In Reply Refer To: June 15, 1992
Mail Stop 445

Memorandum

To: Tim Gauslin, ISD

From: Doug Nebert, WRD

Subject: WAIS enhancements to support spatial query

I have had some discussions and thought about how to incorporate spatial indexing into WAIS and have some feedback to share with you. Here is my initial note to wais-talk:

At the U.S. Geological Survey we would like to be able to manage reference to digital spatial data sets that have an irregular extent and would like to be able to specify an irregular polygon outline (series of coordinates) for a search area in which one or more of our spatial data sets might fall. We could do this with proprietary geographic information software (GIS) but would rather have this capability available to the wider WAIS-using public so they can come and discover what data we have in their area of interest, appraise its documentation, via WAIS, perhaps view a snapshot of the data via a GIF file, then actually retrieve the spatial data sets of interest across the Internet!

Background:

A data set (document) referenced on a server could have an extent, described in a set of bounding coordinates -- a rectangle or a n-gon.

A user may wish to query for data present at a given point location or within a selection region -- again possibly a rectangle, but could be a many-sided polygon defined by a series of x,y coordinates.

The problem I see is that the unstructured form of a text document indexed by WAIS doesn't lend itself to storing a series of coordinates in a directed chain that define a border. There could be four coordinates or hundreds that define the outline of the data set. If we could limit the definition of a target data set to be a bounding rectangle, then each map document could have the following attributes with the set values stored in

parentheses or after an equal sign:

minimum_latitude(45.00)
minimum_longitude(-85.00)
maximum_latitude(46.00)
maximum_longitude(-87.00)

And we could further constrain the query to be initially a single point...
But still we need the semantic support to specify that this is not a literal
string match to be searched for but rather that it falls within the range
of the four document attributes described above. Something like the following
could be entered in the "Tell Me About" pane along with a theme like
"hydrography" to search for:

point(45.435,-86.202) <- defining a search point's lat and long
region(45.435,-86.202;47.000,-86.720;46.564,-87.020;44.881,-86.501)

where the point() would have to fall within the min-max box of any given
document's extent, or the region would either intersect or overlap the
range of min-max coordinates. Then we could further complicate things by
having each data set have a complex chain of bounding coordinates... but
let's keep the initial problem simple -- point in rectangle.

How can the WAIS query system support the query of a specified value in
a range rather than a literal text match? Can some kind of information be
stored with a new indexable data type (e.g. -t map) that the search algorithm
on everyone's WAIS client can forward and servers can adjudicate? There
would seem to be a need to do be able to query specific attributes of a
data set in context, in addition to free-text search, to more quickly locate
desired data sets...

The ability to perform queries within a range or even polygon intersections
seems the perfect kind of intensive operation one would want to execute in
real time -- the kind of operation perfectly suited to a Thinking Machine ;)

What ideas do you WAIS wizards have for amending the query model to permit
this kind of spatial query?

From Simon Spero at Think.Com:

Although the queries you describe cannot be handled by the existing wais
query type, but they fit easily into the Z39.50 model. Z39.50 allows you
to specify a bunch of attributes, and a term to which those attributes apply;
for instance, you could specify the following query: 'inside' 'bounding box'

'lat-long-coordinates' :: { {20,30},{30,40} }.

From Jim Fullton at UNC:

It looks like there are several different problems here. If you want to allow any wais client to access the data, you will have to impose some sort of structure on the query the user enters. Or, you could build some intelligence into the search engine (as opposed to the server) that:

a) checks to see if the query is formatted
b) if it is, passes it to the search engine
c) if it isn't, tries to extract useful information from the query and format it into a suitable form for the search engine.

This is what we did with our prototype astronomical image servers. Our underlying engines were both free-text and relational database. If the query was formatted, it was dispatched with out change, otherwise, the server picked out common terms like OBJECT, OBSERVATORY, RA, DEC, DEWAR, etc and tried to associate nearby terms with those markers. It worked reasonably well, but would have been a lot better if we had put more effort into it.

On the other hand, you could distribute clients that force the user to properly define queries. This is useful for researchers, but the stuff I mentioned above needs to be incorporated for other users.

It seems to me that some combination of these two methods is the best choice. You must have a customized server that controls appropriate search engines and can handle unformatted queries, but you can distribute custom clients to researchers.

The issues thes two raise are pertinent. First, WAIS is supposed to use Z39.50 protocol. If so, then why doesn't the wais query type communicate boolean operations? Is it just so that TMC could build a fast but oversimple search engine? UNC appears to have started context-based string searching and that might be a place to begin both the indexing and the searching modifications we need to access data by extent.

Required modifications to WAIS:

waisindex: I would prefer to see some format that begins to conform with Z39.50 for lack of a better syntax. On the source end, we can define the detected input stream to be anything we are looking for that references the latitude and longitude. The NASA DIF already supports such a bounding field, but so does NEDRES in a different format. I suppose you could search for specific spatial references within the -type portion of irbuild and extract accordingly.

