光栅图形学作业

李晨昊 2017011466

2019-3-23

目录

1 整体介绍

这个小项目实现了区域的填充，多边形的绘制；顺便实现了直线的绘制，在此基础上还实现了参数方程的绘制。

编程语言使用的是 C++，依赖为：

1. 需要使用支持 C++17 的编译器 (虽然就几行用到了…)
2. 需要使用支持 openmp 的编译器 (虽然就一行用到了，去掉也影响不大…)

关于 png 处理的库已经自带，无需额外安装。只需 make run 即可运行程序。

绘图基于一个核心类 Pic，它支持以上描述的这些接口，并且在绘制线条时可以根据坐标来确定颜色。支持 png 作为输入和输出格式，使用了lodepng这个库。

对于区域绘制的边缘锯齿效应，利用了高斯卷积核对其进行卷及处理，顺便非常 naive 地实现了一下卷积的并行计算。

由于 C++ 目前想要实现对于模板类型的限制还比较困难，故代码中都没有加以限制，在注释中说明了模板类型应该具有的特性。

2 基本选题

2.1 区域填充

实现区域填充的函数为

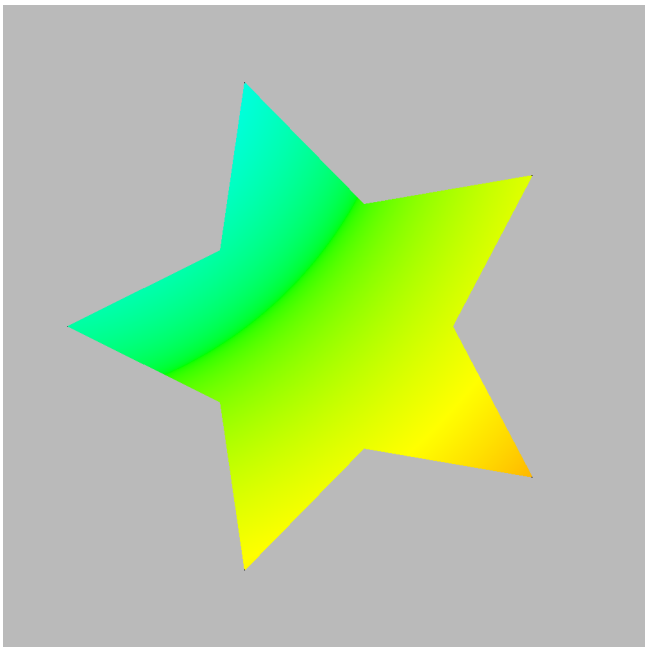
```
Pic &fill(i32 x, i32 y, const Vec3 &col);

template<typename F /* Fn(i32, i32) -> Vec3 */ >
Pic &fill(i32 init_x, i32 init_y, F f);
```

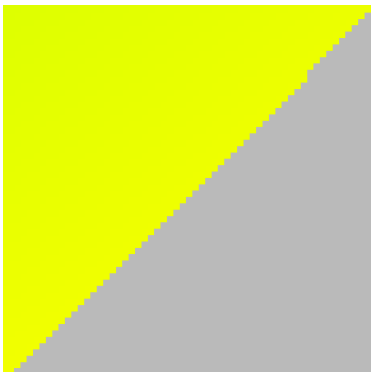
作用是以起始点为源点，填充与它四连通的所有的颜色与它相同的像素点，两个版本分别为用固定的颜色和用随坐标变化的颜色来绘制。它们的一种常见的用法是先用 line 函数绘制出图形的轮廓，再用 fill 函数填充其中的内容。

