《计算机图形学》十月报告

学号：181860077 姓名：佘帅杰，3121416933@qq.com

2020 年 10 月 29 日

1 综述

注：本次报告是增量式的报告，即本次的报告内内容就是这个月做的内容，不包含上个月已经报告过的任何内容

1. 实现 Bezier 曲线绘制算法
2. 实现 B-Spline 曲线绘制算法
3. 在 CLI 做对应的处理，加入曲线绘制算法的对应功能
4. 实现对图元的平移，旋转，缩放以及裁剪的操作，至此已经实现了所有算法模块里的内容
5. 对 CLI 中添加对图元操作的相关支持

2 算法介绍

2.1 曲线的参数化表示

在绘制曲线之前，首先需要了解怎么对曲线进行参数化的表示实际上对于表示一条曲线的各个变量，都可以用某个参数的函数来表示

$$\begin{cases} x = x(u) \\ y = y(u) \\ z = z(u) \end{cases}$$

三个坐标分量就组成曲线上该点的位置矢量，曲线就可表示为参数 u 的矢函数通过这个对曲线进行参数化表示，也是后面绘制曲线的算法的重要基础对应的图元的生成就可以看作是这个联结的变量的变化过程

2.2 Bezier 曲线绘制算法

一阶的 Bezier 曲线就是一条直线，相当于是在两个控制点之间做插值，最后相当于是一个点在两个点之间移动。最后他的轨迹就是一条直线，也就是两个控制点之间的曲线。这

个过程使用公式可以表达为：$B_1(t) = P_0 + (P_1 - P_0)t$。这里的 t 可以和上文的曲线参数化表示关联，t 就是那个负责联结的参数。对于有三个控制点的情况，这个时候是二阶的 Bezier 曲线，可以看作是两层的一阶 Bezier 曲线，基本的理解方式可以看到下面的图

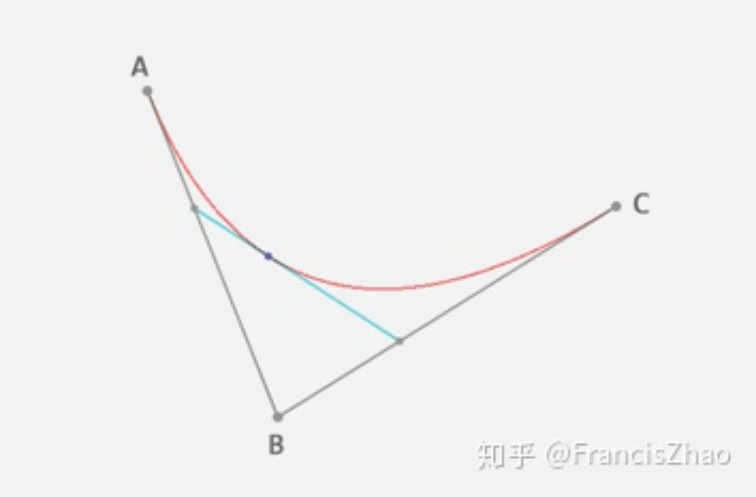

图 1: 二阶 Bezier 算法理解图

上述的图可以看出整个过程还是一个插值的过程，在一次插值的结果的基础上做了第二次插值得到结果同理的更加高阶的 Bezier 都能够一步一步的降阶整个过程还可以发现到系数是二项式的展开，进而得到公式实现算法。

