文章编号:1671-4598(2018)09-0246-04 DOI:10.16526/j.cnki.11—4762/tp.2018.09.052 中图分类号:TP311.5 文献标识码:A

# 函数曲线生成器的设计

毛开梅,邹 星
(西安铁路职业技术学院 电子信息学院,西安 710014)

**摘要**:主要研究了通用函数方程的曲线绘制过程。根据用户输入的函数表达式,使用 VC++编程语言对输入的字符串进行分析,并绘制出相对应的函数曲线;该研究包括函数方程式的解析和逆波兰式求值、规定区域内函数曲线初始值和初始方向的确定,以及函数曲线的逐点绘制过程;根据 MFC 图像绘制方法,对内存 DC 缓冲技术进行了研究,建立基于 Bitmap 的内存兼容 DC,以高效地完成函数图像的平移和缩放操作方法;该研究已实现多项式函数、常用数学函数以及数学函数的复杂嵌套形式的绘制;当用户输入出现错误时,能够智能地提示错误位置;该研究对数学教学和函数模型研究具有深刻的意义。

**关键词**:方程式解析;逆波兰式求值;函数曲线绘制

# Design of Function Curve Maker

Mao Kaimei, Zou Xing
(Electronic Information Department, Xi'an Railway Vocational Technical Institute, Xi'an 710014, China)

**Abstract**: The method of generic function curve drawing is studied in this paper. According to the expression that user input, it is analyzed by VC++ programming language, and a corresponding function curve should be drawn finally. The research includes the function expression parsing and Reverse Polish Notation evaluation, and the evaluation of the initial value and the initial direction of the curve within a certain area, and the rendering process of a function curve point by point. The memory DC buffering technique is also researched in this paper according to MFC drawing method. Compatible memory DC based on Bitmap is established to efficiently complete the translation and scaling of function images. The research has accomplished the rendering of functions constituted of x in polynomial, and common mathematical functions, and other complex functions assembled by simple functions. When an error occurs in the user inputs, the error place can be indicated intelligently. It is significant to mathematics teaching and mathematical functions model building and researching.

**Keywords**: function expression parsing,; RP Notation evaluation; function curve drawing

## 0 引言

随着图形技术的日益广泛应用,计算机绘图方法的研究也就显得愈来愈重要。目前已提出通用的像素级曲线生成算法。其中较为卓著的有我国刘勇奎教授提出的“曲线的逐点生成算法”[1],它只用整数运算,可以绘制多种曲线,包括 Bezier 曲线、B 样条曲线、多项式函数曲线等[2-3]。该算法本身能自动调整前进的方向,因此无论曲线的走向如何变化,该算法都能随着曲线走向的变化而调整自己,使其总能与曲线的走向保持一致。该课题的目的就是为了研究通用函数曲线的绘制算法,实现函数曲线绘制的软件。本课题依据现有的曲线逐点绘制算法,并结合开发语言的特性,实现像素级的函数曲线生成器,并使用图像缓冲技术,解决可能遇到的闪屏等问题。本课题实现函数曲线生成器将解决多项式函数、常用数学函数等的绘制问题,并允许函数相互嵌套组合,用户自由输入表达式,即可得到函数图像。本课题所实现的软件能够智能地判断所输入的方程式的正确性,如果出现错误,可以指示出错误位置。软件不限制显示区域,并可以同时显示多个函数图像,用颜色区分。软件同时可以对可视区域进行坐标的缩放操作,可以观察函数在某个微小区域内的曲线变化。这对数学教学、研究等用途的函数模型参考具有重要的意义。

**收稿日期**:2018-02-26; **修回日期**:2018-03-22。

**作者简介**:毛开梅(1980-),女,四川达州人,硕士,讲师,主要从事计算机科学与技术方向的研究。

## 1 函数方程式的解析求值

### 1.1 方程式的节点解析

本课题研究的函数表达式默认以 $y=f(x)$ 的形式输入。用户需要输入的部分为 $f(x)$,程序将自动添加为 $f(x)-y=0$ 的形式,即最终解析形式仍为 $f(x,y)=0$。课题的方程式解析器,期望输入方程式的字符串和变量参考表,输出解析结果对象。解析结果将包含解析正确与否的信息,若正确,结果对象中将包含分解后的节点对象列表;若失败,结果对象中也应包含错误原因和错误位置。

### 1.2 节点列表的逆波兰处理

通过上文的处理,输入的表达式已分解成由节点组成的列表,并且,除了括号的匹配问题之外,节点列表的顺序是合法的中序表达式顺序。要使表达式可以方便求值,需要将节点顺序转化为逆波兰式。将一个普通的中序表达式转换为逆波兰表达式的一般算法是:

