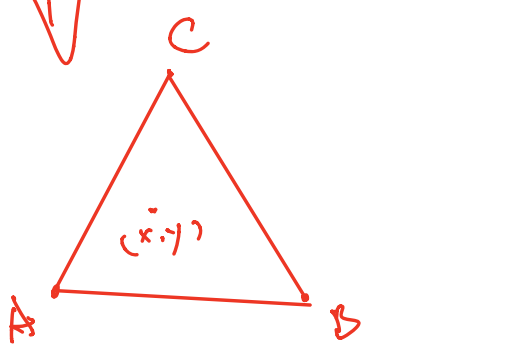
Compute Barycentric 2D 计算2D重心坐标.

w_reciprocal w 倒数.

z_interpolated z 内插值 (使用重心坐标)

$(x, y) = \alpha A + \beta B + \gamma C$

$\alpha + \beta + \gamma = 1.$



① 获取三角形三个顶点数据

auto v = t.toVertex4C();

⇓

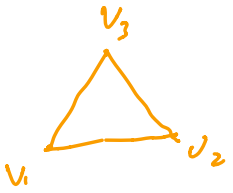
std::array<Eigen::Vector4f, 3UL>

数组.

v = { Vector4f, Vector4f ···· }

② 计算 bounding Box (包围盒)

- 获取三角形的顶点数据 (v1, v2, v3)

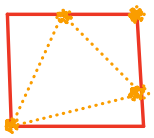


- 获取最大的 y值 x值
  最小的 y值 x值

⇓

- 循环从 $x_{min} \to x_{max}$

  $y_{min} \to y_{max}$



这会是一个矩形的范围

内的 (x, y) 点遍历

③ 接口获得 V 数组 Vector 数据后

不知数组有多少个点

所以用 数组长度

Sizeof(V)

除以

Sizeof(Eigen::Vector4f)

获得

Sizeof(V)

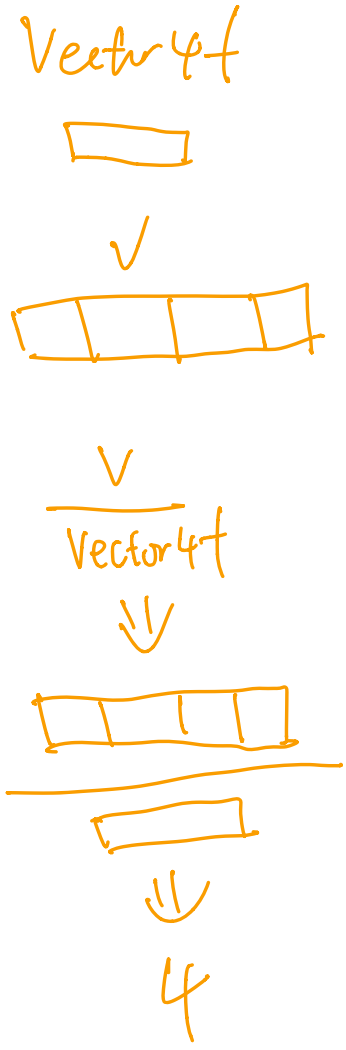
Sizeof(Vector4f)

顶点数量(多少个)

再根据它，遍历数量

for(数量)

Array(Vector3) <= (V[index].x(), V[index].y(), 1.0)



比较 获取最大 最小 x, y

Std: minmax_element (Array (Vector3))

$x_{befor} < x_{later}$

↓

xmin ymin
xmax ymax.

依此不 xmin → xmax
ymin → ymax

① α, β. γ = 计算2D重心坐标 (x, y, √)

② 计算w的倒数, 三个比较、加权计算

$$\frac{1.0}{\frac{\alpha}{V[0].w}+\frac{\beta}{V[1].w}+\frac{\gamma}{V[2].w}} = \frac{V[0].w}{alpha}+\frac{V[1].w}{beta}+\frac{V[2].w}{gama}$$

w_reciprocal

③ 计算 z 的插值

$$\alpha \cdot \frac{V[0].z}{V[0].w} + \beta \cdot \frac{V[1].z}{V[1].w} + \gamma \cdot \frac{V[2].z}{V[2].w} =$$

z_interpolated.

