计算机图形学第三次实验

陈小羽 2016060106020

1 实验名称

画笔软件原型设计与开发

2 实验内容和目的

2.1 实验目的

通过巩固所学的图形学方面的基本知识，了解和掌握图形交互系统设计方法和技能

2.2 实验内容

设计实现一个简单的交互式画笔软件

2.3 基本要求

- 支持矩形绘制、多边形绘制、圆绘制、椭圆绘制
- 支持 Bezier 曲线绘制，鼠标输入控制点
- 支持图形填充
- 程序能正常退出

3 实验原理

通过使用 OpenGL 提供的 glDrawPixels() 函数, 我们可以将一个二进制数组直接写入窗口中的特定位置. 所以, 所有的绘图操作现在转化为了对于数组中元素的读写操作.

问题转化之后, 我们就可以很容易的使用书上讲到的各种画线, 画圆和填充算法来实现一些画图软件基本的功能.

有了绘制图元的算法之后, 我们可以使用这些自己打包好的算法来绘制一个简单的图形界面, 来实现与用户的交互.

对于一些原理的细节 (比如画线算法), 会在实验步骤中提到, 这里就不写了, 免得重复.

4 实验器材 (设备, 元器件)

系统	macOS 10.13.1
处理器	2GHz Intel Core i5
内存	8GB 1867 MHz LPDDR3
图形卡	Intel Iris Graphics 540 1536MB

5 实验步骤和结果分析

5.1 Paper 类型

因为我需要自己实现所有的画线算法, 所以我首先实现了一个比较底层的类型. Paper 类型的主要作用是维护一个数组, 这个数组在这里可以认为是显存 (画布). 然后这个类型提供了一些功能:

- 对于某一个坐标上的像素的读写 (计算偏移量)
- 用某一种颜色清空缓冲区
- 绘制过程中对越界行为的处理
- 整体移动画布的位置

- 更改画布的大小

5.2 Pen 类型

这个类型是为了对 Paper 类型的功能做进一步的封装. 可以用于记录每个物体各自的颜色和绘制时的一些细节 (比如有些算法在绘制的时候要求对颜色的混合).

5.3 菜单

为了方便之后加入各种内容, 我提前就做好了菜单函数和各种和菜单有关的接口和枚举型变量. 菜单使用鼠标右键呼出.

5.4 算法模板类型

我在这个类型中统一打包了绘图程序中用到的所有绘图的算法.

5.4.1 Bresenham 画线算法

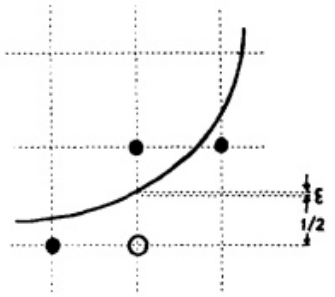
Bresenham 画线算法被封装在 LineAlgorithm 类型中. 这个算法通过将原来 DDA 算法进行了一些推广. 在直线的斜率 $k \in [0, 1]$ 时每次将横坐标 x 加一, 然后通过一个判别式来判断纵坐标 y 是否需要加一 (这个算法的细节部分可以参考课本).

最后, 根据对称性, 我们可以将这个算法推广到所有可能的斜率. 具体的实现可以在 LineAlgorithm.h 中的 LineBresenhamAlgorithm() 中查看. 我的实现和书上的长度差不多, 但是却能支持所有的斜率.