首先需要分配 1 个栈,作为临时存储运算符的栈 S1(含一个结束符节点),一个队列,作为输入逆波兰式的列

表 S2（空队列），S1 栈可先放入优先级最低的运算符‘＃’号节点，注意，中缀式应以此最低优先级的节点结束，这里扩展出‘＃’号操作符节点作为终结符节点。为了与 S1 栈中的终结节点呼应，需要在中缀式最后加入终结符，以便算法可以正确结束[4]。从中缀式的左端开始取出节点，逐序进行如下步骤：

1）若取出的节点是值型节点，则将该节点直接送入 S2 栈；

2）若取出的节点是非值型节点，则将该节点与 S1 栈栈顶元素比较，如果该节点优先级大于 S1 栈栈顶节点的优先级，则将该运算符进 S1 栈，否则，将 S1 栈的栈顶节点弹出，送入 S2 队列中，直至 S1 栈栈顶节点低于（不包括等于）该节点优先级，则将该节点送入 S1 栈；

3）若取出的非值型节点是右括号或终结符节点，则将 S1 栈栈顶弹出，该节点不进入 S2 队列中，直接销毁；

4）重复上面的 1～3 步，直至处理完所有的中缀式节点列表。

完成以上步骤，队列 S2 便为逆波兰式输出结果。

### 1.3　逆波兰式的求值

上文可以得到正确的逆波兰式顺序的节点列表，而得到逆波兰式的目的便是求值。如图 1 所示。

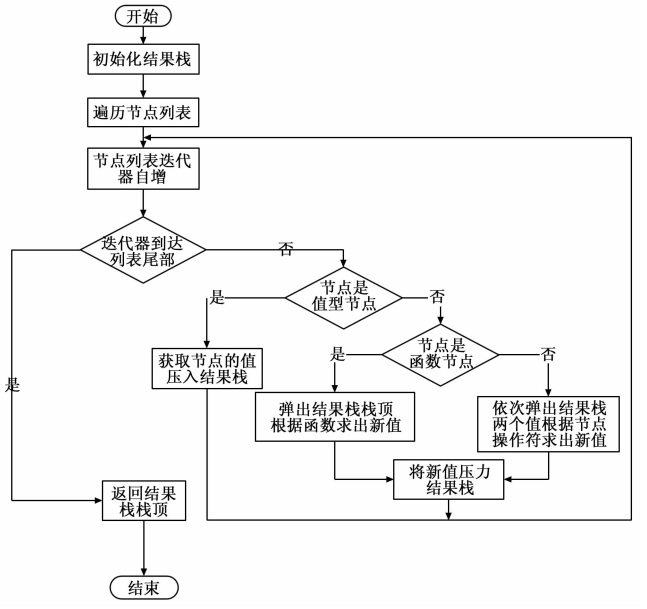


图 1　逆波兰式求值

## 2　绘图区域内曲线初始值的确定

### 2.1　绘图区域、视图区域和坐标区域模型

在研究曲线绘制前，首先应明确程序绘图区域、视图区域和坐标区域的概念模型。为实现平移、缩放和曲线粗细控制，并解决 MFC 闪屏问题，引入绘图区域的概念。定义一个的 Bit Map 对象，其尺寸定义为宽高分别为屏幕大小的 2 倍，使用该 Bit Map 定义一个兼容的 Mem DC 对象，此 Bit Map 可以看成一张画布，即绘图区域，其生成的兼容的 Mem DC 对象即为画布的 DC 对象，它控制着画布的绘制和显示。函数图像都将绘制在此画布上。因此画布需要定义左上角的坐标偏移量和每像素所占的坐标值。将画布放大到像素级，想象成稀疏的网格，以网格的左上角的点代表该像素，则根据左上角像素的坐标偏移量和每像素坐标值，可以计算出画布上任意一点所处的坐标值。同样的，已知某个坐标值，需要定位到画布上的某个像素点，那么，可以做如下假定：若该坐标值的点落在某网格内，则认为其对应的像素点即为该网格所对应的像素点。这样假设的误差在半个像素范围内，因此假设可以接受。

### 2.2　曲线初始像素点的确定

二分法确定曲线的初始像素点：

可以将绘制区域内曲线初始化像素点转化为，确定像素点的 x 值后，寻找距离曲线最近的像素点，即确定像素点的 y 值。图 2 展示了使用二分法寻找距离曲线最近像素点的算法过程。

