第33卷 第24期 计 算 机 工 程 2007年12月
Vol.33 No.24 Computer Engineering December 2007
·博士论文· 文章编号：1000—3428(2007)24—0013—03 文献标识码：A 中图分类号：TP311

数据流可视化语言 LabScene 的连线设计

随阳轶[1]，林 君[1]，范永开[1,2]，张晓拓[3]
(1. 吉林大学仪器科学与电气工程学院，长春 130026；2. 清华大学计算中心，北京 100081；3. 吉林大学通信工程学院，长春 130022)

摘 要：连线作为数据流可视化语言编辑器的重要组成部分，一方面要表示逻辑上的数据依赖关系，另一方面自身要以容易理解的物理形式展示出来。为了有机地整合这两个属性，该文提出了线树的概念，利用线树的结构表示逻辑属性，线树的节点表示物理属性，使得两个属性既相互关联又相互独立，从而为解析运行提供完整的逻辑信息且容易进行编辑和优化。该设计已在面向虚拟仪器开发的数据流可视化语言 LabScene 中实现。

关键词：数据流可视化语言；LabScene 语言；连线；虚拟仪器

Design of Wiring in Dataflow Visual Programming Language LabScene

SUI Yang-yi[1], LIN Jun[1], FAN Yong-kai[1,2], ZHANG Xiao-tuo[3]
(1. College of Instrument Science and Electronic Engineering, Jilin University, Changchun 130026; 2. Computer and Information Management Center, Tsinghua University, Beijing 100081; 3. School of Communication Engineering, Jilin Unviersity, Changchun 130022)

【Abstract】 Wiring is a very important component of editor in dataflow visual programming languages. On the one hand, wiring should represent logical data dependencies; on the other hand, it should be show in an easily comprehended physical form. In order to organically integrate these attributes, the conception of wire-tree is proposed. The relationships of two attributes become both reliant and independent because of using the structure of wire-tree to represent logical attribute and nodes to represent physical attribute. Therefore, wiring supplies enough logical info of interpretive execution and is easy to edit and optimize. This design has been independently implemented in dataflow visual programming language ——LabScene, which can develop virtual instruments.

【Key words】 dataflow visual programming language; LabScene; wiring; virtual instrument

作为改进冯•诺依曼瓶颈[1]的一种备选的计算机体系结构，数据流是一种比基于文本语言的控制流更加丰富的计算模型。随着直接进行交互式的数据流图编辑变得越来越容易，各种数据流可视化语言(DFVPL)相继出现。数据流图的基础是节点和连接节点的线，对绝大多数DFVPL而言，连线的设计是必不可少的。

目前各种DFVPL在连线设计上都有不同程度的缺陷：VPP[2]和E[3]语言虽然都提供算法以绕过自身的元件进行连线，但是缺乏用户对连线本身的参与，因此连线后很难被修改。Prograph[4]只提供输入输出端的斜线连接，容易出现较多交叉线，造成界面混乱。VEE[5]以转角来区分两条线是否真正连接，在交叉较多的地方不容易跟踪数据流动的方向。

本文所提出的连线设计是自主研制的用于虚拟仪器开发的数据流可视化语言 LabScene 中编辑器的一部分，其中线树是其核心技术，它具有以下几个优点：(1)以美观易懂的物理形式展示连线；(2)遍历线树可获得连线的逻辑关系；(3)容易对连线进行编辑和优化。

1 LabScene 中的连线

LabScene 建模的依据是：传统的仪器是由面向最终用户的操作界面和面向工程师的电路图组成的。也就是说，只要提供用户操作的仪器面板和可用来设计的电路图，就可以创建一个虚拟仪器。

电路图基本元素可以抽象为 6 种：基本元件，芯片，管脚，连线，交叉点，线路板。将电路图的基本元素和数据流模型进行映射：元件和芯片都作为一种节点，不可分的元件视为功能节点，芯片及线路板视为容器节点，其内部可以再包容电路。

因此在设计面板中由一个数据流图描述的程序是不同种类的节点以及连接节点的线组成的集合。它可以表示为一个二元组：$Program = (Node, Wire)$。其中，$Node$是一个节点集合，包含不同的功能以及一个输入管脚集合P_{in}，一个输出管脚集合P_{out}；$Wire$表示一条数据流通路径的有序集，它具有逻辑属性和物理属性，可以表示为 $Wire = (a_l, a_p)$，a_l是线的逻辑部分，a_p是线的物理部分。，

2 连线的过程

首先给出源点$A(x_a, y_a)$，目标点$B(x_b, y_b)$来帮助定义，其中x_a, y_a, x_b, y_b为两点在平面中的横纵坐标，并假设两点不重合。源点可以是两种类型的点：(1)能表示一个节点某个输出管脚的点；(2)已连到输出管脚的线上的某点。目标点是能表

