基于 GDI + 的 ColorGrid 控件的设计

惠为君[1],吉善兵[2]
(1. 盐城工学院 电信学院,江苏 盐城 224051; 2. 盐城市无线电管理监测站,江苏 盐城 224001)

摘要:GDI + 是新一代图形设备的交互入口,是封装图形控件的主要工具。基于 GDI + 技术,完成了 ColorGrid 控件的封装,并用该控件实现了无线监测信号的网格显示。该控件使用简单,性能稳定,可以作为数据网格化显示的通用控件。

关键词:GDI + ;VC#. NET;图形控件;平滑滚动
中图分类号:TP393 文献标识码:A 文章编号:1671 - 5322(2008)02 - 0031 - 04

自定义控件是具有可视特征的组件,它是用户接口的一部分。. NET 提供的预定义控件在数量和功能上都是有限的,因此,针对应用程序的特殊的需求,开发者需要封装具有特点接口的自定义控件。

. NET Framework 中构建的自定义控件有 3 种:(1)从头派生自定义控件,控件从 Control 类派生;(2)派生一个现有的 Windows Form 控件;(3)复合控件,控件从 UseControl 类派生。

其中,复合控件通过两个或多个现有控件创建一个自定义控件,也称为用户控件。它可以根据控件的相对位置安置组成控件。可以处理组成控件的每一个事件,实现由添加到自定义控件的属性驱动的附加功能,并触发事件。

复合控件的优点是可以创建更高级别的抽象,还可以创建一组应用程序中使用的复合控件,实现功能使用的一致性。

在构建 Windows Forms 自定义控件时,一般要用 GDI + 绘制自定义控件的外观,在控件内部建立代码,以响应键盘和鼠标事件或自定义事件,实现控件的功能。其中,图形控件可以把数据直观地显示,使人一目了然,因而使用广泛。笔者在项目开发中,需要通过多种图形手段显示无线频谱资源的使用情况。因发现市场上的使用的图形控件不能达到要求,为此,在 Visual Studio 2005 平台上,以基于 VC#. NET 的 GDI + 为工具,设计了 ColorGrid 控件。

ColorGrid 是一个二维网格图形控件,应用程序运行它时,可以控制控件要显示的数据、网格的颜色、可以更改控件的显示模式,实现数据的全部或部分显示。

1 GDI + 简介

GDI + 是与. NET Framework 中的图形设备进行交互的入口,是编写图形交互软件的必备工具。GDI + 包含 1 组非常整洁、一致、强大的高性能的类,提供了我们需要的大部分图形功能。GDI + 被组织到 6 个命名空间中。其中,命名空间 System. Drawing 提供基本的图形功能,包括绘图表面、图像、颜色、笔刷、钢笔、和字体等; System. Drawing. Drawing2D 提供高级光栅和矢量图形功能;System. Drawing. Text 提供高级字体功能,包括描述字体系列、字样、文本、文本格式化等。GDI + 提供的工具可以在窗口、位图、打印机上绘制二维图形,这包括绘制各种直线、曲线、矩形、多边形等,还可以绘制格式文本,控制文本的质量、指定文本的字体等。在此基础上,经过许多类型转换,可以轻松地创建复杂图形、文字效果,从而可以实现自定义控件的符合需要的外观。

2 控件实现

2.1 控件需求

根据无线监控程序的要求,对 ColorGrid 控件提出如下的基本需求:(1)实现对数据的网格显

收稿日期:2007 - 03 - 27
作者简介:惠为君(1969 -),男,江苏盐城,讲师,硕士,主要研究方向为计算机测控、数字信号处理。

示,数据最多可达4000个;(2)对不同性质的数据用不同颜色来表示;(3)绘制行列字符串显示数据单位;(4)全部模式在视区中显示所有的数据,局部模式在视区中显示部分数据,通过滚动条滚动显示所有的数据;(5)上下文菜单中实现对显示模式和区域填充颜色以及区域边框显示的控制;(6)背景显示;(7)统性能稳定,运行时要几乎不占资源。

