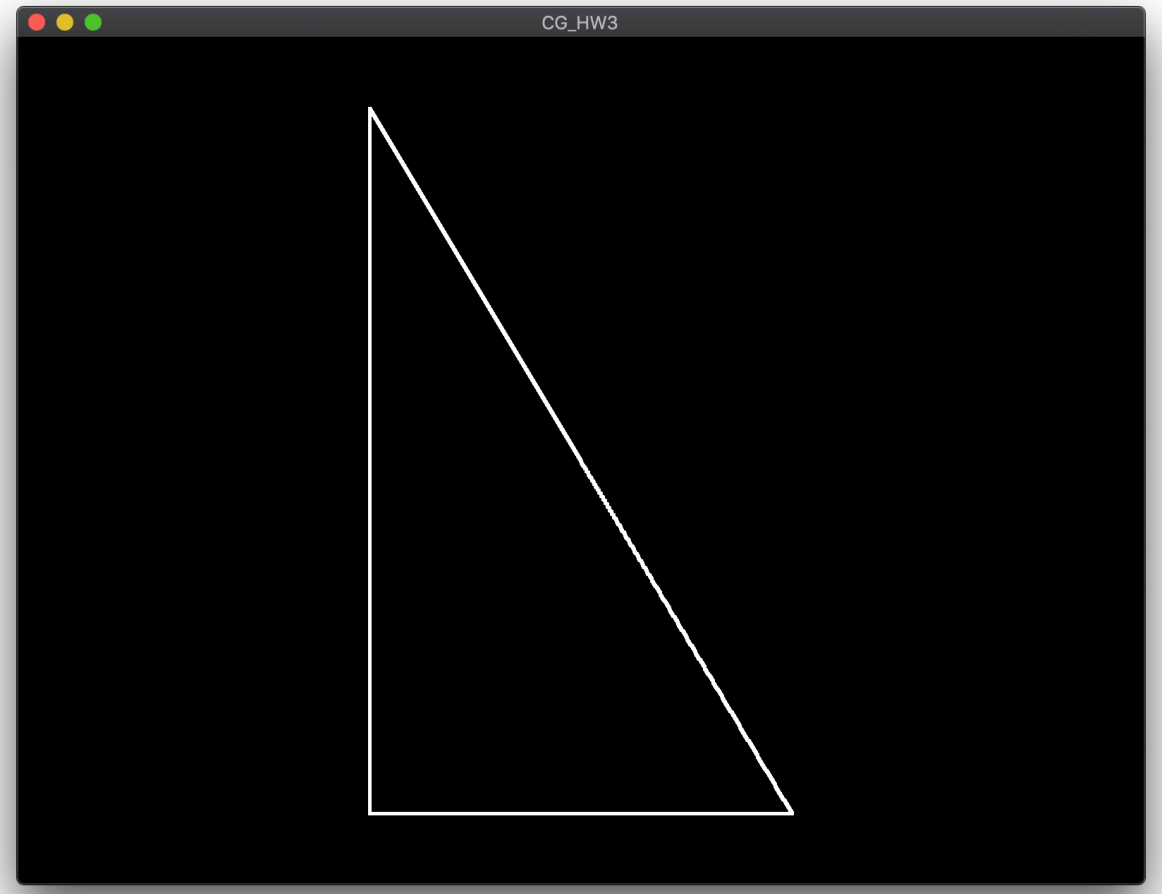
张吉祺 16340286

Basic:

1. 使用Bresenham算法(只使用integer arithmetic)画一个三角形边框:input为三个2D点;output三条直线(要求图元只能用 GL_POINTS ，不能使用其他，比如 GL_LINES 等)。



修改**HW2_v0:Basic:1.**的代码，主要是改变float* coordinates数组的内容，这个数组用来存放点的x、y、z轴坐标，它会被拷贝到VBO输入供顶点着色器进行绘制。

添加void swap(int& a, int& b)函数用于交换两个int变量的值。

添加std::vector<int> bresenhamLine(int x_start, int y_start, int x_end, int y_end)用于给定两个端点的xy坐标，对这两点之间线段进行光栅扫描转换，返回xy坐标int类型的向量。具体实现是先判断第一个点的x分量和第二个点的x分量的大小，如果第一个点的x分量大于第二个点的x分量就将这两个点进行交换（交换后第一个点是原来的第二个点，交换后第二个点是原来的第一个点）。接着判断第一个点的y分量和第二个点的y分量的大小，如果第一个点的y分量大于第二个点的y分量（说明斜率为负数）就将这两个点进行y轴的轴对称变换（变换后第一个点的x分量是变换前第二个点的x分量的相反数，变换后第二个点的x分量是变换前第一个点的x分量的相反

