core::Image4u colorbuffer =core::image::zeros<core::Pixel<std::uint8\_t, 4>>(viewport\_height, viewport\_width);

<R,G,B,alpha>

//存储颜色信息，初始值为0

core::Image1d depthbuffer =core::image::constant(viewport\_height, viewport\_width, std::numeric\_limits<double>::max());

//存储深度信息，初始值为最大值

# 三角形可见性判断

遍历三角形，判断三个3D在像素平面的投影点（X,Y,DEPTH）是否是逆时针（只根据X、Y坐标进行判断，由于opencv坐标系的缘故，投影面的法向的需为负方向才是可见的）

# 对可见的三角形进行光栅化

遍历三角形的矩形框，判断某一点是否在三角形内。

如果点在三角形内：

根据三角形的深度信息加权计算该点的深度信息（因为该点可能没有对应的三维点所以需要这样计算深度信息），Z值越小越朝外，遮挡了该位置的其他点（因为opencv坐标系的缘故）。根据三角形的三个点的颜色的加权值作为该点的颜色值取Alpha=255，令depthbuffer(pixel\_index\_row, pixel\_index\_col) = z\_affine;用新的深度值覆盖该位置。假如计算得到的点被已有的点遮挡就忽略。

因此最后得到的结果为

Colourbuffer depthbuffer

被遮挡的部分的color值为全0

被遮挡的部分的depth值为全max

# 将颜色输出到纹理文件

纹理文件大小固定，3D点和纹理文件的映射关系已知，遍历三角形：

再次判断三角形是否可见，这次增加了一个步骤，假如三角形内部的任意一点被遮挡（根据深度信息判断），就判断三角形不可见。假如三角形可见，计算src到dst（已知的uv坐标）两个三角形的映射关系，将src的三角形投影到dst。