$$f(x) = \begin{cases} P_i \\ (1-u)P_i^{r-1} + (1-u)P_{i+1}^{r-1} \end{cases}$$

对应的图片

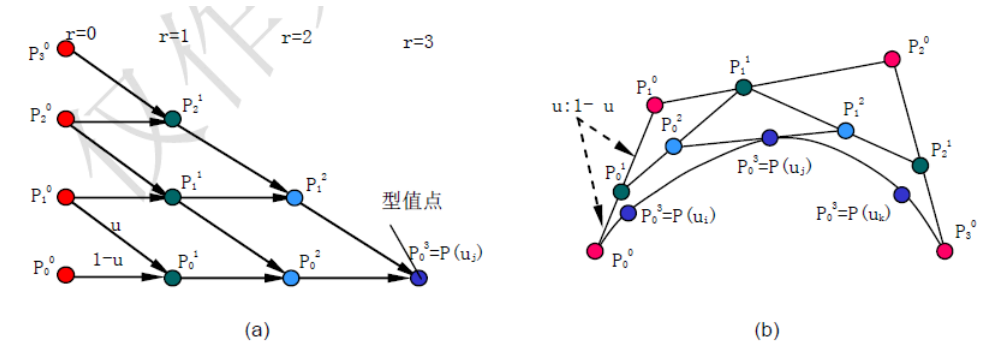

图6-16　三次Bézier曲线的离散生成：(a) 三次Bézier曲线de Casteljau离散生成过程；(b) 三次Bézier曲线作图过程

图 2: Bezier 算法演示图

算法使用过的指令文件：

resetCanvas 600 600

setColor 0 255 0

drawCurve curve1 50 200 100 100 150 200 Bezier

drawCurve curve2 50 400 100 300 150 400 200 300 Bezier

saveCanvas 5

算法结果：

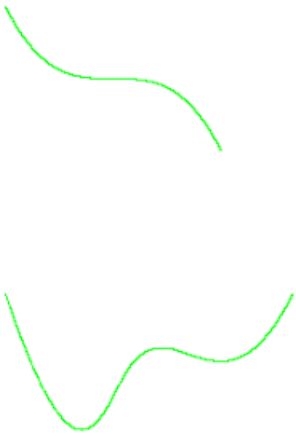

图 3: Bezier 算法测试结果图

2.3 B-spline 曲线绘制算法

B 样条曲线算法涉及到了基函数：什么是基函数？基函数就是一个函数的固定形式，也就是函数只会在这个函数的基础上变化而不会丢掉的函数，在数学中，基函数是函数空间一组特殊的基的元素。对于函数空间中的连续函数都可以表示成一系列基函数的线性组合，就像是在向量空间中每个向量都可以表示成基向量的线性组合一样。

同时由于 Bezier 算法本身的一些问题如：缺少局部性，不够灵活等等，所以 B 样条出现虽然上面说了 B 样条有所不同，但是 B 样条的本质还是一个降阶然后解决问题的过程。

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 1 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

基于上述的公式可以在四个控制点的三次 B 样条进行绘制曲线，但是在实际的情况里控制点可能不止 4 个，对应的方法就是利用一个移动的窗口在控制点数组上滚动，举个例子就是 1，2，3，4，5，得到对应的窗口 1，2，3，4 和 2，3，4，5。这样得到的曲线每一段都能够自然衔接，同时具有连续性。下面的图也可以做辅助说明

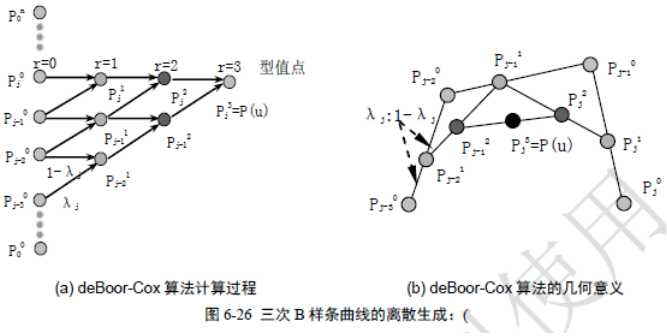

图 4: B-spline 算法示意图

实现之后使用算法进行测试，测试语句如下:

resetCanvas 600 600

setColor 0 0 255

drawPolygon polygon2 250 400 300 300 350 400 400 300 Bresenham

setColor 0 0 0

drawCurve curve3 250 400 300 300 350 400 400 300 B-spline

setColor 0 0 255

drawPolygon polygon3 250 200 300 50 350 250 400 100 450 200 Bresenham

setColor 0 0 0

drawCurve curve4 250 200 300 50 350 250 400 100 450 200 B-spline

saveCanvas 5

测试的结果如下

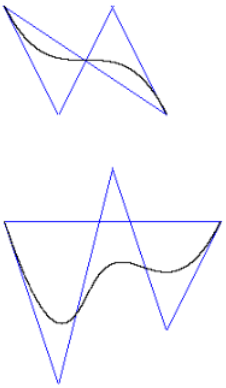

图 5: B-spline 测试结果图

2.4 图元平移算法

平移严格的说是把一个点沿着直线路径从一个坐标位置移动到另一个坐标位置的重定位的过程，说的简单的一些就是，图元的平移就是相对简单一些，图元的构造来源于传入的参数，即“控制点”相对的，如果想要图元的位置进行平移，只需要对图元的控制点进行操作,进行对应的平移,则最后的图形也会做对应的平移算法的测试如下:使用如下的测试指令：

resetCanvas 600 600

setColor 0 255 0

drawLine line1 0 0 500 250 DDA

setColor 255 0 0

drawLine line2 0 0 500 250 Bresenham

translate line2 0 50

saveCanvas 1

测试的结果如下

2.5 图元旋转算法

旋转变化主要是二维的，就是把坐标点关于某个位置转动一个角度，然后确定新的坐标位置的过程，对应也有公式可以进行参考，对应的公式参考如下：

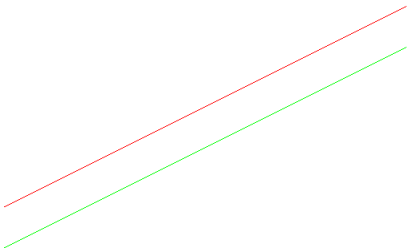

图 6: 平移算法测试示意图

(5) 相对(x_f, y_f)点的旋转变换

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_f & -y_f & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_f & y_f & 1 \end{bmatrix} \tag{3-84}$$

