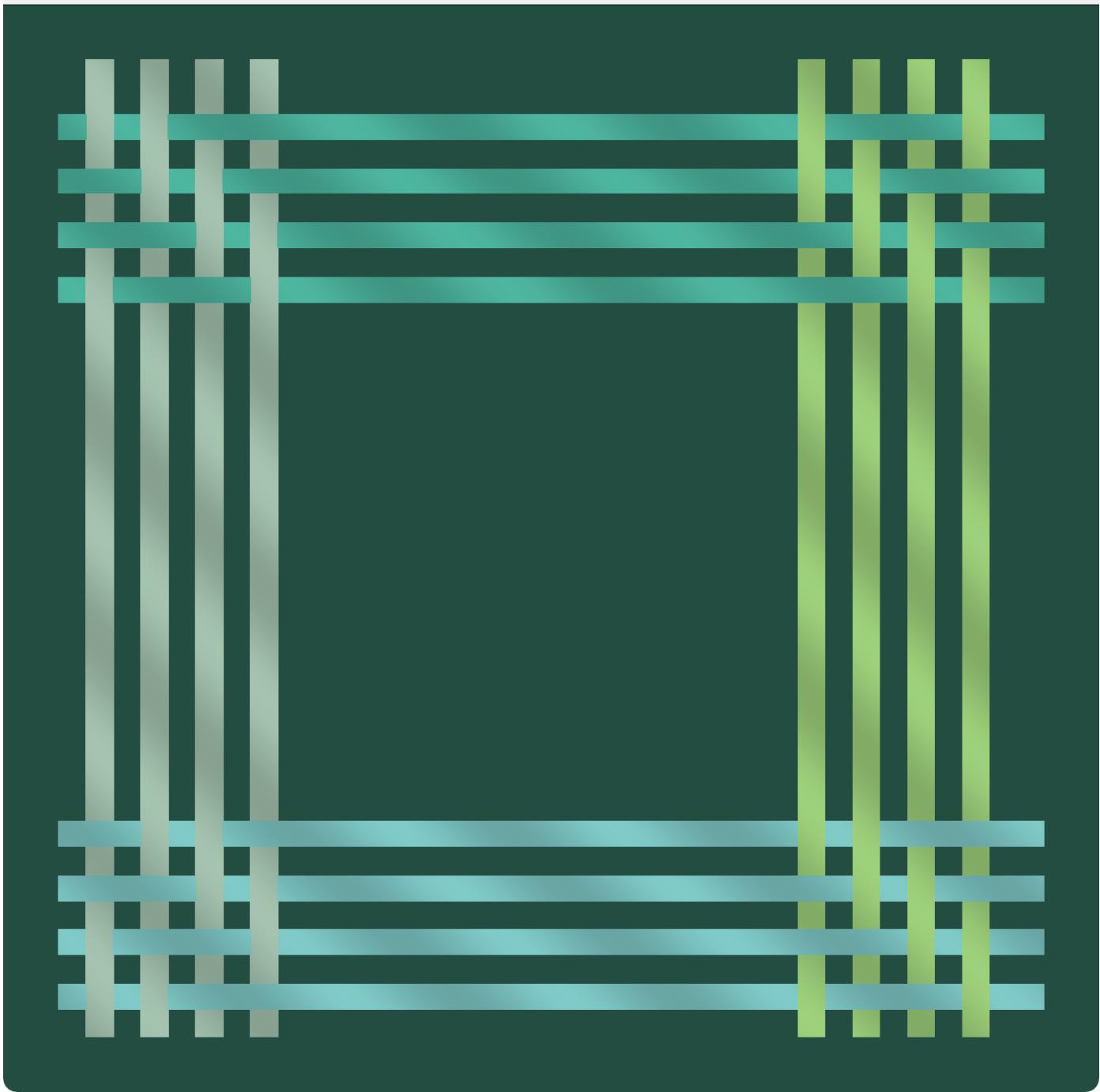
$$Color = ColorOriginal(1 - \alpha \times Depth/maxDepth)$$

