实验报告 2

姓名：蒋卓洋
学号：59119125

1、实现

（1）函数位置：rasterizer.cpp
（2）函数实现：
A.insideTriangle()函数

```
1.  static bool insideTriangle(int x, int y, const Vector3f* _v)
2.  {
3.      // TODO : Implement this function to check if the point (x, y) is inside the triangle repr
    esented by _v[0], _v[1], _v[2]
4.
5.      //////////////////Solution//////////////////
6.      ////Name:JiangZhuoyang
7.      ////StudentID:58119125
8.      ////FinishDate:21/10/15
9.      //Judge with the vectors' sign:
10.     //1.Construct the edge vector
11.     Eigen::Vector2f P1_P2 = _v[1].head(2) - _v[0].head(2);
12.     Eigen::Vector2f P2_P3 = _v[2].head(2) - _v[1].head(2);
13.     Eigen::Vector2f P3_P1 = _v[0].head(2) - _v[2].head(2);
14.
15.     //2.Construct the 'Judge Vectors' with P0 to the three points of triangle
16.     Eigen::Vector2f P0;
17.     P0 << x,y;
18.     Eigen::Vector2f P1_P0 = P0 - _v[0].head(2);
19.     Eigen::Vector2f P2_P0 = P0 - _v[1].head(2);
20.     Eigen::Vector2f P3_P0 = P0 - _v[2].head(2);
21.
22.     //3.Judge the location of P0 with cross product
23.     auto cross_1 = P1_P2[0]*P1_P0[1]-P1_P2[1]*P1_P0[0];
24.     auto cross_2 = P2_P3[0]*P2_P0[1]-P2_P3[1]*P2_P0[0];
25.     auto cross_3 = P3_P1[0]*P3_P0[1]-P3_P1[1]*P3_P0[0];
26.     if(cross_1 < 0){
27.         if(cross_2 < 0 && cross_3 < 0 ){ return true; } //the same direction < 0
28.         else{return false;}
29.     }
```

```
    else{
        if(cross_2 > 0 && cross_3 > 0 ){ return true; } //the same direction > 0s
        else{return false;}
    }

    ////////////////////////////////////////////
}
```

B.rasterize_triangle()函数

```
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 painted.

    ////////////////Solution//////////////////
    ////Name:JiangZhuoyang
    ////StudentID:58119125
    ////FinishDate:21/10/15

    //1.COnstruct the bounding box with for value, the value is difined by the 4 extremums in to directions
    //(1)Get bound
    float bound_L = std::min(v[0][0], std::min(v[1][0],v[2][0]));//Left bound: bounded by the minimum of x-coordinate of three points of triangle
    float bound_R = std::max(v[0][0], std::max(v[1][0],v[2][0]));//Right bound: bounded by the maximum of x-coordinate of three points of triangle
    float bound_T = std::min(v[0][1], std::min(v[1][1],v[2][1]));//Top bound: bounded by the minimum of y-coordinate of three points of triangle
    float bound_B = std::max(v[0][1], std::max(v[1][1],v[2][1]));//Bottom bound: bounded by the maximum of y-coordinate of three points of triangle
    //(2)Nomalize to integer for iteration
```

```
    bound_L = (int)std::floor(bound_L);//round down the left bound
    bound_R = (int)std::ceil(bound_R); //round up the right bound
    bound_T = (int)std::floor(bound_T);//round down the top bound
    bound_B = (int)std::ceil(bound_B);  //round up the bottom bound


    //2.Iterate through the pixel in the bound box and find if the current pixel is inside the triangle
    for(int x = bound_L; x <= bound_R; x++){
        for(int y = bound_T; y <= bound_B; y++){
    //(1)Judge if the current pixel is inside the triangle

            if(insideTriangle(x, y, t.v)){
    //(2)Get the interpolated z value.
                //A.define the min depth, innitialize it with infinite.
                float depth_min = FLT_MAX;
                //B.calculate the min depth
                auto tuple = computeBarycentric2D(x, y, t.v);
                float alpha, beta, gamma;
                std::tie(alpha, beta, gamma) = tuple; // Debug the given method
                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;
                //C.get the min depth
                depth_min = std::min(depth_min,z_interpolated);

    //(3)Judge if the current point's z is more shallow than the value had been stored in the z-buffer (the old depth)
                if(depth_min < depth_buf[get_index(x,y)]){//the current point is more shallow
                    //Renew the z-buffer with current point
                    //a.renew the depth
                    depth_buf[get_index(x,y)] = depth_min;
                    //b.renew the color
                    Vector3f point_current;
                    point_current << x,y,1;
                    set_pixel(point_current, t.getColor());
                }

            }
        }
    }
    ////////////////////////////////////////////////
}
```

2、结果

实验结果如下：

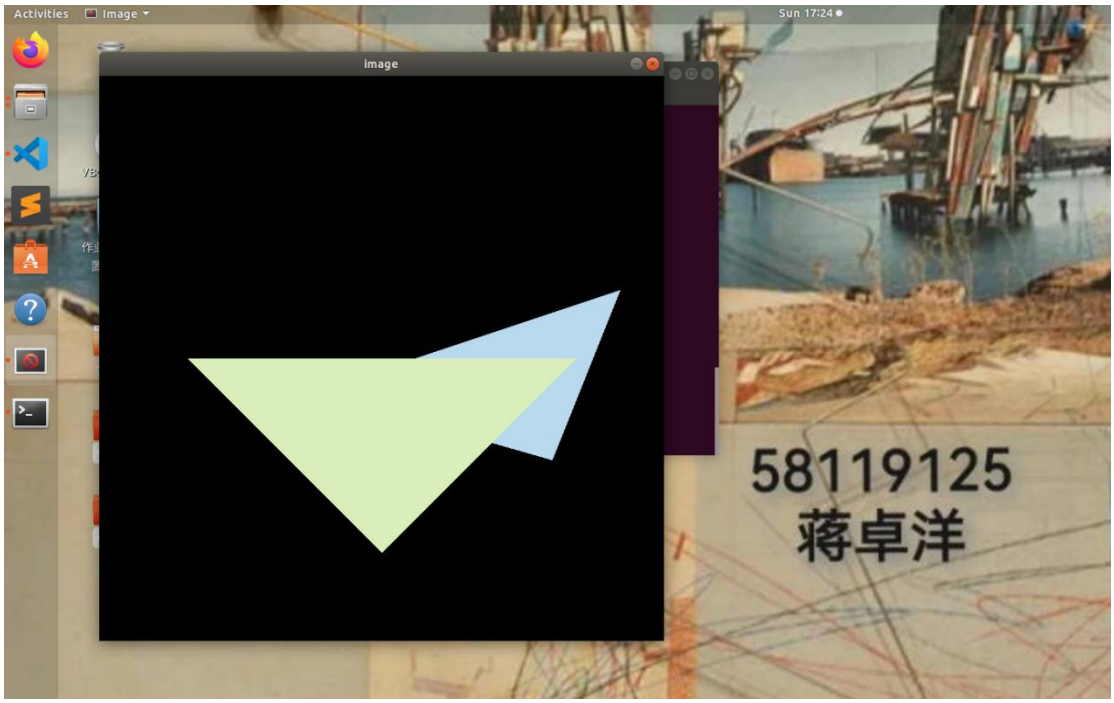

图 1.作业二结果