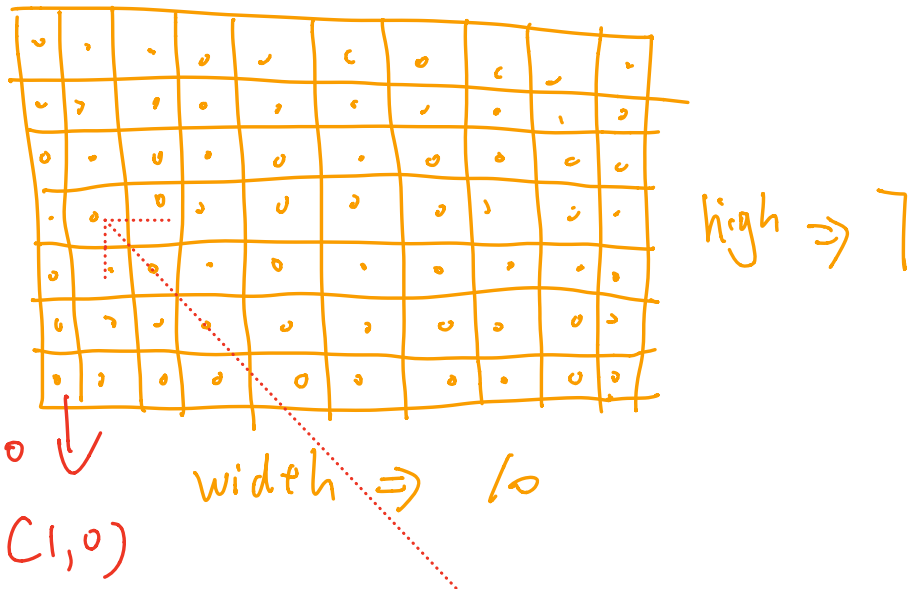
④ 最后获得深度 z 值.

z插值 = z内插 乘 w倒数.

z_interpolated *= w_reciprocal.

8 计算像素索引.

```
int index = x + width * y
```



三行二列

$2 + 10 \cdot 3 = 32$

⑥ 判断像素是否在三角形内.

Vector3f pt = (float)x, (float)y, 0.0f

↓ ↙

像素坐标.

Array<Vector3f> V = {point1, point2

point3}.

三角形3顶点

pt V[0] V[1] V[2]

pt V1 V2 V3

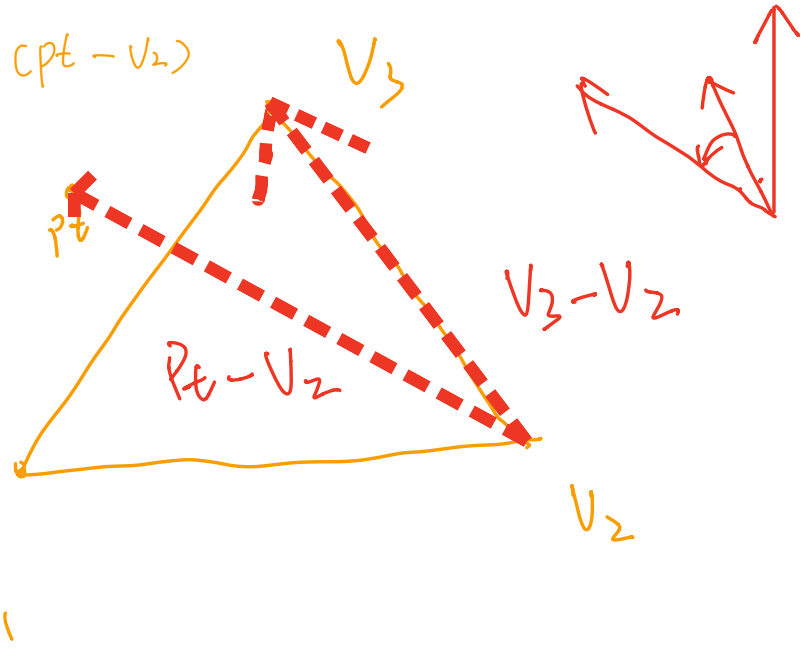
(V3 - V2) cross (pt - V2)

(V3 - V2) cross (V1 - V2)

