Note:

zbuffer 数组的大小为最终渲染图像的大小, 也就是说zbuffer是针对每个像素的

zbuffer 的使用是在绘制每个三角形片元的过程中, 属于三角形内的每个像素点都通过该点的重心坐标计算出Z坐标值. [你判断该像素是不是在三角形里面的时候， 不是计算过该点的重心坐标了吗， 然后直接拿这个坐标计算z]然后比较，绘制, 跟新zbuffer.

- 还有一个重要问题是，check返回的重心坐标的顺序， u/v/1-u-v， 顺序可以随便写， 虽然对像素的z插值的结果不同， 但是只要所有的像素都使用同一个顺序插值得到z， 他们的前后关系就不会变, 没问题【但是对UV来说有问题】

完整代码：  
vec3 check\_(vec3 \*ps, vec3 p){  
 const vec3 X {ps[1].x - ps[0].x, ps[2].x - ps[0].x, ps[0].x - p.x};  
 const vec3 Y {ps[1].y - ps[0].y, ps[2].y - ps[0].y, ps[0].y - p.y};  
  
 vec3 cross\_ = cross(X, Y);  
 float u = (float)(cross\_.x \* 1.0 / cross\_.z);  
 float v = (float)(cross\_.y \* 1.0 / cross\_.z);  
 return vec3{1-u-v, v, u};  
  
}

void triangle\_3(float \*zbuffer, vec3 \*world\_pos, TGAImage &image, TGAColor color){  
   
 *//world\_pos 三角形的三个顶点*

*//*vec2i minbox{image.width() - 1, image.height() - 1};  
 //vec2i maxbox{0, 0};

vec2 minbox{image.width() - 1, image.height() - 1};  
 vec2 maxbox{0, 0};  
 for (int i = 0; i < 3; ++i) {  
  
 *//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* world\_pos[i].x = (world\_pos[i].x + 1) / 2 \* image.width();  
 world\_pos[i].y = (world\_pos[i].y + 1) / 2 \* image.height();  
  
 *//--min  
 //*minbox.x = std::min(minbox.x, (int)world\_pos[i].x);  
 //minbox.y = std::min(minbox.y, (int)world\_pos[i].y);  
 *//--max  
 //*maxbox.x = std::max(maxbox.x, (int)world\_pos[i].x);  
 //maxbox.y = std::max(maxbox.y, (int)world\_pos[i].y);

*//--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);

}  
  
 for (int i = minbox.x; i < maxbox.x; ++i) {  
 for (int j = minbox.y; j < maxbox.y; ++j) {  
 vec3 p{(float)i, (float)j, 0};  
   
 *//还有一个重要问题是，check返回的重心坐标的顺序  
 //* vec3 coord = check\_(world\_pos, p);  
 if (coord.x<0 || coord.y<0 || coord.z<0){continue;}  
  
  
 *//直接使用像素点的重心坐标， 计算出z的插值  
 //三角形顶点的z，还是0-1范围， 所以记得zbuffer 要用float* for (int k = 0; k < 3; ++k) {  
 p.z += world\_pos[k].z \* coord[k];  
 }  
   
 if (zbuffer[i\*image.width()+j] < p.z) {  
 zbuffer[i\*image.width()+j] = p.z;  
 image.set(p.x, p.y, color);  
 }  
 }  
 }  
  
}

*//*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，但是openmp 这地要用int, (int)maxbox.x

