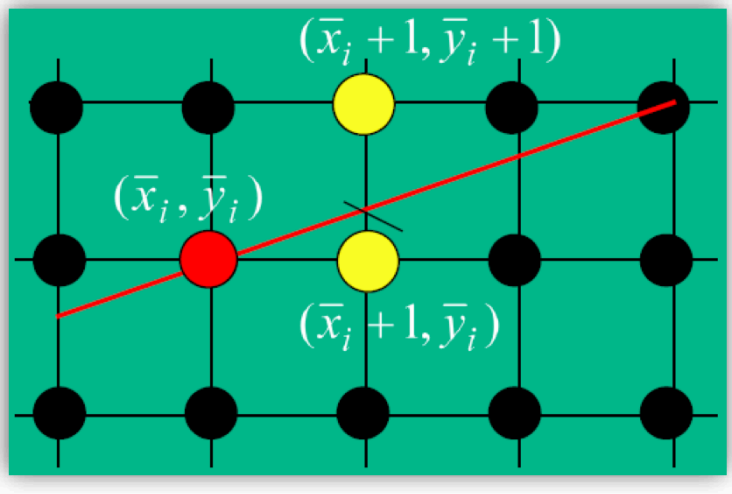
计算机图形学 Homework 3 - Draw Line

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）。
   画三角形边框的方法为对于三角形的三个点，向另外两点作一条直线，使用 Bresenham 算法画线的方法如下：

- 假设线从点(x_0, y_0)到点(x_1, y_1)
- $\Delta x = x_1 - x_0, \Delta y = y_1 - y_0$, 若其$\Delta x < 0$，则交换两点。斜率$m = \frac{\Delta y}{\Delta x}$，若$|m| > 1$则将坐标轴交换后进行画线。
- 设线为$y = mx + B$从x_0开始到 x_1每次步进 1，设当前点为(x_i, y_i)，则根据判定条件决定下一点(x_{i+1}, y_{i+1})为$(\bar{x}_i + 1, \bar{y}_i)$或$(\bar{x}_i + 1, \bar{y}_i + 1)$
- 判别条件如下：

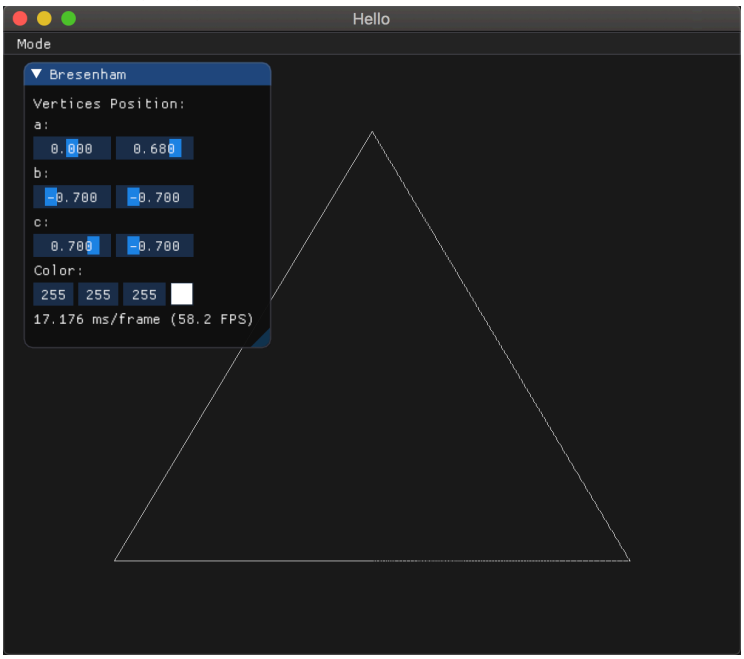


若 $\bar{y}_i + 1 - y_{i+1} < y_{i+1} - \bar{y}_i$，即$p = 2y_{i+1} - 2\bar{y}_i - 1 < 0$ 则取上方黄点，否则取下方黄点。

