doi: 10.3969/j.issn.1006-1576.2012.05.025

高速数据流动态曲线绘图系统

丁颖浩[1]，李滚[1]，刘强[1]，何晓波[2]
（1. 电子科技大学空天科学技术研究院，成都 610054；2. 成都飞机设计研究所，成都 610041）

摘要：针对无人机仿真实验中数据庞大难以处理的问题，基于软件工程思想，详细讨论一种实时曲线绘图系统（RtCurveRenderer）的设计过程。系统包含控件容器外壳和实时曲线绘图控件 2 部分，采用 C++语言编程，以 MFC 应用程序框架为基础，采用异步的数据拾取缓冲与曲线绘图机制，结合环形数据流缓冲技术与移位数据缓冲技术，有效地解决了高速数据覆盖丢失问题。实验证明：该系统能准确地捕获每个实验数据，并流畅地绘制出数据曲线图，有效地提高了实验数据分析效率。

关键词：图形设备接口；曲线控件；多线程；环形数据缓冲；数据绘图；实时曲线绘图系统
中图分类号：TP391　　**文献标志码**：A

High-Speed Data Stream Dynamic Curve Drawing System

Ding Yinghao[1], Li Gun[1], Liu Qiang[1], He Xiaobo[2]
(1. *Aeronautics & Astronautics Academy of Science & Technology*, *University of Electronic Science & Technology of China*, *Chengdu* 610054, *China*; 2. *Chengdu Aircraft Design Research Institute*, *Chengdu* 610041, *China*)

Abstract: Aiming at the problem huge experimental data on UAV simulation experiment, based on software engineering ideas, describes a real-time data curve drawing system (RtCurveRenderer) design process in detail. The system includes control container shell and the real-time curve drawing control, which is programmed by C++ language and based on MFC application framework. Use asynchronous data capture buffer and curve drawing mechanism, combine with the ring data stream buffer technology and the shift data buffer cache technology to effectively solve the problem of data loss due to high-speed data over write. The test proves that this dynamic curve drawing system can capture each of the experimental data accurately, draw the data curve graph smoothly, and improve the efficiency of experimental data analysis effectively.

Key words: graphic device interface; curve control; multi-threading; ring data stream buffer; data curve drawing; RtCurveRenderer

0　引言

无人机地面半物理仿真试验也称为半实物仿真试验[1]，它是将无人机系统的一部分以实物（如液压作动器）方式引入仿真回路，其余部分以数学模型描述。由于这种将数学模型与物理实物相结合的联合仿真可以获得较高的真实度，常用来验证飞行控制律系统设计方案的正确性，找出存在的缺陷。在某型号无人机的地面仿真试验中，仿真目标机通过专用的模型调试器加载并运行一套基于 Matlab 工具开发的数字飞行包，在从模型的 C-API 接口中提取出所有关键仿真变量的地址后，按照 200 Hz 的频率将数据传送到 VMIC 光纤反射内存服务器上。仿真上位机按 5 ms 周期从反射内存服务器上取回数据，交由客户端软件解码显示。由于数据刷新速度太快，屏幕上的数据文本覆盖频率太高，甚至出现闪烁，人眼无法同时阅读所有数据信息，难以看出数据变化趋势。虽然现有的客户端软件可以将数据保存至文件供离线分析处理，但由于数据表格非常庞大，需要消耗很多人力，延缓实验进度。

文中将要讨论的实时曲线绘图系统（以下简称为 RtCurveRenderer）则很好地解决了上述问题。系统使用了与文献[2]类似的多线程结构，并特意将曲线绘图任务置于前台线程，同时避免了曲线图跳帧与数据丢包 2 大难题；文献[3]讨论了基于 COM 技术使用 Visual Basic 语言进行多坐标轴曲线绘图控件编程的方法，但由于 Visual Basic 语言是一种解释性语言，其执行效率无法达到期望的要求，而本系统使用的 C++语言不存在低效率的问题；文献[4]采用动态链表来实现了对实时数据的高效链式存储，虽然链表的确可以快速地插入新数据、删除旧数

收稿日期：2011-12-10；**修回日期**：2012-01-16
基金项目：中央高校基本科研业务费专项资金资助“复杂环境下高空高速长航时无人机多源传感器导航数据融合关键技术研究”（ZYGX2009J087）
作者简介：丁颖浩（1987—），男，四川人，工学硕士，从事检测技术及其自动化装置研究。

据，但是内存浪费极大，因为单个 8 字节的 double 变量节点需要额外付出 8 字节的空间用来存放前驱和后即节点指针，内存利用率最高为 50%，所以本系统使用环形数据缓冲区解决了内存利用率低下的问题。

