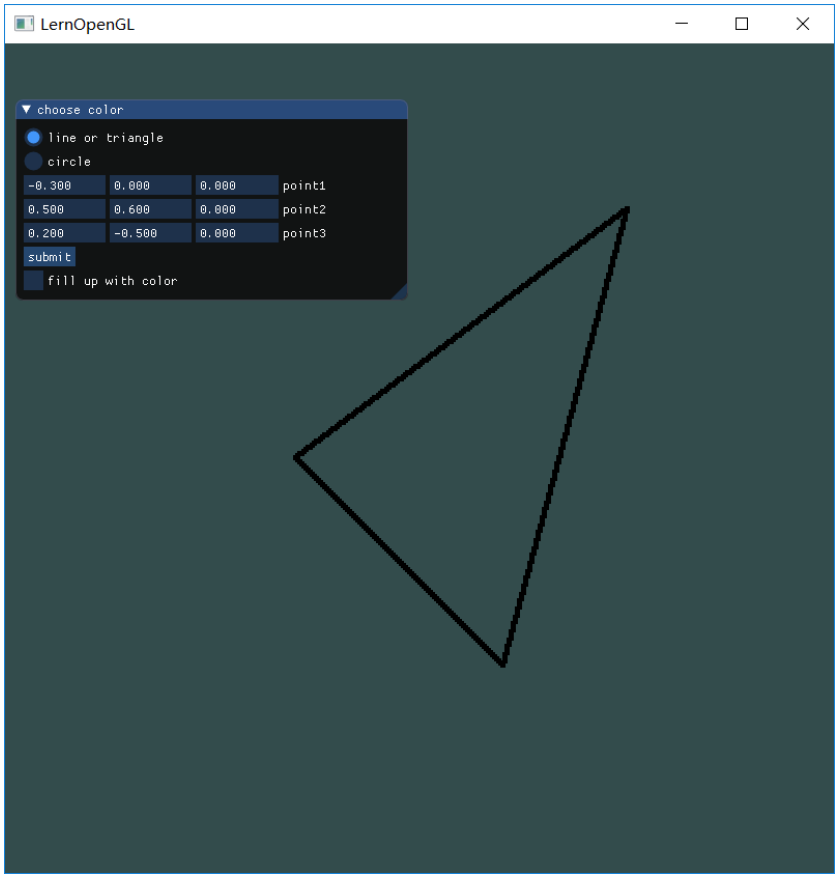
实验报告

1. 用Bresenham算法画出三角形框

实验结果：



实现思路：

1. Bresenham算法

   即用多个点模拟直线。需要计算下一个点的坐标，对于斜率小于1的直线来说，下一个点x的坐标+1，y的坐标需要通过计算y_i和y_i+1和y_{i+1}的距离，取最近的那个点来拟合。

   通过计算可以通过计算两个距离的差的符号来判断，简化计算式子得到

   $$p_i = 2\Delta y * x_i - 2\Delta x * y_i + c$$

   通过计算$p_{i+1} - p_i = 2\Delta y - 2\Delta x(y_{i+1} - y_i)$ 遍历获得每一个p

   对于斜率大于一的直线，只需要将上面式子的x和y互换

2. 实现过程

   三条直线是一条一条画的，下面是一条直线的过程

   （1） 创建窗口、初始化ImGui等（和上一次作业一样）

   （2） 通过计算Δx和Δy，获得斜率是否大于1，下面是斜率小于1的情况，斜率大于一是将下面步骤的xy操作互换

   （3） 将第一个点加入数组，计算p_0

   （4） 如果p>0，将[x+1, y+1]加入数组，通过$p_{i+1} = p_i + 2\Delta y - 2\Delta x$计算p，将x+1和y+1分别替代x,y；若p<0，将[x+1,y]加入数组，通过$p_{i+1} = p_i + 2\Delta y$计算p，将x+1和y分别替代x,y

