



—The look and feel of the Enterra GIS client module.

# Cartographic Object Visualization Using WPF

—Roman Dudarev, Ph.D. Engineering

Because of the widespread use of GPS, GLONASS, and other technologies to position objects on the Earth surface, we have a unique opportunity to create digital mapping systems that can enhance the day-to-day functioning of an organization without considerable expense. Examples are vehicle tracking systems employed by public transportation agencies and medical facilities, as well as the GPS navigators used in cars. All these systems have the same basic feature: a cartographic client module. Our module, Enterra GIS, which was developed in C# .NET 3.5 WPF, displays an interactive map which can be

used to track a given object. The module is part of a new vehicle control system developed by our company for organizations that need to be able to locate their vehicles in real time.

This article focuses on issues concerning the display of cartographic objects. I use as an example a simple geo-information system with two levels of scaling, which, while it has no visible display areas and no search feature, does offer the option of moving the map and obtaining information on the object. Issues related to the vehicles' movements and the display of the vehicles on the map are not specifically discussed here. Instead, I

focus on data storage, choosing the right visual format, and means of optimizing the map.

## Data Warehousing

One of the most widely used formats for storing cartographic information is currently the ESRI shapefile (<http://en.wikipedia.org/wiki/Shapefile>). However, shapefile are not the best format for object searches and criteria-based selections. To obviate the problem, developers in third-party companies have attempted to build special index files for such purposes, such SharpMap (<http://www.codeplex>).

WPF = Windows Presentation Foundation, a graphical subsystem used for developing more powerful and flexible interfacing.

Position	DBMS
1	MSSQL 2008 on the dedicated Server
2	MSSQL 2008 on the one workstation with a Client
3	DBF
4	MSSQL CE 3.5
5	Firebird Embedded
6	SQLite

Table 1. DBMS comparison by averaged performance.

com/SharpMap). This freeware can execute all the actions necessary for working with geo-data.

One problem exists, though. Tests showed that SharpMap's capacity for consumed memory is insufficient, affecting the performance library functions. Further, because the component is covered by an LGPL license, which requires that source code is made public when changes are made, there has been no commercial development of SharpMap.

To improve the performance of shape-files (.shp files) for such actions as map scaling, object movement, and object search, we experimented with importing .shp files into DBMS (Database Management System). The Microsoft SQL Server Compact Edition 3.5 we used as DBMS is part of the VisualStudio 2008 delivery package which has no limitations on distribution (<http://www.microsoft.com/sqlserver/2008/en/us/compact.aspx>). This high-performance DBMS can easily be substituted by other high-performance DBMS, such as Microsoft SQL Server 2008 (<http://www.microsoft.com/sqlserver/2008/en/us/enterprise.aspx>).

The DBMS-based solution is very flexible, enabling developers to segment the software product for in-network and out-of-network work with geospatial information. For users this means that for some mapping needs, they will be able to utilize local databases, while for others, they will be able to access maps stored on the cen-

tral server—which has a number of advantages, i.e. easy update.

Once the .shp files were imported into DBMS, and the information about the object was displayed on the map, an experiment was conducted to compare the efficiency of different embedded DBMS. This was an easy task, because all the requests for objects selection were made on standart SQL. We tested Firebird, SQLite, DBF, MSSQL, but not Oracle DBMS.

The results are given in Table 1.

In the remainder of this article we will discuss the implementation on MSSQL CE 3.5. The structure of the system implemented is given in Table 2.

Name	Data type	Description
Id	Int	Primary key
Name	Nvarchar(255)	Object name
Description	Nvarchar(255)	Object description
Data	Image	Data
ObjectType	TinyInt	Object type
ElementType	TinyInt	Type of object elements
Zoom	TinyInt	Scale
ObjectId	Int	Object identifier
ElementId	Int	Object element identifier
Complex	Bit	Complex element feature

Table 2. Structure of the MSSQL CE 3.5 database.

