《计算机图形学》系统报告

191220164 张廷泽

2021 年 12 月 31 日

1 综述

本课程作业是横跨一学期的项目，要求跟随课程进度在项目中实现各种图形学算法，使用 python 语言，最终完成一个完整的图形学系统。该图形学系统包括核心算法 (algorithms) 程序，包括各种图元的生成、编辑算法；命令行界面（CLI）程序，用来读取包含了图元绘制指令序列的文本文件，并依据指令调用算法模块中的算法绘制图形以及保存图像；用户交互界面（GUI）程序，以鼠标交互的方式，通过鼠标事件获取所需参数并调用核心算法模块中的算法将图元绘制到屏幕上，或对图元进行编辑。

2 算法介绍

1. 直线算法：

   (1) DDA 算法简述：若直线斜率绝对值大于 1，则 y 每增加 1，x 增加 1/k；　若直线斜率绝对值小于 1，则 x 每增加 1，y 增加 k，对最终的结果取整值

   部分代码如下：

```
step=abs(x1-x0)
if x0>x1:
    x0, y0, x1, y1 = x1, y1, x0, y0
fx=float(x0)
fy=float(y0)
for i in range(0,step+1):
    result.append((int(fx),round(fy)))
    fx=fx+1
    fy=fy+k
```

   (2) Bresenham 算法简述：

   原理：选择候选像素中距离直线近的点作为离散点，候选像素包括右边的点和右上的点，循环计算这两个点距离的差，根据其符号判定应选择的点

   实现：（以斜率大于 0 小于 1 为例）利用递增整数运算得到后继决策参数值，决策参数公式：$p_{k+1} = p_k + 2\Delta y - 2\Delta x$，每前进一步需重新计算，根据不同情况更新 p_k

部分代码如下：

```
step=abs(x1-x0)
if ((k>0) and (x0>x1)) or ((k<0) and (x0<x1)):
    x0, y0, x1, y1 = x1, y1, x0, y0
cal_p=2*abs(y1-y0)-abs(x1-x0)
fx,fy=x0,y0
result.append((fx,fy))
for i in range(0,step):
    if k>0 or ((k==0) and x0<x1):
        fx=fx+1
    else:
        fx=fx-1
    if cal_p<=0:
        cal_p=cal_p+2*abs(y1-y0)
    else:
        fy=fy+1
        cal_p=cal_p+2*abs(y1-y0)-2*abs(x1-x0)
        result.append((fx,fy))
```

(3) 总结：直线绘画通过 DDA 和 Bresenham 实现，注意事项在于特殊直线（如垂直 x 轴）、参数顺序大小等问题
该函数参数为两个点的坐标，返回一个点集 list

2. 多边形算法：每相邻两个点对应一条直线，只需多次调用直线绘画函数并将结果集合即可

```
result = []
for i in range(len(p_list)):
    line = draw_line([p_list[i - 1], p_list[i]], algorithm)
    result += line
return result
```

该函数参数为多边形各点的坐标，返回一个点集 list

3. 椭圆算法：Bresenham 画圆法
(1) 原理：与直线类似，计算椭圆轨迹中左右两点与椭圆的距离，取较近的一个像素点以第一象限来说，需按切线斜率（绝对值）是否大于 1 分成两部分计算，对其余象限上的点可以取对称点
(2) 实现：
从函数参数中获取椭圆中心点、长短轴大小，并假定椭圆中心位于原点

计算决策参数初值 p，公式为 $p1_{k+1} = p1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$
计算椭圆处在第一象限的部分，从上顶点开始，到切线斜率大于 1，循环测试选择点并更新 p 的数值
部分代码如下：

```
while ry*ry*fx<rx*rx*fy:
 fx=fx+1
 if p1<0:
     p1=p1+2*ry*ry*fx+ry*ry
 else:
     fy=fy-1
     p1=p1+2*ry*ry*fx+ry*ry-2*rx*rx*fy
```

