《计算机图形学》 6月报告

学号，姓名，171240511@smail.nju.edu.cn

2020 年 6 月 26 日

1 综述

在老师所给框架代码上进行修改完善,完成了所有要求的指令输入输出,GUI界面实现了CLI部分的所有功能,包括设置画笔颜色、重置画布、保存画布、两种算法绘制线段、两种算法绘制多边形、绘制椭圆、两种算法绘制曲线、平移、旋转、缩放和两种算法裁剪线段.

额外实现了如下内容:

- 点击画布右、下、右下角边框调整画布大小; 点击画布选择图元; 编辑时显示平移等操作控制点.
- 删除选中图元,复制选中图元,粘贴已复制图元.
- 绘制虚线段; 凸多边形和椭圆填充.
- 更美观的界面.

2 算法介绍

2.1 绘制线段

2.1.1 DDA算法

基本思想是记下起点,然后让长的一边变量不断加一,短的一边则不断加斜率乘1后近似为int值.代码参照[7].

2.1.2 Bresenham算法

基本思想其实和DDA是一样的.只是Bresenham算法通过变形避免了浮点运算.这样一来,既提高了运算的效率,又避免了浮点数不断累加造成的长线段误差.所以本质上,Bresenham是DDA算法的优化形式.代码参照[7].

2.1.3 代码处理

- 特判使得能够处理两端点相等的情况.
- 因为讲义中说不要求像素级一致,所以和伪代码一样只亮一边端点,即线段点的范围$[P_0, P_1)$.

2.2 绘制多边形

默认指令中的点是排好序的, 直接调用线段绘制算法按顺序连点.

2.3 绘制椭圆

使用ppt上的中点椭圆算法, 先画出1/4椭圆, 然后对称. 画1/4椭圆时,根据是否到达切线斜率为-1的位置, 来判断选点的方向是基于x还是基于y, 再根据实际椭圆上的点离哪个点更近取近似.

实际代码要先求出中心和长短轴, 然后以中心为原点求点, 再映射回原来的坐标系.

对称的时候考虑了x轴和y轴上的特例, 不然x=0和y=0的点对称后相当于1个点出现了两次.

代码流程参考书上ppt相关部分.

2.4 绘制曲线

2.4.1 Bezier曲线

书上给出了Bezier曲线关于控制点的参数方程和矩阵形式, 但直接这样计算参数u对应的点效率太低, 于是又给出了de Casteljau递推算法来快速计算一个u值对应的曲线上点的坐标.

有了这个递推算法后, 由于参数u位于[0,1]之间,理论上我们只要将[0,1]分割的足够小,就可以得到精度足够高的一条Bezier曲线.

但这样分割, 为了确保精度, 我们往往要将步长设置的非常小, 从而导致计算次数过多. 于是书上给出了一种二分逼近的办法.也就是每次先计算u=1/2处的点, 然后用这个点将曲线分为两段, 再对这两段曲线进行u值的二分, 重复这个二分的操作, 最后每段曲线的控制多边形会很接近理论曲线, 就可以直接拿控制多边形当作实际曲线.

实现参考[9]中6.3.3和6.3.4.

2.4.2 B-spline曲线

B样条曲线的绘制本质上和Bezier没什么不同, 都是要根据递推式求参数u对应的离散点. 但具体实现起来还是有所不同的.

B样条曲线实际上是按照次数进行分段绘制的, k次B样条就是每k+1个控制点一段, 相邻两段有k个公共控制点, 这也解释了为什么它具有局部性.

不过, 由于它的控制多边形不像Bezier曲线那样可以随着二分不断逼近, 在具体绘制一段B样条曲线时, 只能设置步长密集取点, 这既导致了步长取密时曲线会变粗(重叠部分), 也导致了效率的降低.

实现参考[8].

2.4.3 代码处理

B-spline曲线绘制其中一段时, 目前选择取100个离散点, 再往大取GUI绘制时就会很卡.

2.5 平移

平移部分参考ppt, 直接令各图形的控制点偏移[dx,dy]即可.

