作业2

基础部分

根据实验指导书，需要完善的函数为rasterizer.cpp文件中的 insideTriangle() 函数和 rasterize_triangle()函数，相关代码实现如下：

```
static bool insideTriangle(float x, float 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]
    Eigen::Vector2f P,AB,BC,CA,AP,BP,CP;
    P << x,y;
    AB = v[1].head(2)-v[0].head(2);
    BC = v[2].head(2)-v[1].head(2);
    CA = v[0].head(2)-v[2].head(2);
    AP = P-v[0].head(2);
    BP = P-v[1].head(2);
    CP = P-v[2].head(2);
    //利用三个叉乘来判断点是否在三角形内
    return ((AB[0]*AP[1]-AB[1]*AP[0]>0)
        && (BC[0]*BP[1]-BC[1]*BP[0]>0)
        && (CA[0]*CP[1]-CA[1]*CP[0]>0))
        ||
        ((AB[0]*AP[1]-AB[1]*AP[0]<0)
        &&(BC[0]*BP[1]-BC[1]*BP[0]<0)
        &&(CA[0]*CP[1]-CA[1]*CP[0]<0));
}
```

```
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
    float min_x,min_y,max_x,max_y;
    min_x = std::min({v[0].x(),v[1].x(),v[2].x()});
    max_x = std::max({v[0].x(),v[1].x(),v[2].x()});
    min_y = std::min({v[0].y(),v[1].y(),v[2].y()});
    max_y = std::max({v[0].y(),v[1].y(),v[2].y()});

    min_x = std::floor(min_x);
    min_y = std::floor(min_y);
    max_x = std::ceil(max_x);
    max_y = std::ceil(max_y);

    for(int i=(int)min_x;i<(int)max_x;i++)
    {
        for (int j=(int)min_y;j<(int)max_y; j++)
        {
            if(insideTriangle(float(i+0.5),float(j+0.5),t.v))
            {
                float alpha, beta, gamma;
                auto baryc = computeBarycentric2D(float(i+0.5),float(j+0.5),t.v);
                std::tie(alpha,beta,gamma) = baryc;
                /*float w_normalized = 1.0f/(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();*/
                float w_normalized = 1.0f/(alpha+beta+gamma);
                float z_interpolated = alpha * v[0].z()+beta*v[1].z()+gamma*v[2].z();
                z_interpolated *= w_normalized;
                if(z_interpolated > depth_buf[get_index(i,j)])
                {
                    depth_buf[get_index(i,j)] = z_interpolated;
                    set_pixel(Eigen::Vector3f((float)i,(float)j,z_interpolated),t.getColor());
                }
            }
        }

    }
    // 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
}
```

需要注意的几点：

- 实验指导书内解释说明了为方便编写代码，深度z的值已经改为正值，但是在先前笔者已经完成的作业1中，笔者所有的变换矩阵中存储的z值均为负值，因此如果没有进行相应的修改，则会出现显示的三角形倒转的情况；对此，有两种解决办法：
  第一种，将帧缓冲区中的所有像素深度初始化为负无穷大，而不是正无穷；
  第二种，在视口变换时，将z轴逆转；

```
//第一种办法
if ((buff & rst::Buffers::Depth) == rst::Buffers::Depth)
    {
        std::fill(depth_buf.begin(), depth_buf.end(), -std::numeric_limits<float>::infinity());
    }
```

```
//第二种办法
//Viewport transformation
        for (auto & vert : v)
        {
            vert.x() = 0.5*width*(vert.x()+1.0);
            vert.y() = 0.5*height*(vert.y()+1.0);
            vert.z() = -vert.z() * f1 + f2;
        }
```

- 另外，在光栅化三角形时，三角形的三个顶点已化为w=1的齐次坐标，后面为了简便运算可以暂时不考虑归一化（当然最好使用归一化公式）
- 在编写代码时，注意float与int类型的相互转化，尽量安全地进行数据类型间地转化