图 7: 旋转公式图

然后对应的对算法进行测试，使用的测试指令文件基本如下

resetCanvas 600 600

setColor 0 255 0

drawLine line1 100 100 400 300 DDA

setColor 0 255 255

drawLine line2 100 100 400 300 DDA

rotate line1 100 100 45

saveCanvas 1

2.6 图元缩放算法

同上所述，图元的构造主要来自于控制点的限制，对应的可以对控制点的位置进行修改，缩放可以得到一个缩放的中心和倍数，对每一个控制点，查看其到缩放中心的位置，然

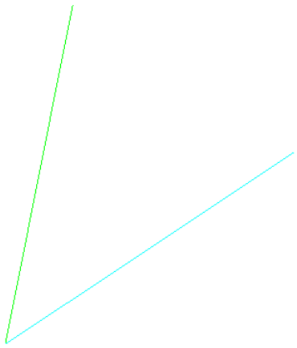

图 8: 旋转算法测试结果图

后把这个位置乘上缩放的倍数就可以很直观的看到，整个图形沿着缩放中心缩小或者是放大的现象，这个算法相对也要简单一些

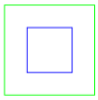

图 9: 缩放测试结果图

2.7 图元裁剪算法

识别图元在指定区域内外部的算法叫做裁剪，用来裁剪对象的区域被称为裁剪窗口。当给定图元和裁剪窗口之后只留下（显示）给定的范围内的部分，而对于区域外的部分则不予显示，图元的裁剪算法很多，但是由于有很多的图元需要执行裁剪操作，对裁剪算法的效率有比较高的要求

2.7.1 Cohen-Sutherland 算法

这个算法的的基本思路就是对整个曲线进行了切分和分类，可以比较显示和可理解的表示出一个线段和裁剪窗口之间的位置关系，对于完全在内部和外部的线段可以快速判断而提高效率，编码的方式可见下图 然后可以直接通过比较或者是作差得到对应的区域码，对得到的两个区域码执行对应的计算来判断直线和裁剪窗口的对应关系，比如说二者的区域码都是 0000 则说明整个线段都在裁剪窗口里。或者是二者相与的结果不是 0000，则说明

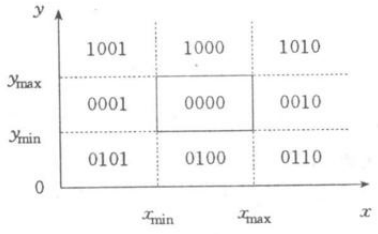

图 1　窗口划分平面示意图

图 10: 裁剪区域编码示意图

整个线段都在区域外。
上述的两种情况是相对简单的两种，都不需要做相对复杂的操作，对于比较复杂度情况就是有一部分的线在区域内，这就需要求解交点等操作，基本的思路就是求交点，然后确定那一部分是要被丢弃的，然后在剩余的部分继续做求解
使用下述的指令文件进行测试:

resetCanvas 600 600

setColor 0 255 0

drawLine line1 0 0 500 250 DDA

setColor 255 0 0

drawLine line2 0 0 500 250 Bresenham

translate line2 0 50

clip line2 50 50 400 200 Cohen-Sutherland

drawPolygon polygon4 50 50 400 50 400 200 50 200 Bresenham

saveCanvas 1

对应的测试结果如下：

2.7.2　梁友栋-Barsky 算法

这个算法的最基本的思路把点和面都看作是大量的点的集合，然后裁剪的结果就应该是二者的交集

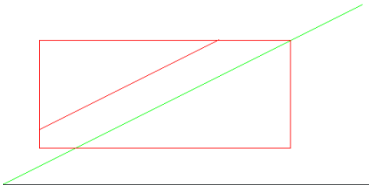

图 11: Cohen-Sutherland 算法测试结果

类似的，这个裁剪算法也是用了一些参数来判断二者的相对的关系，基本的示意图如下对应的使用了好几个变量，包括 δx 表示的是直线左右上的方向然后对应的 δy 上下的方向。