2.6 旋转

旋转部分参考ppt, 直接套用公式, 各图形的控制点相对于旋转中心[x,y]顺时针旋转r度即可.

2.7 缩放

缩放部分参考ppt, 直接套用公式, 控制点相对于缩放中心[x,y]缩放s倍即可.

2.8 裁剪

2.8.1 Cohen-Sutherland算

这个算法的特别之处在于位运算的思想,用4位二进制数表示左右上下四个位置的是否. 接着再分类讨论不同情况下线段裁剪时, 就可以用位运算来判断怎么裁剪线段. 它会重复裁剪直到线段完全在窗口内或者完全在窗口外. 代码参考[6].

2.8.2 Liang-Barskey算法法

这个算法不同于Cohen-Sutherland算法, 它希望一步直接将线段裁剪出来, 而不是重复裁剪. 它用$P = P_1 + u * (P2 - P1)$来表示直线P, 并分情况判断和裁剪窗口相交点处的u_1, u_2值, 根据u_1, u_2的情况来判断结果. 代码参考[2]和ppt.

3 系统介绍

3.1 CLI框架

沿用了老师的框架, 每次进行绘制图元或编辑图元操作时, 将需要的绘制信息储存在字典item_dict中, 等到save_Canvas时再根据每个item的item_type, 调用相应算法计算图元的像素点, 并修改canvas中的值.

3.2 GUI框架

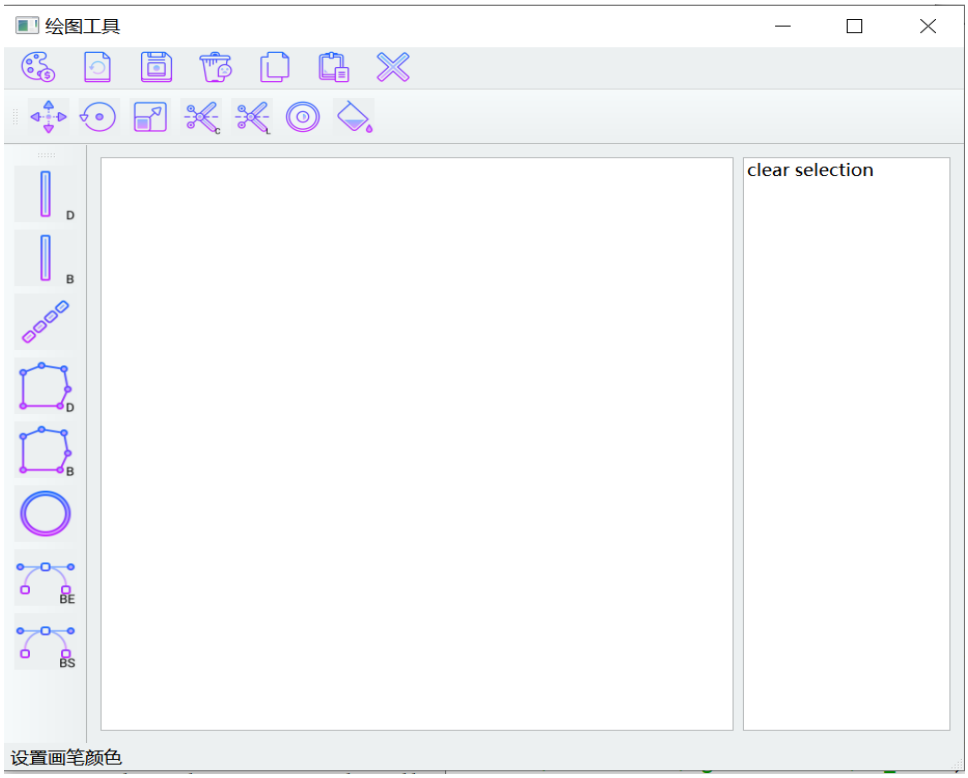
gui部分沿用了老师MainWindow(继承QMainWindow)作为主窗口, MyCanvas(继承QGraphicsView)作为画布, MyItem(继承QGraphicsItem)储存图元信息的框架.

