TkEMA FOR MAPS AND VECTORS

Ted Nolan

SRI International
4210 Columbia Rd, Suite 5A
Augusta, GA 30907
ted@erg.sri.com

ABSTRACT

This paper presents several External Model Access (EMA) utilities built on top of the author's TkEMA package. These utilities create OPNET maps from DCW outline and bit-mapped data, display OPNET maps outside OPNET, and print output vector files from the command line. The paper briefly describes the motivation for the TkEMA package as a whole and for each utility in turn, with explanatory illustrations and, where practical, short examples of code.

TkEMA

Mil3[*] provides a C-based EMA library with OPNET. This library provides a programmatic interface for reading and writing OPNET models. Tool Control Language (Tcl) is a high-level scripting language developed by Dr. John Ousterhout and made freely available for all major platforms. Tk is a cross-platform GUI extension to Tcl; the two are usually considered together. Since Tcl/Tk is designed to be extensible at the C level, it is a natural step to wrap the OPNET EMA library as Tcl commands. I have done so, and call the resulting package TkEMA. (For a fuller if somewhat dated description of TkEMA, see my paper "Tcl Me EMA" in the OPNETWORK '98 proceedings [Nolan 1998].)

OPNET MAPS

When setting up a simulation, especially one that involves radio or other distance-sensitive models, it is desirable to place the simulation's nodes and subnetworks on a map background. Accordingly, OPNET supplies a number of predefined maps that can be used for this purpose.

OPNET map models are stored in Cartographic DataSet (CDS) files, and are searched for along the standard list of directories specified in mod_dirs. CDS files have overall model attributes dealing with the grid to be displayed with the map, and a number of EMA objects specifying lines and cartographic objects such as cities.

Obviously a map must specify geographic positions. CDS files use degrees of latitude and longitude (lat/lon) to do this. Although it has been common practice to divide a geographic degree into 60 minutes and each of those minutes into 60 seconds, CDS files use decimal degrees. This means that only one floating-point (double) number is required rather than three integers, which makes computations much easier.

CREATING OPNET MAPS FROM DCW DATA

If no Mil3-supplied map meets the user's needs, he/she can use EMA to create one. This process requires the user to supply map and geographic information in an electronic format.

One source for such information is the Digital Chart of the World (DCW) server maintained at Pennsylvania State University; the data is available on line at http://www.maproom.psu.edu/dcw. The DCW server allows the user to specify a country (or other political entity) and then download an outline of it as a text file. (Other formats with more information are available). These outline files consist of multiple sequences of lat/lon pairs that, when considered as a sequence of lines, outline the entity in question.

Here, for example, is a portion of the data defining the U.S. state of Wyoming (chosen because its outline is very square and geometrically "clean"):

```
wyoming
1
        -107.500000         43.246983
        -109.841698         45.011566
        -109.822357         45.010571
        -109.804932         45.012745
        .....
        .....
        .....
        -110.000000         45.010296
        -109.864288         45.011402
        -109.841698         45.011566
END
END
```

[*]Mil3: Mil3, Inc. All product and company names mentioned in this document are the trademarks of their respective holders.

The first line identifies the entity in question (Wyoming). The second line indicates that information for the first polygon describing the entity follows (Wyoming requires only one polygon). The third line gives the lat/lon coordinates of the center of mass of the polygon. The next lines, up to the first END line, give longitudes and latitudes for points along the edge of the polygon. The final END indicates the end of the data.

This information is in a very programmer-friendly format; hence, it is easy to write a short TkEMA program called MAP Create (mapc) that builds OPNET CDS files from it. It is not possible to list mapc legibly in this paper, due to column width constraints (it is available on the FTP site http://www.scriptics.com); however, its basic algorithm is to read each set of points and create a CDS line object containing it. When the CDS model is written, the new map appears in OPNET's list of maps, just like those supplied with OPNET. Figure 1 shows an OPNET map of Wyoming created from the outline files with the following command:

```
mapc wyoming2pts.txt
```

As might be expected, the visual result is rather undistinguished, but then the beauty of Wyoming is inside its boundaries, not of them.

