static bool insideTriangle(int x, int y, const Vector4f\* \_v){

Vector3f v[3];

for(int i=0;i<3;i++)

v[i] = {\_v[i].x(),\_v[i].y(), 1.0};

Vector3f f0,f1,f2;

f0 = v[1].cross(v[0]);

f1 = v[2].cross(v[1]);

f2 = v[0].cross(v[2]);

Vector3f p(x,y,1.);

if((p.dot(f0)\*f0.dot(v[2])>0) && (p.dot(f1)\*f1.dot(v[0])>0) && (p.dot(f2)\*f2.dot(v[1])>0))

return true;

return false;

void rst::rasterizer::rasterize\_triangle(const Triangle& t) {

auto v = t.toVector4();

// 构造bounding box

float min\_x = std::min(v[0][0], std::min(v[1][0], v[2][0]));

float max\_x = std::max(v[0][0], std::max(v[1][0], v[2][0]));

float min\_y = std::min(v[0][1], std::min(v[1][1], v[2][1]));

float max\_y = std::max(v[0][1], std::max(v[1][1], v[2][1]));

min\_x = (int)std::floor(min\_x);

max\_x = (int)std::ceil(max\_x);

min\_y = (int)std::floor(min\_y);

max\_y = (int)std::ceil(max\_y);

bool MSAA = true;

//MSAA 4X

if (MSAA) {

// 格子里的细分四个小点坐标

std::vector<Eigen::Vector2f> pos

{

{0.25,0.25},

{0.75,0.25},

{0.25,0.75},

{0.75,0.75},

};

for (int x = min\_x; x <= max\_x; x++) {

for (int y = min\_y; y <= max\_y; y++) {

// 记录最小深度

float minDepth = FLT\_MAX;

// 四个小点中落入三角形中的点的个数

int count = 0;

// 对四个小点坐标进行判断

for (int i = 0; i < 4; i++) {

// 小点是否在三角形内

if (insideTriangle((float)x + pos[i][0], (float)y + pos[i][1], t.v)) {

// 如果在，对深度z进行插值

auto tup = computeBarycentric2D((float)x + pos[i][0], (float)y + pos[i][1], t.v);

float alpha, beta, gamma;

std::tie(alpha, beta, gamma) = tup;

float w\_reciprocal = 1.0 / (alpha / v[0].w() + beta / v[1].w() + gamma / v[2].w());

float z\_interpolated = alpha \* v[0].z() / v[0].w() + beta \* v[1].z() / v[1].w() + gamma \* v[2].z() / v[2].w();

z\_interpolated \*= w\_reciprocal;

minDepth = std::min(minDepth, z\_interpolated);

count++;

}

}

if (count != 0) {

if (depth\_buf[get\_index(x, y)] > minDepth) {

Vector3f color = t.getColor() \* count / 4.0;

Vector3f point(3);

point << (float)x, (float)y, minDepth;

// 替换深度

depth\_buf[get\_index(x, y)] = minDepth;

// 修改颜色

set\_pixel(point, color);

}

}

}

}

}

else {

for (int x = min\_x; x <= max\_x; x++) {

for (int y = min\_y; y <= max\_y; y++) {

//下个采样点是否在三角形内

if (insideTriangle((float)x + 0.5, (float)y + 0.5, t.v)) {

auto tup = computeBarycentric2D((float)x + 0.5, (float)y + 0.5, t.v);

float alpha;

float beta;

float gamma;

std::tie(alpha, beta, gamma) = tup;

float w\_reciprocal = 1.0 / (alpha / v[0].w() + beta / v[1].w() + gamma / v[2].w());

float z\_interpolated = alpha \* v[0].z() / v[0].w() + beta \* v[1].z() / v[1].w() + gamma \* v[2].z() / v[2].w();

z\_interpolated \*= w\_reciprocal;

if (depth\_buf[get\_index(x, y)] > z\_interpolated) {

Vector3f color = t.getColor();

Vector3f point(3);

point << (float)x, (float)y, z\_interpolated;

depth\_buf[get\_index(x, y)] = z\_interpolated;

set\_pixel(point, color);

}

}

}

}

}

}