2.2 控件背景绘制

首先在Visual. Studio2005平台上创建一个项目,项目的类型为Windows控件库。修改控件的名字为ColorGrid。因为要在控件上放置组成控件,ColorGrid类的基类要采用UseControl类。

在控件上放置一个hScrollBar控件,设置其dock属性为Bottom,使其停靠在控件的底部。该滚动条控件将实现在局部模式下的视区的滚动显示。放置一个PictureBox控件,设置其dock属性为为Fill,背景色属性为SYSTEM. WIDNOWS。PictureBox控件将作为一个图形容器,在其中实现网格绘制。

重载UserCnotrol的基类的OnPaint函数,在其中编写代码,实现控件背景的绘制。当控件发布后,将具有如图1所示的简洁的外观。

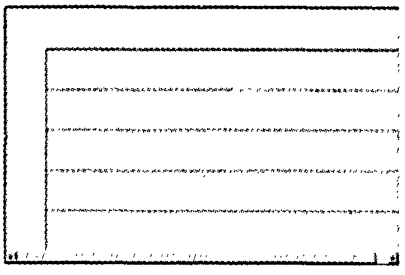

图1 控件的背景图

Fig. 1 Background figure of control parts

控件背景的点划线的默认条数由控件的宽度决定,也可以由应用程序在设计时确定。

2.3 控件视区的绘制

视区的绘制分为两部分:用来显示数据单位的左边矩形和顶部矩形网格绘制,用来显示数据值的矩形网格的绘制。

如果用传统的画频谱图的方法,一维网格显示一维数据的话,由于数据量很大而控件宽度相对较小,每个网格的宽度将可能小于一个象素。如果控件显示部分数据,用滚动条滚动来显示数据文档,则滚动条滚动的范围很大,也不利与数据的直观显示。解决方法是把一维数据分段,转换为二维数据,用二维网格显示数据。这样将充分利用屏幕的宽度,可显示的数据将会大大增多。这是ColorGrid的基本思想。

根据设计需求,设置缓冲区存放接收到的数据。根据数据的段数和控件的宽度,决定顶部矩形框的数目和宽度,根据每一段数据的长度和控件的高度决定左边矩形框的数目和高度。从控件显示数据的美观考虑,每段数据的数目要求较少,一般在10个左右即可。数据的段数一般较多,有时可达上千个,因而上边框矩形的数目可能很多。

顶部矩形的宽度由下式确定:

$$w = (width - b)/n$$

其中,$width$为控件的宽度;b为左边预留下来,用来显示数据单位的矩形的宽度;对于全部模式,n为数据的段数;对于局部模式,n为在视区中看到的数据的段数。由于w一般不为整数,如果取w的整组部分作为网格的宽度,那么$(width - b)/n$的余数大小对应的控件部分将不能参与显示网格。n越大,浪费的控件空间越多。解决的办法是采用非均匀网格间距的方法,即把w四舍五入取整作为第1个网格的宽度,把$n \times w$四舍五入取整的值减去前面$n-1$个网格的宽度之和的差作为第n个网格的宽度。此时,虽然有一些网格比其它网格的宽度多一个象素,但这样的微小差异在视觉上分不出来,感觉上所有网格的宽度还是相等的。

控件左边框的绘制方法与顶边框的绘制方法相同。在左边框和顶部边框绘制完成后,就可以在其中绘制表示数据单位的文本。

每一个数据都对应一个数值矩形框。设数据的段数为MaxCols,即为矩形网格的列数。每一段数据的数目为MaxHors,表示矩形网格的最大行数。应用程序传给控件一个二维坐标(i,j),i表示所在行,j表示所在列。控件把i和j的值转换为数值矩形框的顶点坐标。矩形框的宽度等于i所在列的顶部矩形框的宽度,矩形框的高度由有应用程序决定。数值矩形框画好后,根据应用程序的颜色要求给矩形框填色。