原始代码由于点击绘制图元和在画布上press mouse时都调用get_id方法, 使item_cnt加1,因此图元编号并不连续. 对此进行了修改, 令get_id方法不修改item_cnt, 另外声明方法id_inc来修改item_cnt.

窗口布局情况如下图:

- 最上面是菜单栏,由self.menuBar()方法返回, 可以进行设置画笔颜色等操作.

- 下面一行是编辑工具栏,由self.addToolBar方法添加, 可以对图元进行平移等操作.
- 左边是绘图工具栏,由self.addToolBar方法添加, 可以选择要绘制的图元类型及算法.
- 右边窗口显示图元类型及编号, 由self.list_widget()返回, 点击可以用于选择图元.
- 下方状态栏, 由self.statusBar()方法返回, 显示当前在进行什么操作.
- 最中间的则是画布区域, 定义MyCanvas类, 可以用鼠标操作绘图.



3.3 GUI功能

3.3.1 菜单栏

画笔颜色选择参考了[3], 利用QColorDialog唤出调色板, 将选取的颜色值存在全局变量g_penColor中, 每次要绘制图元的时候将g_penColor作为画笔颜色, 关键代码如下:

```
1 global g_penColor
2 color = QColorDialog.getColor()
3 g_penColor = color
```

重置画布操作也参考了[3], 调用两次QInputDialog获取新画布的x,y轴大小, 然后清空画布和场景, 初始化各个变量.

保存画布的代码参考了[1], 利用QFileDialog.getSaveFileName输入要生成文件的目录

和名字, 然后再用QGraphicsScene的render方法将画布的内容储存到QPixmap类中, 最后用QPixmap类的save方法生成文件.关键代码如下:

```
fname = QFileDialog.getSaveFileName(self, 'Save file',\
'/home/output/default','Image files (*.bmp)')
#cancel save
if(fname[0]==''):
    return
# Get QRectF
rect = self.scene.sceneRect()
# Create a pixmap, fill with white color
pixmap = QPixmap(g_width, g_height)
# set background white
pixmap.fill(QColor(255,255,255))
# painter
painter = QPainter(pixmap)
# Render scene to the pixmap
self.scene.render(painter, rect, rect)
painter.end()
# save bmp file
pixmap.save(fname[0])
```

3.3.2 绘图工具栏

绘制线段和绘制椭圆其实是类似的, 都是先在mousePressEvent中记录一个端点, 然后在mouseMoveEvent中记录另一个端点, 然后利用updateScene方法更新画布, 这样在MyItem类中会自动调用paint方法绘制图形.

绘制多边形又和绘制曲线类似, 都是要输入多个点集, 就添加一个全局变量g_draw_finish来判断绘制是否输入结束.具体来说, 多边形绘制时如果最后一个点在起始点5×5范围内, 就闭合, g_draw_finish=1. 否则, 就和曲线绘制一样, 一直等到下一个任意操作(比如平移)时再调用check_finish方法检测并处理未完成图元.

注意, 绘制曲线时, 每次新控制点加入到末端控制点后面而非前面, 关键代码如下:

```
le = len(self.temp_item.p_list)
if le <= 2:
    self.temp_item.p_list[-1] = [x, y]
else:
    self.temp_item.p_list[-2] = [x, y]
```

3.3.3 编辑工具栏

任何编辑操作前都要点击右边窗口选择图元.

为了处理不同的编辑操作, 在MyItem中添加str类型的trans_type成员, 用于判断编辑操作.

