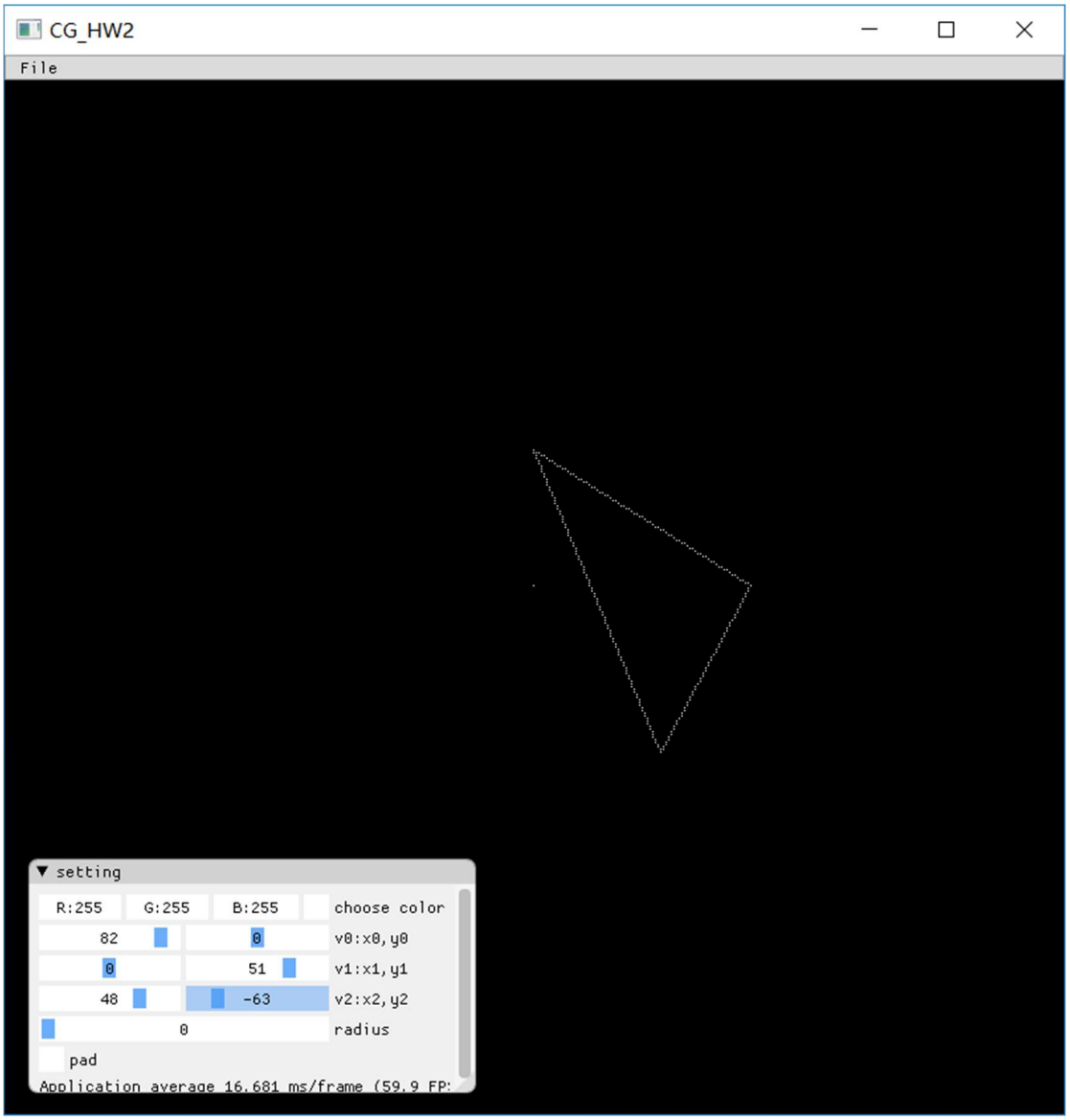