在实际实现中，由于整数运算在计算机中速度更快，可将判别条件改为$\mathrm{p} = \Delta x \cdot p = 2\Delta y \cdot x_i - 2\Delta x \cdot \bar{y}_i + c$, 其中$c = (2B - 1)\Delta x + 2\Delta y$。

代码实现中，关于 Bresenham 画线部分的代码位于 Bresenham/BresenhamLine.cpp 文件下的 `BresenhamLine::getBresenhamLine` 函数中。

运行时的效果如下：

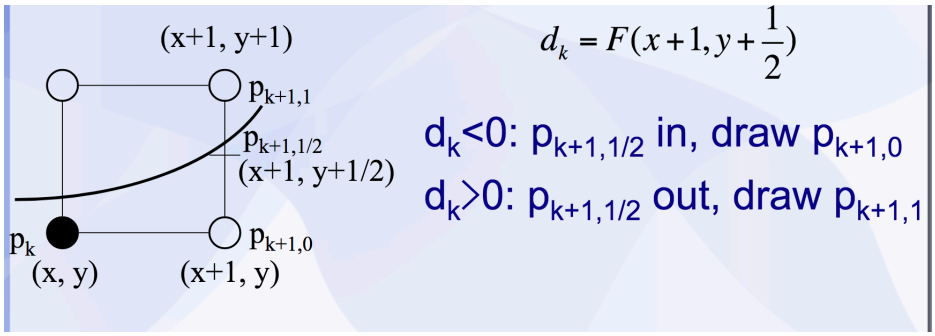


在实现过程中，首先根据用户给出的小数值和目前显示的 `Frame Buffer Size` 计算出整数型的三角形三个顶点的坐标，使用 `Bresenham` 算法获得三条线中所有点的数组后转换回 `OpenGL` 中的小数坐标系数据，最后通过 `glDrawArrays` 使用 `GL_POINTS` 画出。

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

设圆心为(x_c, y_c)，半径为r。Bresenham 算法画圆的方法由 Midpoint Circle Algorithm 派生而来，每次同时画八个点，以圆的每$\frac{\pi}{4}$弧度为一个区间，当第一个区间达到$y = x$时，即到达$\frac{\pi}{4}$弧度。类似于 Bresenham 画线的算法，画圆时从$(x_c + 0, y_c + r)$点出发，x 每次前进 1，根据判定条件决定 y 是否要减 1。根据 x 与 y 的值，可以计算出另外 7 个区间需要画出的点坐标。

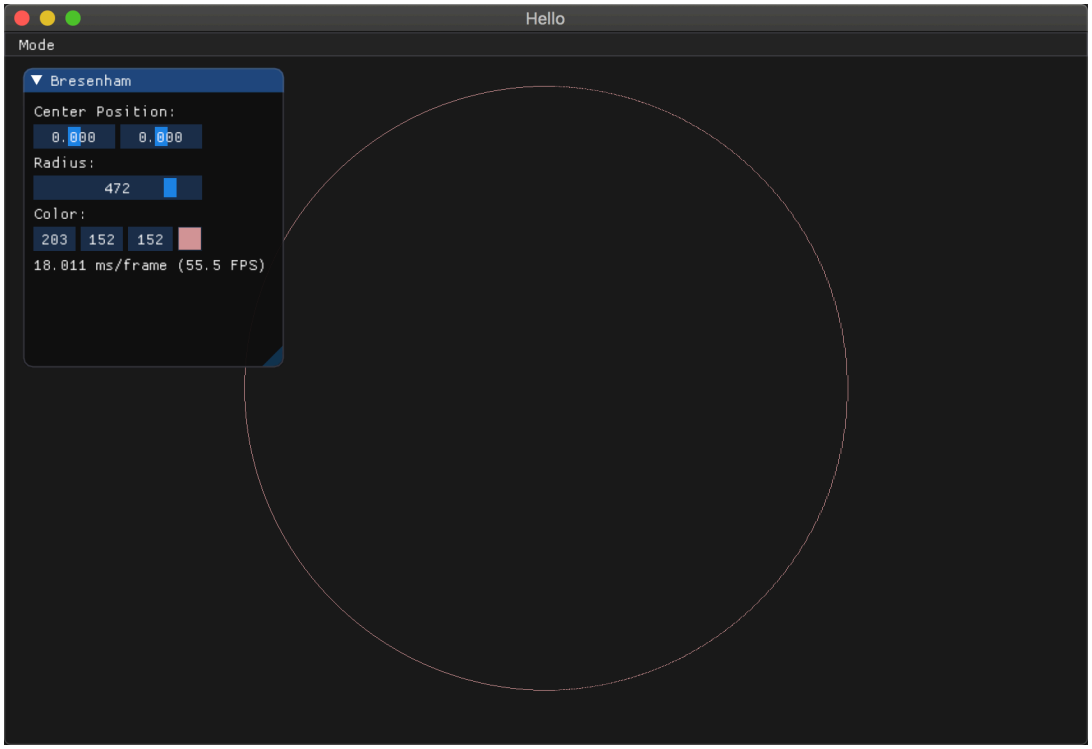
由圆的性质可知，对于点 p，$(x_p - x_c)^2 + (y_p - y_c)^2 - r^2$大于 0 时点处于圆外，小于 0 时点处于圆内，因此对于可能的两个点$(x_i + 1, \overline{y_i}), (x_i + 1, \overline{y_i} - 1)$，求出其中间点$(x_i + 1, \overline{y_i} - 0.5)$的位置，若其位于圆内，则选择$(x_i + 1, \overline{y_i})$，否则选择$(x_i + 1, \overline{y_i} - 1)$。如图：



