第5章

更有效的显示方法

运行第4章的程序，生成随机点。当缩放或移动地图时，画面会闪烁；当调整窗口边框时，地图内容不会自动填充到新出现的窗口区域。当最小化地图窗口，再复原窗口时，地图内容完全消失了。这些问题看来到了需要被解决的时候了。

5.1 闪烁的原因

先分析一下为什么会闪烁。首先，让我们看一下在地图窗口 Form1 中负责画图的函数，UpdateMap 函数代码如下。请注意，这次不是更新代码，所以不必把以下内容复制到项目中，因为这些代码就是来自于项目中的已有内容。

Form1.cs

```
private void UpdateMap()
{
    //生成绘图工具
    Graphics graphics = CreateGraphics();
    //清空窗口
    graphics.FillRectangle(new SolidBrush(Color.White), ClientRectangle);
    //绘制空间对象
    foreach (XFeature feature in features)
        feature.draw(graphics, view, true, 0);
    //回收绘图工具
    graphics.Dispose();
}
```

其中，feature.draw 函数的作用就是在当前视图（view）下面，利用绘图工具（graphics）在窗口上逐个绘制 XFeature 实例。请注意，是一个一个地绘制！而且，因为其 graphics 输入参数就是地图窗口的绘图工具，所以，每一个绘制操作都是直接画在窗口上的，也就是说，窗口要一个一个地显示这个画上去的 XFeature 实例，画一个更新一次，如果需要绘制的内容比较多，那么给人的感觉就是窗口在不断地更新，这就是闪烁的原因所在。

5.2 用双缓冲解决闪烁问题

如何解决闪烁的问题呢？有个办法，就是在内存中用户看不到的地方建立一个和地图窗口一模一样的窗口，把这个窗口叫作背景窗口，与之对应的、用户能够看到的那个地图窗口叫作前景窗口。首先在背景窗口中画上所有需要绘制的地图内容，画好后，一次性把背景窗口中的内容搬到前景窗口，这样就不会闪烁了。

在计算机中，每一个窗口在内存中都有一个存储显示内容的区域，称为显示缓冲区。而上面的方法涉及了两个窗口，所以有两个缓冲区，因此这种方法称为双缓冲方法。根据这个原理，来试一下吧。

首先，将地图窗口 Form1 的标准属性 DoubleBuffered 设成 true，这样，就部分实现了双缓冲的目的，即由操作系统帮忙，定期从一个背景缓冲区中更新前景窗口，而不是窗口内容一有变化就立刻更新。但它的效果是有限的，闪烁还是会发生，还需要定义自己的背景缓冲区。

在 Form1.cs 中定义一个背景窗口作为全局变量，实际上它就是一个 Bitmap 类型的图片，代码如下。

Form1.cs

```
Bitmap backwindow;
```

把背景窗口搬到前景窗口的方法是用语句 graphics.DrawImage (backwindow,0,0)，其中 graphics 就是前景窗口的绘图工具。这个语句的意思是把背景窗口这张图片在前景窗口中画在起点为(0,0)的这个位置上，所以背景窗口的大小就必须与前景窗口相同，否则有些地方就可能画不到了。

有了上述的关键语句，可以修改 UpdateMap 函数，代码如下。

Form1.cs

```
public void UpdateMap()
{
    //如果地图窗口被最小化了，就不用绘制了
    if (ClientRectangle.Width * ClientRectangle.Height == 0) return;
    //更新 view，以确保其地图窗口尺寸是正确的
    view.UpdateMapWindow(ClientRectangle);
    //如果背景窗口不为空，则先清除
    if (backwindow != null) backwindow.Dispose();
    //根据最新的地图窗口尺寸建立背景窗口
    backwindow = new Bitmap(ClientRectangle.Width, ClientRectangle.Height);
    //在背景窗口上绘图
    Graphics g = Graphics.FromImage(backwindow);
    //清空窗口
    g.FillRectangle(new SolidBrush(Color.White), ClientRectangle);
    //绘制空间对象
    foreach (XFeature feature in features)
        feature.draw(g, view, true, 0);
    //回收绘图工具
    g.Dispose();
    //获得前景窗口的绘图工具
    Graphics graphics = CreateGraphics();
    //把背景窗口内容绘制到前景窗口上
    graphics.DrawImage(backwindow, 0, 0);
}
```

