Using Overlapping Polygons

The visual aspects of dynamic feature location are the focus of a companion color plate entitled *Exploring District Services*. The geodata and their interaction in this sample SML Control Script are discussed here to help you further understand this example of dynamic feature location. You can then incorporate this kind of interactive geospatial analysis into your projects. As stated in the companion plate, displaying the schools assigned to every property location is only one of a myriad of possible applications for dynamic feature location. So where the words "school" or "attendance area" are used, substitute the feature name and area appropriate for your application.

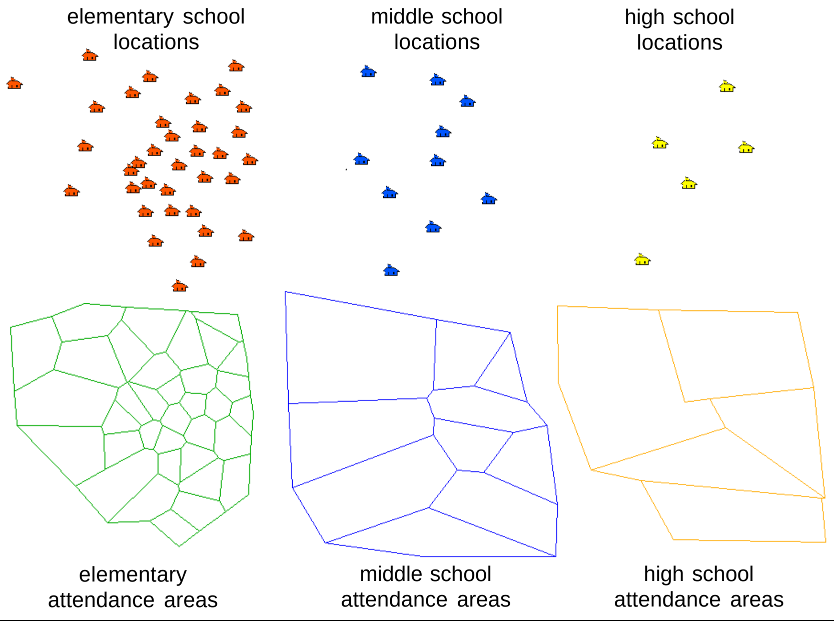
This sample application is a good example of interactive geospatial analysis whose speed and objective (reactive and automatic response) **require vector objects with full topology**. This task would be **difficult and slow to accomplish** with CAD or shape-oriented geodata. It uses seven coincident vector objects: three types of school attendance area polygon vectors (elementary, middle, and high schools, illustrated below), three corresponding school building point location vectors (illustrated below), and a land ownership parcel vector. In this example of dynamic feature location, none of the six school vector layers are directly displayed; they are used only by the Control Script. A 1-foot resolution color orthophoto mosaic lets the user visually locate any property parcel of interest. This image layer could be overlaid by other layers, such as a primary road network, to assist the user in locating a parcel.

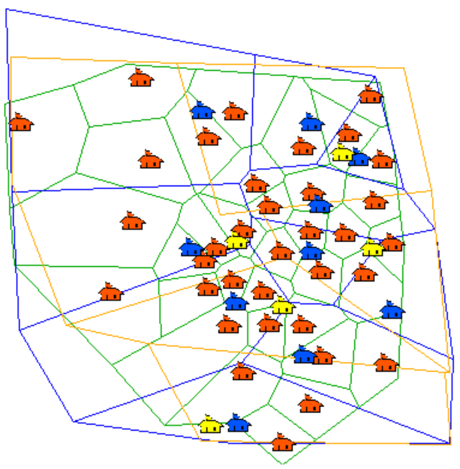
As the cursor is moved over the image in the View, the Control Script detects the parcel when the cursor hovers over it for 0.5 seconds and then displays the parcel boundary if the view scale is appropriate. It also confirms the address of the parcel by presenting it in a DataTip. In this example, the school parcel outlines are displayed by the Control Script at map scales larger than 1:50,000 and the parcel outline for the property beneath the cursor is drawn at scales greater than 1:15,000. Next the Control Script determines the location of the cursor in each of the three overlapping school attendance area polygons. (Smaller scale values mean larger map scales.) It then locates the corresponding school property in the parcels vector object, draws its outline if the map scale is appropriate, and accesses the three school building location vector objects to draw their symbology into the view next to their parcel outlines. All this happens automatically in the Control Script each time the cursor is moved without having to click or perform any other selection procedure as illustrated in the associated color plate noted above.

This example of dynamic feature location permits the user to zoom in and out and explore the underlying spatial relationships of a property of interest using the image layer, other reference overlays, and parcel boundaries. However, it could be modified to pan to, center on, and display these same dynamic feature location results for any specific parcel by incorporating the Property Finder Tool Script, which is described in the color plate of the same name.

Sample sections of the six school-related vector objects are presented below. The parcel boundary layer for this subarea is not shown because it is very complex at the scale illustrated. To provide the context for the actual operation of this example of dynamic feature location, Lincoln, Nebraska (population approximately 260,000) has 36 elementary schools, 11 middle schools, and six high schools. Each time the cursor hovers over a parcel in this sample data, the complete Control Script evaluation and associated display appear in about 0.5 seconds.

sample subset for illustration


elementary school locations
middle school locations
high school locations
elementary attendance areas
middle school attendance areas
high school attendance areas



The six school-related vector objects referenced by the Control Script are shown separately at the left and together above. The parcels vector object is too dense to be shown at this scale.