将第一个中间点代入公式化简后得到判定条件的整数初始值：$D_0 = 1 - r$。当$D_i < 0$时，下一中间点为$(x_i + 2, y_i + \frac{1}{2})$，当$D_i \geq 0$时，下一中间点为$(x_i + 2, y_i + \frac{3}{2})$，代入得$D_{i+1} = \begin{cases} D_i + 2x_i + 3, if\ D_i < 0 \\ D_i + 2(x_i + y_i) + 5, if\ D_i \geq 0 \end{cases}$.

代码实现中使用 Bresenham 算法画圆的部分位于 Bresenham/BresenhamCircle.cpp 文件中的 `getCirclePoints` 函数中，根据传入的圆心坐标和半径的整数值返回圆形各点的坐标数组。

实现效果如下：



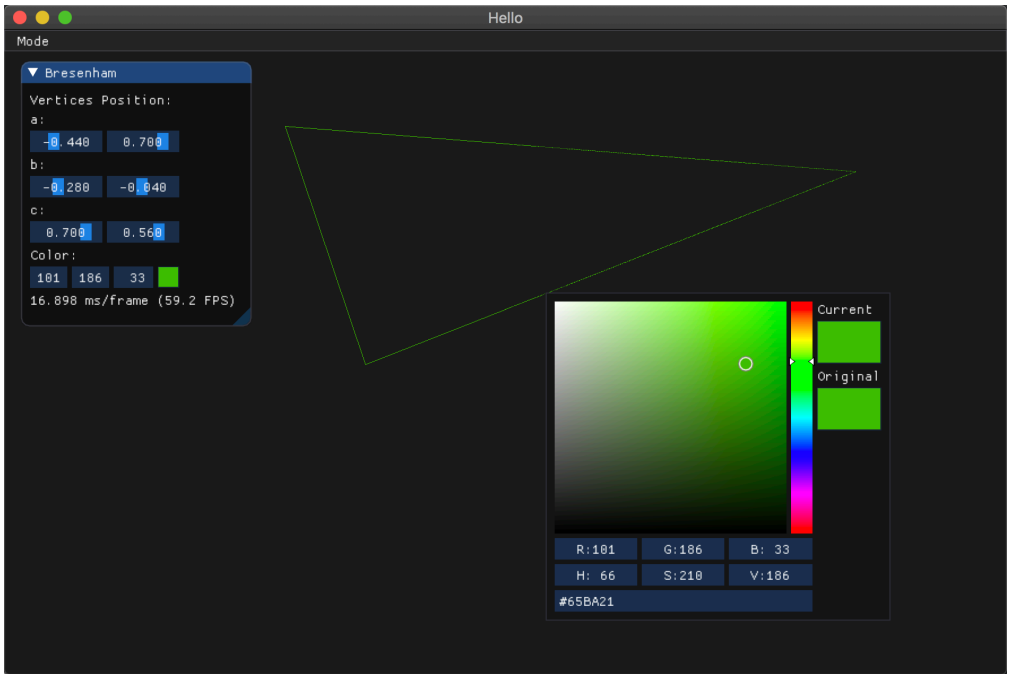
3. 在 GUI 在添加菜单栏，可以选择是三角形边框还是圆，以及能调整圆的大小（圆心固定即可）。