不然无法预测， 转为int 后会有黑色线条， 改为<=即可

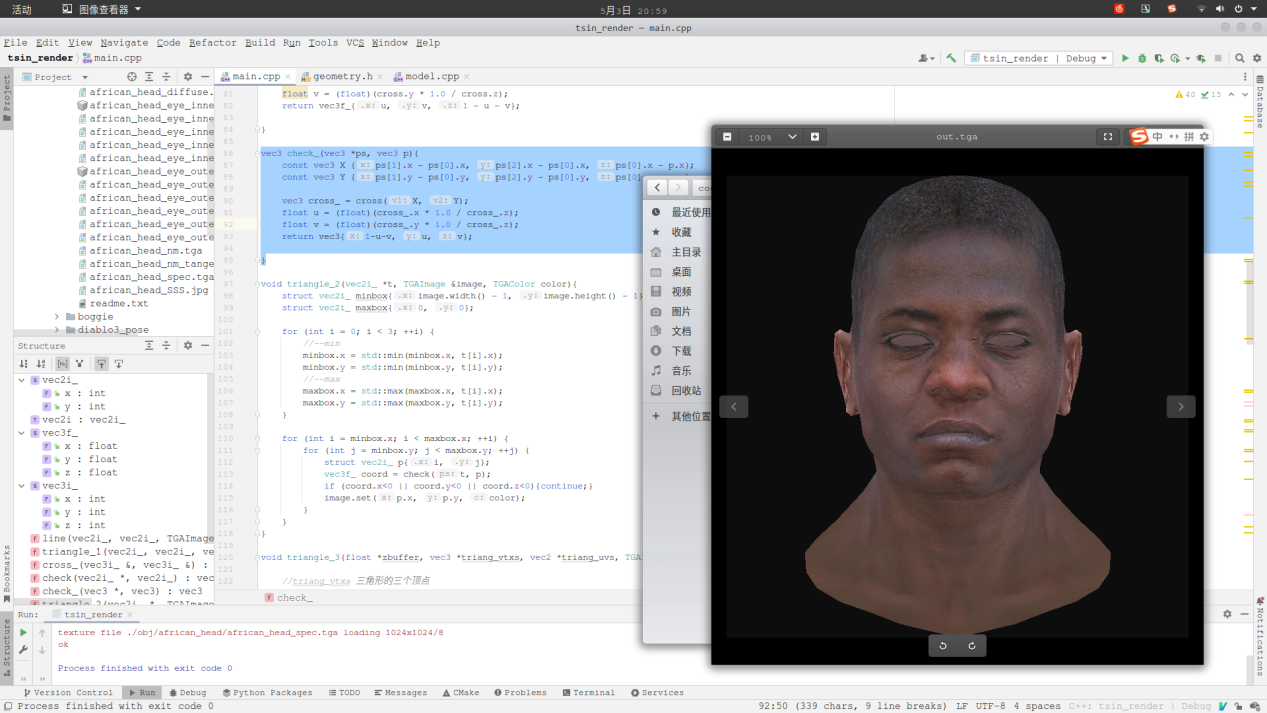
#pragma omp parallel for  
  
 for (int i = minbox.x; i <= (int)maxbox.x; i++) {  
 for (int j = minbox.y; j <= (int)maxbox.y; j++) {

贴图

贴图插值的过程是对 属于三角形内的每个像素点都通过该点的重心坐标计算出uv坐标值， 然后获得贴图的颜色值， 而不是对片元的三个点的颜色进行插值.