数，其他不变），在后面的计算需要对x分量取相反数。接着判断第二个点和第一个点的x分量的差和第二个点和第一个点的y分量的差的大小，如果后者大（说明斜率大于1）就将各个点的x分量换成y分量，y分量换成x分量，在后面的计算换回来。经过这三个变换过程后在变换后的空间上第一个点就在第二个点的左下方且斜率小于1，就可以用Bresenham算法计算。

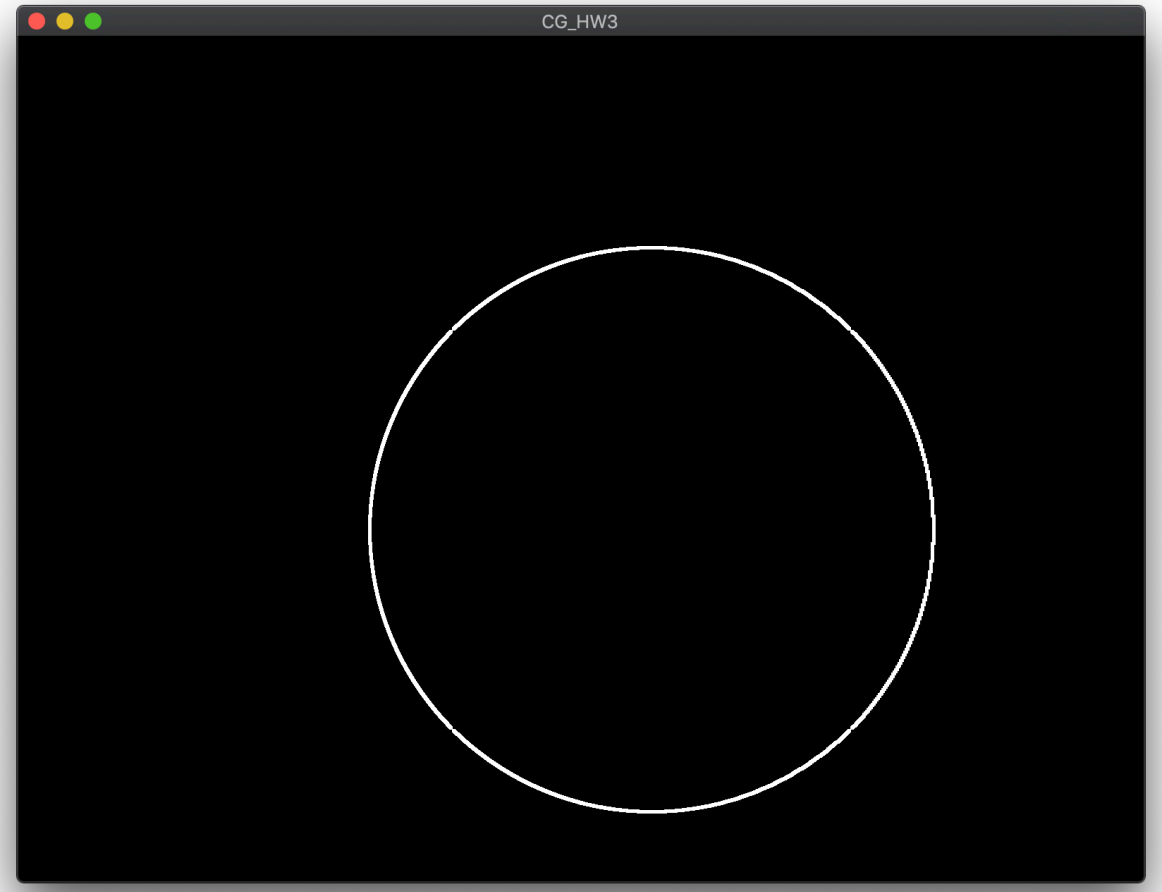
- **draw** (x_0, y_0)
- **Calculate** $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x, p_0 = 2\Delta y - \Delta x$
- **If** $p_i \le 0$ **draw** $(x_{i+1}, \bar{y}_{i+1}) = (x_i + 1, \bar{y}_i)$

  and compute $p_{i+1} = p_i + 2\Delta y$
- **If** $p_i > 0$ **draw** $(x_{i+1}, \bar{y}_{i+1}) = (x_i + 1, \bar{y}_i + 1)$

  and compute $p_{i+1} = p_i + 2\Delta y - 2\Delta x$
- **Repeat the last two steps**

用六个int类型的变量定义三角形的三个顶点的xy坐标，每两个顶点调用bresenhamLine获得线段上离散点的xy坐标的向量，将这三个向量合并，再通过float* normalize(std::vector<int> coordinates_xy, int cnt)规格化到[-1, 1]之间、添加0.0f的z坐标，并返回float* coordinates数组，它会被拷贝到VBO输入供顶点着色器进行绘制。