设置一个ColorGrid类的私有函数:

```
private void drawGrid(Graphics g);
```

在该函数中编写代码,实现上述所有的思想,就可以画出符合要求的二维网格。为了实现网格的正确重画,必须重载控件的onPaint()函数,在

onPaint()函数中调用上述drawDataGrid()函数。

2.4 平滑滚动的实现

由于数据量很大，在视区中不能全部显示数据。因此需要实现视区的平滑滚动，来显示所有数据。在滚动时，需要使控件中表示单位的左矩形框始终显示，也就是只要实现部分视区的平滑滚动，此时，需要添加滚动条来控制视区的滚动。

设置滚动条的Maximum属性等于将要在视区显示的矩形区域的宽度。此宽度随应用程序中待显示的数据的多少而变化。在滚动条的Scroll事件处理函数hScrollBar_Scroll中，实现滚动条的平滑滚动。此时，需要考虑滚动条的滚动类型。滚动条的Scroll事件处理程序传递了一个ScrollEventArgs参数，该参数提供一个属性Type，它说明了滚动事件的种类。其中，ScrollEventArg.SmallIncre和ScrollEventArg.Smalldecrement为增量滚动，说明用户单击了滚动条末端的小箭头。ScrollEventArg.LargeIncrement和ScrollEventArg.LargeDecrement说明用户单击了滚动条的亮边区域或单击并拖拽了滚动条本身，这四种情况下，视区都要重绘自身。同时，在hScrollBar_Scroll函数中调用一个辅助函数Slide(Point oldp, Point newP)。Slide函数是实现视区平滑滚动的主要函数，在该函数中，根据滚动条的始末位置，计算文本中需要显示的部分，把该部分在视区中来显示。Slide函数的关键代码如下：

```
if (oldP.X ! = newP.X)
{
bool goingUp = newP.X - oldP.X > 0;
int delta = (newP.X - oldP.X) / 12;
while (true)
{
if (i = = newP.X)
break;
i + = delta;
if (goingUp && i > newP.X)
{
i = newP.X;
continue;
}
if (goingUp && i < newP.X)
{
i = newP.X;
continue;
}
if(needDisplay(i))
{
drawDataGrid (g);
}
if (i ! = newP.X)
{
Thread.Sleep(10);
}
}
}
```

其中i表示滚动条的位置，如果oldP.X ! = newP.X，说明滚动条发生了滚动。needDisplay(i)根据滚动条滚动位置，计算文档中需要在视区中的显示部分。当需要显示的部分确定以后，就调用drawDataGrid (g)函数来绘制视区。

3 关键技术

3.1 象素控制

当应用程序把数据在横向和纵向的坐标、网格的颜色等参数传给控件后，控件就在给定的位置画矩形网格并且对网格填色。

绘制矩形、填充矩形时，需要考虑绘制区域的边界问题。例如调用DrawRectangle函数绘制矩形$RECT(x_0,y_0,width,height)$，绘制矩形时，从点$(x_0,y_0)$到对角点$(x_0+width,y_0+height)$之间绘制，矩形的实际上占有的宽度为$width+1$个像素，高度为$height+1$个像素。当调用*FillRectangle*函数填充时矩形时，其操作是从点(x_0,y_0)到对角点$(x_0+width-1,y_0+height-1)$范围内填充。因此，应用程序调用控件绘制并填充网格时，将不能看到网格边框，因为它们恰好被填充色遮住了。解决方法是在填充矩形时，将填充矩形改为$(x_0+1,y_0+1,width,height)$，这样边框和填充部分不再重叠。

3.2 抗闪烁技术

由于滚动条移动时，文档的待显示部分发生变化，因此，控件的重绘较频繁。这个问题处理不好，不但会使画面闪烁，还会影响系统的稳定性。

一般情况下，控件的重绘都要重载控件的onPaint()函数或在控件的Paint事件处理函数中完成对控件的重绘功能。不过，由于滚动条在滚动时，需要在合适的位置多次绘制图形，如果调用控件的Paint事件处理程序来绘制，会使得系统性能

