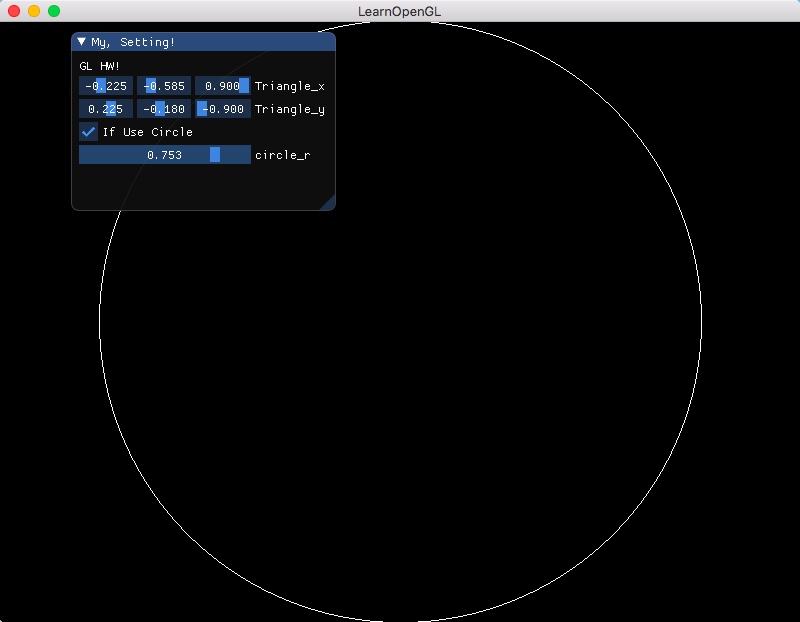
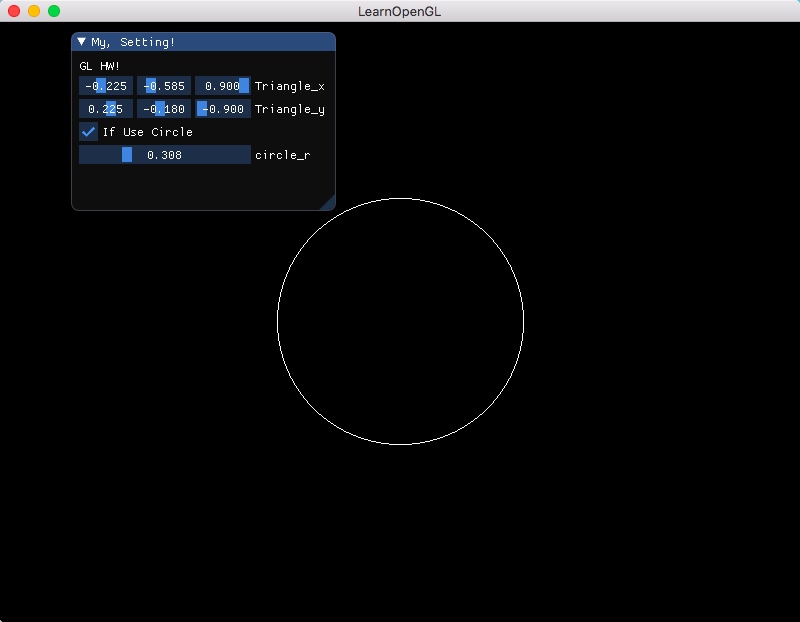
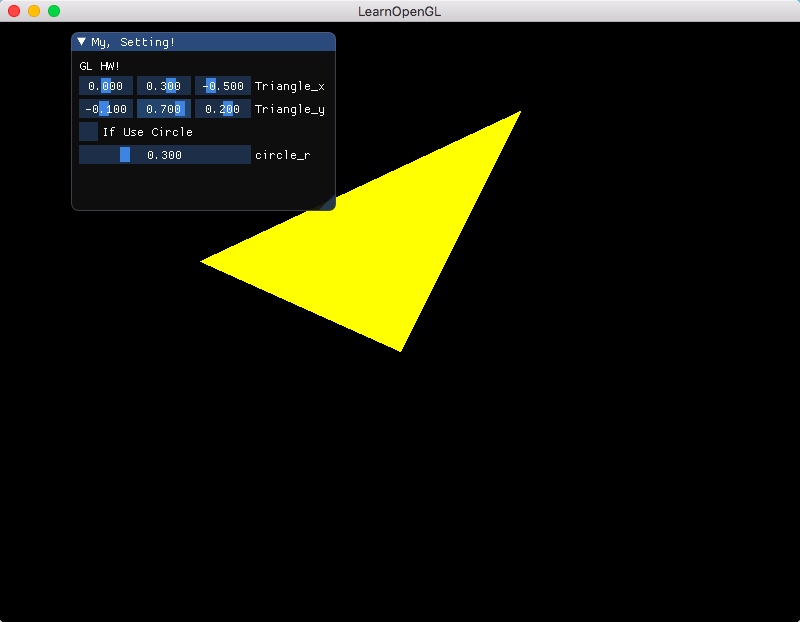
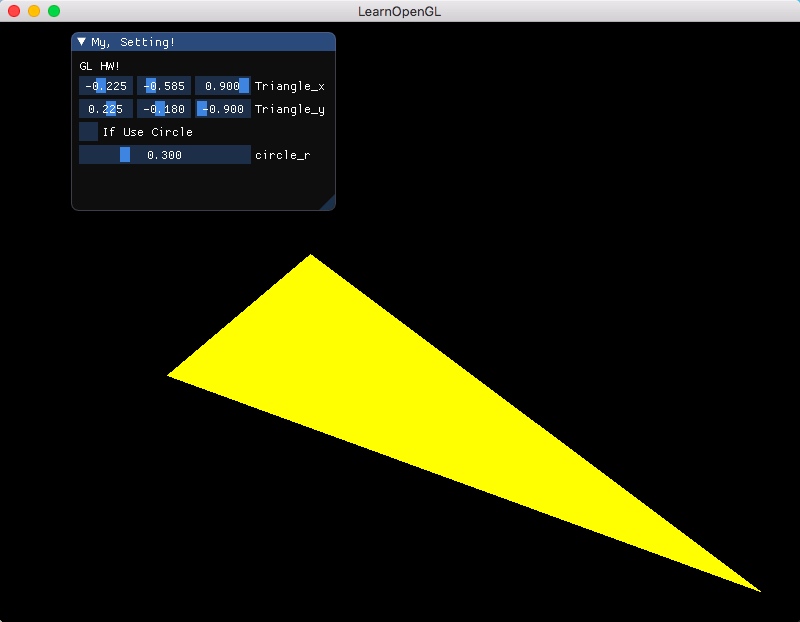
**图形学作业 HW3**