函数的返回值为 `*this`，这是为了提供 `pic.func1(...).func2(...)...` 的便捷写法。

用它来绘制一个实心的五角星 (见函数 `draw::star`)，效果如图



放大发现边缘处有一些锯齿



2.2 多边形绘制

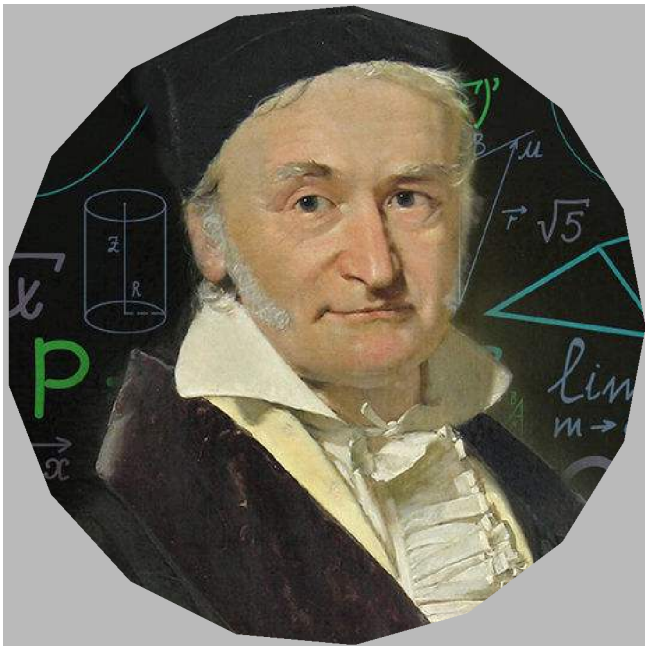
实现多边形绘制的函数为

```
template<typename Pr /* Pair<i32, i32> */, i32 N>
Pic &polygon(const Pr (&ps)[N], const Vec3 &col);

template<typename Pr /* Pair<i32, i32> */, i32 N,
         typename F /* Fn(i32, i32) -> Vec3 */ >
Pic &polygon(const Pr (&ps)[N], F f);
```

其中多边形由数组 `ps` 表示。

它也可以用来绘制上面的五角星的例子，但是没有必要重复。这里用它绘制了一个正 17 边形 (见函数 `draw::polygon`)



2.3 画线

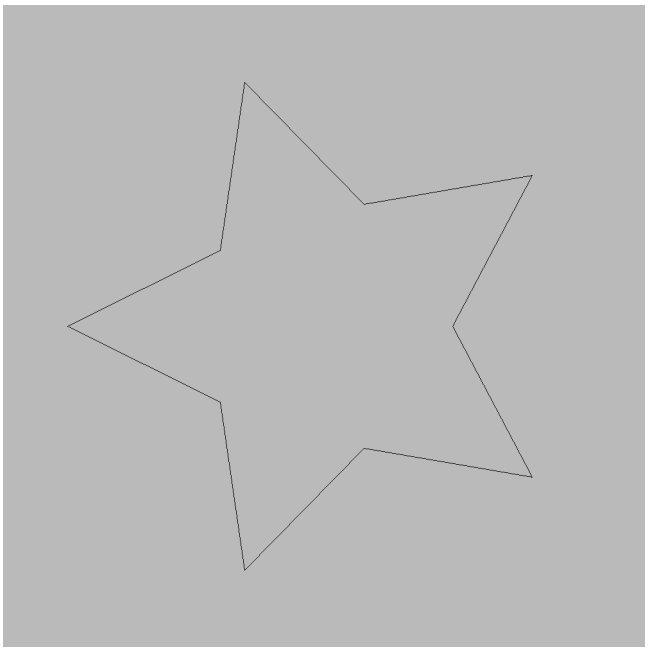
实现画线的函数为

```
Pic &line(i32 x1, i32 y1, i32 x2, i32 y2, const Vec3 &col);

template<typename F /* Fn(i32, i32) -> Vec3 */ >
Pic &line(i32 x1, i32 y1, i32 x2, i32 y2, F f);
```