Partial Script for Exploring District Services (schools.sml)

see back of plate entitled *Exploring District Services* for more of this script

set offsets and map scales for drawing

```
numeric schoolPrclScl = 50000;
numeric mouselocPrclScl = 15000;
numeric labelOffset = 30;
numeric schoolSymOffset = 25;
class GEOREF vecGeoref;
class POINT2D offset;
```

function called to find attendance area polygons at cursor position

```
func FindPointInPoly(
        class POINT2D pointloc,
        class VECTOR vPoints,
        class VECTOR vPolys,
        class POINT2D cursorpos)
{
        numeric closepoly = FindClosestPoly(vPolys, cursorpos.x,
             cursorpos.y);
        numeric i;
        for(i=1; i<=vPoints.$info.NumPoints; i++) {
                numeric polyForPoint = FindClosestPoly(vPolys,
vPoints.Point[i].Internal.x, vPoints.Point[i].Internal.y);
                if(polyForPoint == closepoly) {
                        pointloc.x = vPoints.Point[i].Internal.x;
                        pointloc.y = vPoints.Point[i].Internal.y;
                        return 1;
                }
        }
        return 0;
}
```

function called to find points that fall in attendance area polygons at cursor location

```
func FindPointInPolyElemNum(
        class VECTOR vPoints,
        class VECTOR vPolys,
        class POINT2D cursorpos)
{
        numeric closepoly = FindClosestPoly(vPolys, cursorpos.x,
             cursorpos.y);
        numeric i;
        for(i=1; i<vPoints.$info.NumPoints; i++) {
                numeric polyForPoint = FindClosestPoly(vPolys,
vPoints.Point[i].Internal.x, vPoints.Point[i].Internal.y);
                if(polyForPoint == closepoly) {
                        return i;
                }
        }
        return -1;
}
```

procedure called the first time a GraphTip is activated

```
proc OnInitialize ()
{
        numeric SZ=5;
}
```

function called to determine line from cursor location to school point

```
func getTriOppositeLength(numeric adj, numeric opp, numeric outAdj) {
        if(adj==0) return 0;
        return outAdj * opp / adj;
}
```

function called to determine if school point is outside View

```
func getBinaryOutcode(class POINT2D p, class RECT r) {
    numeric opcode = 0;
    if(p.x < r.x1) opcode = opcode | 1;
    if(p.x > r.x2) opcode = opcode | 2;
    if(p.y > r.y2) opcode = opcode | 4;
    if(p.y < r.y1) opcode = opcode | 8;
    return opcode;
}
```

function called to find point location or, if not in View, intersection of line from cursor to school point

```
func class POINT2D clipPoint(class POINT2D p, class POINT2D
            mouseloc, class RECT ext) {
    class POINT2D out = p;
    numeric outcode = getBinaryOutcode(out, ext);
    while(outcode != 0) {
        class POINT2D temp = out;
        if(outcode & 1 == 1) {
            temp.x = ext.x1;
            temp.y = mouseloc.y + getTriOppositeLength(mouseloc.x -
               out.x, mouseloc.y - out.y, ext.x1 - mouseloc.x);
        } else if(outcode & 2 == 2) {
            temp.x = ext.x2;
            temp.y = mouseloc.y + getTriOppositeLength(mouseloc.x -
               out.x, mouseloc.y - out.y, ext.x2 - mouseloc.x);
        }   else if(outcode & 4 == 4) {
            temp.x = mouseloc.x + getTriOppositeLength(mouseloc.y -
               out.y, mouseloc.x - out.x, ext.y2 - mouseloc.y);
            temp.y = ext.y2;
        } else if(outcode & 8 == 8) {
            temp.x = mouseloc.x + getTriOppositeLength(mouseloc.y -
               out.y, mouseloc.x - out.x, ext.y1 - mouseloc.y);
            temp.y = ext.y1;
        }
        out = temp;
        outcode = getBinaryOutcode(out, ext);
    }
    return out;
}
```

function called to calculate angle of line from cursor to View intersection for school points not in View

```
func calcAngle(class POINT2D a, class POINT2D b) {
    numeric x = b.x - a.x;
    numeric y = b.y - a.y;
    numeric angle = acos(x / sqrt(x^2 + y^2));
    if(y < 0) return PI - angle;
    return angle - PI;
}
```

procedure called to draw line at calculated angle

```
proc drawAngledLine(class GC gc, numeric x, numeric y, numeric angle,
            numeric len) {
    numeric a = len * cos(angle);
        numeric b = len * sin(angle);
        gc.MoveTo(x, y);
        gc.DrawTo(x-a, y-b);
}
proc drawAngledPoint(class GC gc, numeric x, numeric y, numeric angle,
             numeric len) {
        numeric a = len * cos(angle);
        numeric b = len * sin(angle);
        gc.DrawPoint(x-a, y-b);
                }
```