1. Search for strings in source containing the bounding box for each entry with spatial extent (example from DIF):

```
Group:  Coverage
  Minimum_Latitude:     20
  Maximum_Latitude:     50
  Minimum_Longitude:    -132
  Maximum_Longitude:    -62
```

2. Build a new file associated with the index ending in sourcename.bdy (for spatial boundary). You may want to make this file use a hash structure so the entries can be sorted on input. Make this file a binary format file as five tokens in a string with the following format:

```
identity of the source
minimum_latitude
maximum_latitude
minimum_longitude
maximum_longitude
```

3. Sort or index the new file on each of the range values so that when a query comes in those records that fall within the minimum and maximum values will be found quickly.

4. For entries that have a complex polygonal boundary (not a rectangle) follow the preceding steps, including building an index of the bounds of the outline to limit additional processing of point-in-polygon to only those target polygons that at least fall in the correct x and y ranges -- no sense in seeing if you are in every bounding polygon in the index...

5. For complex polygons, extract the chain of coordinates that bound the entry assuming that if the last coordinate pair is not the same as the first, you add it for closure. Determine what the bounding box is for this entry and place it in the sourcename.bdy file. Count the number of coordinate pairs there are in the bounding area as you read them. Write the identity of the entry and the coordinate pairs to a sourcename.sdx (spatial index) file, indexed or sorted by the source identity. This could take the form of a series of records, or as a variable-length string of coordinates, prefaced with the number of coordinates to follow.

The indexing would now be complete.

waisserver: The WAIS server process would need to know about these two new index files and how to process and serve successful entries back to the query. This is where the point-in-polygon and polygon overlap routines would need to be called. A format for the receiving query would have to be established so that the server process could pass control to a couple of spatial processing programs to satisfy a part of the query. Some have suggested that the format be in Z39.50 form. I am not partial to a given format except that it should be English-like and intuitive and not be subject to mis-interpretation by the parser.

A possible format for query at a point and query in an overlap polygon could be:

at { 45.000, -110.45 }
    -or-
in { { 45.000, -110.45 },{ 46.330,-109.58 },{ 46.852,-112.020 } }

This query structure would need to be provided for in both waisserver and xwaisq.

1. The server process would need to identify if the phrase constructs listed above exist in the incoming query. If they do then branch into one of the two following directions:

a. The string includes "at {" and includes two coordinates (real or integer) separated by a comma and/or space and terminated with a "}". Check coordinates against a theoretical limit of latitude and longitude (-90 < lat < 90) and (-180 < long < 180) assuming latitude to be the first token and longitude to be the second. If the number exceed values, reject search and return control to text search program.

Open sourcename.bdy file and compare point values of lat and long against the ranges in each line, flagging all those document identities that fall within the x range and y range. If a sourcename.sdx file exists (meaning there are complex bounded areas as targets) then use the marked target documents as a list to run the point in polygon against. Flag verified targets and return to text searching. Close sourcename.bdy (and sourcename.sdx) files.

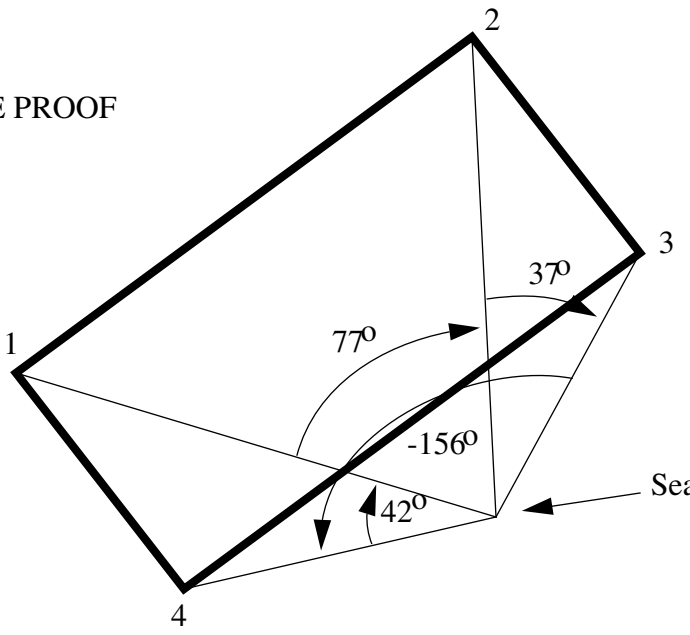
b. The string contains "in {" and includes n number of coordinate pairs within braces, each element and each pair separated by commas and/or spaces and closed by a "}". Check each pair for valid ranges and proceed. If any coordinate falls outside the region, reject search and return control to text search program.

