实验二

实验目的

- 实现一个栅格化的三角形
- 实现深度缓存算法
- 提高部分：采用超采样算法（MSAA）实现抗锯齿的效果

实验过程

1. 修改完成static bool insideTriangle(float x, float y, const Vector3f* _v)

```
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]
    // 采用叉乘法来判断
    Vector3f p = {x, y, 0};
    Vector3f p0 = _v[0] - p;
    Vector3f p1 = _v[1] - p;
    Vector3f p2 = _v[2] - p;

    float c1 = p0.cross(p1).z(); // 取向量叉乘后的z轴坐标来判断正负
    float c2 = p1.cross(p2).z();
    float c3 = p2.cross(p0).z();

    if(c1>=0 && c2>=0 && c3>=0 || c1<0 && c2<0 && c3<0)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

该函数的作用是判断点是否在三角形内，给的三角形是三维，而点是二维，所以实际是判断点是否在空间三角形投影后的平面三角形内，这里采用数学结论跳过投影的操作，提高效率。我们知道在二维平面中，判断点是否在三角形中可以使用叉乘法，这种算法时间复杂度较低，具体步骤如下：

- 首先求出点P到三角形各顶点A,B,C的向量PA,PB,PC
- 将上述三个向量两两叉乘得到三个新的向量C1,C2,C3
- 如果C1~3的方向相同，则点P在三角形内，否则点P在三角形外

三维空间下可以同样使用上述结论，因为二维平面的三角形拉伸到三维后不会改变向量夹角的大小关系。

2. 修改完成void rst::rasterizer::rasterize_triangle(const Triangle& t)函数

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