2.使用Bresenham算法(只使用integerarithmetic)画一个圆:input为一个2D点(圆心)、一个integer半径;output为一个圆。



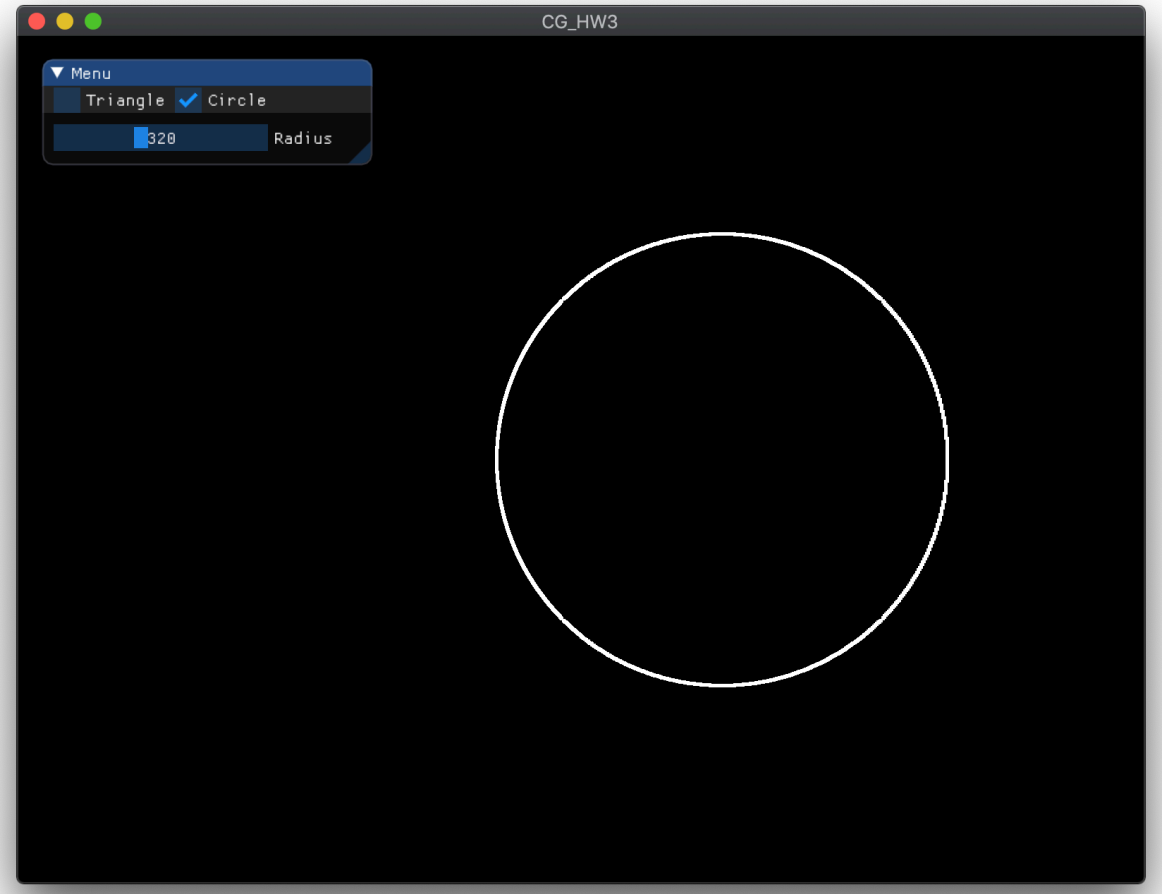
修改**1.**，将std::vector<int> bresenhamLine(int x_start, int y_start, int x_end, int y_end)替换为std::vector<int> bresenhamCircle(int c_x, int c_y, int r)，用于给定圆心的xy坐标和半径，对圆线段进行光栅扫描转换，返回xy坐标int类型的向量。

具体实现是先对从(0,r)到($\frac{r}{\sqrt{2}}$,$\frac{r}{\sqrt{2}}$)的八分之一圆用Bresenham算法计算。

- Draw (x_0, y_0)
- Caculate $d_0 = 3 - 2r$
- If $d_i \leq 0$ draw $(x_{i+1}, \bar{y}_{i+1}) = (x_i + 1, \bar{y})$
  and compute $d_{i+1} = d_i + 4x_i + 6$
- If $d_i > 0$ draw $(x_{i+1}, \bar{y}_{i+1}) = (x_i + 1, \bar{y} - 1)$
  and compute $d_{i+1} = d_i + 4(x_i - y_i) + 10$
- Repeat the last two steps

然后重复利用这八分之一圆的坐标计算剩下的七段八分之一圆，每次根据被计算的段的xy坐标与这八分之一圆的坐标xy坐标的关系赋值，比如是关于坐标轴对称的就取特定的相反数，关于与坐标轴成45°夹角的直线对称的就xy交换位置赋值，并根据情况决定是否取相反数。最后对这八段八分之一圆的x坐标都加上圆心的x坐标，y坐标都加上圆心的y坐标。