5.4.2 吴小林画线算法

因为 Bresenham 画线算法绘制出来的线条大都带有锯齿, 所以我在 LineAlgorithm 类型中加入了另一个绘制直线的算法. 这个算法是吴小林发明的, 所以一般被称作吴小林画线算法. 这个算法在 1991 年上被吴在 «An efficient antialiasing technique» 上提出

idea



从这个图中可以看到, Bresenham 算法在处理中间那两个点的情况的时候, 根据实现, 只会选择两个点中的一个. 这样, 如果选这了上面的那个点, 就会导致我们绘制出来的图形看起来不太平滑. 为了解决这个问题, 我们可以同时绘制两个点. 然后保证两个点的亮度之和为 1. 比如, 如果我们要绘制 $(x, f(x))$ 这个点. 我们就可以同时绘制 $(x, \lfloor f(x) \rfloor)$ 和 $(x, \lceil f(x) \rceil)$ 这两个点. 并且亮度分别为: $\lceil f(x) \rceil - f(x)$ 和 $f(x) - \lfloor f(x) \rfloor$, 相当于这个点离 $(x, f(x))$ 越近, 这个点就越亮. 最后看上去, $(x, f(x))$ 就好像真的被绘制在屏幕上了一样. 下面是放大之后的效果图.

实现 1 这个算法在实现的时候可以说是十分巧妙了. 实现这个算法需要手动维护一个整数 D. D 实际代表了一个小于 1 的定点小数 f, 而且有如下关系 $D = 2^{31} f$. D 的最高位用于检测 D 是否溢出. 假设斜率为 $0 \le k \le 1$, 则按照 DDA 的流程, 我们可以每次使得 $D = D + 2^{31} k$, 如果 D 溢出的话, 说明这个时候纵坐标 y 应该加一了, 否则 y 不变.

实现 2 更加好的是, 我们还可以同时使用这个 D 来维护上下两个点的亮度, 我们可以取 D 的最高的 8 位作为 $(x, \lceil f(x) \rceil)$ 这个点的亮度参考值 (最大设置为 255), 不妨设为 l. 则另外一个点的亮度参考值就为 $\bar{l}$(相当于按位取反操作). 可以这么做是因为 D 的最高 8 位相当于 $(f(x) - \lfloor f(x) \rfloor) \times 2^8$, 这恰恰就是我们上面讨论出来上面的那个点需要的亮度值.

5.4.3 中点画圆和吴小林画圆算法算法

画圆的算法和画线的算法基本上是相同的套路, 基本上也是每次 $x = x + 1$, 然后这个时候使用一个判别式来判断这个时候 y 是不是应该减一 (第一象限上半部分), 最后使用对称性补全这个图形中的其他的部分.

5.4.4 椭圆绘制算法

椭圆的绘制我没有使用书上的公式, 主要使用的是椭圆的参数方程做的近似, 然后使用画线算法绘制的. 这样子可以做到使用 3 个控制点来控制椭圆, 可以绘制出长轴和短轴不平行与坐标轴的椭圆.

5.4.5 填充算法

填充算法我只实现了洪泛填充. 最开始书上介绍了一种基于深度优先搜索的写法, 但是他没有考虑到这种写法容易爆栈. 所以我稍微做了一下改进, 用队列实现的相同的功能, 而且我没有使用递归, 我的程序应该没有调用函数的开销, 应该会跑得快一些.

5.4.6 Bezier 曲线绘制算法

贝塞尔曲线的绘制也是十分简单的, 只需要将书上的公式抄一遍, 就完成了编写.

6 界面类型

有了绘制图形的各种算法之后, 我制作了绘制图形的各种功能. 并且, 除此之外, 我还制作了简易的交互系统. 有了这个简易的交互系统之后, 用户可以调整颜色, 选择一些图形并进行操作. 下面将单独介绍这些功能.

6.0.1 关键点类型

我写的画板程序使用的是矢量图形, 虽然我没有去实现放缩, 平移和旋转的功能, 但是我在图形的表示方式这个问题上却是参考的矢量图形的标准. 在这个程序中, 所有的图元都是通过绑定在一些关键点上从而进行绘制的.

用户在绘制图形的时候可以使用之前已经存在的关键点, 也可以添加新的关键点. 由于关键点和图元是分开维护的, 他们之间只使用指针传递信息. 所以, 在绘制完成之后, 用户仍然可以使用鼠标拖动关键点来实现对图元的控制.

考虑到关键点有可能会遮挡住绘制的图, 所以当按下 h 键的时候会停止绘制所有的关键点 (隐藏).

