第3章

屏幕坐标与实际坐标

能够在地图窗口中无极缩放及四处漫游以浏览各种空间对象实体是GIS的魅力所在，但是前两章所做的GIS是不能移动和缩放的，而且，当绘图窗口坐标值超过窗口像素范围时，超过的部分也是没有办法看到的。本章就是要解决这个问题。

3.1 坐标系统

为了将一个空间对象实体显示到窗口中，需要知道它的坐标值，比如经纬度或X、Y坐标等。而每一组坐标值都是与给定的坐标系统相关的，只有在给定的坐标系统下，它的值才具有意义。

由于地球是三维椭球体，而电脑屏幕是二维平面，因此必须选择一种方法或函数把分布在球面上的空间对象投影到平面上，实现在屏幕上的显示，这也就是地图投影概念，包括投影方式、椭球体定义等，相关知识在地图学当中有完整的介绍。上述讨论是针对二维电子地图的，当然，在电脑屏幕上也可以直接以立体的形式显示三维球体，这时我们似乎不需要涉及地图学当中的投影知识。的确是这样，地图投影解决了从地球这一椭球体上将地物投影到二维地图中的问题，而三维地图则直接展示地物的三维坐标，然后借助计算机图形学中通用的透视变换实现三维对象在二维电脑屏幕上的显示。显然，三维地图更具直观的视觉显示效果，但它在空间分析、量测和浏览等方面需要更复杂的处理和操作，在观测视角上不可避免地存在相互遮挡的问题，对计算机硬件也有特别的要求。相比之下，二维地图通常以“上帝视角”展示各种对象，一览无余。概括来说，三维地图由于更接近现实，因此比较适用于普通地图用户，而二维地图更适合GIS专业用户，支持各种专业的空间分析。本书的讲授重点将以二维地图为主。

在二维电子地图上，所有空间对象的位置都需要用单个或一系列坐标对来描述，这些坐标对被称为地图坐标。地图坐标必定是投影后的坐标，通常为米制坐标，具备量测特性，同时，也可借助投影转换公式，动态地将米制坐标转换成地理坐标，也即经纬度坐标显示出来。经纬度坐标可以很好地实现全球目标定位，但不具备量测特性，也就是说，地图上同样经度差或纬度差在不同位置的实际长度（也即实际的球面距离）可能是不一样的。目前来说，我们假设处理的地图坐标都是可量测的米制坐标，在本书后续章节，会讲解米制坐标与经纬度坐标之间的转换。

地图坐标是独立于显示设备存在的，当需要将用地图坐标描述的空间对象显示到电脑屏幕上时，我们就需要将地图坐标转成屏幕坐标。屏幕坐标也可以称作绘图窗口坐标，就是在屏幕上绘图涉及的那个窗口部分的坐标，如图3-1所示。

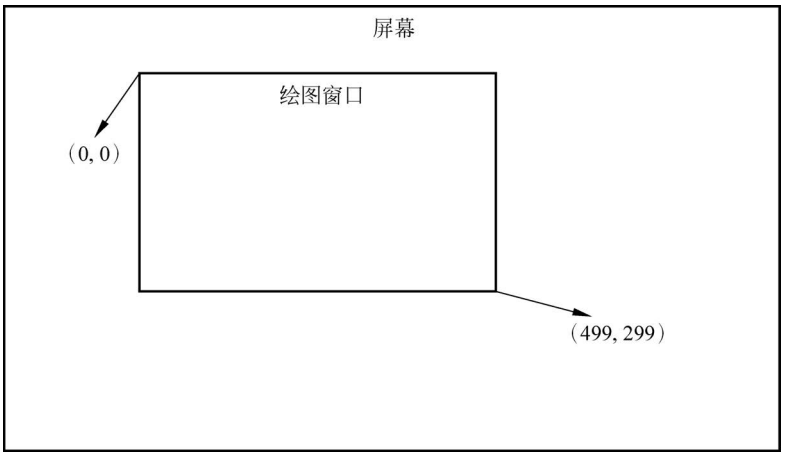

图 3-1　屏幕坐标示意图

由图 3-1 可知，大的矩形代表一个电脑屏幕，小的矩形代表一个绘图窗口。首先，绘图窗口通常是一个矩形，其次，它是由一系列像素构成的，因此，屏幕坐标值肯定是整数，而且，坐标原点在左上角，横坐标是 0，纵坐标也是 0，横纵坐标的最大值在右下角。以图 3-1 为例，其最大值分别是 499 和 299，则此绘图窗口的大小是 500×300，即横坐标 500 个像素，纵坐标 300 个像素。从上述介绍可以看出，横坐标取值是从左到右递增的，纵坐标取值是从上到下递增的。