基金项目：国家自然科学基金资助项目科学仪器专项(40127003)；吉林省科技厅科技引导项目(20030324)
作者简介：随阳轶(1980－)，男，博士研究生，主研方向：可视化编程技术，虚拟仪器；林 君，教授、博士生导师；范永开，博士；张晓拓，助理工程师
收稿日期：2006-12-20 **E-mail：**suiyangyi@163.com

示一个节点的输入管脚的点。

连线的过程是用户从源点出发连一条线到目标点的过程。用户在确定源点之后，点击源点进入连线状态，然后通过移动鼠标使系统根据鼠标运动轨迹产生正确的线段，并画出这些线段，如果鼠标沿先前运动的轨迹反方向移动则把先前画的线擦去。当达到目标点时，点击目标点完成连线，此时系统已连接所有线段并形成了需要的数据流线。在连线过程中用户也可以点击右键来取消连线操作，所有本次操作中已经产生的线段都将被擦除。

3 线的属性

3.1 线的逻辑属性

线的逻辑属性a_l可以表示为 $a_l = (dt, dc, u.p_{\text{out}}, v.p_{\text{in}})$，意思是数据可以从源节点$u$的输出管脚$p_{out}$流到目标节点$v$的输入管脚$p_{in}$中，其数据的类型和数据复杂度分别是$dt$和$dc$。

3.2 线的物理属性

线的物理属性a_p处于表现层，是用户可见的源管脚与目标管脚之间的表面连接。它可以表示为顺序连接在一起的线段集。下面先介绍为求解物理属性所引入的概念，再提出求解的过程：

(1)线段：为了使连线交叉较少，本文在两点间连线不采用斜线或曲线，而是添加必要的点产生水平或垂直的线段，每个线段可以表示为 $seg = (P_s, P_d, Color, Thickness)$，$P_s$和$P_d$分别表示线段两端的坐标，*Color*和*Thickness*分别表示绘制线段的颜色和粗细。而线段的颜色和粗细是由线的逻辑属性中携带的信息决定的。为了表示数据的类型和复杂度的不同，使用线的颜色来表示数据类型，用线的粗细表示数据复杂度。

(2)转折点：如果 *A*、*B* 两点的横纵坐标之一相等，那么它们之间的连线方式只有唯一的一种，且满足水平或垂直的要求。但是如果 *A*、*B* 两点的横纵坐标都不相等，需要添加一个转折点 *T*，那么 *T* 的产生方式有两种。在第 4 节中将详述这两种方式的取舍。

(3)交叉点：当两条线段交叉，且这两个线段是要表示连接在一起，则在交叉的地方增加一个交叉点 *C*。交叉点是用户从已有的线上某点进行连接时产生的。在实际绘图中交叉点表现为一个和线颜色相同的圆点。

(4)交互点：在绘图过程中，用户使用鼠标来参与连线的操作。当用户想经过某一个坐标点时，并在这个位置上点击一下，则产生一种点，称之为交互点 *P*，交互点的特点是连线过程中必须经过的点。交互点和转折点的区别是：前者是由交互产生的，后者是根据需要自动生成的。在交互点生成的时候，可能生成转折点。交叉点和交互点的区别是：前者改变了数据流的依赖关系，而后者不会。

多个交互点的连接，是一个递归的过程，即从起始点开始和第 1 个交互点以两点连接的方法进行连接，当第 1 个交互点连接好之后，再以此交互点作为起始点和第 2 个交互点进行连接，直到连接到结束点。图 1 展示了这个过程。

为了区别上述的各种点，将源点、目标点、交叉称之为核心点，而转折点、交互点称之为非核心点，核心点是连线逻辑意义上不可缺少的点，而非核心点只是一种绘制时的辅助点，缺少它们只是在物理表示方面产生影响，并不会改变连线所表达的逻辑含意。

求解线物理属性的过程就是在连线中根据鼠标移动的轨迹产生一系列正确的点并按颜色和粗细信息顺序连接的过程，具体是：

(1)首先记录出发时的核心点坐标并把它记作起始点。

(2)如果到达目标核心点，跳到(3)；否则表示用户产生了交互点，记录交互点坐标，然后根据起始点和交互点按前面提到的规则自动产生相应的转折点，记录这些转折点的坐标，并插入到起始点和交互点之间，把起始点、转折点及交互点顺次连接起来，然后把交互点记为起始点。重复(2)。