考虑到我自己也不知道关键点做多大合适, 所以我在程序中使用加号和减号来控制控制点的大小.

6.0.2 选择功能

因为需要添加擦子和更改颜色的功能, 这些后续功能都需要知道用户想对那一个图元进行操作. 所以这就需要一个用于选择的功能.

选择功能能够在菜单之中找到, 使用之后拖动鼠标就可以在屏幕上看到一个红色的框, 被框中的关键点会被选中 (变为绿色). 为了应对有一些情况, 我对于直接点击某一个关键点的行为做了特判, 这样也是可以进行选择的.

考虑到有可能需要多选, 所以我设置了 a 键作为多选的开关. 按下 a 键之后会开启多选功能.

在选择完成过后, 用户再次点击鼠标是不会取消选择或者选中新的关键点的. 也就是说选择操作是和其他的操作独立的.

后续的功能在选择之后进行. 根据我的设计, 我使用如下的策略来判断到底那一个图元被选中了:

- 如果一个图元的所有的关键点都被选中了, 那么这个图元被选中了.
- 如果有多个图元满足上个条件, 那么去最早创建的图元作为被选中的图元.

6.0.3 擦子

这个功能实现出来是用来作为这个绘图程序的橡皮擦的. 因为这个程序是矢量绘图程序, 所以不能使用常规意义上的那种橡皮擦, 所以, 这个功能的代替就是删除操作. 这个程序可以支持删除被选中的关键点和所有用到这个关键点的图元.

6.0.4 颜色面板

得益于上面介绍的选择功能, 我现在可以实现这个调色板的功能, 这个功能可以改变当前被选中的图元的颜色, 而且还有一个简洁的界面.

选中一个图元之后, 点击菜单中的附着颜色到调色板就可以更改一个图元的颜色.

6.0.5 保存功能

这个功能可以说是我写的最随意的一个功能了, 因为我不太想继续写了, 所以, 为了完成任务, 我从 libjpeg 的官网搞了一份他们的示例代码, 然后稍微魔改了一下, 就得到了我自己额保存功能. 但是这个功能并不能保存矢量信息, 我也懒得去编码并一个一个输出了. 所以就做了一个这样简易的保存 (导出) 功能. 直接利用现成的 libjpeg 库导出到一个 jpeg 格式的图片中. 而且导出的图和原图是反过来的 (没有改).