地图坐标则完全是另外一回事。首先，其坐标值通常为实数，可能有正有负。其次，地图纵坐标取值通常是从下到上递增，这与屏幕纵坐标取值的递增方向是相反的，地图横坐标取值的递增方向与屏幕横坐标一致，都是从左到右逐渐增大。

了解上述基本概念后，我们可观察图 3-2。地图范围通常是给定的，而其中半透明矩形范围指代当前显示在绘图窗口中的部分地图内容，之后，可以通过缩放和平移来浏览地图其他部分。

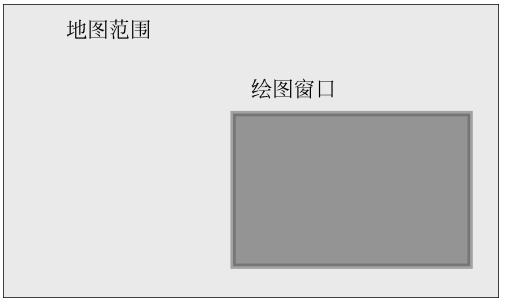

图 3-2　地图范围与绘图窗口的空间关系

显然，当前显示的地图部分的范围和窗口大小之间的对应关系决定了地图显示的内容和比例尺，因此，我们需要时刻记录这样的对应关系，实际上，就是用一个视图类记下当前显示的地图范围及绘图窗口范围。这个视图类的名字可以是 XView，把它放入 BasicClasses.cs 文件中，它的类定义如下。

BasicClasses.cs

```
class XView
{
    XExtent CurrentMapExtent;
    Rectangle MapWindowSize;

    public XView(XExtent _extent, Rectangle _rectangle)
    {
        CurrentMapExtent = _extent;
        MapWindowSize = _rectangle;
    }
}
```

其中 CurrentMapExtent 记录的是当前绘图窗口中显示的地图范围，MapWindowSize 记

录的是绘图窗口的大小，这里“范围”和“大小”是不一样的概念，“范围”代表的区域并不要求某一个角点必须是地图坐标原点，而“大小”代表的绘图窗口的左上角必定是坐标原点，因此有意义的信息就是长、宽。如前所述，Rectangle 是 C# 语言标准类库中提供的一个类，它通常包括 4 个成员，左上角横坐标(x)，左上角纵坐标(y)，窗口高度(Height)和宽度(Width)，显然，x 和 y 在这里取值为 0。

CurrentMapExtent 与 MapWindowSize 在显示上是重叠的，已知 CurrentMapExtent 及 MapWindowSize，则可以实现地图坐标与屏幕坐标之间的转换，如图 3-3 所示。

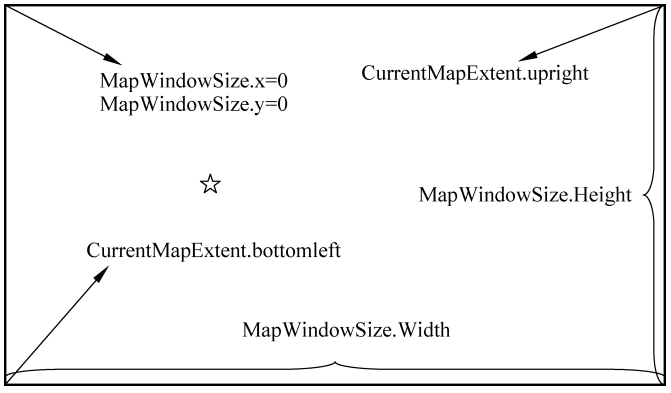

图 3-3　CurrentMapExtent 与 MapWindowSize 在显示上的重叠关系

3.2　两种坐标之间的转换

为简化描述，先定义以下变量：

```
MapMinX = CurrentMapExtent.bottomleft.x
MapMinY = CurrentMapExtent.bottomleft.y
WinW = MapWindowSize.Width
WinH = MapWindowSize.Height
MapW = (CurrentMapExtent.upright.x - CurrentMapExtent.bottomleft.x)
MapH = (CurrentMapExtent.upright.y - CurrentMapExtent.bottomleft.y)
ScaleX = MapW/WinW
ScaleY = MapH/WinH
```