1. 在计算深度时，让一个短截线上的深度也发生变化，这样就出现了更有趣的剪开的效果



phase 3.2

1. 采用4组带子，各不一样，相互交叉



phase 3.3

## 库和函数

使用了<GL/freeglut.h>库

函数说明

**void glBegin(GLenum mode)**

glBegin()是和glEnd()结合起来使用。

mode:创建元素的类型，比如：点，线等。可以是以下数值：

GL\_POINTS: 把每个顶点作为一个点进行处理，顶点n定义了点n，绘制N个点。

**GL\_LINES:** 把每个顶点作为一个独立的线段，顶点 $2n-1$ 和 $2n$ 之间定义了 $n$ 条线段，绘制 $N/2$ 条线段

**GL\_LINE\_STRIP:** 绘制从第一个顶点到最后一个顶点依次相连的一组线段，第 $n$ 和 $n+1$ 个顶点定义了线段 $n$ ，绘制 $n-1$ 条线段。

**GL\_LINE\_LOOP:** 绘制从第一个顶点到最后一个顶点依次相连的一组线段，然后最后一个顶点和第一个顶点相连，第 $n$ 和 $n+1$ 个顶点定义了线段 $n$ ，绘制 $n$ 条线段。

**GL\_TRIANGLES:** 把每个顶点作为一个独立的三角形，顶点 $3n-2$ ,  $3n-1$ 和 $3n$ 定义了第 $n$ 个三角形，绘制了 $N/3$ 个三角形。

原文链接：<https://blog.csdn.net/aa941096979/article/details/50843596>

### **glColor3f (float r, float g, float b)**

在OpenGL中设置颜色，一般可以使用glColor3f()。从函数名字就可以看出，它的参数应该有三个，类型是float型的。另外一点是它的参数值的范围是[0.0,1.0]一般的，可以将这三个参数值视为颜色的成分。

需要注意的是，如果在glBegin()与glEnd()函数之间多次连续调用颜色函数，那么，只会显示出最后一次的颜色，例如：

```
glBegin(GL_POINTS)
    glColor3f(0.0, 1.0, 0.0); //绿色
    glColor3f(1.0, 1.0, 0.0); //黄色
    glVertex(0.25, 0.75, 0.0);
glEnd();
```

那么，画出来的这条线只是黄色的。

### glVertex 指定点

如何指定一个点呢？OpenGL提供了一系列函数。它们都以glVertex开头，后面跟一个数字和1~2个字母。例如：

glVertex2d

glVertex2f

glVertex3f

glVertex3fv

等等。

数字表示参数的个数，2表示有两个参数，3表示三个，4表示四个（我知道有点罗嗦~）。

字母表示参数的类型，s表示16位整数（OpenGL中将这个类型定义为GLshort），

i表示32位整数（OpenGL中将这个类型定义为GLint和GLsizei），

f表示32位浮点数（OpenGL中将这个类型定义为GLfloat和GLclampf），

d表示64位浮点数（OpenGL中将这个类型定义为GLdouble和GLclampd）。

v表示传递的几个参数将使用指针的方式，见下面的例子。

这些函数除了参数的类型和个数不同以外，功能是相同的。例如，以下五个代码段的功能是等效的：

(一) glVertex2i(1, 3);

- (二) glVertex2f(1.0f, 3.0f);
- (三) glVertex3f(1.0f, 3.0f, 0.0f);
- (四) glVertex4f(1.0f, 3.0f, 0.0f, 1.0f);
- (五) GLfloat VertexArr3[] = {1.0f, 3.0f, 0.0f};

### **glVertex3fv(VertexArr3);**

以后我们将用glVertex\*来表示这一系列函数。

注意：OpenGL的很多函数都是采用这样的形式，一个相同的前缀再加上参数说明标记

假设现在我已经指定了若干顶点，那么OpenGL是如何知道我想拿这些顶点来干什么呢？是一个一个的画出来，还是连成线？或者构成一个多边形？或者做其它什么事情？

为了解决这一问题，OpenGL要求：指定顶点的命令必须包含在glBegin函数之后，glEnd函数之前（否则指定的顶点将被忽略）。并由glBegin来指明如何使用这些点。

例如我写：

```
glBegin(GL_POINTS);
    glVertex2f(0.0f, 0.0f);
    glVertex2f(0.5f, 0.0f);
glEnd();
```

则这两个点将分别被画出来。如果将GL\_POINTS替换成GL\_LINES，则两个点将被认为是直线的两个端点，OpenGL将会画出一条直线。

reference: [http://blog.sina.com.cn/s/blog\\_986c47a70101hujw.html](http://blog.sina.com.cn/s/blog_986c47a70101hujw.html)

### **glFlush()**

glFlush()是OpenGL中的函数，用于强制刷新缓冲，保证绘图命令将被执行，而不是存储在缓冲区中等待其他的OpenGL命令。

## **配色**

---

采用日系配色

NIPPON COLORS - 日本の伝統色

<https://nipponcolors.com/#aimirucha>

藍海松茶  
AIMIRUCHA

MUNSELL  off

Copyright © 2010 ONO TAKEHIKO. Some Rights Reserved. HOSTED BY [hetem!](#)  
Color data cited: "日本の伝統色 The Traditional Colors of Japan". PIE BOOKS, 2007.



## reference

[扫描线算法完全解析](#)

[计算机图形学（三）扫描线多边形填充算法讲解与源代码菜鸟CSDN博客计算机图形学多边形填充代码](#)

[drawline\\_\(1\).rar](#)

[Z-buffer算法](#)

<https://www.youtube.com/watch?v=fcMqynrVao8>

[单色多边形](#)

[OpenGL自带的深度检测](#)

<https://learnopengl.com/Advanced-OpenGL/Depth-testing>

配色

[NIPPON COLORS - 日本の伝統色](#)