- 线扫描line sweeping

**==> 将三角形的三个点按y值从小到达排序，最下面为t0, 中间的为t1, 最上面为t2.**

**==> 过t1的水平直线将三角形分成两个部分, 从t0.y向上遍历, 像往漏斗里加水一样, 利用相似三角形性质，得到水平面的左右端点，然后绘制这条水平线.**

void triangle(Vec2i t0, Vec2i t1, Vec2i t2, TGAImage &image, TGAColor color) {  
 *// sort the vertices, t0, t1, t2 lower−to−upper (bubblesort yay!)* if (t0.y>t1.y) std::swap(t0, t1);  
 if (t0.y>t2.y) std::swap(t0, t2);  
 if (t1.y>t2.y) std::swap(t1, t2);  
 int total\_height = t2.y-t0.y;  
 for (int y=t0.y; y<=t1.y; y++) {  
 int segment\_height = t1.y-t0.y+1;  
 float alpha = (float)(y-t0.y)/total\_height;  
 float beta = (float)(y-t0.y)/segment\_height; *// be careful with divisions by zero* Vec2i A = t0 + (t2-t0)\*alpha;  
 Vec2i B = t0 + (t1-t0)\*beta;  
 if (A.x>B.x) std::swap(A, B);  
 for (int j=A.x; j<=B.x; j++) {  
 image.set(j, y, color); *// attention, due to int casts t0.y+i != A.y* }  
 }  
 for (int y=t1.y; y<=t2.y; y++) {  
 int segment\_height = t2.y-t1.y+1;  
 float alpha = (float)(y-t0.y)/total\_height;  
 float beta = (float)(y-t1.y)/segment\_height; *// be careful with divisions by zero* Vec2i A = t0 + (t2-t0)\*alpha;  
 Vec2i B = t1 + (t2-t1)\*beta;  
 if (A.x>B.x) std::swap(A, B);  
 for (int j=A.x; j<=B.x; j++) {  
 image.set(j, y, color); *// attention, due to int casts t0.y+i != A.y* }  
 }  
}

虽然不是很复杂，但是行扫描的源代码有点混乱。此外，它实际上是一种为单线程CPU编程设计的老式方法。让我们看看下面的伪代码:

triangle(vec2 points[3]) {

vec2 bbox[2] = find\_bounding\_box(points);

for (each pixel in the bounding box) {

if (inside(points, pixel)) {

put\_pixel(pixel);

}

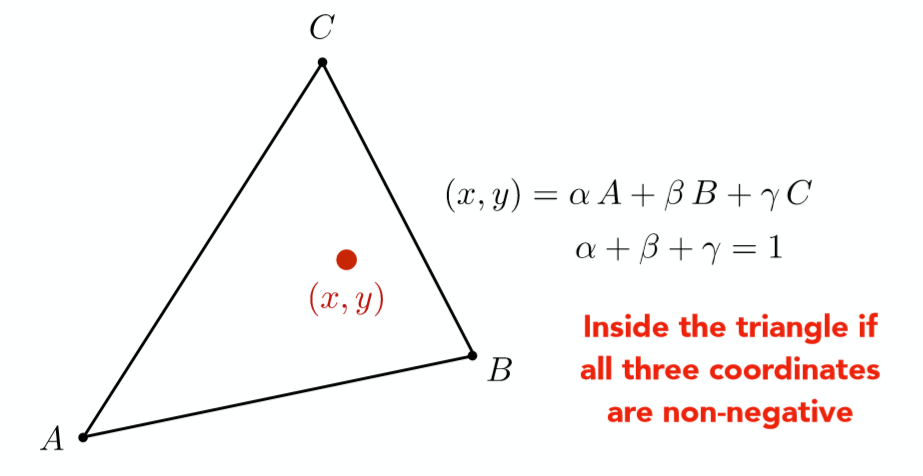
}

}

先找出包裹这个三角形的最小矩形盒子, 然后遍历里面的像素是否在三角形内|

所以关键的问题是：如何判断一个给定的点是否在一个给定的三角形内部|

解决方案:利用重心, 找到该点在该三角形的重心坐标, 都大于0, 该点即在三角形内|



给定三角形的三点坐标A, B, C，该平面内一点P(x,y)可以写成这三点坐标的线性组合形式，即 P( x , y ) = α A + β B + γ C 且满足 α + β + γ = 1 则称此时3个坐标A,B,C的权重 α , β , γ 为点P(x,y)的重心坐标。（特别的，如图中所说的，如果该点在三角形内部则三点坐标都为非负数，我们会在1.3节给出简要的叙述。）