MapMinX 是当前屏幕显示的地图范围的最小横坐标，而 MapMinY 是最小纵坐标；WinW 是绘图窗口宽度，WinH 是绘图窗口高度；MapW 是地图横坐标长度，MapH 是地图纵坐标长度；ScaleX 及 ScaleY 分别是横、纵坐标比例尺，即绘图窗口中的一个像素分别代表多少个横、纵地图坐标单位。上述这些变量对屏幕坐标与地图坐标之间的转换有重要的作用。

假设图 3-3 中五角星位置的屏幕坐标是(ScreenX，ScreenY)，它对应的地图坐标是(MapX，MapY)，则它们相互之间转换公式如下。

$$\frac{\mathrm{MapX}-\mathrm{MapMinX}}{\mathrm{ScreenX}}=\mathrm{ScaleX}$$

$$\frac{\mathrm{MapY}-\mathrm{MapMinY}}{\mathrm{WinH}-\mathrm{ScreenY}}=\mathrm{ScaleY}$$

也即

$$\mathrm{MapX}=\mathrm{ScaleX}\times\mathrm{ScreenX}+\mathrm{MapMinX}$$

$$\mathrm{MapY}=\mathrm{ScaleY}\times(\mathrm{WinH}-\mathrm{ScreenY})+\mathrm{MapMinY}$$

$$\mathrm{ScreenX}=(\mathrm{MapX}-\mathrm{MapMinX})/\mathrm{ScaleX}$$

$$ScreenY = WinH - (MapY - MapMinY)/ScaleY$$

实现屏幕坐标与地图坐标之间的转换是 XView 的主要工作，只要将上述公式转换成代码即可实现此功能。出于简化代码的考虑，我们先优化一下 XExtent 的类定义，令它如 Rectangle 类型的数据一样，可以直接提供宽度、高度等信息，代码如下。

BasicClasses.cs

```
class XExtent
{
    public XVertex bottomleft;
    public XVertex upright;

    public XExtent(XVertex _bottomleft, XVertex _upright)
    {
        bottomleft = _bottomleft;
        upright = _upright;
    }

    public double getMinX()
    {
        return bottomleft.x;
    }

    public double getMaxX()
    {
        return upright.x;
    }

    public double getMinY()
    {
        return bottomleft.y;
    }

    public double getMaxY()
    {
        return upright.y;
    }

    public double getWidth()
    {
        return upright.x - bottomleft.x;
    }

    public double getHeight()
    {
        return upright.y - bottomleft.y;
    }

}
```

从补充的 XExtent 类定义中可以看到，出现了一些带有前缀 public 的函数，通过这些函数可以直接获得地图范围的坐标极值及高、宽的范围。也许读者感觉仅针对 XView 来说这样做的价值似乎不大，但今后我们将大量使用 XExtent，有了这些函数，就会方便很多。

现在，根据新的 XExtent 类定义来完善 XView 的类定义。

BasicClasses.cs

```
class XView
{
```

```
        XExtent CurrentMapExtent;
        Rectangle MapWindowSize;
        double MapMinX, MapMinY;
        int WinW, WinH;
        double MapW, MapH;
        double ScaleX, ScaleY;

        public XView(XExtent _extent, Rectangle _rectangle)
        {
            Update(_extent, _rectangle);
        }

        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 = MapW / WinW;
            ScaleY = MapH / WinH;
        }

        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);
        }
    }
```