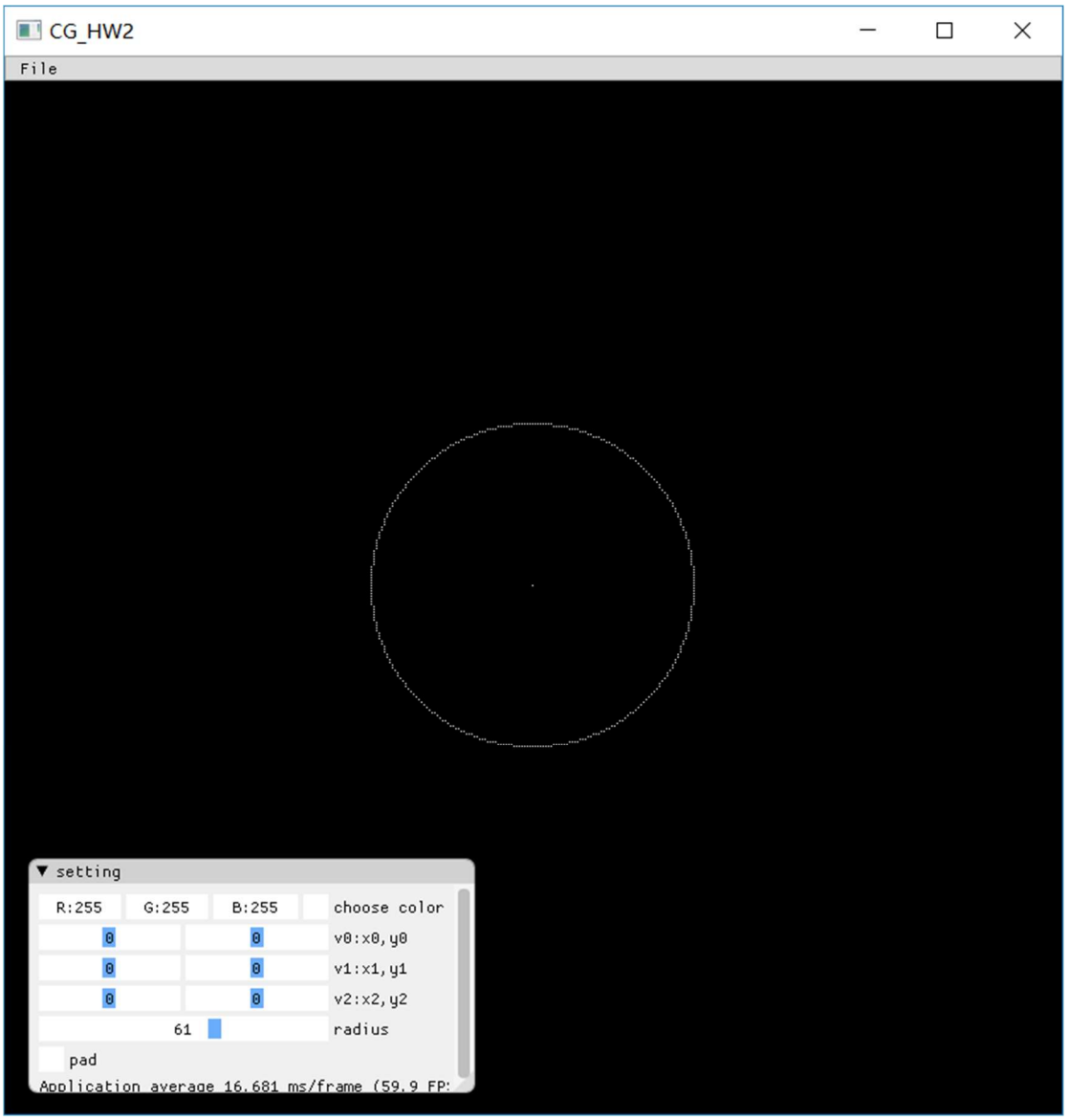
BASIC：