1 GDI 简介

图形设备接口(graphics device interface，GDI)的主要任务是负责系统与绘图程序之间的信息交换，处理所有 Windows 程序的图形输出。GDI 是位于应用程序与不同硬件之间的中间层，这种结构让程序员从直接处理不同硬件的繁重工作中解放出来，把硬件间的差异交给了 GDI 处理。GDI 通过将应用程序与不同输出设备特性相隔离，使 Windows 应用程序能够毫无障碍地在 Windows 支持的任何图形输出设备上运行。它把 Windows 系统中的图形输出转换成硬件命令然后发送给硬件设备。GDI 以文件的形式存储在系统中，系统需要输出图形时把它载入内存，如果转换成硬件命令时遇到非 GDI 命令，系统还可能载入硬件驱动程序，驱动程序辅助 GDI 把图形命令转换成硬件命令。图 1 给出了基于 GDI 的应用程序绘图的主要步骤。

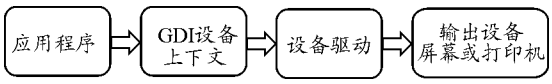

图 1 基于 GDI 的应用程序绘图步骤

2 系统结构设计

RtCurveRenderer 绘图系统在 MFC 多文档(MDI)应用程序框架的基础上进行开发，每个曲线图单元独占 1 个子窗口，子窗口框架类 CChildFrame 以 MFC 默认的 CMDIChildWndEx 类作为基类，子窗口视图类 CChildView 以曲线绘图控件 CCurveCtrl 类作为基类。

CChildFrame 类使用 2 个定时器分别完成数据与曲线图的定时刷新：

1) RTX 实时定时器以 200 Hz 频率准确地从 VMIC 反射内存上取得实验数据，并同时存入 CurveCtrl 控件的实时数据流缓冲区。

2) Windows 多媒体定时器以 25 Hz 频率通知 CChildView 类刷新移位数据缓冲区并完成数据绘图任务。

RtCurveRenderer 绘图系统结构如图 2。

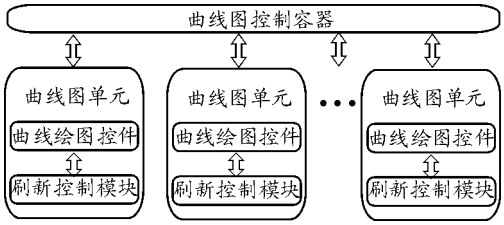

图 2 RtCurveRenderer 系统结构图

3 曲线绘图控件设计

曲线绘图控件 CCurveCtrl 类主要提供数据绘图服务，由于其功能比较复杂，因而被划分为图形绘制、曲线管理、移位数据缓冲、文件数据加载以及实时数据流缓冲共 5 个功能模块，图 3 给出了这几个功能模块之间的关系。

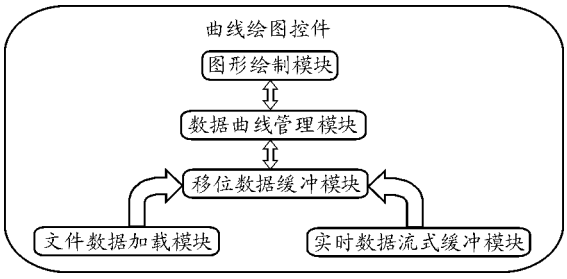

图 3 曲线绘图控件功能模块

3.1 文件数据加载模块

文件数据加载模块负责将保存在文件中的实验数据缓冲到内存。曲线绘图控件在每次刷新移位数据缓冲区时都会从文件缓存中读取适当数量的数据，具体文件数据读取步骤如下：

1) 从文件中提取可用数据总量及数据时间密度信息。

2) 开辟一个足够大的内存缓冲区，并预先使用部分文件数据将缓冲区完全填满。

3) 将此内存缓冲区等分为 4 段，初始情况下，读写指针都停在缓冲区起点位置。移位数据缓冲模块使用数据时间密度信息控制数据读取量，文件数据加载模块需要在数据被取走后立即更新读指针位置，并递减可用数据量。

4) 当读指针越过第 1 段缓冲区后，利用事件对象通知文件加载线程使用新文件数据覆盖第 1 段缓冲区内容。当读指针越过第 2 段缓冲区后，文件加载线程再次使用新文件数据覆盖第 2 段缓冲区内容。