#### 由于α = 1 - β - γ； 所以整个问题变为求解P(m,n) = (1 - β - γ)A + β B + γ C; (其中β， γ是未知数) 方便起见用u v替换未知数: 【非常关键好好看看】

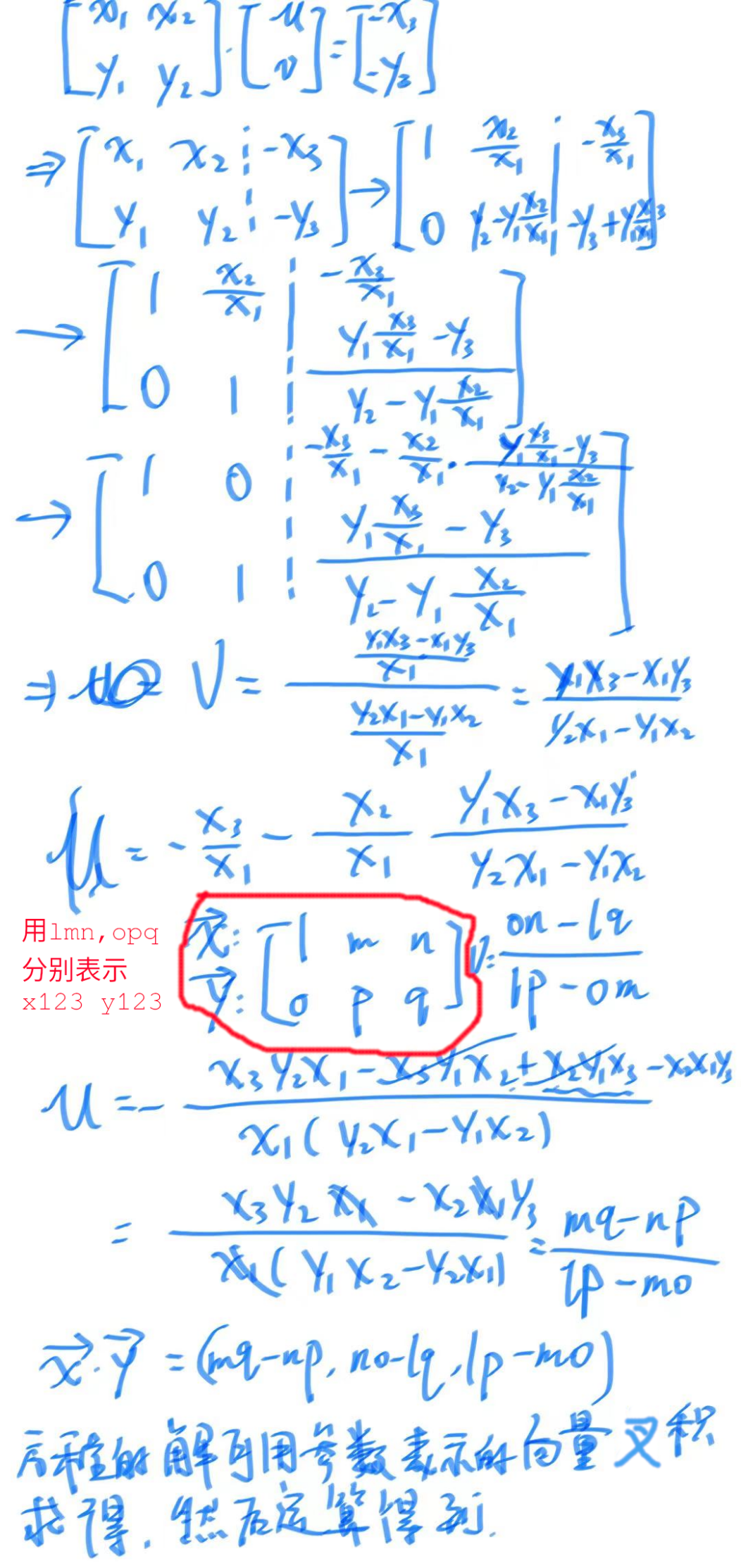
IMG_256

IMG_256

IMG_256

NOTE: B - A = AB

**【关于一些实现的具体细节见ssp\_offer项目 代码注释】**



举例来计算一个点(可用于调试):

三角形三个点分别为 a(10, 70) b(50, 160) c(70, 80)

一个在三角形内部的点为 p(40, 90)

ab = (40, 90), ac = (60, 10), pa = (-30, -20)

40 60 -30

90 10 -20

XY = (-1200+300, -2700+800, 400-5400) = (-900, -1900, -5000)

u = 9/50 v = 19/50 , 1-u-v = 22/50

这里面的XY 容易写错， 一定要小心细节[排查半天发现是手误!!!!]

**代码为:**

vec3f check(vec2i \*ps, vec2i p){  
  
 vec3i X {ps[1].x-ps[0].x, ps[2].x-ps[0].x, ps[0].x-p.x};  
 vec3i Y {ps[1].y-ps[0].y, ps[2].y-ps[0].y, ps[0].y-p.y};  
  
 vec3i Cross{  
 X.y\*Y.z - X.z\*Y.y,  
 X.z\*Y.x - X.x\*Y.z,  
 X.x\*Y.y - X.y\*Y.x  
 };  
 float u = (float)(Cross.x\*1.0/Cross.z);  
 float v = (float)(Cross.y\*1.0/Cross.z);  
 return vec3f{u, v, 1-u-v};  
  
}  
  
void triangle\_2(vec2i \*t, TGAImage &image, TGAColor color){  
 struct vec2i minbox{image.width()-1, image.height()-1};  
 struct vec2i maxbox{0, 0};  
  
 for (int i = 0; i < 3; ++i) {  
 *//--min* minbox.x = std::min(minbox.x, t[i].x);  