从上述代码看到，XView 的所有属性成员都没有前缀 public，这是一个有效的保护机制，因为，很多成员之间都是相互关联的，必须通过小心地计算才能确保它们的一致性。为此，我们把对这些值的编辑限定在类的内部，并通过 Update 函数来保持更新，就连构造函数也是引用了 Update 函数。使用 Update 函数的另外一个好处就是，如果今后需要更新 XView，只要外部调用 Update 函数即可，而不需要建立一个新的 XView 类实例，减少了系统的内存开销。ToScreenPoint 及 ToMapVertex 函数用于地图坐标与屏幕坐标之间的转换，它们就是简单实现了上述的坐标转换公式。屏幕坐标的点用 Point 记录，Point 是 C＃语言提供的标准类，它有 X 和 Y 两个整数成员。

有了坐标转换函数，我们就可以来更新 XGIS 类库中涉及屏幕绘图的函数，包括 XAttribute 中的 draw 函数，以及 XSpatial 中的 draw 函数，目前，在这两个函数中，坐标转换实际上是用“(int)”执行强制类型转换，现在，我们用 XView 中的函数来替代它。

针对不同大小的绘图窗口，不同的地图范围，坐标转换的结果是不一样的，所以，需要传递 XView 的实例来记录这些信息，在 draw 函数中增加一个 XView 类型的参数，代码修改如下。

BasicClasses.cs/XAttribute

```
public void draw(Graphics graphics, XView view, XVertex location, int index)
{
    Point screenpoint = view.ToScreenPoint(location);
    graphics.DrawString(values[index].ToString(),
        new Font("宋体", 20),
        new SolidBrush(Color.Green),
        new PointF(screenpoint.X, screenpoint.Y));
}
```

XSpatial 的 draw 函数为抽象函数，其实现代码是在 XSpatial 各个子类中给出的，所以也要相应修改其子类的 draw 函数，代码更改如下。

BasicClasses.cs/XSpatial

```
public abstract void draw(Graphics graphics, XView view);
```

BasicClasses.cs/XLine

```
public override void draw(Graphics graphics, XView view)
{
}
```

BasicClasses.cs/XPolygon

```
public override void draw(Graphics graphics, XView view)
{
}
```

BasicClasses.cs/XPoint

```
public override void draw(Graphics graphics, XView view)
{
    Point screenpoint = view.ToScreenPoint(centroid);
    graphics.FillEllipse(new SolidBrush(Color.Red),
    new Rectangle(screenpoint.X - 3, screenpoint.Y - 3, 6, 6));
}
```

其中，在 XLine 及 XPolygon 中，draw 函数还是空的，以后再补充。在 XAttribute 及 XPoint 的 draw 函数中，用到了 XView 的 ToScreenPoint 函数实现坐标转换。这时我们发现，在 BasicClasses.cs 中，XFeature 中的 draw 函数出现了红色波浪线，因为它还是调用了以前的 XAttribute 及 XSpatial 的 draw 函数，所以需要进行修改，代码如下。

BasicClasses.cs/XFeature

```
public void draw(Graphics graphics, XView view, bool DrawAttributeOrNot, int index)
{
    spatial.draw(graphics, view);
    if (DrawAttributeOrNot)
        attribute.draw(graphics, view, spatial.centroid, index);
}
```

XFeature 的 draw 函数也增加了一个 XView 类型的参数 view，并且传递给 XSpatial 及 XAttribute 的 draw 函数。这时，大家可能会想，到底这个 view 是如何赋值的？它来自哪里？在 3.3 节读者将会发现答案。

3.3 迷你 GIS 的再次更新

如图 3-4 所示，我们需要在窗口 Form1 中增加 4 个文本框及一个按钮。4 个文本框的名

称分别为 tbMinX、tbMaxX、tbMinY、tbMaxY，按钮的文本属性为“更新地图”，其名称为 bRefresh。

图 3-4　更新的程序界面

在这个更新的界面中，我们希望用户可以在 4 个文本框控件中输入要显示的地图范围，即最大、最小横纵坐标值，然后单击“更新地图”按钮即可重绘地图窗口。

为实现上述目的，我们需要考虑如何将 XView 融入程序当中。首先，在 Form1.cs 中声明一个 XView 的全局变量 view，用于时刻记录当前绘图窗口的大小及地图显示范围，这个全局变量在 Form1 的构造函数中被初始化。代码如下。

Form1.cs

```
XView view = null;
public Form1()
{
    InitializeComponent();
    view = new XView(new XExtent(new XVertex(0, 0), new XVertex(100, 100)), ClientRectangle);
}
```