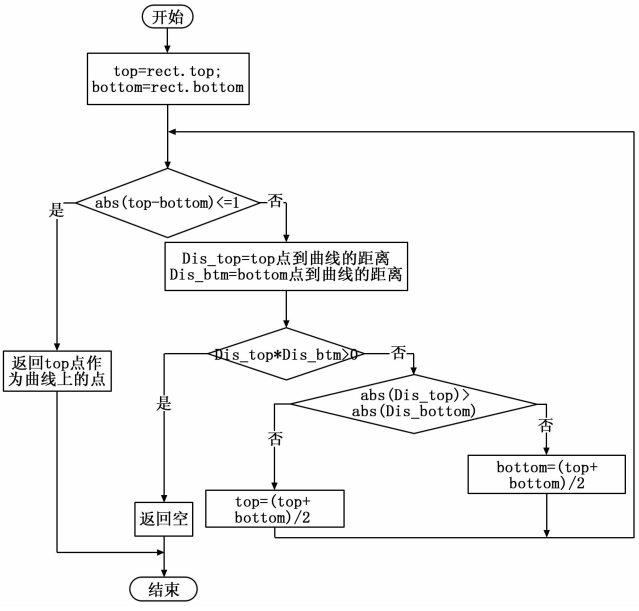


图 2　二分求解曲线的起始像素点流程图

如图 2 的流程。Get Point On Curve 函数接受 3 个参数，第一个参数为当前像素列的 x 值，第二个参数为求得的像素点的 point 对象，第 3 个参数为画布的矩形对象。初始时，top、bottom 分别为画布的顶端和底部，即指示出该列像素的顶部和底部两个像素。将这两个像素所对应的坐标值 $(x,y)$ 带入函数方程式 $f(x,y)$，得到两个值。

### 2.3　曲线初始点的运动方向确定

要确定曲线的运动方向，即需要计算曲线在该点处的导数值。根据导数值就可以判断出曲线在该点处的走向。导数的定义如下：

$$f'(x0)=\frac{\Delta y0}{\Delta x0}=\frac{f(x0+dx)-f(x0)}{dx} \tag{1}$$

式（1）中的 $dx$ 期望取得足够小，而实际 $dx$ 的值取绘

图区域的每像素所占坐标值的 0.5 倍，产生的误差就可以在可接受的范围内。

根据这个导数值，可以确定曲线的走向。以该点位中心，划分出 8 个方向，这 8 个方向分别记为 $m1$ 至 $m8$，则导数值和方向区间的对应关系如图 3。

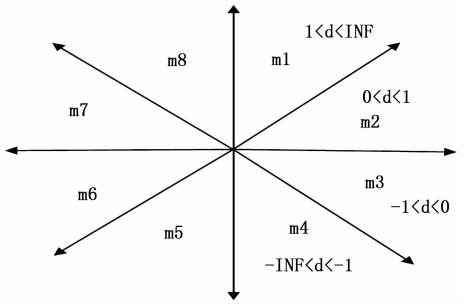


图 3 曲线导数值和方向向量

## 3 曲线的逐点绘制算法

### 3.1 一个通用的像素级曲线生成算法

二维平面上曲线的一般表达式为

$$f(x,y)=0 \tag{2}$$

根据曲线的正负性质提出的算法如下：一个像素级绘制算法就是要逐点地选择距离实际曲线最近的那些象素点。设当前点的坐标为 $(x,y)$，则下一步就要从其相邻象素点中选择一个距离实际曲线最近的象素。8 个前进方向依次设为 $m1$ 至 $m8$，如图 4 所示。为实现曲线绘制，在算法中，根据曲线的不同走向被分为 8 个部分，并进行了处理。例如，根据曲线趋势，假如某一小段曲线的切线方向在 $m1$ 和 $m2$ 之间，即大于 0°而小于 45°，则由算法的 $P1$ 部分进行绘制。假如这小段曲线的切线方向在 $m2$ 和 $m3$ 之间，则由 $P2$ 部分进行绘制，以此类推，如图 4 所示。