5) 当读指针达到缓冲区末端时，通知文件加载线程填充第 4 段缓冲区内容，并让读指针返回缓冲

区首端，继续读取数据。

6) 当可用数据量减至 0 时，移位数据缓冲模块将不再继续读取数据。

3.2　实时数据流缓冲模块

曲线绘图控件只有在一帧画面绘制完成后才能再次刷新移位缓冲区内部数据。在通常情况下，实验用工控机 1 s 内最多能完成 100 帧画面的绘制任务，则移位数据缓存的刷新周期最短为 10 ms，远大于反射内存数据刷新周期。如果只在刷新移位缓存时才从反射内存上取数，刷新率的不匹配必将导致实验数据因覆盖而大量丢失。实时数据流缓冲模块从绘图线程上分离出来，充分利用当前多核心处理器的并行处理能力，在一个专用线程上独立运行，从而达到数据与曲线的异步刷新要求。由于控件外部程序可以使用任意高的频率向实时数据流缓冲模块写入数据，如果此频率恰好等于反射内存数据刷新频率，则可确保实验数据不会丢失，图 4 给出了其工作流程示意。

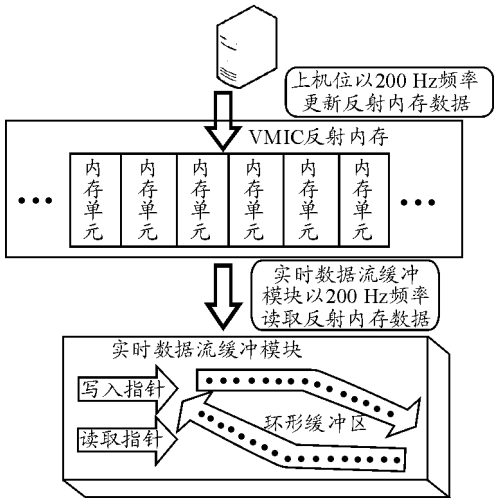

图 4　实时数据流缓冲模块工作流程

3.3　移位数据缓冲模块

移位缓存将实验数据按先后顺序存储起来，其容量和数据刷新频率决定了曲线在横轴上的持续时间。曲线绘图控件在每次绘图操作开始之前，会首先刷新全部移位数据缓冲区，移位数据缓冲区更新步骤如下：

1) 如果这个移位缓存与某个文件数据加载器相关联，则通过当前时刻与上次刷新时刻的时间差计算出需要读取的记录数据数量。如果这个移位缓存与某个实时数据流缓存相关联，则查询实时数据流缓存在 2 次刷新操作之间所获得的新数据总数量。

2) 在新数据数量 N 被确定之后，将移位缓冲区中的数据向尾端移动 N 步，以便丢弃尾端 N 个旧数据，并在起始端为新数据预留出存储空间。

3) 从相应数据源(即文件数据加载器或实时数据流缓存)复制 N 个新数据到移位缓冲区起始端空闲区域。

3.4　曲线管理模块

曲线管理模块使用高效的数据结构将绘图控件中所有曲线信息组织管理起来，包括曲线颜色，曲线名称，曲线数据移位缓存对象等，图形绘制模块将使用这些信息来绘制曲线、坐标系等画面元素。

3.5　图形绘制模块

3.5.1　GDI 绘图方式选择

曲线绘图控件 CurveCtrl 的图形绘制模块以 Windows GDI 为基础，负责所有与绘图相关的操作。使用 GDI 有 2 种绘图方式：

1) 直接方式：直接在硬件设备上下文(hardware device-context)中执行绘图命令。由于每个操作都必须通过图形设备驱动程序直接访问硬件来执行，不仅绘图效率低下，而且清屏操作还会带来剧烈的画面闪烁。

2) 间接方式：使用兼容设备上下文(compatible device-context)完成绘图，然后将结果显示到屏幕上。这种绘图方式效率较高，并且由于曲线图是直接被“贴”出来的，所以根本看不到任何闪烁[5]。唯一的缺点是兼容设备上下文中的兼容位图会耗费额外的系统内存。

通过分析可以看出，间接方式比直接方式更加适合图形绘制模块的设计。

3.5.2　图形绘制模块结构

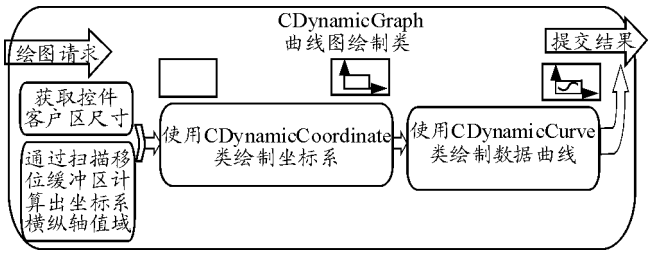

图 5　曲线图绘制过程

因为一幅完整的曲线图总是由坐标系和曲线 2 部分组成，所以图形绘制模块也顺理成章地被划分为 2 个子模块：坐标系绘制模块和曲线绘制模块，

分别由 CDynamicCoordinate 和 CDynamicCurve 类实现。CDynamicCoordinate 坐标系绘制类负责完成坐标系边框、坐标系背景、坐标轴、坐标刻度、刻度网格线、刻度文本及注释文本的绘制任务，CDynamicCurve 曲线绘制类只负责完成数据曲线的绘制任务。曲线图绘制过程如图 5。