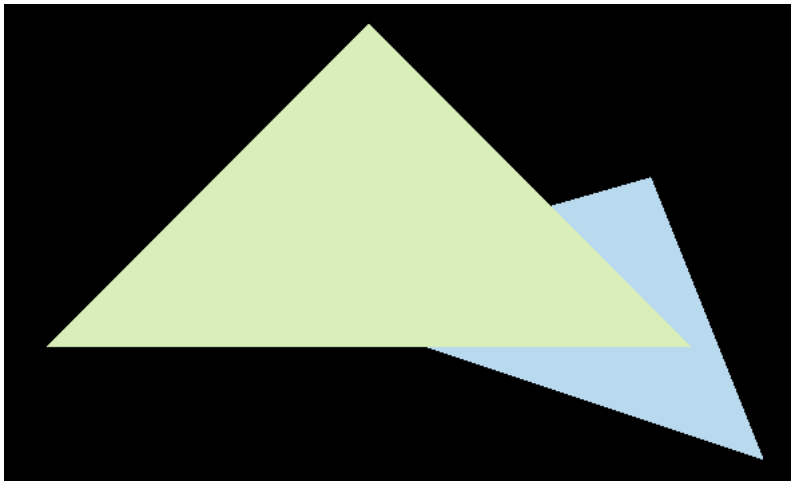
提高部分

提高部分则为实现4倍MSAA采样，实现代码如下：

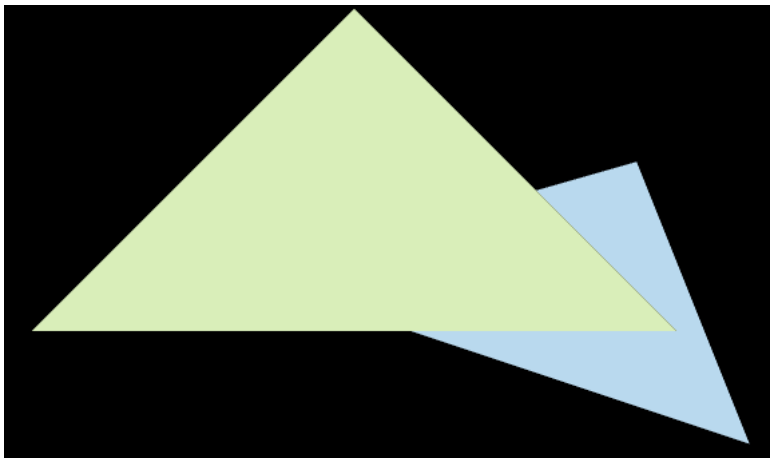
```
void rst::rasterizer::rasterize_triangle(const Triangle& t) {
    auto v = t.toVector4();

    // TODO : Find out the bounding box of current triangle.
    // iterate through the pixel and find if the current pixel is inside the triangle
    float min_x,min_y,max_x,max_y;
    min_x = std::min({v[0].x(),v[1].x(),v[2].x()});
    max_x = std::max({v[0].x(),v[1].x(),v[2].x()});
    min_y = std::min({v[0].y(),v[1].y(),v[2].y()});
    max_y = std::max({v[0].y(),v[1].y(),v[2].y()});

    min_x = std::floor(min_x);
    min_y = std::floor(min_y);
    max_x = std::ceil(max_x);
    max_y = std::ceil(max_y);

    std::vector<float> s{0.25,0.25,0.75,0.75,0.25}; //存储4个采样点
    //该部分代码为4倍MSAA采样
    for(int i=(int)min_x;i<(int)max_x;i++)
    {
        for (int j=(int)min_y;j<(int)max_y; j++)
        {
            int count =0;
            float maxDepth = -std::numeric_limits<float>::infinity();
            for (int k = 0; k < 4; k++)
            {
                if (insideTriangle(float(i)+s[k],float(j)+s[k+1],t.v))
                {
                    float alpha, beta, gamma;
                    auto baryc = computeBarycentric2D(float(i)+s[k],float(j)+s[k+1],t.v);
                    std::tie(alpha,beta,gamma) = baryc;
                    float w_normalized = 1.0f/(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_normalized;
                    maxDepth = std::max(maxDepth,z_interpolated);
                    count++;
                }

            }
            if(maxDepth > depth_buf[get_index(i,j)])
            {
                depth_buf[get_index(i,j)] = maxDepth;
                set_pixel(Eigen::Vector3f((float)i,(float)j,maxDepth),t.getColor()*count/4);
            }
        }
    }
}
```

- 对于提高部分，笔者也是按照模型位于-z轴来实现的
- 笔者将4倍超采样简单实现为该像素的周围4个点的加权平均

普通采样效果图：



4倍MSAA采样效果图：



至此，作业2完成！