$(V_3 - V_2)$ cross $(pt - V_2)$

⇓

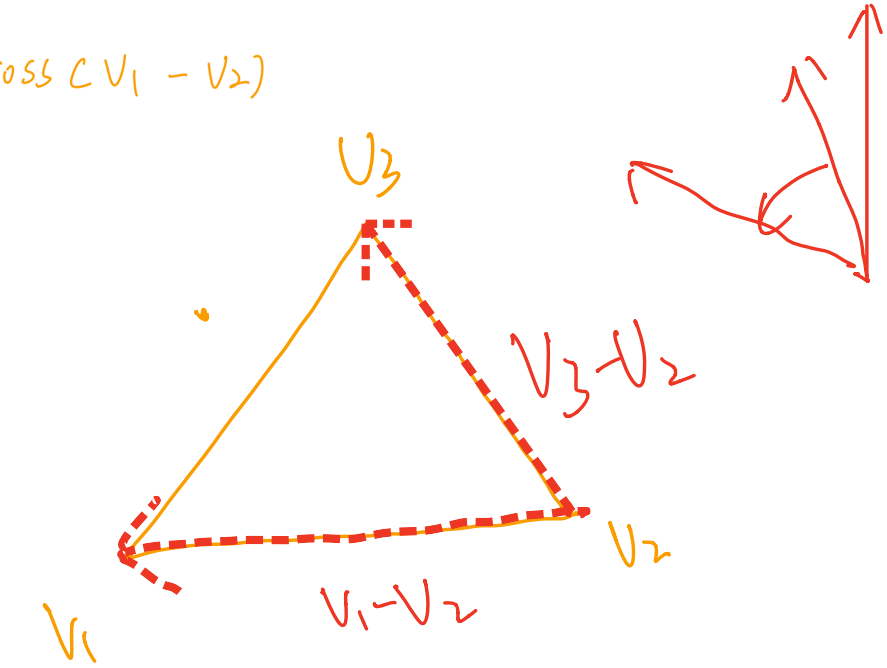
cp1



$(V_3 - V_2)$ cross $(V_1 - V_2)$

⇓

cp2



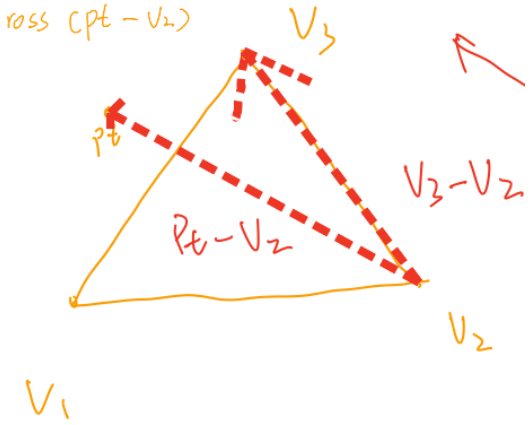
cp1 dot cp2 = ?