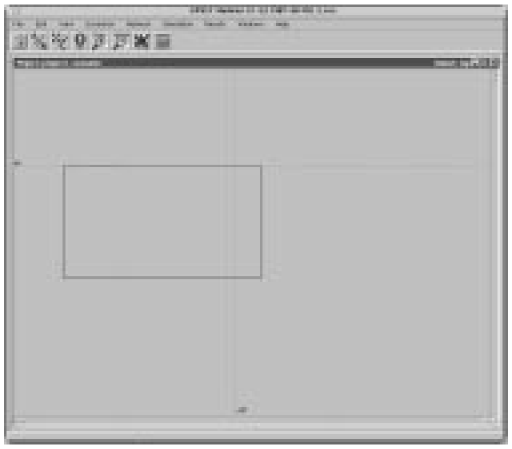

Figure 1. OPNET Map of Wyoming, Created from DCW Data

CREATING OPNET MAPS FROM BITMAP DATA

OPNET maps are displayed by means of vector graphics; aside from icons used to display cartographic objects, the basic drawing unit is a line, not a pixel. This works well for defining political boundaries, contour lines, and the like, but does not fulfill many people's expectations that a map will look like what they see when they open an atlas. Such traditional maps are available in electronic format; they are stored in bit-mapped file formats and are pixel rather than vector based. At first glance, it would seem impossible to use pixel-based maps in OPNET, but this is not quite the case.

Pixel-based maps can indeed be used because pixels are not perfect, dimensionless mathematical points, but actually occupy an area, both physically on the screen and notionally in the realm of latitudes and longitudes. In fact, pixels are just short, stubby lines.

To use a pixel-based map in OPNET, it is necessary to have accurate geographical information for pixel positions. Thus, the map that was scanned to produce the bit-mapped file (or the algorithm that generated the bit map) must be geographically accurate—there is probably no way to use a map drawn to direct guests to a birthday party. Secondly, there must be a way to deduce a geographical lat/lon coordinate from an x/y pixel coordinate. One way is to use the concept of tie points and pixel scales. A tie point is an x/y coordinate with a known matching lat/lon coordinate, and a pixel scale is a floating point number indicating how many degrees of latitude or longitude to add or subtract for a one-pixel movement on the x- or y-axis. With these two pieces of information, lat/lon coordinates can be assigned to every pixel in a map image.

I should insert a note of caution here; so far we have been assuming that the bit map in question is already represented in terms of latitude and longitude. In many cases, geographically accurate maps have been projected, and pixels represent distances in meters (or other linear units) rather than degrees. Map projection is a complicated process in which I am not at all adept. However, in some cases it may be possible to perform an inverse projection on each pixel to take it back into the lat/lon domain.

For a specific example of dealing with bit-mapped data, consider the electronic maps produced by the National Imagery and Mapping Agency (NIMA). NIMA distributes scanned versions of defense-related paper maps on CD-ROM in a number of formats (such as ADRG and CADRG). These formats are somewhat arcane, and it is hard to write programs for them; but NIMA also distributes a free software package, called NIMA MUSE,[*] that runs on several UNIX and NT platforms. NIMA MUSE provides a tool, called the raster importer, that can be used to select part of a NIMA ADRG or CADRG map by location, extent, and resolution, and save it in a variety of bit-mapped file formats.

[*]NIMA MUSE is available at http://www.nima.mil:80/geospatial/ SW_TOOLS/ NIMAMUSE.

For our purposes, the most useful of these formats is GEOTIFF, an extension to the Tagged Image File Format (TIFF). GEOTIFF stores geographic location information along with the actual pixels of an image. Accessing both the geographic and bitmap data is easy with the basic TIFF library (ftp://ftp.sgi.com/graphics/tiff/tiff-v3.4beta037-tar.gz) and the GEOTIFF extensions (ftp://ftpmcmc.cr.usgs.gov/release/geotiff/jpl_mirror/code/libgeotiff.1.02.tar.gz).