3.5.3 坐标系绘制过程

该系统的曲线绘制要求美观、精确，为此坐标环境的设置非常重要，坐标环境由 CDynamicCoordinate 类负责绘制，主要操作步骤及要点如下：

1) 使用坐标系边框色填充整个空白曲线图。

2) 在曲线图中使用坐标系背景色绘制出一个圆角矩形，并围绕圆角矩形区域绘制一个边框，使其具有立体效果，作为坐标系背景。

3) 在以上边框所包围的圆角矩形范围内绘制直角坐标系，坐标轴需要带有箭头，并且能够自动调整横纵坐标的摆放位置以防止和坐标刻度文本相重叠。

4) 使用 PS_DASH 风格的画笔绘制坐标轴刻度线，坐标刻度必须以 1、2、5 为单位进行增长，如：“3.14, 3.15, 3.16, 3.17, 3.18”、“0, 2, 4, 6, 8”、“50, 100, 150, 200, 250”。为了防止由于横轴时间刻度分割算法切换而带来的标度跳变现象，需要锁定首次计算出的分割步长值。

5) 使用 CDC::Textout()函数在坐标刻度附近输出刻度文本，当最小刻度值大于 1 000 或者最大刻度值小于 0.1 时，用科学计数法显示刻度数值，指数显示在坐标轴箭头附近。

6) 使用曲线颜色在坐标系顶部输出曲线变量名字符串，需要根据曲线数量调整输出字号：变量少时使用大字号，变量多时使用小字号。

3.5.4 曲线绘制过程

在 CDynamicCoordinate 类绘制完成后，通过调用 GDI 接口中的 BitBlt 函数把坐标系背景图传递给 CDynamicCurve 类，CDynamicCurve 类将实验数据点在坐标系背景图上连成曲线。设曲线显示范围内的数据最大值为 yMax，最小值为 yMin。由于数据可能在某段时间内保持不变，使得 yMax 等于 yMin，又因为(yMax−yMin)项会出现在坐标转换公式中，所以必须对(yMax−yMin)等于 0 这种情况作特殊处理。设 yValue 为数据值，coordinateRect 为屏幕空间坐标区域，则坐标转换代码如下：

```
#define FLOAT_ZERO 1e-300
xScreen = (xValue - xMin) / (xMax - xMin)
   * (coordinateRect.right - coordinateRect.left)
   + coordinateRect.left;
if ((yMax-yMin) < FLOAT_ZERO) {
   yScreen = (coordinateRect.bottom
      + coordinateRect.top) / 2.0;
} else {
  yScreen = (yMax - yValue) / (yMax - yMin)
      * (coordinateRect.bottom - coordinateRect.top)
      + coordinateRect.top;
}
```

由于数据点非常密集，常常导致相邻的 $(xValue_{[n]}, yValue_{[n]})$、$(xValue_{[n+1]}, yValue_{[n+1]})$等数据点经过坐标转换后得出完全相同的(xScreen, yScreen)。如果避免在同一屏幕像素上重复执行画线操作，将会极大地降低 GDI 函数的调用量，使得绘图性能大幅提高，因此应该在画线循环中加入重复像素过滤功能，简化代码如下：

```
int xLastDest = xScreen[0];
int yLastDest = yScreen[0];
curveDC.MoveTo(xScreen[0], yScreen[0]);
for (DWORD loop = 1; loop < varCount; ++loop) {
   if ((xScreen[loop] != xLastDest)
    || (yScreen[loop] != yLastDest)) {
      curveDC.LineTo(xScreen[loop], yScreen[loop]);
      xLastDest = xScreen[loop];
      yLastDest = yScreen[loop];
   }
}
```

4 系统测试

系统允许用户方便地设置曲线的绘制时间长度，新数据从曲线图右侧进入坐标系，并随时间的推进向左移动，在经过指定时间长度后从左侧移出坐标系范围。用户不但可以从实时曲线图中读取历史数据，直观地观察数据的总体变化趋势，还可以重放保存在文件中的实验数据，在期望数据曲线与实验数据曲线之间做同步对比，大大降低了实验数据的分析难度。经过测试，系统表现良好且运行效率较高，在 1 台 CPU 为 Intel Core i7 860 的工控计算机上，可同时以 25 Hz 绘制 20 幅持续时间为 60 s 的数据曲线图。由于实时数据流缓冲模块采用异步数据刷新机制，数据写入速度非常快，即使数据更新频率达到 1 000 Hz 也不会出现覆盖丢失现象。

（下转第 93 页）