再计算另一部分，从右顶点开始，到切线斜率小于 1，将代码对应 x、y 互换即可
最后将得到的点求其在其他象限的对称点，最后对所有点进行相应平移操作

4. 曲线生成算法：

(1) Bezier 算法：

Bézier 曲线段可拟合任何数目的控制点，通过逼近这些控制点控制多边形勾画曲线形状这些控制点的相关位置决定了 Bézier 多项式的次数
一条 n 次 Bézier 曲线被表示成它的 n+1 个控制顶点的加权和，权是 Bernstein 基函数
Bernstein 基函数由 de Casteljau 算法生成，公式如下
$P_i^r = (1-u)P_i^{i-1} + uP_{i+1}^{r-1}$
需要计算曲线上的点 $P(u) = P_0^n$
部分代码如下：

```
if algorithm=='Bezier':
    point_num=10000
    for i in range(0,point_num):
        u=float(i/point_num)
        res_x=int(de_Casteljau(x,n,0,u)+0.5)
        res_y=int(de_Casteljau(y,n,0,u)+0.5)
        result.append([res_x,res_y])

def de_Casteljau(p, n, i, u):
    if n == 1:
        return (1-u)*p[i] + u*p[i+1]
    else:
        return (1-u)*de_Casteljau(p, n-1, i, u)
        + u*de_Casteljau(p, n-1, i+1, u)
```

(2) B 样条曲线（三次四阶）:

给定 n+1 个控制点，n+k+2 个参数节点向量，k 次 B 样条曲线有如下形式:

$P(u)=\sum_{i=0}^{n}P_iB_{i,k+1}(u)$

其中 $B_{i,k+1}(u)$ 为 k 次 B 样条基函数，递推公式如下:

$B_{i,k+1}(u)=\dfrac{u-u_i}{u_{i+k}-u_i}B_{i,k}(u)+\dfrac{u_{i+k+1}-u}{u_{i+k+1}-u_{i+1}}B_{i+1,k}(u)$ （规定 $0/0=0$）

$B_{i,1}(u)=1$,$u\in[u_i,u_{i+1}]$, 否则 $B_{i,1}(u)=0$

可以看到基函数只在 (u_i,u_{i+k+1}) 内取值，其余情况都为 0，这样曲线形状只与某个点周围的几个点有关

绘制过程代码如下:

```
elif algorithm=='B-spline':
    k=3
    node=[i/(n+k+1) for i in range(0,n+k+2)]
    point_num=1000*(n+k+1)
    for t in range(int(node[k-1]*point_num),int(node[n+1]*point_num)):
        u=float(t/point_num)
        x,y=0,0
        for i in range(0,n+1):
            B_ik = deBoor_Cox(u, k, i,node)
            x += B_ik * p_list[i][0]
            y += B_ik * p_list[i][1]
        result.append([int(x+0.5),int(y+0.5)])
```

递归计算 B 样条基函数过程代码如下:

```
def deBoor_Cox(u, k, i,node):
if k==1:
    if (u>=node[i]) and (u<=node[i+1]) :
        return 1
    else:
        return 0
else:
    l1=node[i+k-1]-node[i]
    if l1==0:
        l1=1
    l2=node[i+k]-node[i+1]
    if l2==0:
        l2=1
    u1=(u-node[i])/l1
    u2=(node[i+k]-u)/l2
```

```
return u1*deBoor_Cox(u,k-1,i,node)+u2*deBoor_Cox(u,k-1,i+1,node)
```

5. 平移变换：图形上每个点移动相同的坐标

函数接收点集 plist，对其中每个点都加上 dx、dy 变量，返回经过平移处理后的点集

部分代码如下：

```
result=[]
for point in p_list:
    result.append(point[0]+dx,point[1]+dy)
```

6. 旋转变换：物体沿平面内圆弧路径重定位

函数接收点集 plist 及旋转中心、旋转角度

先将问题转化为基准点为原点的旋转，再代入公式计算，最后还原并得出结果

$x_1 = x_r + (x - x_r)cos\theta - (y - y_r)sin\theta$,

$y_1 = y_r + (x - x_r)sin\theta - (y - y_r)cos\theta$

部分代码如下：

```
result=[]
r_sin=math.sin(r*(math.pi)/180)
r_cos=math.cos(r*(math.pi)/180)
for point in p_list:
    fx=x+(point[0]-x)*r_cos-(point[1]-y)*r_sin
    fy=y+(point[0]-x)*r_sin+(point[1]-y)*r_cos
    result.append(round(fx),round(fy))
```

7. 缩放变换：改变物体尺寸　　函数接收点集 plist 及缩放中心、缩放倍数

先将问题转化为基准点为原点的缩放，再将顶点坐标乘缩放系数，最后还原并得出结果

$x_1 = xs_x + x_f(1 - s_x)$,$y_1 = ys_y + y_f(1 - s_y)$

部分代码如下：

```
result=[]
for point in p_list:
    fx=point[0]*s+x(1-s)
    fy=point[1]*s+y(1-s)
    result.append(round(fx),round(fy))
```