With this software, three new commands can be added to TkEMA: *Geotiff_Read*, *Geotiff_Destroy*, and *Geotiff_Cdsscan*. As might be expected, the *Geotiff_Read* command reads an image from a GEOTIFF file produced by NIMA MUSE (of the many possible parameters in GEOTIFF files, *Geotiff_Read* implements only a few); and *Geotiff_Destroy* frees the resources consumed by that image.

Somewhat more interesting is the *Geotiff_Cdsscan* command. *Geotiff_Cdsscan* scans the bitmap image from left to right and top to bottom. On each scan line, it looks for runs of pixels that are the same color. There may, of course, be as many “length one” runs as there are pixels, but the limited number of colors used in NIMA maps makes it quite likely that there will be a significant number of longer runs. For each run, Geotiff_Cdsscan creates an OPNET CDS *line* object with start coordinates given by the first pixel in the run, and end coordinates given by the last pixel in the run. The bitmapped image is thus rendered into a collection of vectors in an OPNET CDS model.

This process works surprisingly well, but the user should be aware of several issues. First, there is the issue of color. OPNET uses a color scheme that allows 64 colors specified with 2-bit red, green, and blue components. 64 colors are more than enough to represent NIMA maps; they are not, however, the *right* 64 colors. The TIFF software uses 24-bit pixels internally; I use a scheme that maps each color component down to 2 bits, such that levels 0–63 map to 0, levels 64–127 map to 1, levels 128–191 map to 2, and levels 192–255 map to 3. This scheme is imperfect but seems generally adequate. Secondly, there is no free lunch. When a bit-mapped peg is hammered into a vector hole, the disk space needed to store a map increases (in the worst case, each pixel consumes 36 bytes + overhead), as does the RAM required to keep track of the display list. Thirdly, there are some odd color interactions with icons dropped onto such a map. Still, the technique may be useful in some cases. The same algorithm could be applied to other bitmapped formats, as long as some geographic information is known. In cases where only the locations of specific tie points are known and uniform pixel scales are not guaranteed, interpolation may still produce an adequate result.

Here is a short TkEMA program that reads an OPNET-supplied map and merges a GEOTIFF file into it:

```
#!/usr/local/bin/tclsh8.0
#
# Merge a Geotiff file given on
# the command line into an Opnet MAP.
# Write it to the specified name
#
# Usage:
# map_merge geotiff in_map out_map
#

lappend auto_path ../tkema/lib
lappend auto_path ../tkgt/lib
package require Tkema
package require Tkgt

#
# Get the map file name
# set infile [lindex $argv 0]
set in_map  [lindex $argv 1]
set outfile [lindex $argv 2]

Ema_Init

set model_id \
    [Ema_Model_Read \
    MOD_CARTO $in_map]

set tiff_id [Geotiff_Read $infile]
Geotiff_Cdsscan $model_id $tiff_id
Ema_Model_Write $model_id $outfile
exit
```

If we run this program as

```
map_merge blackwell.tiff \
      namerica blackwell ,
```

it will merge the given GEOTIFF file (which happens to depict an area around the Blackwell Mountains of Texas) into the OPNET-supplied map of North America, and save the result as *blackwell.cds*.

Once this is done, the *blackwell* map appears in the OPNET tool and can be used in OPNET projects. Figure 2 shows a subnetwork icon positioned on an airstrip in the *blackwell* map.

Now, about that birthday party map—while it is true that there is not much prospect of using it in OPNET as an accurate map, that is perhaps not the whole truth. If all we are interested in is using an image as wallpaper behind a network diagram, then geographic accuracy is