 minbox.y = std::min(minbox.y, t[i].y);  
 *//--max* maxbox.x = std::max(maxbox.x, t[i].x);  
 maxbox.y = std::max(maxbox.y, t[i].y);  
 }  
  
 for (int i = minbox.x; i < maxbox.x; ++i) {  
 for (int j = minbox.y; j < maxbox.y; ++j) {  
 struct vec2i p{i, j};  
 vec3f coord = check(t, p);  
 if (coord.x<0 || coord.y<0 || coord.z<0){continue;}  
 image.set(p.x, p.y, color);  
 }  
 }  
}

- 简单光照

Note： 利用光照向量点积三角形片元的法向量得到光照强度{intensity = |a||b|cos(heta)}，负值代表光从反面照过来，用以反面片元剔除|

void draw\_head\_light(TGAImage &testImage){  
 Model model {"african\_head.obj"};  
  
 vec3 light\_dir{0, 0, -1};  
  
 for (int i = 0; i < model.nfaces(); ++i) {  
 struct vec2i\_ screen\_pos [3];  
 vec3 world\_pos [3];  
 for (int j = 0; j < 3; ++j) {  
 auto v = model.vert(i,j);  
 screen\_pos[j].x = (v.x + 1) / 2.0 \* testImage.width();  
 screen\_pos[j].y = (v.y + 1) / 2.0 \* testImage.height();  
  
 world\_pos[j] = v;  
 }  
  
 const vec3 a = world\_pos[2] - world\_pos[0];  
 const vec3 b = world\_pos[1] - world\_pos[0];  
 vec3 normal = cross(a, b);  
 normal.normalize();  
  
 float intensity = normal \* light\_dir;