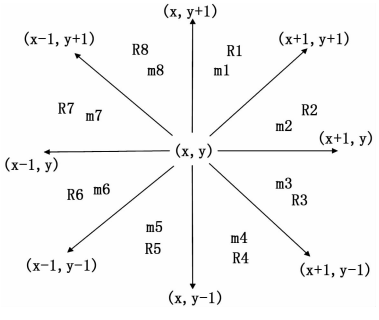


图 4 一个像素的 8 个相邻像素及对应的移动方向

下面以第一部分 $P_1$ 为例讨论算法的实现。首先判断象素 $(x,y+1)$ 和 $(x+1,y+1)$ 中距离曲线较近的像素，即将 $(x,y+1)$ 和 $(x+1,y+1)$ 分别代入式 (2)，比较两者绝对值的大小。如果 $|f(x,y+1)|$ 的绝对值较小，则说明 $(x,y+1)$ 点距离曲线较近，该点就成为下一步的当前点，否则下一步就前进到 $(x+1,y+1)$ 点。

如果当曲线的走向发生变化时，如何判断，以便转到算法中相应的部分。这是该算法的一个关键技术，即算法的自动方向调整。具体方法如下：

当 $f(x,y+1)$ 和 $f(x+1,y+1)$ 皆为正或皆为负，则说明 $(x+1,y+1)$ 和 $(x,y+1)$ 两点在曲线的同一侧。这时曲线的走向已脱离 $R1$ 方向的范围，曲线的走向最大可能为 $R2$ 或者 $R8$。通过判断 $f(x,y+1)$ 和 $f(x+1,y+1)$ 之间的绝对值大小可以确定是 $R2$ 还是 $R8$。若后者小于前者，则是 $R2$，否则是 $R8$。如果曲线的走向既不在 $R2$ 方向域内，也不在 $R8$ 方向域内，那么在转到算法的 $P2$ 或 $P8$ 部分后仍可以继续判断曲线的走向（见下面的算法，其中首先要判断的是曲线走向是否属于其它方向，以便转到算法的相应部分），否则，算法是不前进的。因此，它可以生成任何弯曲度的曲线，包括在某些点处具有很大转向的曲线。

### 3.2 绘图区域内的函数绘制

根据上节的算法，可以从已知点，逐点绘制出函数曲线。该算法的迭代出口是处理点不在绘图区域内。然而，考虑到绘图区域的特殊性，算法退出后，绘图区域的右边可能仍有函数图像。根据 1.2 节所指示，算法退出后，将继续进入下一个初始值寻找的迭代过程，直到横坐标超出绘图区域。这样就保证了绘图区域图像的完全覆盖。

## 4 函数图像的平移和缩放

根据 2.1 小节的定义，为实现平移、缩放和曲线粗细控制，并解决 MFC 闪屏问题，引入绘图区域、视图区域和坐标区域的概念[5]。定义一个的 Bitmap 对象，其尺寸定义为宽高分别为屏幕大小的 2 倍，使用该 Bitmap 定义一个兼容的 MemDC 对象，此 Bitmap 可以看成一张画布，即绘图区域。视图区域一定是从绘图区域截取的某个矩形，视图区域的大小由程序窗体的大小决定。视图区域相对于绘图画布有两个像素级的偏移量（OffsetX，OffsetY）。坐标区域即绘图区域所映射的坐标矩形范围。坐标区域由绘图区域的坐标偏移量和每像素坐标值决定。这里的每像素所占坐标值的定义，是支持视图缩放功能的本质原理。每像素所占的坐标值越大，坐标轴刻度越精确，函数图像即显示某个细节部位，便于观察某个小范围内函数图像的变化过程。每像素所占的坐标值越小，绘图区域所代表的坐标区域就越大，函数图像即显示在更宏观的视图状态下的情形。

区域模型如图 5 所示。

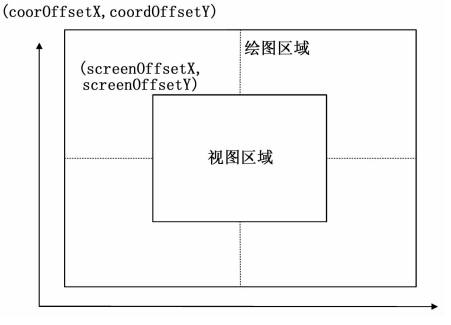


图 5 视图区域、绘图区域、坐标区域模型

### 4.1　函数图像的平移操作

根据 3.1 小节对绘图区域和视图区域的定义，要实现函数图像的平移操作，即实现视图区域的平移。我们已知，视图区域包含在绘图区域的矩形中，根据像素偏移量（OffsetX，OffsetY）进行偏移定位。在窗体需要重绘时，程序从画布上复制视图区域所对应的像素数据，从内存 BitMap 中快速批量地复制视图区域的数据到显存[6]。为了提高效率，避免阻塞 UI 线程，如果用户的操作没有引起绘图区域产生重绘，而只是视图区域的重绘，那么函数曲线的逐点绘制过程就不用进行，而只需要更新偏移量（OffsetX，OffsetY），然后重新刷新内存数据到显存即可，这个效率是非常高的[7]。

