PA 0

赵晨阳 2020012363 计06

PRINCIPLE

直线段扫描转换

在解析几何中直线的方程可以表示为 $y = kx + b$，为了将连续的线"离散化"到像素矩阵网格（即计算机屏幕）中，我们采用光栅图形学中采用最广泛的 Bresenham 直线扫描转换算法。对于斜率 $0 \leq k \leq 1$ 的直线而言，我们循环起点到终点的 x 列像素坐标 x_i，依次计算对应 y_i 的坐标。每当 x_i 增加一个像素的时候，y_i 要么保持不变，要么也增加一个像素。是否增 1 取决于误差项 d 的值。误差项 d 的初值 $d_0 = 0$，x_i 每增加 1，d 的值就要增加 k，当 $d \geq 0.5$ 时，y_i 就要增 1，同时误差项 d 要减 1。

圆的扫描转换

一个圆心为 (x_c, y_c)，半径为 r 的圆的隐式表达式为 $(x - x_c)^2 + (y - y_c)^2 = r^2$。由于圆形
具有高度的对称性，我们将其分成 8 份，因此只需要扫描转换 1/8 的圆弧，就能够利用对称性
绘制出整个圆形。请参见图形学课本的 2.2.2 节进行代码实现。

区域填充

在 Windows 的“画图”软件 2 中，有一个工具名为“油漆桶”，当用户给定一个种子点之后，该种子点周围相同颜色的像素都会被染成新的颜色，这种填充方式叫做“漫水填充”（Flood Fill）。这种技术实际上是通过宽度优先遍历实现的，通过一个遍历队列，就能够选取到所有符合条件的待染色点。请参见图形学课本的 2.3.2 节进行代码实现，我们推荐使用非递归版本的实现，因为实际操作的图像可能会很大。

代码框架

官方给了个很简单的 shell 脚本：

```
#!/usr/bin/env bash

# If project not ready, generate cmake file.
if [[ ! -d build ]]; then
    echo "good"
else
    rm -rf build
fi
mkdir -p build
cd build
cmake ..
make -j
cd ..

# Run all testcases.
# You can comment some lines to disable the run of specific examples.
mkdir -p output
bin/PA0 testcases/canvas01_basic.txt output/canvas01.bmp
bin/PA0 testcases/canvas02_emoji.txt output/canvas02.bmp
```

IMPLEMENT

画线

对斜率 k 进行讨论

- if k is not exist：直线竖直，直接描绘竖直的像素点
- $k \in [-1, 1]$，参考 Bresenham 算法，从最左侧端点 (x, y) 开始，取 d = 0，设 X_e 为右侧终点：
    1. while $x \leq X_e$
    2. x +=1 ,d += k
    3. 当 $d \geq \frac{1}{2}$ ，y += 1 ，d -= 1
    4. 当 $d < -\frac{1}{2}$，y -= 1， d += 1
- $k \in (-\infty, -1) \cup (1, +\infty)$，交换 x 轴与 y 轴即可。

画圆

参考课程说明，依据圆的对称心，我们先绘制圆心角 $(0, \frac{\pi}{4})$ 的部分。

- 以圆心为坐标原点，先对 $(x, y) = (0, R)$ 处染色。
- while x <= y
    1. if (x + 1，y - 1 / 2) 在圆内，则 x += 1，y 不变
    2. if (x + 1，y - 1 / 2) 在圆外，则 x += 1，y += 1
    3. 对 (x, y) 染色
- 按照如此算法，对关于圆心对称的八个点染色

区域填充

- 记录当前点的颜色，记为 `origin_color`，将其染上新的颜色，并加入队列，采用 BFS
- 当队列不空，循环此队列
    1. 检查上下左右四个方向是否在图片范围内，如果在图片范围内，且颜色与 `origin_color` 相同，则将元素加入队列
    2. 元素出队，且染色
- 队列空了之后，也即没有像素需要染色，结束算法。

遇到的问题

长期写 Python，让我对 C++ 产生了很强的不适应。

首先，我一开始写了函数内定义函数：

```
    void draw(Image &img) override {
    void symmetry_draw(int x, int y, Image& img) {
        img.SetPixel(cx + y, cy + x, color);
        img.SetPixel(cx + y, cy - x, color);
        img.SetPixel(cx - y, cy + x, color);
        img.SetPixel(cx - y, cy - x, color);
        img.SetPixel(cx + x, cy + y, color);
        img.SetPixel(cx + x, cy - y, color);
        img.SetPixel(cx - x, cy + y, color);
        img.SetPixel(cx - x, cy - y, color);
    }
        printf("Draw a circle with center (%d, %d) and radius %d using color (%f, %f, %f)\n", cx, cy, radius,
               color.x(), color.y(), color.z());
        int x = 0, y = radius;
        double d = 1.25 - radius;
        symmetry_draw(x, y, img);
        while(x <= y)
        {
            if (d < 0){
```

```
                d += 2 * x + 3;
            } else{
                d += 2 * (x -y) + 5;
                y--;
            }
            x++;
            symmetry_draw(x, y, img);
        }
    }
```

才想起来 C++ 并不支持这种语法。

以及，我还使用了如下的语法：

```
int start = (xA <= xB) ? xA : xB,  y = (xA <= xB) ? yA : yB,
end = (xA <= xB) ? xB : xA;
```

结果刚开始忘了给 `(xA <= xB)` 打括号...

不会写 C++ 了...

讨论和借鉴

习题课讲的蛮不错，借鉴了习题课的算法，没和同学讨论...

未解决的 BUG

目前没有发现 bug，看着还行。

如果有更多的时间，我可能会自己设计一些测例对程序进行进一步测试。

建议

总体上文档清晰，框架友好。

不过认真地说，shell 脚本不太优雅，建议改成 Python 脚本。

如果不在 WSL 里面，`#!/usr/bin/env bash` 这个 `heredoc & shebang` 指令能用吗，有些怀疑，可能用 Python 的话能兼容三个系统吧…

还有一点不理解，我没有给这个脚本更改权限，居然也能直接用，但是自己其他脚本都在 `chmod u+x <whatever.sh>` …