UpdateMap 函数首先检查当前地图窗口是否可见，如果不可见就不用绘制了，直接返回；然后调用了一个 XView 的 UpdateMapWindow 函数(该函数还没有偏写)，其目的是将当前地图窗口的范围告诉 view，让它能及时更新，其中 ClientRectangle 是窗口的标准属性，记载了窗口的大小；接下来，在背景窗口中绘图；最后把背景窗口的内容复制到前景窗口。

现在把 UpdateMapWindow 函数的代码补充如下。

BasicClasses.cs/XView

```
public void UpdateMapWindow(Rectangle rect)
{
```

```
        MapWindowSize = rect;
        Update(CurrentMapExtent, MapWindowSize);
    }
```

现在试试运行程序，读者应该惊喜地发现，地图窗口不再闪烁了！

5.3 解决显示内容消失的问题

尽管显示效果提升了，可是还有不尽如人意的地方，本章之前提到的一些问题依然存在。如图 5-1 所示，当移动窗口时，窗口中的内容如果被移到屏幕外边，再移回来时，被移出的内容就没有了；当最小化窗口，再复原窗口时，窗口中的地图内容也全都没有了。

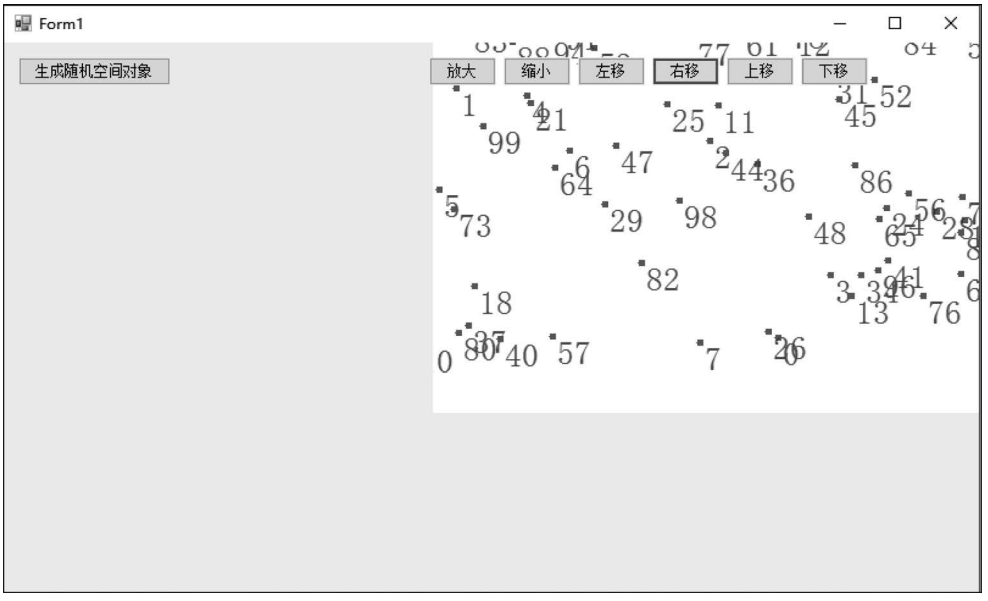

图 5-1 移动窗口后被覆盖的内容消失了

内容消失是什么原因呢？原来，窗口被部分或全部遮挡时，程序会自动重绘之前在窗口中添加的一些可视化控件，如上述界面中的那些按钮，因为它们是由程序自动维护的。但是，窗口中编程绘制的地图内容却无法享受到自动重绘的待遇，这就需要编程者主动维护，否则不会进行自动重绘，所以也就造成内容不见了。

那么如何知道什么时候该重绘，什么时候不该重绘呢？Windows 会提醒，当需要重绘时，地图窗口会收到一个 Paint 事件，只要重写这个 Paint 事件就行了。

但是下一个问题又来了，重绘什么呢？是不是把 UpdateMap 函数重新执行一遍？这是不需要的，因为已经有了背景窗口，所以在重绘时，只需要把背景窗口内容再重新复制到前景窗口中就好了。现在来试一下，为地图窗口 Form1 添加一个事件处理函数 Paint，代码如下。