   （5） 重复步骤3，直到 x = 终点的x坐标

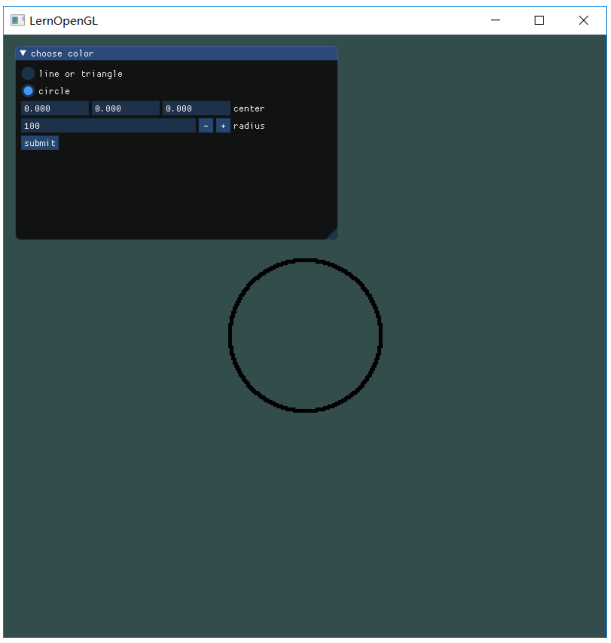
(6) 顶点数组对象：通过glGenVertexArrays函数创建一个VAO对象，通过glBindVertexArray绑定，通过glBindBuffer将VBO存到缓冲中，通过glBufferData将数组复制到缓存，通过glVertexAttribPointer和glEnableVertexAttribArray函数设置顶点属性指针。

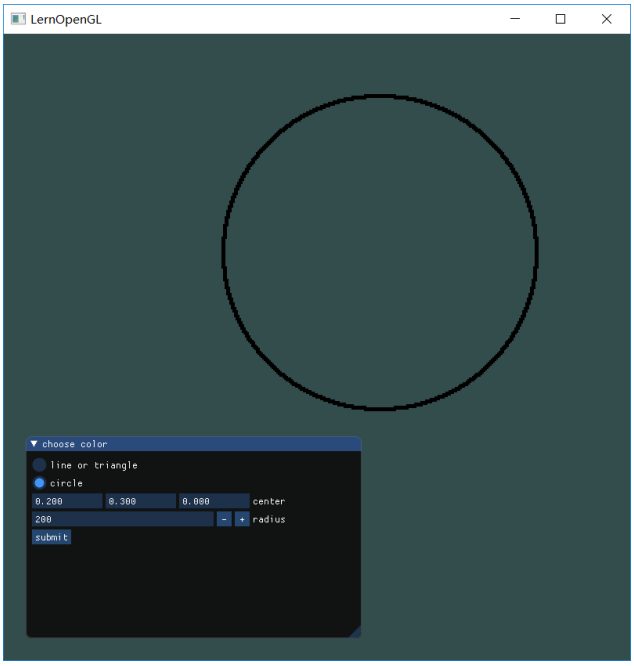
(7) 绘制物体：通过glUseProgram函数激活程序对象，通过glBindVertexArray绑定VAO，再通过glDrawArrays函数绘制图形。（由于+的值不是一个像素的值，这里通过函数将点的大小设为5倍）

3. 由于这里的坐标是相对坐标，不能对应相应像素点，因此在程序代码里依旧涉及到浮点运算（这里将坐标值+0.005视为像素+1）
4. 输入参数：三个点的坐标（-1，1），点击submit即可绘制