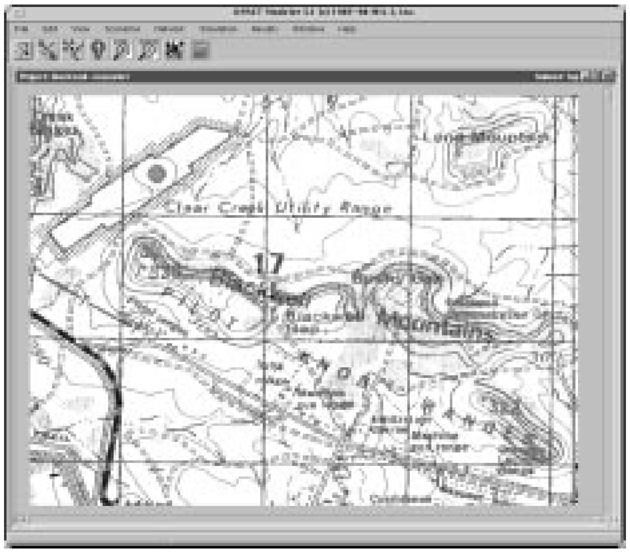

Figure 2. OPNET Map of Blackwell Mountain Area, Created from ADRG Data

not necessary. To handle this case, I have defined a command called *Geotiff_Image_Read*. This command simulates reading a GEOTIFF image from an ordinary TIFF file with no georeferencing information. This can be useful in cases where the georeferencing information for a map is contained in a README file and not in the image file itself. The format is

```
Geotiff_Image_Read \
   tiff_file lon lat \
   xscale yscale ,
```

where lon and lat give the lat/lon coordinates for the (0,0) pixel, and xscale and yscale give the number of degrees represented by each pixel. Now, consider a simple bit-mapped image that is not a map of any sort. In this case, the user simply makes up values for the position and scaling parameters. On a good day, with plenty of RAM, the results can be rather pleasing, as in Figure 3.

VIEWING OPNET MAPS

We have talked about importing maps into CDS format so they can be viewed under OPNET; conversely, it sometimes may be useful to view CDS maps outside OPNET. Given the GUI capabilities inherent in Tk, this is not especially difficult. The main stumbling block, to which I alluded earlier, is the issue of projection.

CDS map files store positions in terms of latitude and longitude, whereas the Tk canvas widget works in terms of x/y pixel positions. Representing points from an essentially spherical object (the Earth) on a flat surface (paper) is a problem that has occupied cartographers for centuries. There are many different types of projection,

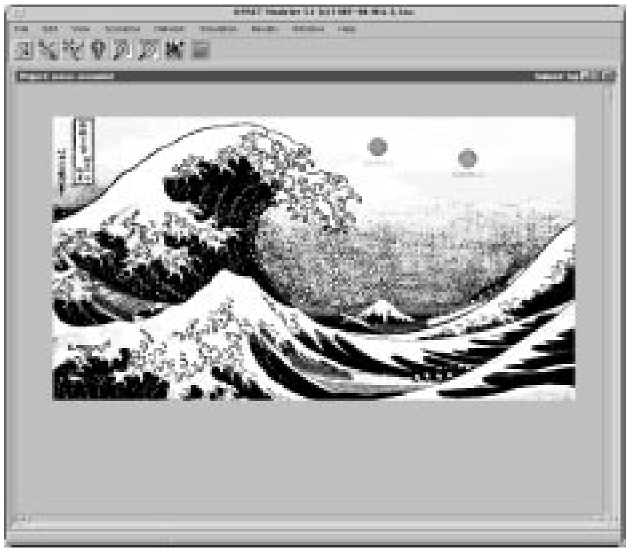

Figure 3. OPNET Map Created from Arbitrary Bit-Mapped Image (Hokusai: “The Great Wave off Kanagawa”)

and I cannot claim to understand the mathematical basis for any of them. Fortunately, there are software packages to handle these issues. For previewing CDS maps, I chose to use the General Cartographic Transformation Package (GCTP) from the United States Geologic Service (USGS) (ftp://edcftp.cr.usgs.gov/pub/software/gctpc/gctpc20.tar.Z) and the Mercator projection.

For the purposes of displaying CDS maps, it is only necessary to define two new Tcl commands, *Geo_Init*, and *Geo_To_Merc*. The first initializes the GCTP package, and the second translates a point specified in lat/lon to a point specified in x/y. With these two new commands, it is relatively easy to write a TkEMA script, called *map_load*, that reads and displays CDS maps.

When invoked as

```
map_load world  ,
```

map_load reads the OPNET supplied world map and displays it as shown in Fiugre 4. Having conquered the world, we now leave maps behind.