if (intensity>0) {  
 triangle\_2(screen\_pos, testImage,

TGAColor(intensity \* 255, intensity \* 255, intensity \* 255));  
 }

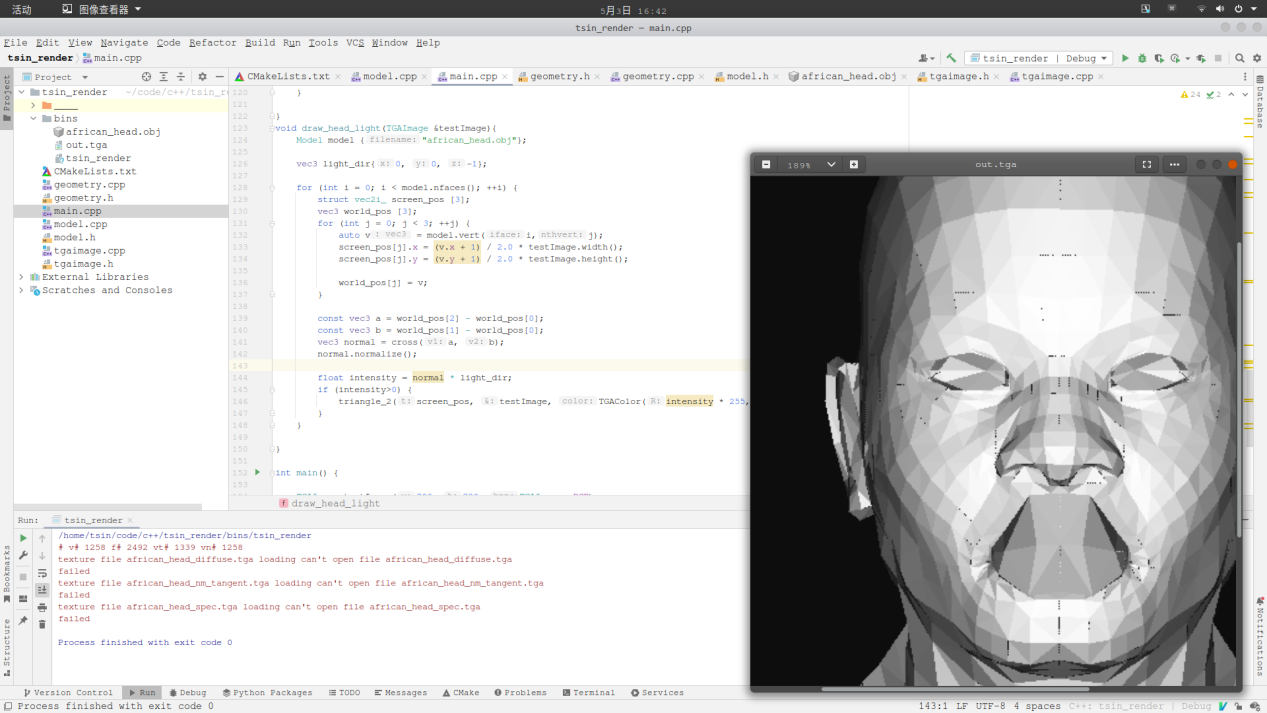
}  
  
}

使用自己的画线算法，有些黑点， 三角形的包围盒要用float格式，

就不会有黑点，

*//模型坐标范围是0-1的float, 将每个顶点的xy转换到屏幕空间范围*triang\_vtxs[i].x = (triang\_vtxs[i].x + 1) / 2 \* image.width();  
triang\_vtxs[i].y = (triang\_vtxs[i].y + 1) / 2 \* image.height();  
  
*//--min*minbox.x = std::min(minbox.x, triang\_vtxs[i].x);  
minbox.y = std::min(minbox.y, triang\_vtxs[i].y);  
*//--max*maxbox.x = std::max(maxbox.x, triang\_vtxs[i].x);  
maxbox.y = std::max(maxbox.y, triang\_vtxs[i].y);

而不是minbox.x = std::min(minbox.x, (int)triang\_vtxs[i].x);

****

注意嘴的内腔是画在嘴唇上的。这是因为我们对看不见的三角形进行了不干净的剪裁:它只对凸形有效。我们会在下次编码z缓冲区时去掉这个伪影。