在初始化 view 时，我们用了(0,0)和(100,100)两个角点定义了一个地图范围，实际上随便用什么值都可以，反正之后是需要修改的。ClientRectangle 是一个 Form1 内置的 Rectangle 类型的成员，记载的是 Form1 中有效的窗口范围，不包含无法使用的标题栏和边框。

在 Form1 的 UpdateMap 函数中，我们发现其中的 draw 函数调用被提示错误，这是因为缺少一个 XView 类型的参数。把 view 参数增加进去，代码修改如下。

Form1.cs/button1_Click

```
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();
}
```

鼠标单击文件处理函数 Form1_MouseClick 要多做一些改动，代码如下。

Form1.cs

```
private void Form1_MouseClick(object sender, MouseEventArgs e)
{
    //检查是否有空间对象
    if (features.Count == 0)
    {
        MessageBox.Show("没有任何空间对象!");
        return;
    }
    //将鼠标单击位置转换成地图坐标
    XVertex onevertex = view.ToMapVertex(e.Location);
    //查找距离上述地图坐标最近的空间对象
    double mindistance = Double.MaxValue;
    int findid = -1;
    for (int i = 0; i < features.Count; i++)
    {
        //计算空间对象与某一地图位置之间的距离
        double distance = features[i].Distance(onevertex);
        if (distance < mindistance)
        {
            mindistance = distance;
            findid = i;
        }
    }
    //计算与地图距离对应的屏幕距离
    double ScreenDistance = view.ToScreenDistance(mindistance, onevertex);
    //如果屏幕距离过大，则表示单击不准确
    if (ScreenDistance > 5)
    {
        MessageBox.Show("鼠标单击位置不准确!");
    }
    //找到一个空间对象，显示其属性信息
    else
    {
        MessageBox.Show(features[findid].getAttribute(0).ToString());
    }
}
```

上述函数在对空间对象数量进行有效性检查后，利用 view 参数的 ToMapVertex 函数将鼠标的单击位置转换成地图坐标；通过计算距离找到最近的空间对象；然后将此地图距离转换成屏幕距离，判断它是否在一个较小的范围以内(这里设定的阈值是 5)；如果在阈值范围以内，则显示其属性信息，否则会提示单击不准确。

这里有两个函数：用于计算空间对象与某一空间位置之间距离的 XFeature.Distance 函数，以及将地图距离转成屏幕距离的 XView.ToScreenDistance 函数，代码如下。

BasicClasses.cs/XFeature

```
public double Distance(XVertex vertex)
{
    return spatial.Distance(vertex);
}
```

BasicClasses.cs/XSpatial

```
public double Distance(XVertex vertex)
{
    return centroid.Distance(vertex);
}
```

BasicClasses. cs/XView

```
public double ToScreenDistance(double mapDistance, XVertex vertex)
{
    Point p1 = ToScreenPoint(vertex);
    Point p2 = ToScreenPoint(new XVertex(vertex.x - mapDistance, vertex.y));
    return Math.Sqrt((p1.X - p2.X) * (p1.X - p2.X) + (p1.Y - p2.Y) * (p1.Y - p2.Y));
}
```

XFeature. Distance 函数直接调用了 spatial 属性的 Distance 函数，而此函数通过计算中心点与输入参数 vertex 之间的直线距离作为返回值，这对于点对象来说是准确的，但对于线或面对象来说，这只是一个权宜之计，我们会在本书后续章节逐渐完善。

XView. ToScreenDistance 函数构造了两个屏幕点 P1 和 P2。前者是输入的地图位置对应的屏幕位置，后者是一个距离输入位置给定地图距离的地图位置对应的屏幕位置。然后，利用两点间直线距离公式，计算 P1 与 P2 之间的距离。

现在，需要为程序界面中新出现的按钮“更新地图”建立一个单击事件处理函数，代码如下。

Form1. cs

```
private void bRefresh_Click(object sender, EventArgs e)
{
    //从文本框中获取新的地图范围
    double minx = Double.Parse(tbMinX.Text);
    double miny = Double.Parse(tbMinY.Text);
    double maxx = Double.Parse(tbMaxX.Text);
    double maxy = Double.Parse(tbMaxY.Text);
    //更新 view
    view.Update(new XExtent(minx, maxx, miny, maxy), ClientRectangle);
    //更新地图
    UpdateMap();
}
```