Form1.cs

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    if (backwindow != null)
        e.Graphics.DrawImage(backwindow, 0, 0);
}
```

该函数相当简单，就是把背景窗口搬到前景窗口，其中的绘图工具 Graphics 直接从 Paint 事件的参数 e 中获得就好了。现在试一下运行程序，发现无论怎样移动窗口，或切换窗口状

态，里面的地图内容都不会消失了。

此外，值得注意的是，在这里，我们并没有像 UpdateMap 函数中一样，在执行完绘图操作后，运行 graphics.Dispose()命令回收绘图工具。其原因是，此处的 e.Graphics 并非由我们生成，而是直接引用的，而且在执行完毕上述语句后，可能还需要继续执行 Paint 事件的其他响应函数，因此我们不应该主动回收。一个简单原则就是：谁生成的，谁回收。这个原则其实也适用于所有涉及资源回收的类对象，比如 Bitmap。

有了 Paint 函数，我们发现，其实可以在 UpdateMap 函数中，直接调用 Paint 函数把背景窗口的内容绘制到前景窗口中，因为它们做的是同一件事。调用 Paint 函数的方法有些特殊，因为 Paint 函数是一个系统定义的窗口事件，我们需要通过系统的消息函数激活这个事件，这个消息函数就是 Invalidate，其含义就是令窗口内容失效，重绘窗体。代码如下。

Form1.cs

```
public void UpdateMap()
{
    //如果地图窗口被最小化了，就不用绘制了
    if (ClientRectangle.Width * ClientRectangle.Height == 0) return;
    //更新 view，以确保其地图窗口尺寸是正确的
    view.UpdateMapWindow(ClientRectangle);
    //如果背景窗口不为空，则先清除
    if (backwindow != null) backwindow.Dispose();
    //根据最新的地图窗口尺寸建立背景窗口
    backwindow = new Bitmap(ClientRectangle.Width, ClientRectangle.Height);
    //在背景窗口上绘图
    Graphics g = Graphics.FromImage(backwindow);
    //清空窗口
    g.FillRectangle(new SolidBrush(Color.White), ClientRectangle);
    //绘制空间对象
    foreach (XFeature feature in features)
        if(feature.spatial.extent.IntersectOrNot(view.CurrentMapExtent))
            feature.draw(g, view, true, 0);
    //回收绘图工具
    g.Dispose();
    //重绘前景窗口
    Invalidate();
}
```

5.4 解决显示内容变形的问题

当然，还存在其他问题，例如当拖动地图窗口的边框时，其地图内容不随之变化。其实在拖动边框、改变窗口大小时，也会自动引发 Paint 事件，但由于此时背景窗口已经不适应于新的窗口大小了，所以简单地复制背景窗口变得没有意义。这时就需要调用 UpdateMap 函数重新绘制背景窗口和前景窗口了。窗口大小发生变化时，会触发 SizeChanged 事件，我们为地图窗口 Form1 添加一个事件处理函数 SizeChanged，代码如下。

Form1.cs

```
private void Form1_SizeChanged(object sender, EventArgs e)
{
    UpdateMap();
}
```

运行程序，拖动窗口边框改变地图窗口大小，地图内容可以自动填充了，但是新问题又出现了，如图 5-2 所示，填充的内容是变形的，所有的点都压缩到一起。确切地说，是由于 XView

中的 ScaleX 与 ScaleY 是根据地图窗口的宽、高和地图范围确定的，而实际上，这两个参数之间的比例关系，也就是地图的纵横比，应该是保持不变的。

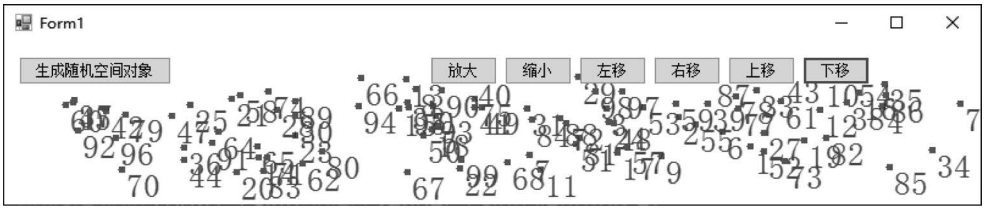

图 5-2　由于改变窗口大小造成的地图变形

如何保持 ScaleX 与 ScaleY 之间的比例关系不变，最简单的，令二者永远相等，这对大部分地图来说都是这样的。要实现这一点非常简单，只要在 XView 的 Update 函数中令二者都等于其中一个值即可，可以是其中的最大值或最小值，这样，就保证了显示内容不变形。修改后的 Update 函数代码如下，它令 ScaleX 及 ScaleY 均等于宽度或高度比值的最大值。

BasicClasses.cs/XView