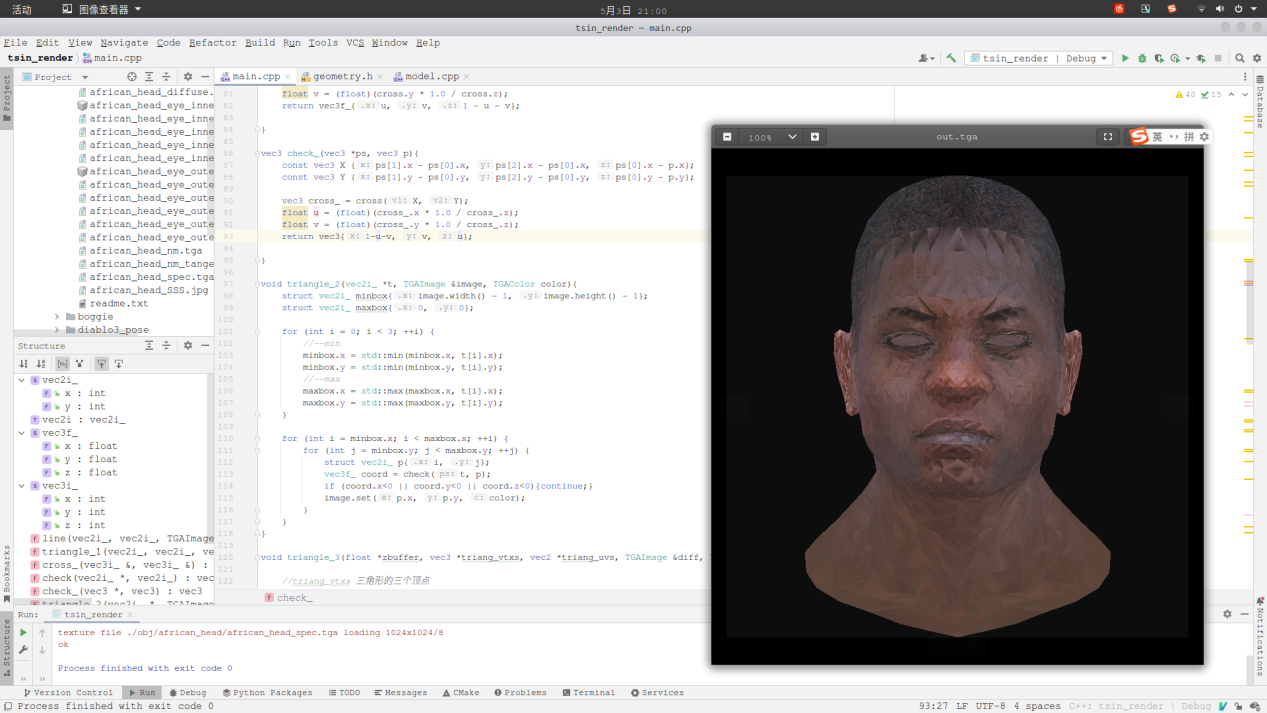
- 还有一个重要问题是，check返回的重心坐标的顺序， u/v/1-u-v, 在ssp\_render中又不同的情况有空请好好研究一下。

vec3 check\_(vec3 \*ps, vec3 p){  
 const vec3 X {ps[1].x - ps[0].x, ps[2].x - ps[0].x, ps[0].x - p.x};  
 const vec3 Y {ps[1].y - ps[0].y, ps[2].y - ps[0].y, ps[0].y - p.y};  
  
 vec3 cross\_ = cross(X, Y);  
 float u = (float)(cross\_.x \* 1.0 / cross\_.z);  
 float v = (float)(cross\_.y \* 1.0 / cross\_.z);  
 return vec3{1-u-v, u, v};  
  
}



如果改为

return vec3{1-u-v, v, u};



代码  
  
void triangle\_3(float \*zbuffer, vec3 \*triang\_vtxs, vec2 \*triang\_uvs, TGAImage &diff, TGAImage &image, TGAColor color){  
  
 *//triang\_vtxs 三角形的三个顶点* vec2 minbox{(double )image.width() , (double)image.height()};  
 vec2 maxbox{-1, -1};  
  
 for (int i = 0; i < 3; ++i) {  
  
 *//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();  
  
 triang\_uvs[i].x = triang\_uvs[i].x \* diff.width();  
 triang\_uvs[i].y = triang\_uvs[i].y \* diff.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);  
  
 }  
  
  
 for (int i = minbox.x; i < maxbox.x; ++i) {  
 for (int j = minbox.y; j < maxbox.y; ++j) {  
 vec3 p{(float)i, (float)j, 0};  
  
 *//还有一个重要问题是，check返回的重心坐标的顺序, 没关系* vec3 coord = check\_(triang\_vtxs, p);  
 if (coord.x<0 || coord.y<0 || coord.z<0){continue;}  
  
 *//直接使用像素点的重心坐标， 计算出z的插值  
 //三角形顶点的z，还是0-1范围， 所以记得zbuffer 要用float* for (int k = 0; k < 3; ++k) {  
 p.z += triang\_vtxs[k].z \* coord[k];  
 }  
  
 *// u v* int u\_ = coord \* vec3 (triang\_uvs[0].x, triang\_uvs[1].x, triang\_uvs[2].x);  
 int v\_ = coord \* vec3 (triang\_uvs[0].y, triang\_uvs[1].y, triang\_uvs[2].y);  
  
  
 if (zbuffer[i\*image.width()+j] < p.z) {  
 zbuffer[i\*image.width()+j] = p.z;  
 image.set(p.x, p.y, diff.get(u\_, v\_));  
 }  
 }  
 }  
  
}