```
    // TODO : Find out the bounding box of current triangle.
    int xmin = std::min(std::min(v[0].x(), v[1].x()), v[2].x());
    int xmax = std::max(std::max(v[0].x(), v[1].x()), v[2].x());
    int ymin = std::min(std::min(v[0].y(), v[1].y()), v[2].y());
    int ymax = std::max(std::max(v[0].y(), v[1].y()), v[2].y());
    // iterate through the pixel and find if the current pixel is inside the
triangle
    for(int x = xmin; x<=xmax; x++)
    {
        for(int y = ymin; y<=ymax; y++)
        {
            if(insideTriangle(x+0.5, y+0.5, t.v)) // 注意使用像素中心点来判断
            {
                // If so, use the following code to get the interpolated z
value.
                float alpha, beta, gamma;
                std::tie(alpha, beta, gamma) = computeBarycentric2D(x+0.5,
y+0.5, t.v); // 函数返回值是一个tuple元组, 可以使用std::tie接收
                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.
                if(z_interpolated < depth_buf[get_index(x, y)])
                {
                    depth_buf[get_index(x, y)] = z_interpolated;
                    Vector3f point = {(float)x, (float)y, 0};
                    set_pixel(point, t.getColor());
                }
            }
        }
    }
}
```

该函数的作用是栅格化三角形。函数内部工作流程如下：

- 创建三角形的 2 维 bounding box
- 遍历此 bounding box 内的所有像素(使用其整数索引)。然后,使用像素中心的屏幕空间坐标来检查中心点是否在三角形内
- 如果在内部,则将其位置处的插值深度值 (interpolated depth value) 与深度缓冲区 (depth buffer) 中的相应值进行比较
- 如果当前点更靠近相机,请设置像素颜色并更新深度缓冲区 (depth buffer)

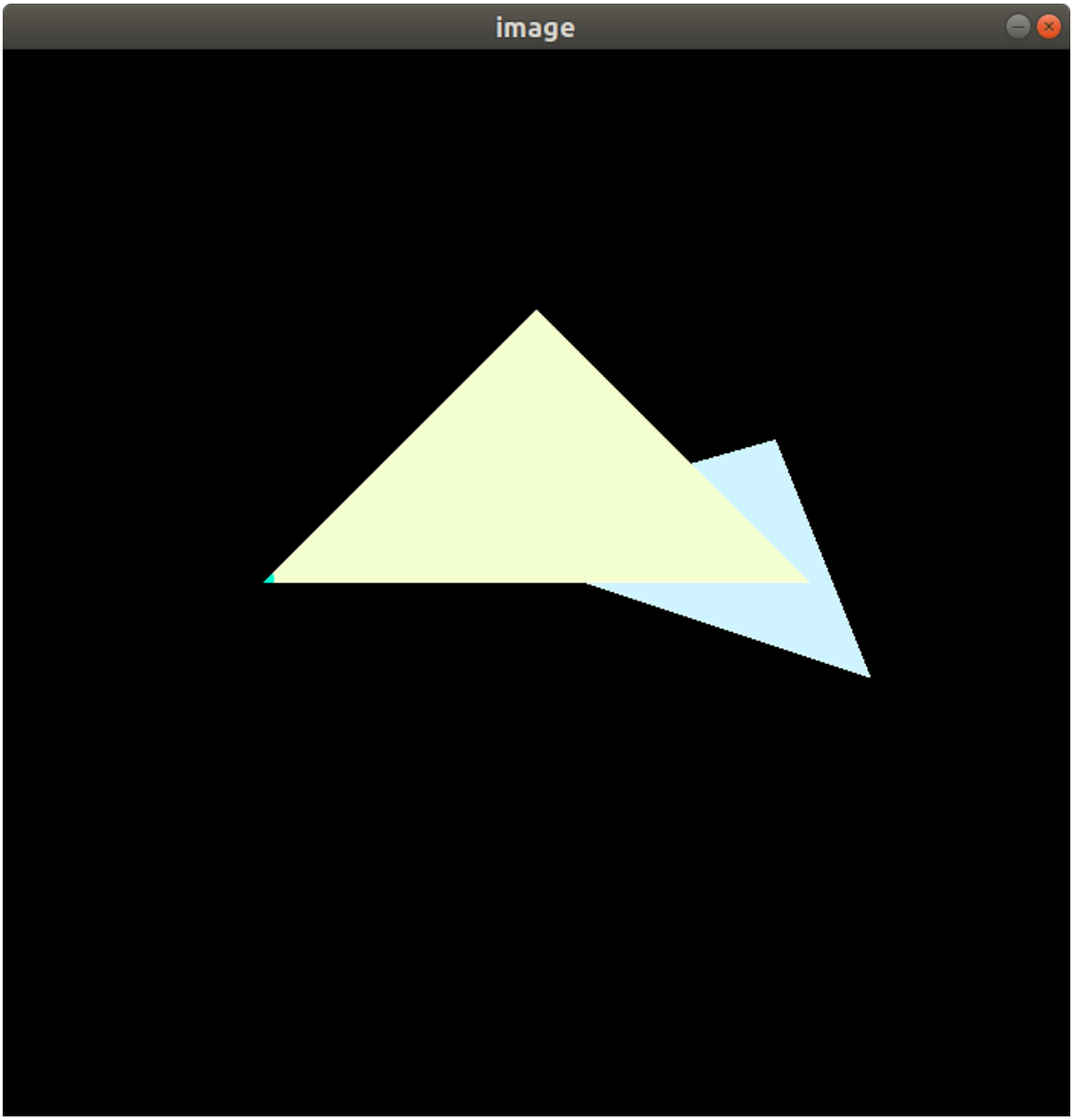
3. 执行代码

输入如下shell命令：