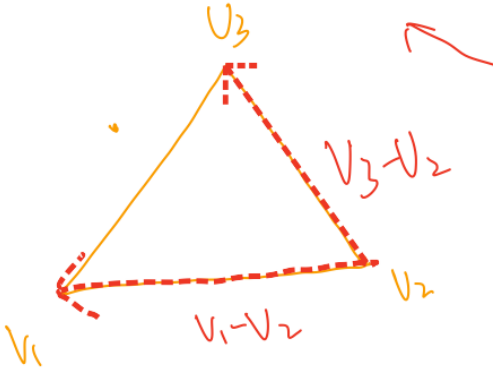
点乘

同方向 ⇒ 大于0

反方向 ⇐ 小于0

★ 即 $P_t - V_2$ 在 $V_3 - V_2$、$V_1 - V_2$ 之间

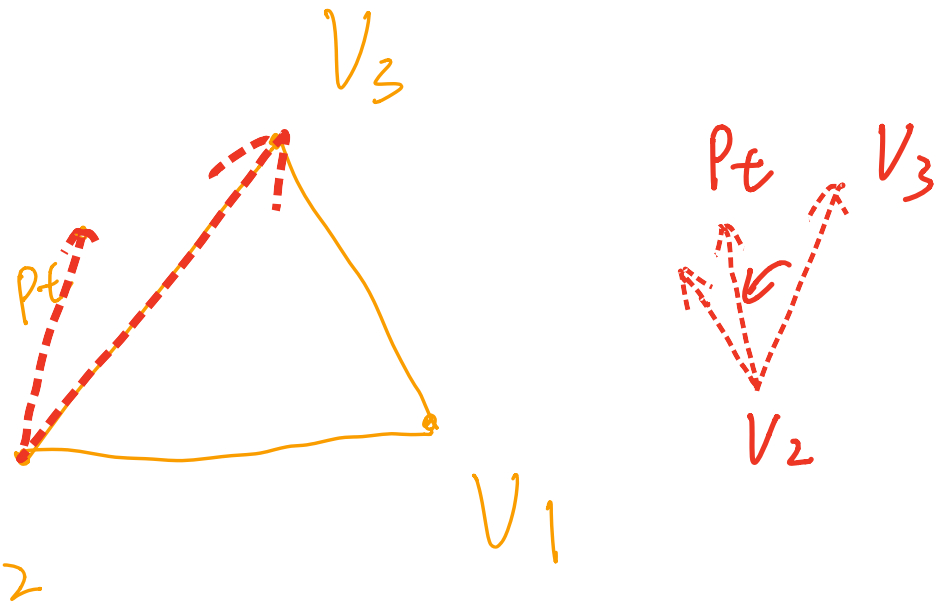




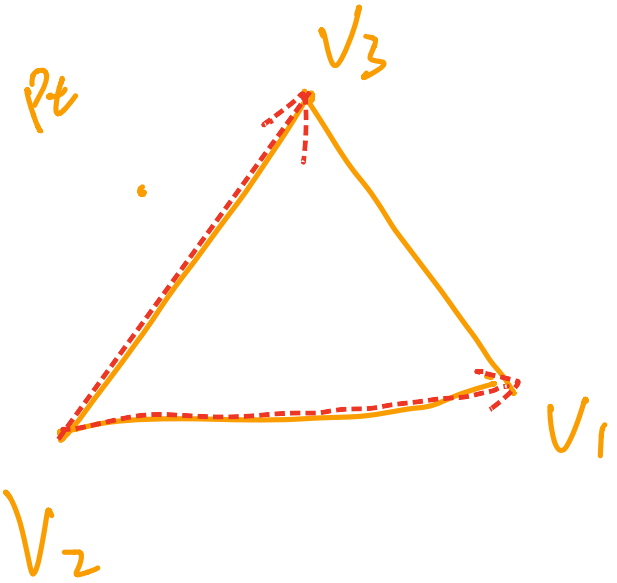
	V[1]	V[0]	V[2]
Pt	V_1	V_2	V_3

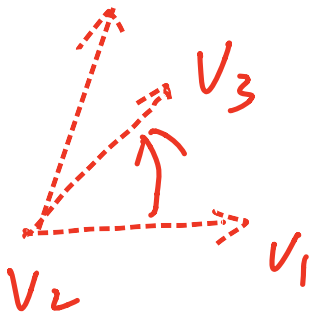
$(V_3 - V_2)$ cross $(Pt - V_2)$

$(V_3 - V_2)$ cross $(V_1 - V_2)$



$(V_3 - V_2)$ cross $(Pt - V_2)$





(V2 - V2) Cross (V1 - V2)