```
public void Update(XExtent _extent, Rectangle _rectangle)
{
    CurrentMapExtent = _extent;
    MapWindowSize = _rectangle;
    MapMinX = CurrentMapExtent.getMinX();
    MapMinY = CurrentMapExtent.getMinY();
    WinW = MapWindowSize.Width;
    WinH = MapWindowSize.Height;
    MapW = CurrentMapExtent.getWidth();
    MapH = CurrentMapExtent.getHeight();
    ScaleX = ScaleY = Math.Max(MapW / WinW, MapH / WinH);
}
```

当然，读者也可以把上述函数中的 Math.Max 改成 Math.Min，这样变形问题也不会发生，但显示内容可能有所变化。为理解此变化，我们添加一个新功能，就是“显示全图”。在 form1.cs 中，我们增加一个名为“bFullExtent”的按钮“全图”，其单击事件处理函数代码如下。

Form1.cs

```
private void bFullExtent_Click(object sender, EventArgs e)
{
    if (features.Count == 0) return;
    XExtent fullextent = new XExtent(features[0].spatial.extent);
    for (int i = 1; i < features.Count; i++)
        fullextent.Merge(features[i].spatial.extent);
    view.Update(extent, ClientRectangle);
    UpdateMap();
}
```

该函数首先计算当前所有空间对象的最小外接地图范围(fullextent)，然后据此更新当前视窗(view)重绘地图，达到显示全图的作用。这其中用到了 XExtent 的两个新的函数：一个是新的 XExtent 构造函数，它通过复制输入的地图范围构造新的地图范围；另一个是合并地图范围的 Merge 函数。为什么需要构造一个新的地图范围(fullextent)，而不是直接等于 features[0].spatial.extent 呢？其原因是在调用 Merge 函数时，fullextent 的值是会发生变化的，如果它的值是直接引用自 features[0].spatial.extent，那么 features[0]的空间范围也会发生变化，这显然是不对的。上述两个函数的实现代码如下。

BasicClasses.cs/XExtent

```
public XExtent(XExtent extent)
{
    upright = new XVertex(extent.upright);
    bottomleft = new XVertex(extent.bottomleft);
}
public void Merge(XExtent extent)
{
    upright.x = Math.Max(upright.x, extent.upright.x);
    upright.y = Math.Max(upright.y, extent.upright.y);
    bottomleft.x = Math.Min(bottomleft.x, extent.bottomleft.x);
    bottomleft.y = Math.Min(bottomleft.y, extent.bottomleft.y);
}
```

Merge 函数的原理是根据输入的地图范围，计算新的地图范围坐标极值，并更新当前地图范围；新的 XExtent 构造函数则通过复制节点的方法获得两个角点，它也引用了 XVertex 的一个新的构造函数，该构造函数其实也是复制了一个新的 XVertex 实例，避免修改原有实例，代码如下。

BasicClasses.cs/XVertex

```
public XVertex(XVertex v)
{
    x = v.x;
    y = v.y;
}
```

完成上述用于显示全图的代码后，重新运行程序，生成随机点，然后经过缩放、移动等浏览地图动作，再单击“全图”，会看到所有空间对象都会显示在地图窗口中。此时，如把 XView.Update 函数中的 Math.Max 改成 Math.Min，则发现，单击“全图”并不能保证显示的内容是完整的，而仅能保证单个轴向上是完整的。此外，不管是用 Math.Max 还是用 Math.Min，当显示全图时，地图内容都不在窗口中间，而是靠窗口的左方或下方。这是 XView 中的 ToMapVertex 和 ToScreenPoint 两个函数的问题造成的，这两个函数的代码如下。

BasicClasses.cs/XView

```
public Point ToScreenPoint(XVertex onevertex)
{
    double ScreenX = (onevertex.x - MapMinX) / ScaleX;
    double ScreenY = WinH - (onevertex.y - MapMinY) / ScaleY;
    return new Point((int)ScreenX, (int)ScreenY);
}

public XVertex ToMapVertex(Point point)
{
    double MapX = ScaleX * point.X + MapMinX;
    double MapY = ScaleY * (WinH - point.Y) + MapMinY;
    return new XVertex(MapX, MapY);
}
```

它们的坐标转换都是基于比例尺（ScaleX 和 ScaleY）及地图范围的最小横纵坐标值（MapMinX 和 MapMinY）的。而在新的 Update 函数中，由于 ScaleX 或 ScaleY 为保持地图不变形，被修改了取值，那这其实就意味着，当前在地图窗口中显示的地图范围与 Update 函数输入的地图范围已经不一致了，因此，MapMinX 和 MapMinY 也就不再是 Update 函数输入的地图范围的左下角 X 及 Y 坐标了。这就好比，我们希望把一个长方形的地图范围放进一个正

方形的地图窗口中显示，那么，为了保证原有地图不变形，实际显示出来的地图范围也会是一个正方形，而不再是输入的长方形范围。基于上述考虑，我们就需要在 Update 函数中计算并更新实际显示的地图范围。新的 Update 函数代码如下。

BasicClasses.cs/XView

