Assignment2

Task

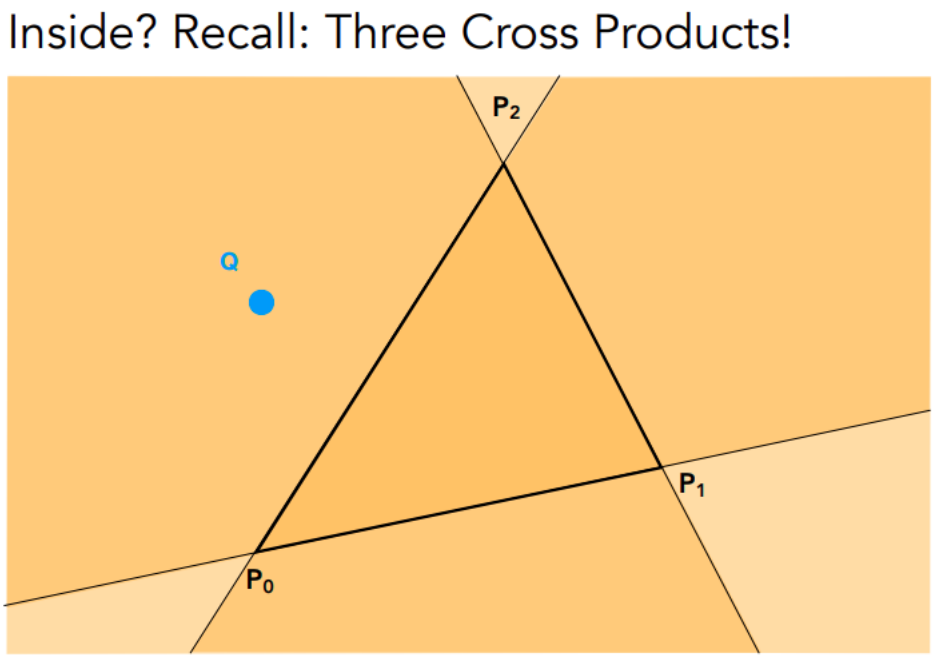
实现光栅化

需要补全的函数

- static bool insideTriangle(int x, int y, const Vector3f* _v)
- void rst::rasterizer::rasterize_triangle(const Triangle& t)

Method

判断点在三角形内部——叉积



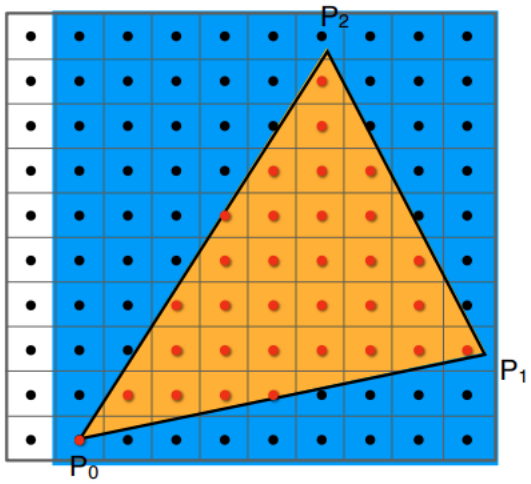
对于三角形ABC，

如果该点P，AB*AP，BC*BP，CA*CP方向相同，则在内部，否则在外部

光栅化

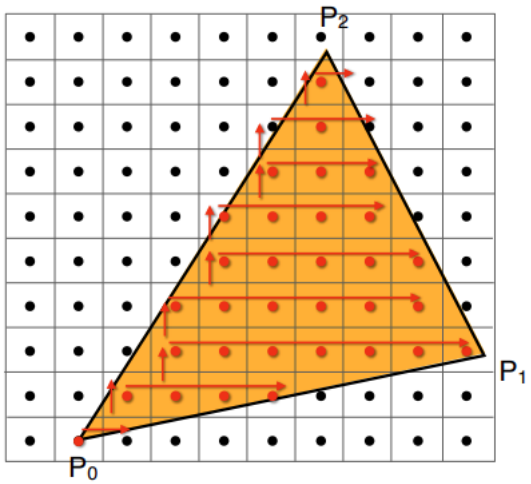
首先是确定bounding box

Checking All Pixels on the Screen?



Use a **Bounding Box!**

随后在bounding Box中逐个像素去询问是否在需要画出的三角形中



之后使用Z-buffer算法

Idea:

- Store current min. z-value for each sample (pixel)
- Needs an additional buffer for depth values
  - frame buffer stores color values
  - depth buffer (z-buffer) stores depth

Code

insideTriangle

```
static bool insideTriangle(int x, int y, const Vector3f* _v)
{
    // TODO : Implement this function to check if the point (x, y) is inside the triangle represented by _v[0], _v[1], _v[2]
    Vector3f q(x, y, 0);
    Vector3f ab = _v[1] - _v[0], bc = _v[2] - _v[1], ca = _v[0] - _v[2];
    Vector3f aq = q - _v[0], bq = q - _v[1], cq = q - _v[2];
    bool ans = ab.cross(aq).dot(bc.cross(bq)) > 0 && ab.cross(aq).dot(ca.cross(cq)) > 0 && bc.cross(bq).dot(ca.cross(cq)) > 0;
    return ans;
}
```

rasterize_triangle

```
void rst::rasterizer::rasterize_triangle(const Triangle& t) {
    auto v = t.toVector4();

    // TODO : Find out the bounding box of current triangle.
    // iterate through the pixel and find if the current pixel is inside the triangle

    // If so, use the following code to get the interpolated z value.
    //auto[alpha, beta, gamma] = computeBarycentric2D(x, y, t.v);
    //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;

    // TODO : set the current pixel (use the set_pixel function) to the color of the triangle (use getColor function) if it should be paint
        // Bounding Box
    int min_x = std::min(v[0].x(), std::min(v[1].x(), v[2].x())),
        min_y = std::min(v[0].y(), std::min(v[1].y(), v[2].y())),
        max_x = std::max(v[0].x(), std::max(v[1].x(), v[2].x())),
        max_y = std::max(v[0].y(), std::max(v[1].y(), v[2].y()));
    // 采样
    for (int x = min_x; x <= max_x; x++)
        for (int y = min_y; y <= max_y; y++)
        {
            if (insideTriangle(x + 0.5, y + 0.5, t.v))
            {
                // Z-Buffer算法

                // 获取当前像素的深度
                auto tmp = computeBarycentric2D(x + 0.5, y + 0.5, t.v);
                float alpha, beta, gamma;
                std::tie(alpha, beta, gamma) = tmp;
                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;//深度信息
                // 深度测试
                if (z_interpolated < depth_buf[get_index(x, y)])// 深度越大，离视角越远
                {
```

```
                    depth_buf[get_index(x, y)] = z_interpolated;// 更新深度缓存
                    Vector3f point = { (float)x,(float)y, z_interpolated }, color = t.getColor();
                    set_pixel(point, color);
                }
            }
        }
}
```