(3)记录目标核心点坐标，连接起始点和目标核心点，完成连线。

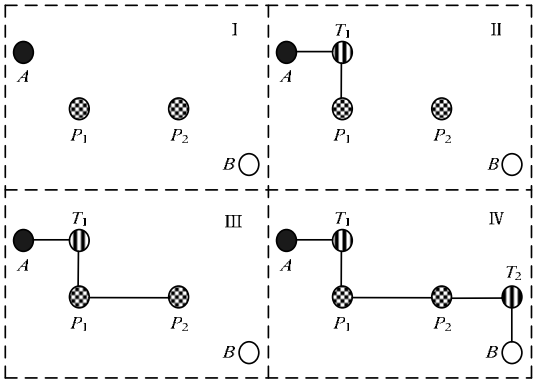

图 1 多个交互点的连接

4 引入线树

为了清晰地表示连线的逻辑属性和物理属性，提出线树的概念。基本思路是以树的结构表示连线的逻辑属性，以树的节点表示连线的物理属性。这样当线树构建完成之后只要遍历线树就可以得到逻辑属性，而通过访问线树的节点就可以获得物理属性。因此，两个属性既相互联系又相互独立。例如用户连线完成之后，移动某个线段的位置，只会改变其中拥有此线段的线树节点的坐标集的值，而不会改变此线的逻辑结构。

4.1 线树的节点

如果两个核心点之间没有其他核心点(有可能有非核心点)，这两个核心点之间的连线可以看成为一个线树节点(*WNode*)，$WTNode = (P_{\text{in}}, P_{\text{out}}, Parent, Children, Points)$，其中，$P_{\text{in}}$和$P_{\text{out}}$是连接的输入管脚和输出管脚；*Parent*是指向本节点的父节点；*Children*是指向本节点的各个孩子节点；*Points*是上节中根据连线操作所得到的点的坐标集。

4.2 线树

线树(*Wire-Tree*)是$n(n>0)$个线树节点的有限集，为了方便确定数据的流动路径，把线树定义为一个二元组 $WT = (root, F)$，其中：*root*是一个*WTNode*节点，它是线树的根节点；*F*是*m*棵子树，表示为 $F = (T_1, T_2, L, T_m)$，其中T_i是根节点第*i*棵子树。则从定义可知，从根节点到某棵子树的叶子节点的路径正表示了数据流动的路径。在同一个线树中的所有节点都具有相同的数据类型、数据复杂度和数据。

(1)生成线树的过程就是在已有的线树上逐渐插入节点的过程，算法如下：

1)首先根据输入管脚和输出管脚生成根节点，在设计面板中把根节点坐标集字段的点用直线连接起来，颜色和粗细由输出管脚的数据类型和数据复杂度所确定。

2)如果要从已有的线上某点连线到新目标点，就在连接的地方加一个交叉点，用圆点表示，同时将生成两个新的线

树节点，其中一个节点的坐标集是交叉点到新目标点的那些点，另一个节点的坐标集是把已有的线节点从交叉处分成两部分的后一部分。去掉已有线节点的坐标集中交叉点之后的部分。把代表已有线的节点作为新生成的节点的父，修改 3 个节点的管脚字段。把连到新目标点的线树节点的坐标点用直线连接起来。

图 2 是线树创建的一个示例，其中字段是按定义的顺序来表示。

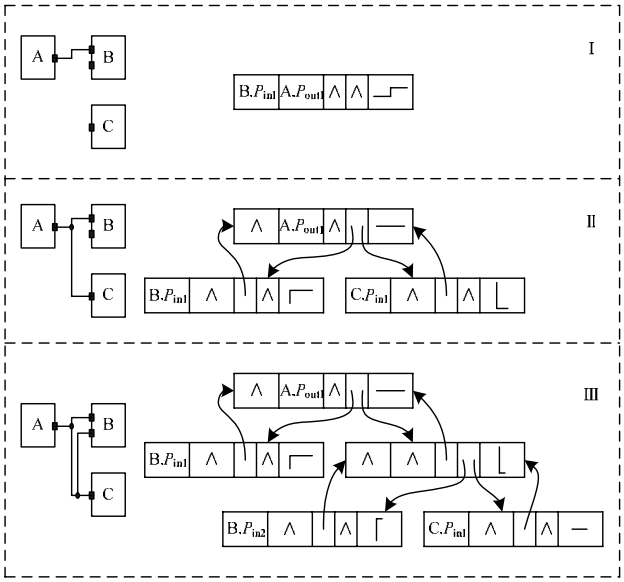

图 2　线树的创建示例

(2)节点的选择：以线树节点中各个线段为中心线，上下或左右几个像素所形成的矩形作为选择区域。如果鼠标点击的位置在某个节点的选择区域中，就表示被选中。

(3)删除节点的算法如下：1)选择要删除的节点，执行删除操作，如果删除的是根节点，把所有节点的坐标集表示的连线擦除，并把所有指向根节点的引用变成空，则删除结束。2)否则擦除它的坐标集的连线，去除其父节点的孩子列表对它的指向，如果父节点孩子列表中不只一个孩子，则删除结束。3)否则父节点要与此孩子进行合并，即把此孩子节点的坐标集添加到它的父节点坐标集，并把它的孩子节点的父改为它的父节点，然后删除孩子节点，最后擦除交叉点。