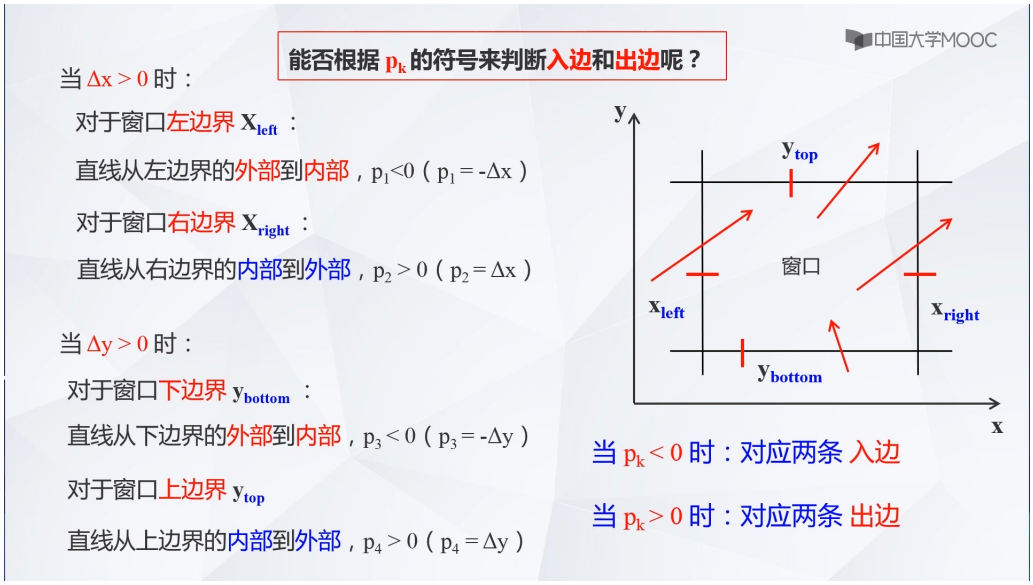

图 12: 梁友栋-Barsky 算法示意图

具体的公式如下

$$\begin{cases} p1 = -\Delta x \\ p2 = \Delta x \\ p3 = -\Delta y \\ p4 = \Delta y \\ q1 = x_1 - x_{left} \\ q2 = x_{right} - x_1 \\ q3 = y_1 - y_{bottom} \\ q4 = y_{top} - y_1 \end{cases}$$

对应的对上述的参数的数值进讨论可以得到线段和裁剪窗口的相对的位置关系

(1) 若 pk=0，直线平行于裁剪边界之一。此时，如果同时满足 qk $<$ 0，则线段完全在边界外；如果同时满足 qk 0，线段平行于裁剪边界，并且在窗口内；这种情况是相对简单一些的情况

(2) 当 pk $<$ 0, 线段从裁剪边界延长线的外部延伸到内部

(3) 当 pk $>$ 0，线段从裁剪边界延长线的内部延伸到外部

对应的算法伪代码

(1) 将线段交点的参数初始化为 u1=0，u2=1；

(2) 定义一个函数，用 p、q 来判断是舍弃线段还是改变交点的参数 r：当 p $<$ 0 时，参数 r 用于更新 u1；当 p $>$ 0 时，参数 r 用于更新 u2：如果更新 u1 或 u2 后使 u1 $>$ u2，则舍弃该线段；否则，更新适当的 u 值仅仅求出了交点，缩短线段。

(3) p、q 的四个值经过测试后，当 p=0 且 q $<$ 0 时，说明该线段平行于边界且位于边界之外，舍弃该线段。假如该线段未被舍弃，则裁剪线段的端点由 u1、u2 值决定。

(4) 反复调用算法进行计算

使用下述的指令文件进行测试：

resetCanvas 600 600

setColor 0 255 0

drawLine line1 0 0 500 250 DDA

setColor 255 0 0

drawLine line2 0 0 500 250 Bresenham

translate line2 0 50

clip line2 50 50 400 200 Liang-Barsky

drawPolygon polygon4 50 50 400 50 400 200 50 200 Bresenham

saveCanvas 1

最后的测试结果

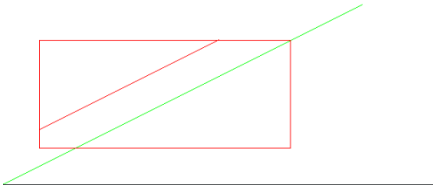

图 13: Liang-Barsky 算法测试结果

3 系统介绍

目前的 CG2020 图形学系统已经支持

- 命令行和 GUI 界面调用直线生成算法
- 命令行调用多边形生成算法和中点圆生成算法
- 在 GUI 界面中使用鼠标直接的选中所需图元
- 命令行调用曲线生成算法，可选算法 Bezier 和 B-spline
- 在命令行调用指令，对图元做平移，旋转，缩放，裁剪操作

4 总结

本次实验基本是在实现和完善算法模块，集中对需求的图形学绘制和图元操作算法进行研究和实现，至此已经实现了所有的算法函数，下一步是更加细致的查错和完善在 GUI 里对算法的调用和支持

参考文献

[1]《计算机图形学教程》孙正兴编

[2] Google 图片

[3] CSDN 博客