Zooming on a map is accomplished through object selection executed as follows:

```
select ObjectId, ObjectType,
Name as ObjectName, ElementType, Data, Description,
Complex, ElementId from
MapObject where @Zoom =
Zoom order by ObjectType
```

Note that additional functions can be implemented by changing the structure shown in Table 2 and possibly also adding a few new tables. In the real application, Enterra GIS, we have

five tables with three of them designated for object search. The following section describes the implementation of object visualization (of which zooming is part) using WPF.

### Choosing a Drawing Format

WPF uses vector graphics as a drawing format. Typically this means saving the data as an image accompanied by an instruction kit. The kit, which is sent to output subsystem, contains instructions on how the drawing should be completed using graphical primitives (i.e. line, curve, etc.). Using this method enables scaling without any loss in data quality.

There are two drawing functions in WPF: Shape and DrawingVisual objects.

Shape objects are represented as rectangles, ellipses, and other primitive shapes. These objects can be described using the xml markup language, and they support anti-aliasing and events processing, however, any time there is need to draw a large number of such objects, system performance is negatively affected.

The second function, DrawingVisual objects, can be used to draw images as well as text. However, this functionality does not support re-arranging and events. Additionally, VisualObjects can't be described in xml.

Terminology: C# = C"sharp"; .NET = a component of several Microsoft Windows systems; LGPL = Laser General Public License; MSDN = Microsoft Development Network.

A container (inherited from FrameworkElement class) needs to be created. This container, termed VisualCollection class, will store DrawingVisual objects. To be able to arrange containers as desired, two simple functions need to be overlaid, thus:

```
public class BaseRenderer : FrameworkElement
{
    //member for optimize
    protected VisualCollection ObjectChildrenList;
    protected override int VisualChildrenCount
    {
        get
        {
            return ObjectChildrenList.Count;
        }
    }
    protected override Visual GetVisualChild(int index)
    {
        return ObjectChildrenList[index];
    }
}
```

Created DrawingVisual objects should be placed into the ObjectChildrenList container so that the WPF graphical subsystem can execute tier drawing. Note that to maximize performance it is necessary to use the Add function and add the objects sequentially. Otherwise, the Insert function, for example, will recalculate the visual parent of all elements with an index higher than the one of insertion. The Remove function is analogous. That is why it is better to clear the container completely. This can be ensured by using any profiler, for example, jetBranche (<http://www.jetbrains.com/profiler/index.html>) or by reviewing the source code using, for instance, Reflector (<http://www.lutzroeder.com/dotnet/>).

MapDrawingVisual object is thus created first. The object is a descendant from the DrawingVisual object and has a link to the business-object of the map for executing the HitTesting procedure (hit check). There is also a style for object drawing, and when the geometry is created, the drawing is executed. The process is coded thus:

```
private void CreateDrawing(MapElement element, bool closed)
{
    MapDrawingVisual drawingVisual = new MapDrawingVisual();
    drawingVisual.MapObject = element.MapObject;
    DrawingContext drawingContext = drawingVisual.RenderOpen();
```

```
GeometryStyle style = StyleManager.
GetStyle(element);
    Geometry geometry = CreateBaseGeometry(element, closed);
    drawingContext.DrawGeometry(style.Brush, style.Pen, geometry);
    drawingContext.Close();
    AddDrawingVisual(drawingVisual);
}
private void AddDrawingVisual(DrawingVisual drawingVisual)
{
    ObjectChildrenList.Add(drawingVisual);
}
```

Drawing of simple graphical primitives is executed with the help of the high performing StreamGeometry class. This class is "frozen," which makes sense, because our map objects are not expected to be modified.

```
private static Geometry CreateGeometry(Point[] points, bool closed, bool freeze)
{
    Geometry geometry = new StreamGeometry();
    using (StreamGeometryContext ctx = ((StreamGeometry) geometry).Open())
    {
        ctx.BeginFigure(points[0], true, closed);
        ctx.PolyLineTo(points, true, false);
    }
    // Freeze the geometry (make it unmodifiable)
    // for additional performance benefits.
    if (freeze)
    {
        geometry.Freeze();
    }
    return geometry;
}
```