通过 ImGui::BeginMainMenuBar 和 ImGui::BeginMenu 创建菜单，在菜单中使用 ImGui::MenuItem 创建菜单选项，在选项中切换显示的状态，添加的菜单栏的代码如下：

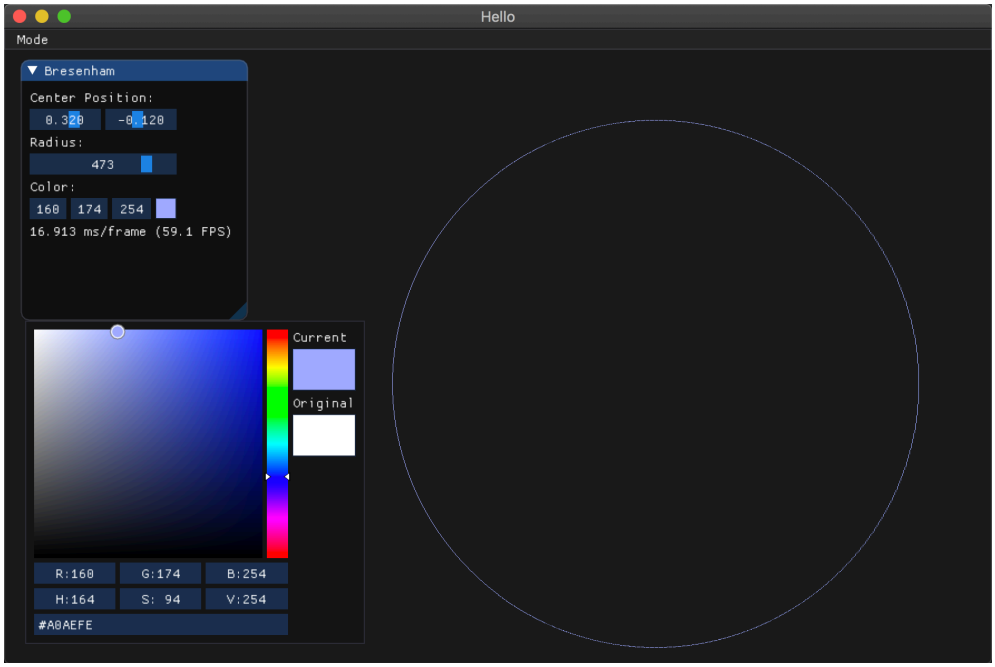
```
{
        if (ImGui::BeginMainMenuBar()) {
            if (ImGui::BeginMenu("Mode")) {
                if (current_mode != CIRCLE && ImGui::MenuItem("Circle")) {
                    current_mode = CIRCLE;
                }
                if (current_mode != TRIANGLE_OUTLINE &&
ImGui::MenuItem("Triangle Outline")) {
                    current_mode = TRIANGLE_OUTLINE;
                }
                if (current_mode != TRIANGLE_FILLED &&
ImGui::MenuItem("Triangle Filled")) {
                    current_mode = TRIANGLE_FILLED;
                }
                ImGui::EndMenu();
            }
            ImGui::EndMainMenuBar();
        }
    }
```