(V3-V2) Cross (Pt - V2) = Pt1

(V3-V2) Cross (V1 - V2) = Pt2

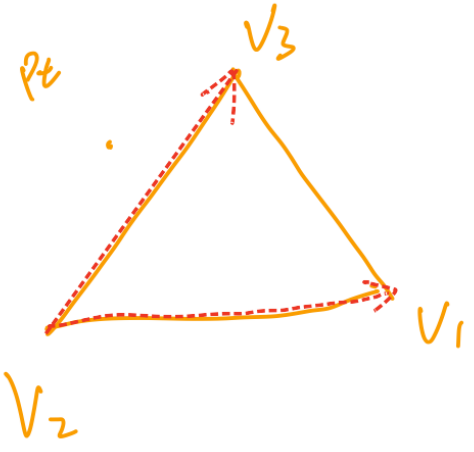
Pt1 ↗

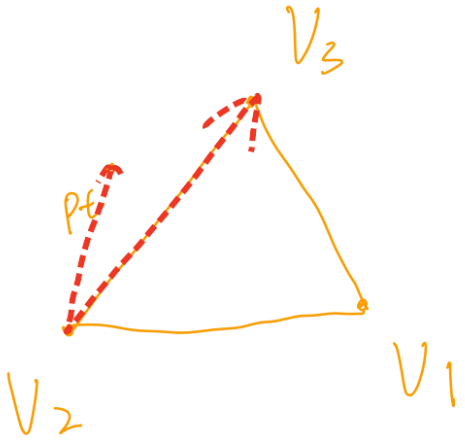
Pt2 ↙

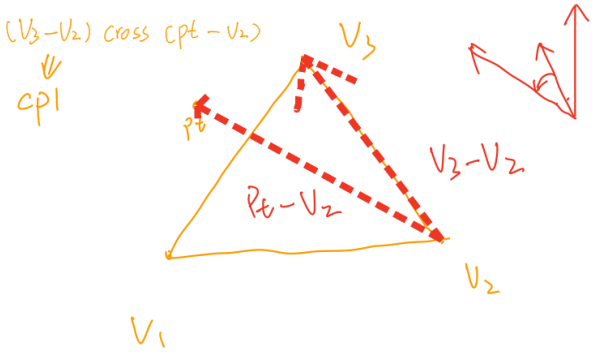
Pt1 dot Pt2 < 0

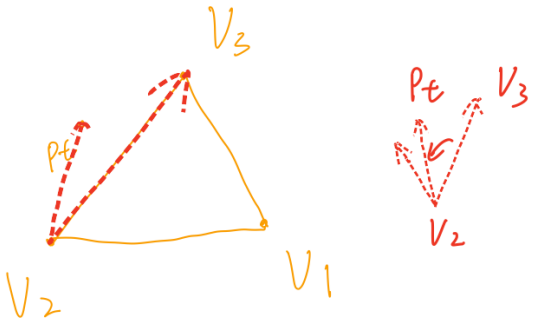
同方向大于
反方向小于

所以 V2Pt 不在 V2V3 - V2V1 之间

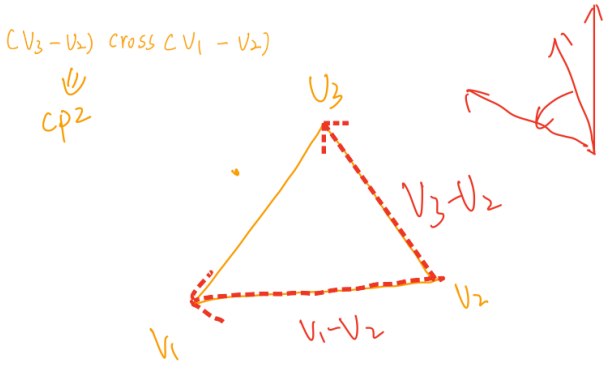


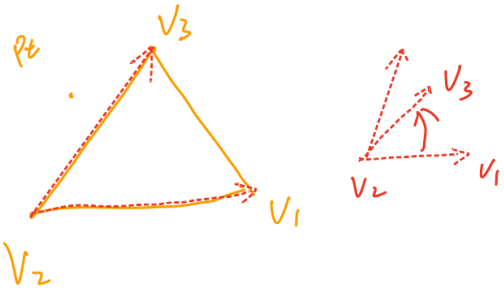




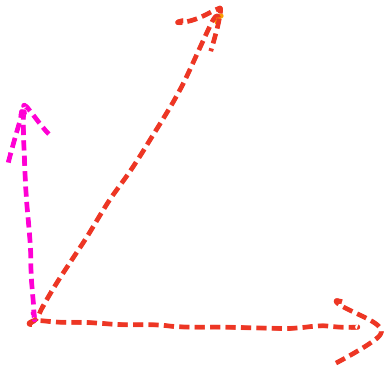


(V3-V2) cross (pt - V2)

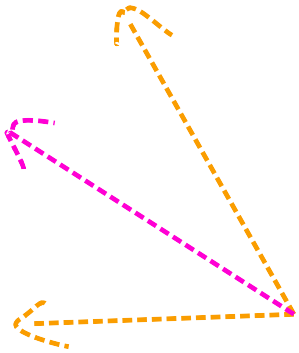




(V3 - V2) cross (V1 - V2)



不在



在

所以点不在三角形内

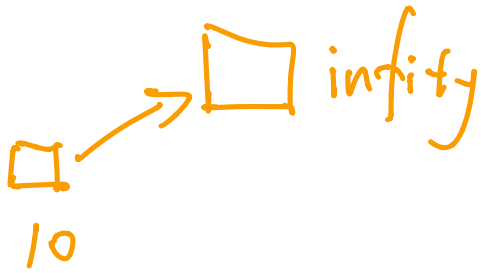
---

① 如果在三角形内 ✓

判断 z 深度，读 buffer

z_interpolated < depth[index]

初始为无限大



更新深度值

depth_buf[index] = z_interpolated

以及更新像素颜色

set_pixel(point, t.getcolor())

---

if(insideTriangle(x, y, v3f) && (z_interpolated
< depth_buf[index]))
{
vector3f poi = (x, y, 1.0f)
setpixel color (poi, t.getcolor())
depth_buf[index] = z_interpolated
}