二者分别为用固定的颜色和用随坐标变化的颜色来绘制线段 `(x1, y1) -> (x2, y2)`，对于坐标的大小没有限制，函数内部会做处理。

用它来绘制一个五角星 (见函数 `draw::star`)，效果如图

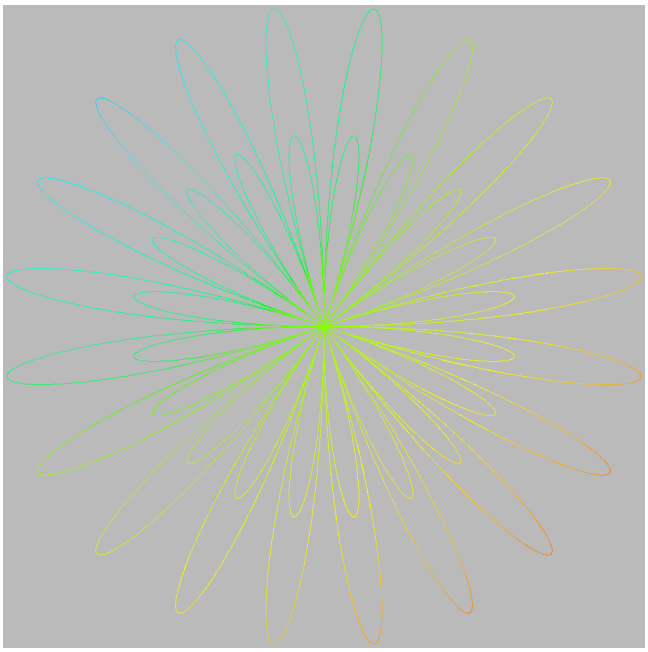


在实现了画线段的基础上，很容易就能用它来绘制一个用参数方程表示的曲线，相关函数为

```
template<typename F /* Fn(f64) -> (i32, i32) */ >
Pic &func(F f, f64 s, f64 e, f64 step, const Vec3 &col) {

template<typename F /* Fn(f64) -> (i32, i32) */,
         typename G /* Fn(i32, i32) -> Vec3 */  >
Pic &func(F f, f64 s, f64 e, f64 step, G g);
```

二者分别为用固定的颜色和用随坐标变化的颜色来绘制参数方程 `f(t)`，其中 `t` 从 `s` 增长到 `e`，每次增长 `step`。用它来绘制 $\rho = r\sin(k\theta)$(见函数 `draw::func`) 效果如下



3 加分项

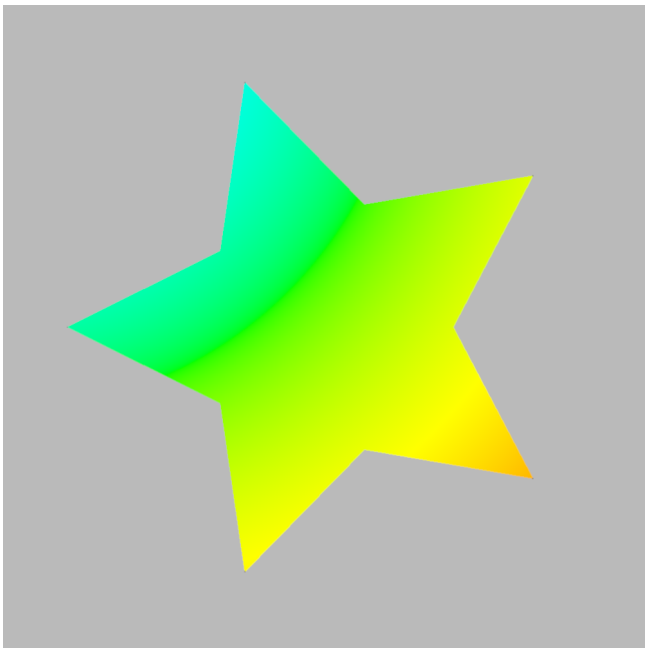
3.1 抗锯齿

使用卷积操作使图像平滑，提供卷积操作的函数为

```
Pic conv(const Kernel &kernel) const;
```

它将返回原图的一个副本而不是修改原图。函数的内部实现了非常 naive 的并行化，可以起到一定的加速效果。

这里使用了 3x3 的高斯卷积核，效果如图



放大，发现锯齿已经有一定的缓解