8. 线段裁剪:

（1）Cohen-Sutherland 算法：

通过编码测试减少要计算交点的次数，将线段端点按区域赋以四位二进制码
根据区域码进行判定，是否完全在窗口内、在窗口外、与窗口相交的边
计算交点，确定裁剪的部分，剩下部分重复上述过程
部分代码如下：

```
k=float((y1-y0)/(x1-x0))
Q = [x0-x1, x1-x0, y0-y1, y1-y0]
D0 = [x0-x_min, x_max-x0, y0-y_min, y_max-y0]
D1 = [x1-x_min, x_max-x1, y1-y_min, y_max-y1]
if algorithm == 'Cohen-Sutherland':
    s1, s2=0, 0
    for i in range(4):
        if D0[i]<0:
            s1+=pow(2,i)
        if D1[i]<0:
            s2+=pow(2,i)

    if s1==0 and s2==0 :
        result=[[x0,y0],[x1,y1]]
    elif s1 & s2 !=0:
        result=[[0,0],[0,0]]
    else:
        if (s1 & 1) !=0:
            y0 = int(y0 + ((x_min-x0) * k)+0.5)
            x0 = x_min
        if (s1 & 2) !=0:
            y0 = int(y0 + ((x_max-x0) * k)+0.5)
            x0 = x_max
        if (s1 & 4) !=0:
            x0 = int(x0 + ((y_min-y0) / k)+0.5)
            y0 = y_min
        if (s1 & 8) !=0:
            x0 = int(x0 + ((y_max-y0) / k)+0.5)
            y0 = y_max
        '''
        重复以判断s2
        '''
    result=[[x0,y0],[x1,y1]]
```

(2) 梁友栋-Barsky 裁剪算法

二维裁剪问题化简为一维裁剪问题，待裁剪线段及裁剪矩形窗口均看作一维点集

线段参数表达式：$P = P_1 + u(P_2 - P_1)$，目的即找到待裁剪线段与窗口（及延长线）交点

令 $Q_l = -\Delta x, Q_r = \Delta x, Q_b = -\Delta y, Q_t = -\Delta y$

$D_l = x_0 - x_l, D_r = x_r - x_0, D_b = y_0 - y_b, D_t = y_t - y_0$，

则交点参数 $t_i = \dfrac{D_i}{Q_i}$，还可据此分辨出边入边部分代码如下：

```
elif algorithm == 'Liang-Barsky':
u1, u2 = 0, 1
for i in range(4):
    if Q[i] < 0:
        u1 = max(u1, D0[i]/Q[i])
    elif Q[i] > 0:
        u2 = min(u2, D0[i]/Q[i])
    if u1 > u2:
        result = [[0,0], [0,0]]
        return result

res_x0 = int(x0 + u1*(x1-x0) + 0.5)
res_y0 = int(y0 + u1*(y1-y0) + 0.5)
res_x1 = int(x0 + u2*(x1-x0) + 0.5)
res_y1 = int(y0 + u2*(y1-y0) + 0.5)
result = [[res_x0, res_y0], [res_x1, res_y1]]
```

3 命令行程序

1. 简介：命令行界面程序接受两个外部参数：指令文件的路径和图像保存目录，通过解析指令文件的每行命令完成任务

2. 运行逻辑：逐行解析命令，先观察命令行第一个词（即 line[0]），
   若对应某个绘制算法，则分配参数将其保存至 item_dict 中；
   若对应某个变换算法，则修改 item_dict 中对应元素的值；
   若对应设置（重置画布、颜色），则修改系统变量中相关部分；
   若为保存画布，则查询 item_dict，调用算法模块进行绘画

3. 需要完成的部分：包括解析并分配参数、保存画布时调用算法、文件画笔相关操作
   (1) 解析绘制命令（以线段为例），代码如下：

```
elif line[0] == 'drawLine':
```

```
        item_id = line[1]
        x0 = int(line[2])
        y0 = int(line[3])
        x1 = int(line[4])
        y1 = int(line[5])
        algorithm = line[6]
        item_dict[item_id] = ['line', [[x0, y0], [x1, y1]],
                                    algorithm, np.array(pen_color)]
```

说明：在线段的前提下继续解析命令行，找到线段的端点参数，并以 [类型、参数、算法名、颜色] 的形式保存在字典中，关键字是 id

(2) 解析变换命令（以平移为例），代码如下：

```
    elif line[0] == 'translate':
        _,tran_list,_,_=item_dict[line[1]]
        tran_result=alg.translate(tran_list,int(line[2]),int(line[3]))
        item_dict[line[1]][1]=tran_result
```