平移操作在用户对视图区域按下左键开始，弹起左键结束。当按下左键时，在消息映射的处理函数中，首先记录下拖拽起始点，并置拖拽标识量为真，设置捕获鼠标事件，让鼠标离开窗口后程序依然可以捕获消息[8]。

在鼠标移动的消息处理函数中，判断如果目前正处于拖拽状态，则获得当前鼠标位置，计算出鼠标偏移量，将这个偏移量累加到视图区域相对绘图区域的偏移量上，然后触发视图区域重绘，并更新拖拽起始点。

在鼠标弹起时，释放捕获动作，并设置拖拽标识量为非真。再次触发鼠标移动事件时，则只更新状态栏而不移动视图区域中的函数图像。这样，给用户的感觉是，函数图像随着鼠标的拖拽而移动。

在更新视图区域的偏移量时，应注意，避免让视图区域超出绘图区域。如果视图区域因偏移，到达绘图区域的边缘，则说明绘图区域需要进行重绘。该重绘动作需要使得视图区域偏移回到中心，这需要重新计算绘图区域的左上角坐标偏移量，使得视图区域的坐标不发生变化，即绘图区域的函数图像不发生位移。

### 4.2　函数图像的缩放操作

函数图像的缩放操作与平移类似。其不同点在于，缩放操作必然会触发绘图区域的重绘，因为缩放操作需要更新绘图区域的画布的每像素所占坐标值大小[9-10]。在缩放前后，需要保证视图区域的中心点的坐标值不变，这样产生的缩放动作才符合逻辑。这就需要在缩放前记录下视图区域中心的坐标值，在改变每像素所占坐标值大小后，将视图区域平移到绘图区域中心，同时根据当前视图区域的偏移，改变绘图区域的左上角坐标偏移，使得视图区域中心坐标值不变。这样，在缩放前后，视图区域总是以中心点在进行放大或缩小操作。

## 5 运行及测试

测试用例 1：

用例名称：不连续函数曲线绘制测试。

测试用例输入和预期结果如表 1。

测试结论及说明：

程序能够正确处理非连续函数的绘制，包括对极限的处理和不存在导数值的点的处理。

表 1　测试用例 1 输入预期表

| 表达式 | 线宽 | 颜色 | 预期结果 |
|---|---|---|---|
| y=0.5/x | 4 | rgb(0,0,160) | 反比例曲线，两侧无限逼近 y 轴，但 y 轴上无图像 |
| y=x/abs(x) | 4 | rgb(255,128,0) | 分断函数，y 轴右侧为 y=−1，左侧为 y=1 的曲线 |
| y=−x*(x/abs(x)) | 4 | rgb(255,0,0) | 第二象限是反比例曲线，第 4 象限是 y=−x 的直线 |

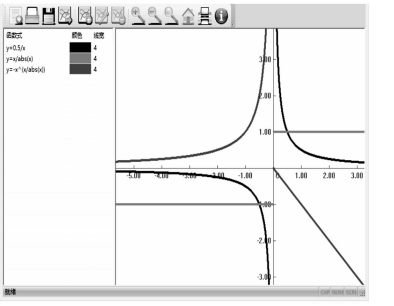

图 6　非连续函数绘制图

测试用例 2：

用例名称：三角函数周期极限测试。

测试用例输入和预期结果如表 2。

预期结果：呈现有规则的正弦线而不出现紊乱。

测试结果如图 2～9。

测试结论及说明：

在能够接受的精度范围内，程序正确地绘制出了函数曲线，当精度要求更高时，程序绘制出现交叠。当出现精

表 2　测试用例 2 输入预期表

| 表达式 | 线宽 | 颜色 |
|---|---|---|
| y=2sin(10x) | 1 | rgb(0,0,0) |
| y=2sin(20x) | 1 | rgb(0,0,0) |

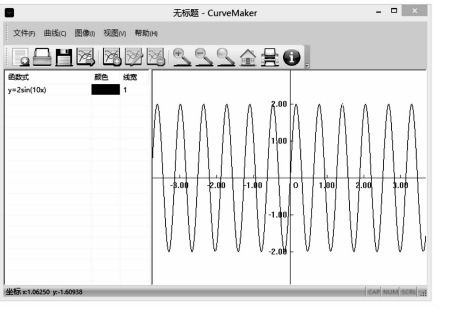


图 7　y=2sin（10x）测试截图

（下转第 263 页）