5　布局优化

在连线过程充分考虑到用户的参与一方面增加了用户对程序的理解，另一方面可能造成一些界面上的不美观，图 3 中左列图展示了这些不美观的情况，图 3 中右列图是经过优化后的结果。

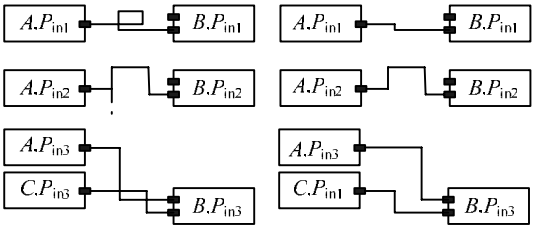

图 3　造成不美观的各种情况

(1)去掉环线：关键要找到进入环的 Q 点。若 Q 点不存在于坐标集中，则问题转化为找出两条绝对相交的线段。即判断以下条件：1)垂直线段的横坐标是否在另一水平线段两端的横坐标之间；2)水平线段的纵坐标是否在此垂直线段两端的纵坐标之间。

如同时满足，则得到 Q 点，其横坐标与垂直线段相同；纵坐标与水平线段相同。然后分别插入 Q 点到两条正交线段中，形成新的坐标集。

此时坐标集中重复出现的点即为进入环的点。删除重复点之间的所有点及一个重复点，并在屏幕上擦除相应的线段。

(2)剪除重复线段：用户在产生交互点之后，沿着已有的轨迹反向运动鼠标，再继续连线，这样会产生重复线段。剪除的方法是首先检查是否存在重复线段。对于垂直线段，判断以下条件：1)两条垂直线段相连；2)有一垂直线段的不相连点在另一垂直线段上。

如果都满足，则有产生重复的垂直线段。水平线段的判断方法与之类似。删除相连的点并在屏幕上擦除两条线段，然后连接上述两条线段的不相连点。

(3)减少交叉线：如图 3 所示，在$A.P_{out1}$到$B.P_{in1}$已经连上的情况下，再连接$C.P_{out1}$到$B.P_{in2}$则需要添加一个转折点。两种方式都可以，但是为了减少线交叉，故增加两条规则：

规则 1　如果使用其中一种方式增加的线交叉数目多于使用另一种，则使用交叉数目增加少的一种方式；

规则 2　如果两种方式都没有增加或都增加了相同数目的线交叉数，则增加的转折点的方法是：设$\triangle x = |x_b - x_a|$，$\triangle y = |y_b - y_a|$；如果$\triangle x > \triangle y$，那么从起始点$A$开始，先画出水平线，再画出垂直线，同时，两者相交的地方自动产生一个转折点$T(x_b, y_a)$。反之，如果$\triangle x < \triangle y$，那么从起始点开始，先画出垂直线，再画出水平线，同时，两者相交的地方自动产生一个转折点$T(x_a, y_b)$。

6　结束语

本文提出的线树模型可以同时表示连线的物理属性和逻辑属性。这使得在连线结束后可以容易地对物理属性进行优化而不会影响数据逻辑关系的表达。从而改进了目前流行的 DFVPL 中连线的缺陷。该连线设计在自主研发的面向虚拟仪器的数据流可视化语言 LabScene 中实现并优化。结果证明，设计是有效和可行的。

参考文献

[1] Wesley M J, Hanna J R P, Richard J M. Advances in Dataflow Programming Languages[J]. ACM Computing Surveys, 2004, 36(1).

[2] 耿晨歌. 面向虚拟仪器系统的可视化编程语言研究[D]. 杭州: 浙江大学, 1999.

[3] 王瑞荣, 汪乐宇. 事件触发并发数据流模型[J]. 软件学报, 2003, 14(3): 410.

[4] Cox P T, Pietrzykowski T. Advanced Programming Aids in PROGRAPH[C]//Proc. of ACM SIGSMALL Symposium on Small Systems. [S. l.]: ACM Press, 1985: 27.

[5] Motti K. Reusable Test Executive and Test Programs Methodology and Implementation Comparison Between HP VEE and LabView[C]//Proc. of IEEE Systems Readiness Technology Conference. Antonion, USA: IEEE Press, 1999: 305.