为了更好展示算法的效果用 400*400 的区域均匀展示 200*200 的点阵，从而能看出算法的颗粒效果，动态效果见附视频文件 hw3.mp4。

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）。



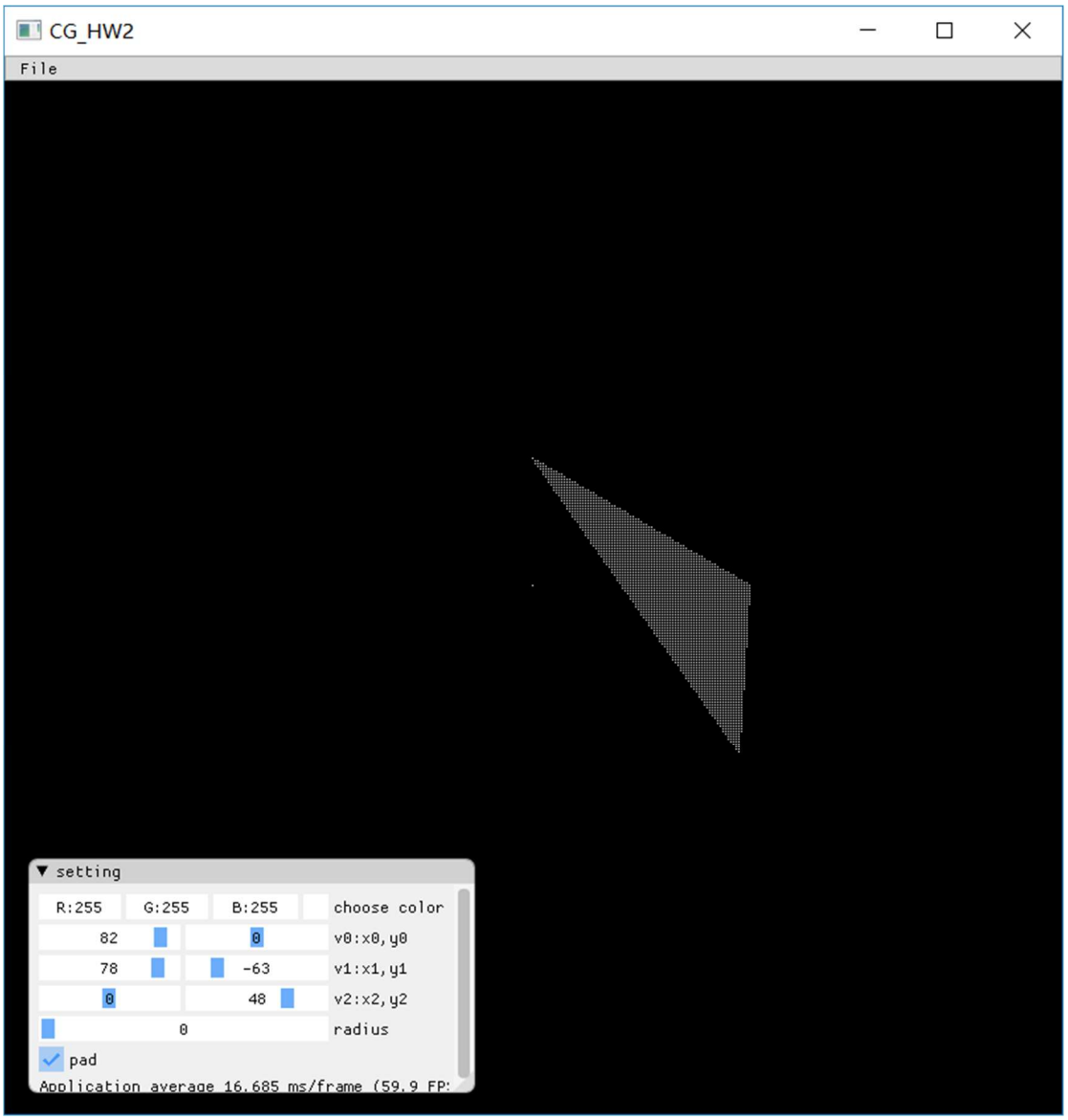
2. 使用 Bresenham 算法(只使用 integer arithmetic)画一个圆：input 为一个 2D 点(圆心)、一个 integer 半径； output 为一个圆。



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)。
   如上图可以通过菜单调节三角形三个顶点的坐标，和圆的半径，pad 勾选时则填充三角形。

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



代码思路解析

附代码文件（本次作业的核心代码在 hw3.cpp 内）

1. 直线算法按照课件的标准实现
   使用下面划线函数在三点之间都画一条直线

```
int bresenham_1ine(float vertices[], int v0[2], int v1[2], int size)
{
    int dx = v1[0] - v0[0], dy = v1[1] - v0[1];
```

2. 画圆算法采用 Bresenham 算法（圆心固定 0，0）

```
int bresenham_circle(float vertices[], int center[2], int radius ,int size)
{
```

3. Imgui 控件的实现部分

```
// -----------
{
    ImGui::Begin("setting");
    ImGui::ColorEdit3("choose color", (float*)&color, 1);

    ImGui::SliderInt2("v0:x0,y0", v[0], -99, 99);
    ImGui::SliderInt2("v1:x1,y1", v[1], -99, 99);
    ImGui::SliderInt2("v2:x2,y2", v[2], -99, 99);

    ImGui::SliderInt("radius", &radius, 0, 99);

    ImGui::Checkbox("pad", &isPad);

    ImGui::Text("Application average %.3f ms/frame (%.1f FPS)", 1000.0f / ImGui::GetIO().Framerate
    ImGui::End();
}
```

4. 填充三角形采用线性方程的方法

主要使用两个函数

根据两点生成直线方程的函数

```
int* lineEquation(int para[3], int v0[2], int v1[2], int vout[2])
{
```

使用直线三条方程设置点亮三角形内所有的点的函数

```
int rasterize_triangle(float vertices[], int v[3][2], int size)
{
    int maxX = std::max(v[0][0], std::max(v[1][0], v[2][0]));
```