3. 在GUI在添加菜单栏，可以选择是三角形边框还是圆，以及能调整圆的大小(圆心固定即可)。

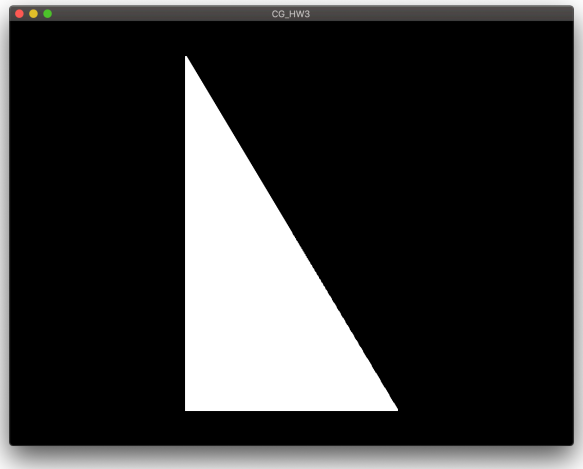


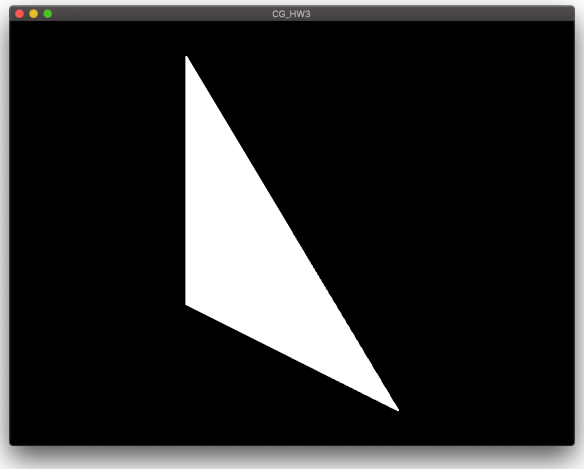
修改**1.2.**，引入ImGui，在渲染循环中加入if(ImGui::BeginMenuBar())创建一个菜单栏，放入ImGui::Checkbox("Triangle", &showTriangle)和ImGui::Checkbox("Circle", &showCircle)选框，通过对布尔值showTriangle和showCircle的控制选择显示三角形还是圆。当选择显示圆时，ImGui::SliderInt("Radius", &r, 2, 800)显示滑块，可进行半径值的调节，每个循环都会调用std::vector<int> bresenhamCircle(int c_x, int c_y, int r)重新生成坐标再拷贝给VBO。

张吉祺 16340286

Bonus:

1. 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。





修改**Basic:1.**的std::vector<int> bresenhamLine(int x_start, int y_start, int x_end, int y_end)为void bresenhamLine(std::priority_queue<std::pair<int, int>>& coordinates_xy, int x_start, int y_start, int x_end, int y_end)，原来是返回一个std::vector<int>容器，调用三次函数（三角形每两个点调用一次得到一条边上点的坐标）后合并三个std::vector<int>，现在改成输入一个std::priority_queue<std::pair<int, int>>容器的引用，调用三次函数，对这个容器添加三条边的坐标元素，并且每个元素std::pair<int, int>的first赋值为y坐标，second赋值为x坐标，使用priority_queue的好处是所有坐标会跟据y坐标降序排列（便于使用水平扫描线方法），如果y坐标相同，就按x坐标降序排列。

再添加std::vector<std::pair<int, int>> rasterization(std::priority_queue<std::pair<int, int>> coordinates_xy)，输入包含三角形三条边的坐标的coordinates_xy容器，输出为std::vector<std::pair<int, int>> rasterized_coordinates_xy。进行如下循环直到coordinates_xy容器为空：pop出top元素赋值给end变量（扫描线的右端点），再检查跟新的top元素的y坐 标是否相同，如果相同就pop出新的top元素赋值给start变量（扫描线的左端点），如果不同就让start变量等于end变量；接着就从start的x坐标遍历到end变量的x坐标（y坐标不变），push_back到rasterized_coordinates_xy容器（这里的元素std::pair<int, int>的first赋值成x坐标，second赋值成y坐标），这样这条扫描线上的点就都保存到rasterized_coordinates_xy容器中了，接着进入下个循环。当coordinates_xy空时，三角形的三条边中相同y坐标的点对形成的水平扫描线上的点就都保存在rasterized_coordinates_xy容器中了。

接着再将rasterized_coordinates_xy输入到float* normalize(std::vector<std::pair<int, int>> rasterized_coordinates_xy)，输出规格化后的float*坐标数组，再将其拷贝给VBO进行绘制。