Open sourcename.bdy file and identify entries which overlap each of the ranges. If a sourcename.sdx file exists (meaning there are complex bounded areas as targets) then use the marked target documents as a list to run the polygon overlap against. Flag verified targets and return to text searching. Close sourcename.bdy (and sourcename.sdx) files.

2. Point-in-polygon processing:

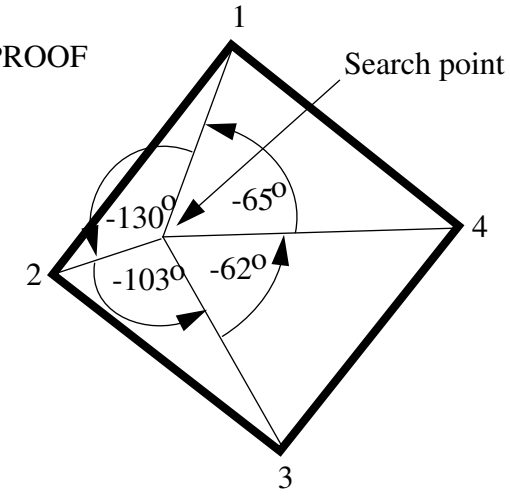
a. Open sourcename.sdx file
b. Move to first/next entry that has an extent overlap with search point
c. Read in the number of coordinate pairs to work through
d. For each "chord" or point-to-point pair that defines the perimeter, calculate the angle between the first point, the search point (vertex or angle point), and the second point. Place this angle value (positive or negative) into a summed variable and continue.
e. The sum of the angles measured from each point in sequence as positive (clockwise) or negative (counter-clockwise) is equal to 360 (or -360) if the point is on the boundary or inside the boundary, or zero if it is outside, as illustrated in the following diagram:

OUTSIDE PROOF



77
+37
-156
+42
0
(outside)

INSIDE PROOF



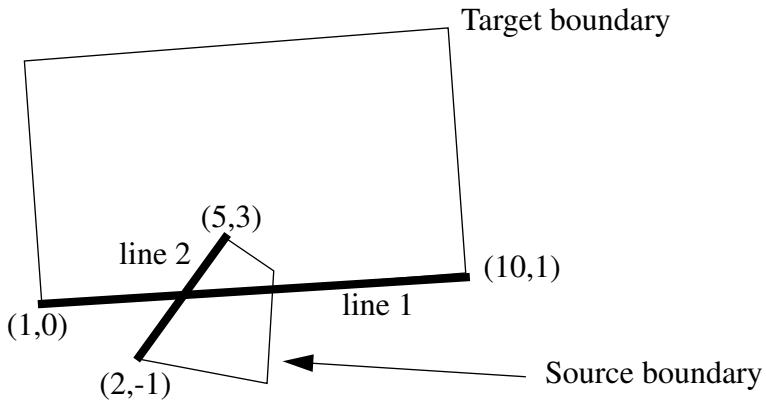
-130
-103
-62
-65
-360
(inside)

Note: bounding polygon does not need to be clockwise or counter-clockwise. By taking the absolute value of the sum the result is the same.

3. Polygon overlap processing:

a. Open sourcename.sdx file
b. Load the query bounding coordinates into a 2-D array
c. Identify a bounding box that contains the query bounding coords as they are loaded in array
d. Move to first/next entry that has an extent overlap with search polygon
e. Determine if any line segment from the query coordinates intersects any line segment of the bounding region of each indexed data set:

Determine the equation of each pair of line segments, one from target, one from query
Increment through all target segments:



Equation for a line segment $y = mx + b$ $b = y_1 - \frac{\text{delta y}}{\text{delta x}}\ x_1$

$x = \frac{y - b}{m}$ $m_1x + b_1 = m_2x + b_2$

where $m = \frac{\text{delta y}}{\text{delta x}}$ $\frac{y - b1}{m1} = \frac{y - b2}{m2}$

Line 1

$y = \frac{(1-0)}{10-1}x + b$

$y = 1/9x + b$

$b = 0 - 1/9(1)$

$b = -1/9$

$y = 1/9x - 1/9$

Line 2

$y = \frac{(3+1)}{5-2}x + b$

$y = 4/3x + b$

$b = 3 - 4/3(5)$

$b = 3 - 6.66667$

$b = -3.66667$

$y = 4/3x - 3.66667$

Solve for x value:

$1/9x - 1/9 = 4/3x - 3.6667$

$x = 2.909$

Solve for y value:

$\frac{y + 1/9}{1/9} = \frac{y + 3.6667}{4/3}$ $y = 0.43$

To avoid syntactical confusion and promote interactive feedback, an index map of the US could be prepared with state outlines as a backdrop and points could be picked from the GIF image with a translation between the row-column of the image and the lat long of the position. This could assist the user in filling in either the “at” or “in” arguments of the “Tell me about...” line. This would really be a client (xwaisq) feature.