平移操作时, 鼠标按下的点为[x0,y0], 移动时为[x1,y1], 计算出dx,dy并paint.(如果完全平移出画布,接下来这个图元的任何编辑操作都不生效了,但复制粘贴等菜单操作还可以)

旋转操作需要鼠标点击两次, 第一次点击确定了旋转中心坐标, 记录在MyItem的center中, 第二次点击和移动则确定了两个新的点存在MyItem的poi和poi1中,这时poi1-center和poi-center的夹角即为旋转角.注意椭圆不能旋转, 关键代码如下:

```
if self.item_type == 'ellipse':
    print("Can't rotate ellipse")
    return
if self.param_cnt == 2:
    # center and poi, poi1 all gotten
    theta = angle([self.center, self.poi], [self.center, self.
        poi1])
    new_p_list = alg.rotate(self.p_list, \
                self.center[0], self.center[1], theta)
```

编辑时有些操作参数较多,需要点击两次,比如旋转,这时如果点击一次就切走做别的,第一次点击选择的点仍然会保留在对应item中.但是每次进行编辑操作前都会调用selectedTransClear()方法,确保选择的item的相应参数恢复到初始状态.

缩放基本的实现思路和旋转一致, 先确定缩放中心, 再根据两线段x方向比值确定缩放倍数.

裁剪操作时,绘制裁剪窗口,并高亮窗口内线段,关键代码如下:

```
# draw the clip window
painter.setPen(QColor(255, 0, 0))
painter.drawRect( self.regionRect([self.center,self.poi]) )
tmp_p_list = alg.clip(self.p_list, self.center[0], self.center
    [1], self.poi[0], self.poi[1], self.trans_algorithm)
if tmp_p_list != []:
    # highlight the line in clip window
    tmp_pixels = draw(tmp_p_list,self.algorithm)
    painter.setPen(QColor(255, 0, 0))
    for p in tmp_pixels:
        thick_draw_point(painter, p)
```

如果裁剪对象不是线段, 则不起作用. 而如果裁剪后线段什么点也没剩下,仍然保留QListWiget中该图元,但是无法对其进行任何操作.

3.3.4 右侧列表栏

这部分专门用于图元选择.最上面的clear selection点击可以清除对图元的选择.

3.4 额外功能

3.4.1 界面美化

首先是通过谷歌搜索得到相应图标,并手动修改,然后通过QAction类初始化的时候设置Icon.

然后是设置了主窗口的颜色和style,部分代码如下:

```
# The proxy style should be based on an existing style,
# like 'Windows', 'Motif', 'Plastique', 'Fusion', ...
myStyle = MyProxyStyle('Fusion')
app.setStyle(myStyle)
palette1 = QPalette()
palette1.setColor(palette1.Background, QColor(236,240,241))
mw.setPalette(palette1)
```

3.4.2 菜单部分

首先是实现了点击画布右、下、右下角边框后拖动调整画布大小.

这个功能并不会对正在进行的绘制有影响. 具体是通过MyCanvas中的is_image_scaling来判断情况的, 关键代码如下:

```
if g_width-5 <= x <= g_width+5 and g_height-5 <= y <= g_height+5:
    self.is_image_scaling = 3
elif g_width-5 <= x <= g_width+5:
    self.is_image_scaling = 1
elif g_height-5 <= y <= g_height+5:
    self.is_image_scaling = 2
```

接着是删除图元操作. 删除图元操作的难点在于要同步MyCanvas和QListwidget的更新,只要一个更新除了问题,将来跑着跑着就会莫名崩溃.具体代码细节比较多,就不贴了.

复制图元和粘贴图元的思想是用一个全局变量来保存复制的图元,粘贴的时候将其加入,更新MyCanvas和QListWidget.这部分的重点在于复制的时候要注意深度拷贝,不然将来

操作被复制图元和复制图元的时候可能会出问题,例如:item.pixels = src.pixels[:]

3.4.3 绘图工具栏

这部分就额外实现了一个虚线段,基本和线段是一致的.

3.4.4 编辑工具栏

先是选择图元操作.它最复杂的部分和删除图元其实类似,要同步更新QListWidget中的Select, QListWidget里面的selectionChanged函数的参数不好获取,查了半天没查明白,最后换成了下面的形式,参考了[4]:

```
for i in range(g_list_widget.count()):
widget_item = g_list_widget.item(i)
strList = widget_item.text().split()
if(strList[-1] == id):
blocker = QSignalBlocker(g_list_widget)
widget_item.setSelected(True)
break
```

然后是填充操作,该操作只能填充凸多边形和椭圆,对其他类型图元会在控制台提示有问题,对凹多边形则不保证结果,只能保证不把程序弄崩溃(就我的尝试,基本填充都会空一点).代码部分的基本思想就是一排排垂直线推过去,找寻要填充的部分,然后一条条填充.