```
public void Update(XExtent _extent, Rectangle _rectangle)
{
    //给地图窗口赋值
    MapWindowSize = _rectangle;
    //计算地图窗口的宽度
    WinW = MapWindowSize.Width;
    //计算地图窗口的高度
    WinH = MapWindowSize.Height;
    //计算比例尺
    ScaleX = ScaleY = Math.Max(_extent.getWidth() / WinW, _extent.getHeight() / WinH);
    //根据比例尺计算新的地图范围的宽度
    MapW = ScaleX * WinW;
    //根据比例尺计算新的地图范围的高度
    MapH = ScaleY * WinH;
    //获得地图范围中心
    XVertex center = _extent.getCenter();
    //根据地图范围的中心，计算最小坐标极值
    MapMinX = center.x - MapW / 2;
    MapMinY = center.y - MapH / 2;
    //计算当前显示的实际地图范围
    CurrentMapExtent = new XExtent(MapMinX, MapMinX + MapW,
        MapMinY, MapMinY + MapH);
}
```

其过程就是，先计算比例尺、实际的地图范围（宽度和高度），然后根据希望显示的地图范围的中心，计算当前显示的坐标极值和地图范围。其中，获得地图范围中心的 getCenter 函数代码如下。

BasicClasses.cs/XExtent

```
public XVertex getCenter()
{
    return new XVertex((upright.x + bottomleft.x) / 2, (upright.y + bottomleft.y) / 2);
}
```

重新运行程序，发现当窗口尺寸发生变化时，地图内容的改变比较符合正确的感觉了。

5.5 提高显示效率

目前显示的地图数据都是比较简单的，所以显示速度好像还可以，但如果需要显示大量的空间对象，可能就要慢很多了。为此，一个提高显示效率的办法就是不要绘制那些不可能出现在当前地图窗口中的对象。如何判断一个空间对象是否会出现在当前窗口呢？只要判断当前地图窗口对应的地图范围与空间对象的地图范围是否相交即可，如不相交，那就不需要绘制了。

据此，修改 Form1.cs 中的 UpdateMap 函数，在绘制空间对象时，完成这一判断即可，部分修改代码如下。

Form1.cs

```
public void UpdateMap()
{
    ……
```

```
        //绘制空间对象
        foreach (XFeature feature in features)
            if(feature.spatial.extent.IntersectOrNot(view.CurrentMapExtent))
                feature.draw(g, view, true, 0);
        ……
    }
```

XExtent 的 IntersectOrNot 函数是用来判断当前的地图范围(也就是 feature.spatial.extent)是否与输入的地图范围(即 view.CurrentMapExtent)相交,如果相交,就继续画下去。IntersectOrNot 函数定义如下。

BasicClasses.cs/XExtent

```
public bool IntersectOrNot(XExtent extent)
{
    return !(getMaxX() < extent.getMinX() || getMinX() > extent.getMaxX()
        || getMaxY() < extent.getMinY() || getMinY() > extent.getMaxY());
}
```

该函数就是排除所有不相交的可能,剩下的就必然是相交,其中只有逻辑判断,所以效率会很高。

现在把随机点的数量从 100 提高到 10 000,重新运行程序。尝试浏览地图,观察一下,当地图窗口中空间对象数量变少时,地图显示速度是否变快了。如果肉眼观察比较困难,可以在 Form1.UpdateMap 函数中增加一个时间(DateTime)变量,计算每次完成 UpdateMap 函数的运行时长,并把结果显示出来,或者通过 Console.WriteLine 打印出来看看。

5.6 总结

本章介绍了一些系统开发中的细节问题。经过上述的调整,相信迷你 GIS 变得更加强大了。但在实际的使用中,也许还会遇到各种奇怪的问题,希望读者能从本书中得到一些启发,试着解决这些问题。