上述函数首先读入描述新地图范围的 4 个 double 类型数字，其中，Double. Parse 是一个将字符串转换成 double 类型数字的函数；然后，根据新的地图范围，更新现有的 view；最后，更新地图。在更新 view 时，我们使用了一个新的 XExtent 构造函数，它通过顺序输入 4 个坐标极值，构造一个 XExtent 的实例，代码如下。

BasicClasses. cs/XExtent

```
public XExtent(double x1, double x2, double y1, double y2)
{
    upright = new XVertex(Math.Max(x1, x2), Math.Max(y1, y2));
    bottomleft = new XVertex(Math.Min(x1, x2), Math.Min(y1, y2));
}
```

看到上述函数的实现过程，相信读者领会了它的价值，其输入参数分别是两个横坐标和两个纵坐标，然后在初始化右上角及左下角角点时判断了坐标值的大小，实现正确的赋值，显然这个构造函数比另外一个更强大，因为它保证了角点的有效性。函数使用者不必担心输入的坐标值到底谁大谁小、顺序如何，因为这些问题在函数内部都会自动判断和解决。

运行程序，单击“生成随机空间对象”按钮。接下来，输入新的地图窗口范围(0,0),(1000,1000)，看看现在窗口中能显示几个点，是否跟图 3-5 一样。当然，如果觉得白色的背景不太好看，试试修改一下。进一步调整地图范围，看看地图对象的显示是不是能够按照读者的想象实现。

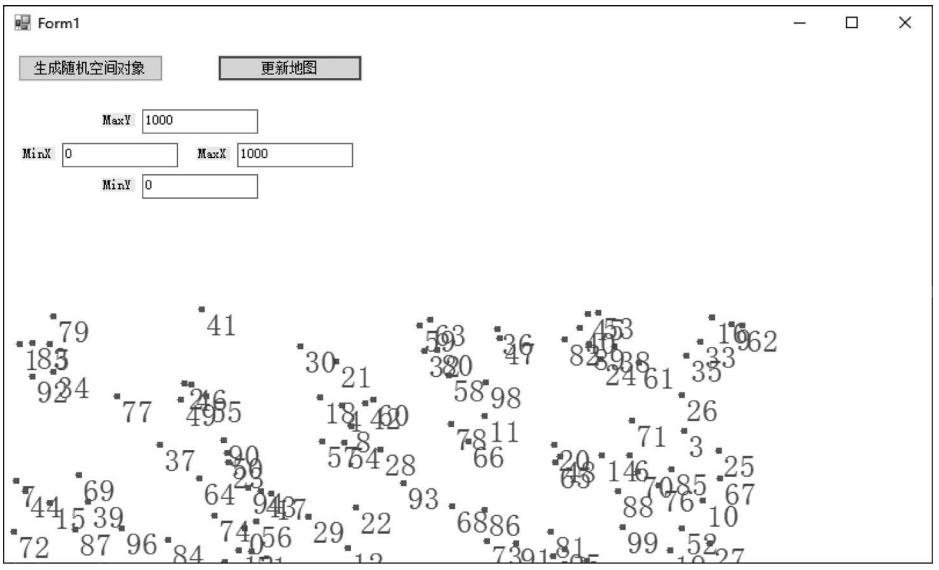

图 3-5　可控制地图显示范围的迷你 GIS 运行结果

3.4　总结

本章介绍了两种坐标系统，实际上都是平面坐标系统，只不过计量单位和原点的位置不同。本章的实现并没有涉及复杂的投影知识，因为这并不影响地图的显示。如果把投影问题考虑进来，需要做的就是在存储坐标或显示坐标时，按照投影描述文件的信息，把读到的坐标转换成显示地图所需要的坐标即可。当然，如果显示地图所需要的坐标与空间对象本身的坐标是一致的，那么连这样的转换都可以省掉了。相关的知识会在后续章节介绍。

如果能够按照本书的步骤进行，会发现我们已经可以实现地图的自由浏览了。当然，方法还比较笨拙，但原理是一样的，只要调整 view 的取值即可，可见 XView 的价值是相当大的。