说明：在平移的前提下继续解析命令行，读取字典中对应元素，通过执行平移算法计算出新的端点参数并写入字典

(3) 解析保存画布，代码如下：

```
    elif line[0] == 'saveCanvas':
        save_name = line[1]
        canvas = np.zeros([height, width, 3], np.uint8)
        canvas.fill(255)
        for item_type, p_list, algorithm, color in item_dict.values():
            if item_type == 'line':
                pixels = alg.draw_line(p_list, algorithm)
                for x, y in pixels:
                    canvas[height - 1 - y, x] = color
            elif item_type == 'polygon':
                ...
            elif item_type == 'ellipse':
                ...
            elif item_type == 'curve':
                ...
        Image.fromarray(canvas).
        save(os.path.join(output_dir, save_name + '.bmp'), 'bmp')
```

说明：遍历 item_dict，调用算法模块返回应该画的点，再形成图像并保存

4 用户交互界面

1. GUI 程序简介

   (1) MyItem 类：图元类，每一个图形即是一个 MyItem，主要包括图元信息（类型、参数等）、绘画函数、边界函数

   (2) Mycanvas 类：画布类，功能包括状态控制、绘制辅助、存储图元等，是需要完成的主体部分

   (3) Mainwindow 类：窗口类，功能包括 UI 设计、触发与操控，用户直接面向此类

2. GUI 程序框架与运行逻辑分析

   (1)Mainwindow 中定义了一个画布类实例 (canvas_widget) 及 pyqt 库中列表窗口 (list_widget)，前者用来保存图元，后者用来选择图形，用户在 mainwindow 上方点击的操作选项将通过以 action 结尾的函数来修改其中的 canvas 状态 (status)，在 mainwindow 右方选择对应图元的操作将修改其中的 canvas 的选择 id(selected_id)

   (2)Mycanvas 中定义了 item_dict、status、selected_id 三个重要变量，分别用来保存图元、状态控制和编辑图元。

   另外还有三个重要函数，分别为鼠标的点击事件、移动事件和释放事件。对于图元绘画，进行点击操作代表一定会形成一个新图元，因此点击操作对应生成一个暂时图元 (temp_item)，并将其加入 scene；进行移动操作代表图元控制点参数变化，对 temp_item 相应部分进行修改；进行释放操作代表图元已确定不可修改，将 temp 图元加入 item_dict 中；

   对于图元编辑，前提是有选中的图元，点击、移动、释放代表编辑参数的变化，在移动时调用编辑算法修改对应图元的控制点参数

   (3)MyItem 中需要在 paint 函数中补充调用图形算法 (algorithms 模块)，及对应的边界

3. 运行流程及代码示例

   (1) 图元绘制（以椭圆为例）：

   点击程序选项” 椭圆”, 对应的触发器生效，调用 canvas 中 start_draw_ellipse

```
def ellipse_action(self):
    self.canvas_widget.start_draw_ellipse(self.get_id())
    self.statusBar().showMessage('绘制椭圆')
    self.de_selection()
```

   canvas 中状态、算法等变化，准备绘画直线

```
def start_draw_ellipse(self, item_id):
```

```
    self.status = 'ellipse'
    self.temp_id = item_id
```

鼠标点击对应事件，产生暂时图元并加入 scene

```
if self.status == 'ellipse':
    self.temp_item = MyItem(self.temp_id, self.status, [[x, y], [x, y]], sel
    self.scene().addItem(self.temp_item)
```

鼠标移动，暂时图元参数控制点变化，调用算法模块对应函数进行绘画

```
if self.status == 'ellipse':
    self.temp_item.p_list[1] = [x, y]
```

鼠标释放，item_dict 加入暂时图元，该椭圆确定，结束这次绘画

```
if self.status == 'ellipse':
    self.temp_item = MyItem(self.temp_id, self.status, [[x, y], [x, y]], sel
    self.scene().addItem(self.temp_item)
```

(2) 图元编辑（以平移为例）:
点击程序选项” 编辑”->” 平移”, 对应的触发器生效, 若有选中的图元，则调用 canvas 中 start_translate

```
def translate_action(self):
    self.statusBar().showMessage('平移图像')
    if self.canvas_widget.selected_id !='':
        self.canvas_widget.start_translate()
```

canvas 中状态变化，准备平移图元

```
def start_translate(self):
    self.status = 'translate'
```