2. 使用Bresenham算法画圆

实验结果：



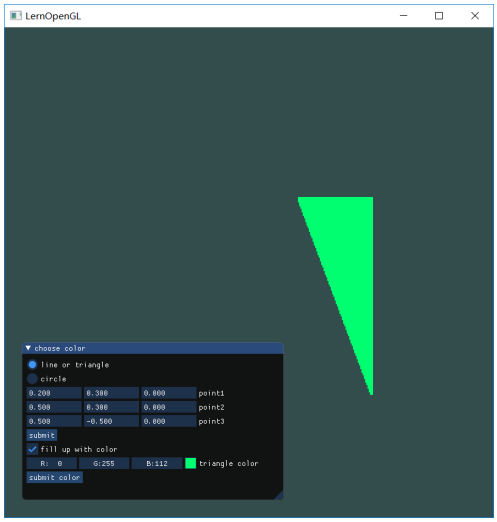


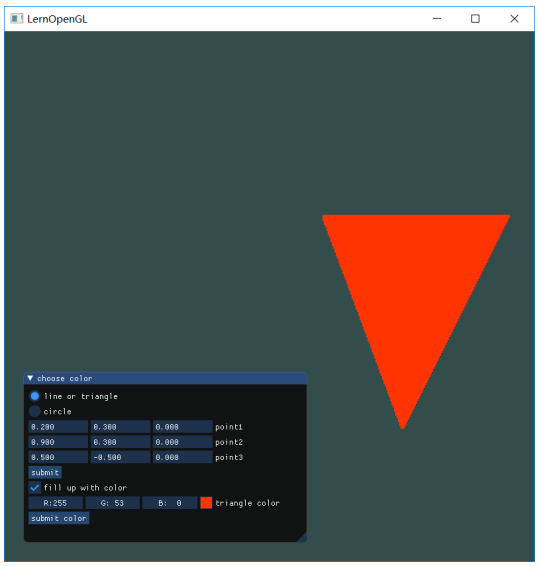
实现思路：

1. Bresenham算法画圆公式
   (1) 第一个点是圆心上方的点，计算下一个点为(x_i+1, y_i+1)或者(x_i+1, y_i)哪一个点和本来的下一个点更近
   (2) 计算和圆上的点的距离的平方的差得到$p_i = 2r^2 - 2(x_i+1)^2 - 2y_i^2 + 2y_i - 1$
   (3) 计算 $p_{i+1} - p_i$，得到 $4x_i + 6 + c$，其中当 $y_{i+1} = y_i$时，c=0，当$y_{i+1} = y_i - 1$，$c = -4y_i + 4$
   (4) 其中$p_0 = 3 - 2 * r$
2. 实现过程：由圆的对称性，只需要画1/8个圆，然后通过与x轴对称，y轴对称原点对称，直线y=x和直线y=-x对称获得其他七个点
   (1) 创建窗口、初始化ImGui等（和上一次作业一样）
   (2) 将原点上的点（圆心x坐标，圆心y坐标+r）和这个点的其他对称点放入vector，计算p_0
   (3) 如果p>0，将点(x+1, y-1)及其对称点放入vector中，通过$p = 4(x - y) + 10$计算p，将x+1和y-1替换x,y；如果p<0，将点(x+1, y)及其对称点放入vector中，通过$p = 4x + 6$计算p，将x+1和y替换x,y
   (4) 重复步骤3，直到x>y
   (5) 顶点数组对象：通过glGenVertexArrays函数创建一个VAO对象，通过glBindVertexArray绑定，通过glBindBuffer将VBO存到缓冲中，通过glBufferData将数组复制到缓存（其中传入参数中，个数为sizeof(float)* vertice.size()，传入数据为vertice.data()），通过glVertexAttribPointer和glEnableVertexAttribArray函数设置顶点属性指针。
   (6) 绘制物体：通过glUseProgram函数激活程序对象，通过glBindVertexArray绑定VAO，再通过glDrawArrays函数绘制图形。（由于+的值不是一个像素的值，这里通过函数将点的大小设为5倍）
3. 这里依旧是相对坐标，因此传入的r会通过计算获得对于这个坐标系的r值，由于全部的1都是变成0.005，因此上面式子里的所有3、6、10都要乘0.005
4. 输入参数：点的坐标（相对坐标），半径（像素值）

3. 三角形光栅转换算法填充三角形

实验结果：





实现思路：

1. 判断点是否在三角形内
   根据向量叉乘的定理可以得出，只要向量间夹角同样小于或大于180°，叉乘结果的正负是相同的。也就是说在三角形内任意一点P，PA×PB、PB×PC、PC×PA三个值正负相同。
2. 实现过程：这里三角形用第一题画出来的三角形
   (1) 创建窗口、初始化ImGui等（和上一次作业一样）
   (2) 遍历由三角形三个顶点最小x，最大x，最小y，最大y构成的矩形内的每一个点，计算PA×PB、PB×PC、PC×PA的值（叉乘：PA.x*PB.y-PA.y*PB.x）当这三个值同正负时，将这个点加入vector，同时将选择的颜色加入vector
   (3) 顶点数组对象：通过glGenVertexArrays函数创建一个VAO对象，通过glBindVertexArray绑定，通过glBindBuffer将VBO存到缓冲中，通过glBufferData将数组数据复制到缓冲中，通过glVertexAttribPointer和glEnableVertexAttribArray函数设置顶点属性指针，其中这里要设置位置属性和颜色属性。
   (4) 绘制物体：通过glUseProgram函数激活程序对象，通过glBindVertexArray绑定VAO，再通过glDrawArrays函数绘制图形。
3. 这里用到的三角形是前面画出来的三角形