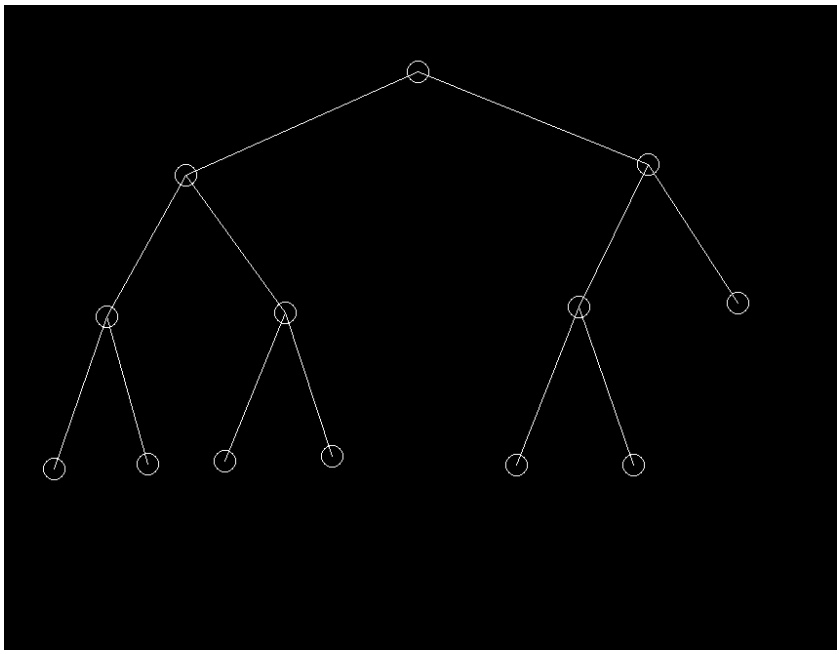
6.0.6 键盘快捷键

我在自己实际使用的时候, 发现有可能会有一些工作会重复很多次, 每次去菜单中选择就会显得很慢, 所以我为菜单中的常用功能做了对应的快捷键. 这个函数被实现在 Automaton.cpp : keyboard() 中. 其中每个按键对应的功能可以在 Menu.cpp : createGLUTMenus() 这个函数中查到. 如果对快捷键不熟悉的话, 也可以在菜单中寻找相应的功能.

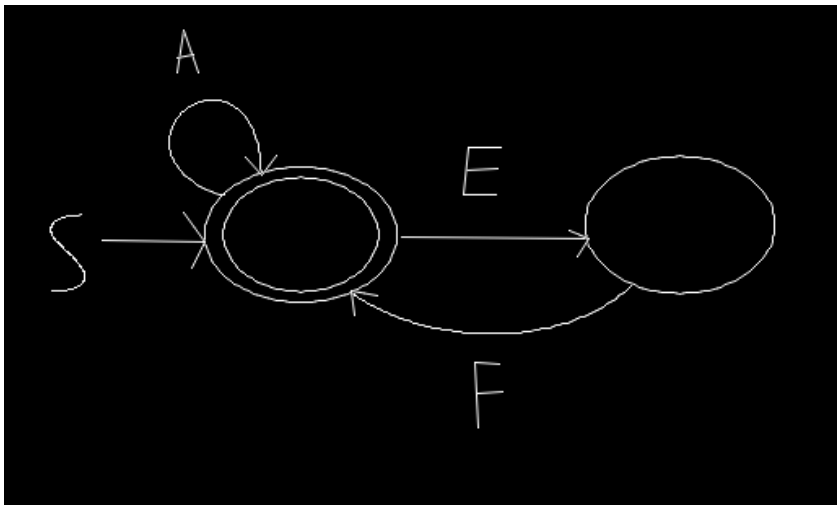
6.1 创造一些作品

为了体现我自主设计的绘图程序的强大之处, 我提供了一些我自己用这个程序制作的绘图作品. 可以看到, 我制作的这个程序不仅可以用于艺术创作, 还可以用于平时一些简单问题的思考.

完全二叉树



确定有限自动机



如图所示的自动机可以识别任意个 A 和任意个 EF 拼接而成的串. 图中的 S 使用 bezier 曲线绘制.

海贼王的 logo

这是我模仿海贼王里面的海贼旗, 用这个程序绘制的图形. 其中分别使用了圆, 椭圆, 线段, bezier 曲线, 洪泛填充等图元. 这个图在绘制的过程中使用了大概 200 多个控制点和 10 多个洪泛填充, 在调试模式下, 我的程序运行已经有一点卡顿, 但是加了 O2 优化之后, 流畅了不少.

7 实验结论, 心得体会和改进建议

这次实验看上去还是比较成功的. 但是, 在实现这些东西的过程中, 我还是发现了很多的问题. 我对图形学的这一套东西还是不太熟悉, 有很多图元在绘制的过程中明明可以使用更好的算法来得到更好的效果, 但是为了让他们能够互相配合, 同时达到比较好的效果, 还是需要做很多取舍的.

比如, 我在写吴小林画线算法的时候, 就和我之前写的那些填充之类的算法起了冲突. 当我使用了吴小林的算法之后再使用填充算法, 在边界处会

出现一些黑孔, 这些东西是吴小林算法在绘制过程中在线段周围留下的 “阴影”, 但是, 之后的填充算法遇到这些阴影的时候, 自动的就结束了, 最后就会在一些曲线的边界处留下一些难看的黑孔. 我知道这个问题是怎么发生的, 但是我却不知道要如何解决.

这样的事情一多, 自然就会留下很多的遗憾.