### Optimization Tools: Brushes, Pens, Text, and Object Movement

In our application, many different types of object need to be drawn: houses, rivers, roads, etc. Unfortunately, a simple solution which comes to mind (using one brush object for one object type), leads to a significant slowdown of the system. Thus we faced the challenge of finding an optimal method of creating brushes and pens that would increase performance. Creating a new brush right before it is used didn't lead to a significant increase in performance. Maximum effect was however reached when a brush or pen was used as a template to create copies for use with GetCurrentValueAsFrozen(). The simplified code describing this method reads:

```

public GeometryStyle Clone()
{
    return new GeometryStyle
    {
        Brush = (Brush)Brush.GetCurrentValueAsFrozen(),
        Pen = (Pen)Pen.GetCurrentValueAsFrozen()
    };
}

```

An important issue to note is the realization of object signatures. In our application, we signed objects in the background of a semitransparent rectangle. In the delimitation of this rectangle, the consideration of the amount of text as well as the width and height of the FormattedText object will however be non-optimal. To avoid repeated recalculation, text drawing needs to be executed first, followed by rectangle drawing and adding the created DrawingVisual objects to the collection of the visual objects in a specified order.

Map movement with the mouse is quite an easy task and in fact consists of the correct calculation of the TranslateTransform class parameters executing object shifts. By using transformations, a number of sophisticated effects can be created, such as changing the viewing angle and map rotation. But performance, again, leaves much to be desired.

We noticed substantial performance decrease when drawing lines, which is particularly troubling because one needs them to draw such cartographic objects as roads, rivers, region borders, etc. To speed up the drawing of such objects, we recommend assigning only integer numbers for pen thickness. We also found that when anti-aliasing is disabled, drawing performance decreases, hence the following coding:

```
RenderOptions.SetEdgeMode(this, EdgeMode.Aliased);
```

It's our hope that others may be able to explain these two rather anomalous system behaviors and recommend additional ways of increasing performance.

Performance profiling is an important tool. We used WFP Perforator (<http://windowsclient.net/wpf/perf/wpf-perf-tool.aspx#perforator>) to profile our WFP application, in addition to considering Microsoft recommendations found in MSDN.

### Hit testing

Hit testing for a given event is an easy task and can be implemented using a simple class, such as the one written below:

```

MapDrawingVisual:
public class MapDrawingVisual : DrawingVisual
{
    public MapObject MapObject;
}

```

The entire code for the hit testing procedure reads:

```

public MapInfo GetInfo(Point point)
{
    MapInfo info = new MapInfo();
    VisualTreeHelper.HitTest(Viewer, null,
        delegate(HitTestResult result)
        {
            if (result.VisualHit is MapDrawingVisual
                && ((MapDrawingVisual)result.VisualHit).MapObject
                != null)
            {
                MapObject mapObject =
                    ((MapDrawingVisual)result.VisualHit).MapObject;
                if (!String.IsNullOrEmpty(mapObject.Object-
                    Name))
                {
                    info.Name = mapObject.ObjectName;
                    info.Description = mapObject.Description;
                    return HitTestResultBehavior.Stop;
                }
            }
            // Stop the hit test enumeration of objects in the
            // visual tree.
            return HitTestResultBehavior.Continue;
        }
        , new PointHitTestParameters(point));
    return info;
}

```

### Summary

WPF simplifies application development. Animations, compositing, hit testing, and other complicated tasks can be executed with few lines of code. WPF is recommended when there is no need to sort a large number of visual objects. It is also the application to use when one wants to create interesting effects. In situations where many complex objects and lines need to be drawn there is likely to be trade-off with system performance, especially if the computer on which the application will run is not high-end. Our application will not work on Celeron 2000 for instance, which has GeForce 4 MX400 video card and 1 Gb memory, despite the fact that one can play Counter Strike and other rather complex games on this computer.

Dr. Dudarev will respond to comments on this article, the application, and the code used to describe its functionality at [Roman.Dudarev@enterra-inc.com](mailto:Roman.Dudarev@enterra-inc.com).