学号：16340315

姓名：朱俊凯

gif录制文件在(./mv.gif)里面

**运行截图：**



**作业里提出的问题：**

本次作业无问题。

**实现思路：**

Bresenham其实就是中点在线上方，那么就取下面的点，中点在线下方，那么就取上面的点，另外，Bresenham将计算简化为只用简单的加乘法和判断，加快了点的判断和画点的效率。

1.画线

\* 使用Bresenham得 计算带入得,将原式子乘2得到，重新计算得。通过这条式子我们就能快速的通过这个点来确定下一个中点的值，当F（中点）<0时，说明中点在直线上方，那么下一个点应该取（x+1，y），当F(中点)>0，则中点在直线下方，那么下一个点应该取（x+1，y+1）；通过这样类推就可以将所有的点的位置。即，每次循环x都加1，当F值>0时，y才加1

\* 因为上式是当的情况，所以当就不一样，但大致差不多，就是把上式的x和y的位置对调即可，即，每次循环y都加1，当F值>0时，x才加1。

\* 由于不是所有的x1>x,y1>y，因此我们要提取出系数，让他们符合这个条件，之后再乘回来即可。

2.画三角形

\* 就是将三个点两两画线，就画出了三角形的三条边了。

3.画圆

\* 因为圆关于x,y轴很对称，所以，我们将圆8等分，计算一份的点即可推导出其他的点

\* 我们取的是右上的1/8的圆，即x的变化率大于y的变化率，同理，先计算误差函数F(x,y)=，当F((x,y)+)<0，取下方的点，反之，取上方的点。同理计算得F(x + 2, y - 1.5)= F(x + 1, y - 0.5) + 2x - 2y + 5以及F(x + 2, y - 0.5)= F(x + 1, y - 0.5) + 2x + 3，这样就能快速推导出下一个点的y值是否需要-1，就可以画出所有的点了。

4. 选择是三角形边框还是圆，以及能调整圆的大小

\* 用一个bool变量，绑定ImGui::Checkbox，控制true和false，让后判断来显示三角或者圆。

\* 用一个float变量，去绑定一个ImGui::SliderFloat，设置范围0.1f-0.9f，将这个变量作为半径参数输入到圆的绘制函数里面。

5. 使用三角形光栅转换算法，用和背景不同的颜色，填充你的三角形。

\* 使用线扫描，取y值中等的点，水平切割，把三角形分为2块。

\* 从切割处根据线的数据向上和向下扫描，将两个y值相等的点中间的所有的点都加到点的数组里面。

**主要function/algorithm：**

**class** location{} //记录像素的横纵坐标

vector<location> getLine(**float** x,**float** y,**float** x1,**float** y1); //根据两个点，使用Bresenham算法，获取所有中间的像素位置数组

vector<location> getinsidepoint(**float** x,**float** y,**float** x1,**float** y1,vector<location>& result,vector<location>& result1); //根据两个点的位置，和两个线上的像素位置，获取上（下）半三角形中间的所有填充像素的位置

vector<location> getTriangle(**float** x,**float** y,**float** x1,**float** y1,**float** x2,**float** y2); //根据3个点，获取3条边上的像素位置，和上（下）半三角形中间的所有填充像素的位置

**int** setup\_Triangle(**float** x,**float** y,**float** x1,**float** y1,**float** x2,**float** y2,**unsigned** **int** &VAO,**unsigned** **int** &VBO,**int** shaderProgram); //绘制三角形

**int** setup\_Circle(**unsigned** **int** &VAO,**unsigned** **int** &VBO,**int** shaderProgram,**float** circle\_r); //绘制圆形

**void** plotCirclePixel(**int** cx, **int** cy,**int** x, **int** y,vector<location>& points); //根据圆的对称原理，将一个像素对称成8个像素