菜单效果如下：

当位于显示三角形边框的状态时，在 GUI 中添加了可以切换边框颜色和三角形三点坐标的选项，效果如下：



位于显示圆形的状态时，GUI 提供了可以切换圆心位置和圆形半径（像素整数数值）的滑动条，以及可以选择圆形的颜色：



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。

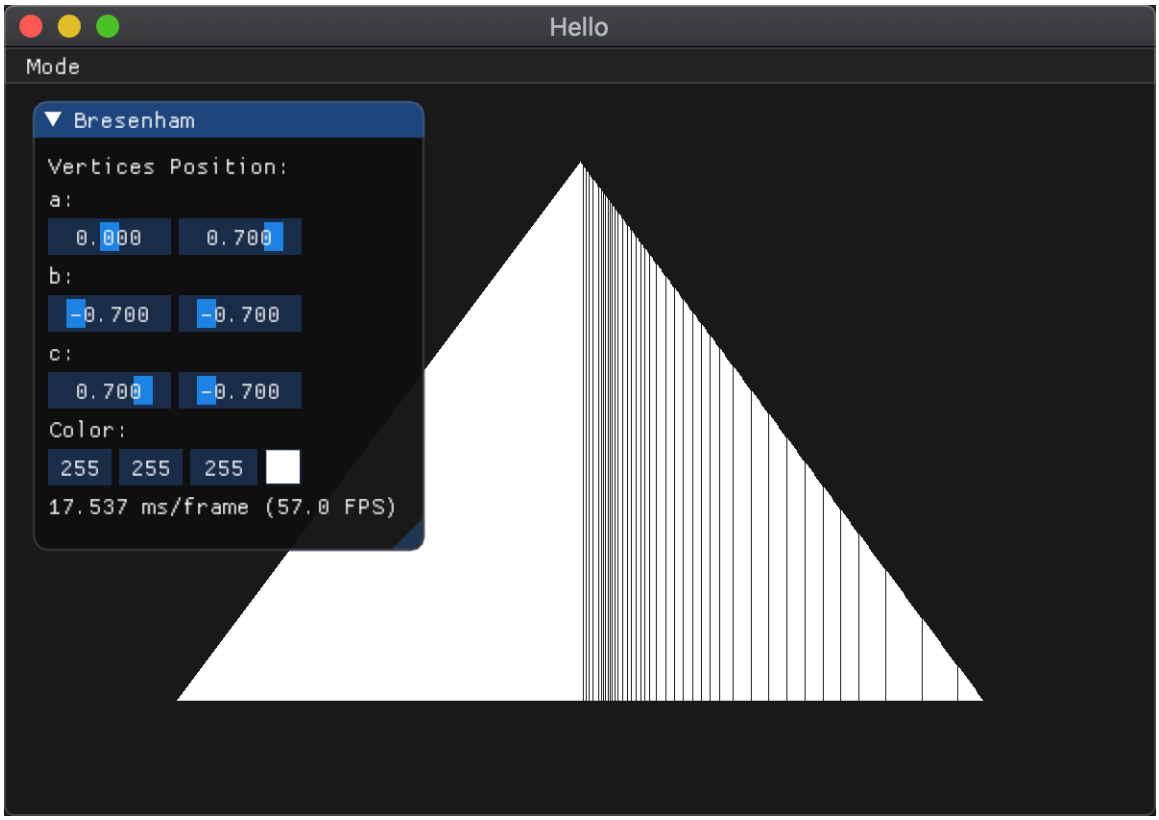
   使用了 Edge Equation 方法进行三角形光栅化，过程如下：

- 对于三角形的三个顶点间的连线，分别计算其直线公式$Ax + By + C = 0$，并保证直线不经过的第三个点代入公式结果小于 0
- 创建矩形的三角形边框，边框四点基于三角形最大和最小的横纵坐标。
- 对边框内的所有像素点进行扫描，若该像素点代入三个直线公式结果均不大于 0，则该点位于三角形内部。

代码实现中与三角形填充相关的内容位于 Bresenham/RasterizedTriangle.cpp 文件中的 `RasterizedTriangle::draw` 函数。

实现中将根据当前显示大小将 OpenGL 中的坐标转为像素的坐标，计算出填充的每一个像素点坐标的数组后将其转回至 OpenGL 坐标进行显示。

效果如下：



程序中 GUI 提供了对三个顶点坐标进行修改的选项，以及更改颜色的选项。效果如下：