不稳定,因为无法保证 Paint 事件会按照进度发生,这将会影响滚动效果。

可以在滚动事件处理函数中对不同的滚动类型加以处理来解决这个问题。系统默认对每一次滚动事件都重绘控件,对于大幅滚动,由于每一个相素的滚动都要重绘显示区域,因此重绘十分频繁,使得画面出现闪烁现象。处理的方法是,对于小幅度滚动,先调用 Slide( ) 函数,计算待显示的区域,再调用 drawDataGrid (g) 完成网格的直接绘制。对于大幅度滚动,可以只考虑滚动条的始末位置,由这两个位置计算待显示的区域,然后重绘网格。这种方法的好处是整个视区只在末位置重绘一次,因而有效的克服了闪烁现象。

4 在频谱分析中的应用

当 ColorGrid 控件接受矩形坐标、高度、填充色彩等数据以后,调用控件的 refresh 函数就可以实现有关数据的网格显示。无线电频率资源经过国家频率划分后,分成很多段频谱,再经过规划后,分成很多信道。每个信道用一个中心频率和频率宽度表示。对每一个信道进行无线电监测,判断各信道是否占用,在无线电监测中具有重要意义。如图 2 所示,ColorGrid 控件显示了无线信道的分配情况,灰色网格表示已占用信道,白色网格表示未占用信道。

图 3 显示了控件的各个信道的状态。网格中用不同的颜色表示不同的信道状态。信道状态分为如下几种:空闲频率、合法在用频率、合法未用频率、非法未判明频率、非法已判明频率等。这 5 种状态分别用不同的颜色表示,颜色可以由使用者选择。

MHz	100.0	101.0	102.0	103.0	104.0
.000 .025 .050 .075	100.000	101.000	102.000	103.000	104.000
.100 .125 .150 .175	100.100	101.100	102.100	103.100	104.100
.200 .225 .250 .275	100.200	101.200	102.200	103.200	104.200
.300 .325 .350 .375	100.300	101.300	102.300	103.300	104.300
.400 .425 .450 .475	100.400	101.400	102.400	103.400	104.400

图 2 无线信道分配

Fig. 2 Wirless channel assignment

MHz	93.0	94.0	95.0	96.0	97.0
.000 .025 .050 .075	93.000	94.000	95.000	96.000	97.000
.100 .125 .150 .175	93.100	94.100	95.100	96.100	97.100
.200 .225 .250 .275	93.200	94.200	95.200	96.200	97.200

图 3 信道状态

Fig. 3 Channel states

参考文献:

[1] 杨浩,张哲峰. Eric White. "GDI + 程序设计"[M]. 北京:清华大学出版社,2002:175 - 178.

[2] 陈新,王宝宝. "基于 GDI 的二维图形绘制打印及预览"[J]. 微计算机应用,2005,26(2):248 - 251.

[3] 李顺亮,张均东,甘辉兵. "GDI + 技术在综合船舶监控系统中的应用"[J]. 大连海事大学学报,2005,31(1):42 - 44.

The Design of ColorGrid Control Based on GDI +

HUI Wei-jun[1], JI Shan-bing[2]

(1. School of Electric and Information Engineering, Yancheng Institute of Techonology; Jiangsu Yancheng 224051, China;
2. Yancheng Station of Wireless Manager and Monitor, Jiangsu Yancheng 224001, China)

Abstract: GDI + is a new interface of graphics device, and it is the primary tool to encapsulate graphics control. ColorGrid control is encapsulated based on GDI + technique , and it has been used to display wireless spectrum with grid mode. The control is stable and simple to use, and it can display data with grid mode as a general control.

Keywords: GDI + ; VC#. NET; graphics control; slide smoothly