4 总结

4.1 遇到的一些问题

- gui中sys.exit(app.exec_())改为app.exec_(),自己手动退出,不然Spyder运行手动退出时会报错. 此外,还得加上del app,不然重新运行的时候也会有点问题.参考[5]
- 框架代码是鼠标一动就将MyItem中的图元绘制, 这导致绘制直线的时候直线随着拖动不断绘制,这包括了最开始端点重合的情况, 对于DDA会出现除0错误,因此要特判.
- 画笔颜色选择分两步,一个是会用QColorDialog调出调色版,另一个是要将画笔颜色传给MyItem,我用全局变量g_penColor传递.

4.2 6月更新

4.2.1 添加功能

- 添加了点击画布选择图元功能. 最开始试图使用QGraphicsScene的itemAt方法,发现

两个item重叠多的时候很难选,于是改成记录图元的所有点,与鼠标点击的坐标逐一比对.此外,清除对图元的选择点击右边clear selection即可.

- 添加了删除选中图元,复制选中图元和粘贴选中图元的功能.
- 添加了平移缩放等操作时的控制点显示.
- 界面进行修改,按钮用icon代替,微调了界面的颜色.
- 添加了凸多边形和椭圆填充.(填充凹多边形最难的是斜率比较大的时候竖着一个点周围还有其他点)

4.2.2 bug修正及其他更新

- 修正多边形没画完就edit不会自动终止的bug.
- 对于当前不在修改中的图元,记录下它的所有像素点,update的时候就不再用算法生成,这避免了曲线太多时卡顿的情况.
- 优化了下GUI中鼠标点击事件的分支代码,使其更为精简.
- 修正了重置画布时输入奇怪字符,导致之后text转换成int报错的问题,具体办法是不用int()函数,而是自己写atoi函数.
- 修正了连续绘制多个多边形时,由于第一个完成后g_draw_finish等于1,最后一个多边形如果没完成就做别的事,不会自动补上最后一边的bug.
- 修正了只剩一个图元时无法选中两次的bug.(QListWidget是根据currentTextChanged来切换item的,这就导致如果就一个item,选中一次后,这时如果进行别的操作时clearSelection,再点这个item就选不中了,因为它认为你的text没有变化)
- 修正了鼠标点击越边界时的神秘bug.限制鼠标点击事件的点在[0,g_width], [0,g_height]之间, 不然经常往外拖就会崩溃.同时,如果图元被操作到完全处于边界外,比如平移,接下来对它的任何编辑操作都不起作用,也就是说无法再移回来了.

参考文献

[1] How to save images from qgraphicsview? https://stackoverflow.com/questions/35191327/how-to-save-images-from-qgraphicsview?r=SearchResults.

[2] liang-barskey算法. https://blog.csdn.net/sinat_34686158/article/details/78745492.

[3] Pyqt5中文教程. http://code.py40.com/pyqt5/24.html.

[4] Qlistwidget: change selection without triggering selectionchanged. https://stackoverflow.com/questions/31341442/qlistwidget-change-selection-without-triggering-selectionchanged.

[5] simple ipython example raises exception on sys.exit(). https://stackoverflow.com/questions/10888045/simple-ipython-example-raises-exception-on-sys-exit.

[6] 运用python实现cohen-sutherland直线段裁剪算法. https://www.cnblogs.com/aliali/p/12623642.html.

[7] David F.Rogers, 石教英, and 彭群生. 计算机图形学的算法基础. 机械工业出版社, 2002.

[8] HachiLin. B样条曲线——de boor递推算法实现. https://blog.csdn.net/Hachi_Lin/article/details/89812126.

[9] 孙正兴. 计算机图形学教程. 机械工业出版社, 2006.