```
mkdir build
cd build
cmake ..
make -j4
./Rasterizer
```

编译成功

```
cs18@games101vm:/mnt/hgfs/GAMES101/2/代码框架/build$ make -j8
Scanning dependencies of target Rasterizer
[ 25%] Building CXX object CMakeFiles/Rasterizer.dir/rasterizer.cpp.o
[ 50%] Linking CXX executable Rasterizer
[100%] Built target Rasterizer
```



3. **提高部分**

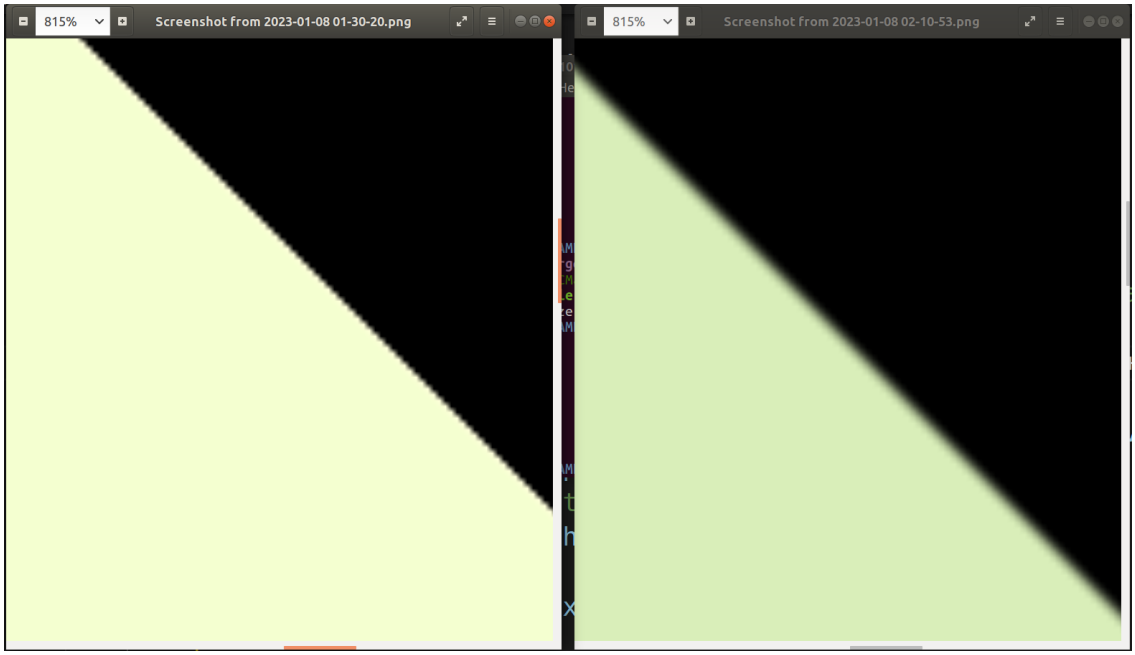
上述算法由于只考虑了单个像素导致边缘锯齿的产生，超采样算法（MSAA）是一种图形技术，用于提高图片质量。它的原理是在每个像素周围采样多个像素，对这些像素进行平均，从而达到抗锯齿的效果。下面是实现3x3超采样的实验代码：

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

    // TODO : Find out the bounding box of current triangle.
    int xmin = std::min(std::min(v[0].x(), v[1].x()), v[2].x());
    int xmax = std::max(std::max(v[0].x(), v[1].x()), v[2].x());
    int ymin = std::min(std::min(v[0].y(), v[1].y()), v[2].y());
    int ymax = std::max(std::max(v[0].y(), v[1].y()), v[2].y());
    // iterate through the pixel and find if the current pixel is inside the
triangle
```

```
    for(int x = xmin; x<=xmax; x++)
    {
        for(int y = ymin; y<=ymax; y++)
        {
            // 提高部分 MSAA超采样(3x3)
            std::vector<Vector3f> sample_list;
            int inside_num=0;
            for(int dx = -1; dx<=1; dx++)
            {
                for(int dy = -1; dy<=1; dy++)
                {
                    if(insideTriangle(x+dx+0.5, y+dy+0.5, t.v)) // 如果不在三角
形内则默认加上背景色RGB{0, 0, 0}
                    {
                        sample_list.push_back(t.getColor());
                        inside_num++;
                    }
                }
            }
            Vector3f sample_color={0, 0, 0};
            for(auto & s : sample_list)
            {
                sample_color += s;
            }
            sample_color /= 9;
            std::cout<<sample_color.x()<<std::endl;
            // If so, use the following code to get the interpolated z value.
            float alpha, beta, gamma;
            std::tie(alpha, beta, gamma) = computeBarycentric2D(x+0.5, y+0.5,
t.v); // 函数返回值是一个tuple元组, 可以使用std::tie接收
            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.
            if(z_interpolated < depth_buf[get_index(x, y)])
            {
                depth_buf[get_index(x, y)] = z_interpolated;
            }
            if(inside_num>0)
            {
                // std::cout<<"inside_num = "<<inside_num<<std::endl;
                Vector3f point = {(float)x, (float)y, 0};
                set_pixel(point, sample_color);
            }
        }
    }

}
```

代码思路是对于每一个像素，采样它周围其他8个子像素的颜色，作平均，然后赋值给原像素。

比较使用MSAA前后图像的变化：



实验总结

在写栅格化代码时，我一开始疏忽了题目所述对于像素采用整数索引的条件，而是使用浮点数，导致结果图像位置有偏移，究其原因是因为浮点数在转化为整数时有精度损失，损失程度不同，导致图像偏移。写MSAA算法也犯了没有初始化三维向量和除以0导致NAN的错误，好在Debug解决了，要注意算法逻辑。