鼠标点击对应事件，修改图元编辑指示变量 self.exit

```
if self.status == 'translate' and self.selected_id !='':
    self.edit=[x,y]
```

鼠标移动，通过当前点与 self.exit 间的运算得到平移相关参数，调用算法模块函数修改选择的图元控制点参数，图像发生移动

```
if self.status == 'translate' and self.selected_id !='':
    if [x,y] != self.edit:
        dx,dy=x-self.edit[0],y-self.edit[1]
        self.item_dict[self.selected_id].p_list=alg.translate(self.item_dict
        self.edit=[x,y]
```

鼠标释放，结束平移操作，清空指示变量值

```
if self.status == 'translate' and self.selected_id !='':
    self.edit=None
```

4. 特殊操作处理及函数介绍

(1) 多边形绘制:

与绘画直线不同，多边形绘画需要依次确定各个控制点，且最后一个点要回归初始点

方法: 设置 polygon_point, 记录多边形点的个数; 构造修改 temp_item 时注意数组长度及对应位置

当鼠标移动到非常靠近初始点的时候（而不是完全贴合）自动结束（不需要手动点击结束）

```
if self.status == 'polygon':
    if self.polygon_point>0:
        if abs(x-self.temp_item.p_list[0][0])<=3 and abs(y-self.temp_item.p_
            self.end_draw_polygon()
        else:
            self.temp_item.p_list[-1] = [x, y]
```

(2) 曲线绘制:

同样时由不确定个数的点控制的图元，曲线不需要在结束的时候靠近初始点，而是手动结束，在" 曲线" 下加入" 结束绘制" 来结束绘画

```
def end_draw_curve(self):
    self.item_dict[self.temp_id] = self.temp_item
    self.list_widget.addItem(self.temp_id)
    self.curve_point=0
    self.finish_draw()
```

运算优化:

(3) 旋转:

在进行旋转操作时，需要先用鼠标点击旋转中心，代码中使用 self.core 保存; 之后鼠标移动确定旋转角度，原理为向量夹角

```
def cal_angle(self,p1,p2,core):
        a=sqrt(pow((p1[0]-core[0]),2)+pow((p1[1]-core[1]),2))
        b=sqrt(pow((p2[0]-core[0]),2)+pow((p2[1]-core[1]),2))
        c=sqrt(pow((p1[0]-p2[0]),2)+pow((p1[1]-p2[1]),2))
        if a==0 or b==0 or c==0 :
            return 0
        return math.degrees(math.acos((c*c-a*a-b*b)/(-2*a*b)))
```

(4) 缩进：

与旋转相同，需要先用鼠标点击旋转中心；除此之外，缩进还需要计算缩进倍数，缩进倍数计算基于鼠标移动的距离，距离越大，相对倍数越大；缩进方向基于鼠标移动向量与缩进中心-图元中心夹角

```
def cal_ss(self,p1,p2):
    dis_1=pow((p1[0]-p2[0]),2)+pow((p1[1]-p2[1]),2)
    arr_1=[self.center[0]-self.core[0],self.center[1]-self.core[1]]
    arr_2=[p1[0]-p2[0],p1[1]-p2[1]]
    cos_theta=arr_1[0]*arr_2[0]+arr_1[1]*arr_2[1]

    if cos_theta < 0:
        return 2-math.exp(-dis_1/20)
    else:
        return math.exp(-dis_1/20)
```

(5) 文件选项与取消选择：

重置画布函数需要清空画布窗口、选择列表,同时注意重置 mycanvas 类中 temp_item 等变量

设置画笔可以调用 pyqt 中对应模块简单实现

对于取消选择,我们需要注意到源代码中列表窗口切换时选择的触发器 currentTextChanged 只有在 listwidget 类内部 text 值变化时才会改变，因此我特别加入了取消选择选项，主要作用为重置 text

5 总结

1. 这个学期的图形学课程都伴随着这个重要的大作业，在完成大作业的过程中，既有恍然大悟的快乐，也有阻塞不前的烦恼；无论怎样，这份 1300 多行的代码及 10 多页的报告和说明书，都为这门课的学习和自己的代码能力起着作用。

2. 算法方面主要是将课堂上讲的算法转换为代码，部分简单的算法比较好实现，但有些算法较难落实到代码上，而且难以 debug

3. 命令行和 gui 方面，重点在于读懂代码并自学部分内容，理解画图的整体运作模式

4. 联系方式：手机号 15948464742 QQ：1711556082