OPNET VECTOR FILES

It is common for a running simulation to write results into OPNET vector files. These files can be read with OPNET’s analysis facilities to produce various types of graphical output. However, they are in a binary format that cannot be directly read by other programs.

Often a person may be running a simulation on behalf of a principal who does not have OPNET and wishes to do his/her own analysis. In this case and many others, it is advantageous to be able to export OPNET vector files to a text-based format.

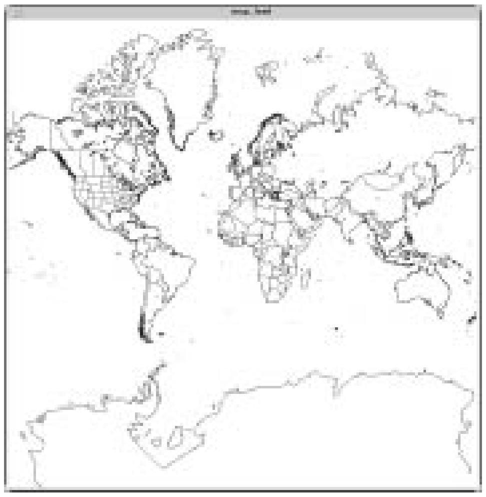

Figure 4. OPNET World Map Displayed by *map_load*

Vector files in OPNET 5 are (or can be) laid out in an hierarchical manner. The top object has child objects that may have other children. The actual vector data is stored in leaf nodes. I have written a TkEMA script, called *vector_print*, that extracts individual vectors from a vector file by name. The convention I have adopted is that first the vector file name (without the .ov) is given, then the names of each element in the vector hierarchy, and finally the ordinate label for the desired vector.

For instance, suppose we use the output vector file from Lesson 2 in the OPNET tutorial and give the command

```
vector_print \
  Lesson_2_ref-no_back_util \
  “New_York <-> Washington_DC [0]” \
  point-to-point \
  “utilization <--” > vec.out .
```

Then the vector with the ordinate label “utilization <--” from the element “point-to-point” under the element “New_York <-> Washington_DC [0]” is stored in the file *vec.out*. (Note that the arguments with spaces and UNIX shell metacharacters must be quoted.)

The data is output as x/y pairs such as these:

```
   112.0 1.35925722225
   119.0 1.40713656673
   126.0 1.39071279509
   133.0 1.59961897783
   140.0 1.26681383887
   147.0 1.4949627804
   154.0 1.6223538
   161.0 1.3734747349
```

These x/y pairs can be directly utilized by common plotting tools such as *gnuplot*.

Figure 5 shows the same vector plotted in both the OPNET analysis tool (lower left) and gnuplot (upper right).

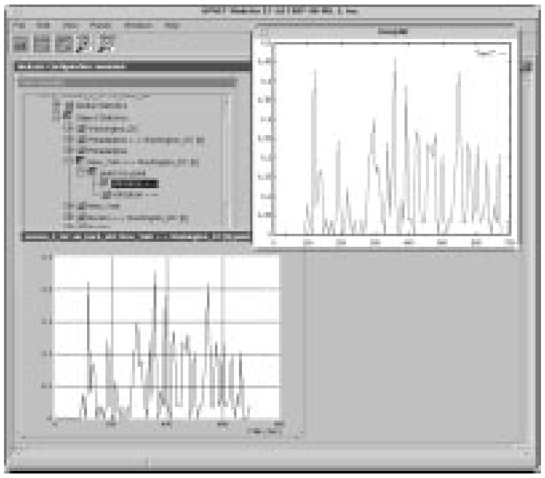

Figure 5. The Same Vector, Plotted with Both OPNET and *gnuplot*

Although I have not done so, it should also be possible to go in the opposite direction: a user who has data created in another tool and wants to use the analysis and graphing functionality of OPNET, should find it fairly easy to write a script to create an OPNET vector file from that data.

AVAILABILITY

Tcl and Tk are available at http://www.scriptics.com.

I have used version 8.0 for TkEMA, but I imagine 8.1 should work as well.

The core TkEMA package and all the scripts mentioned in this paper are available at ftp://ftp.erg.sri.com/pub/people/ted/tkema99.tar.gz.

The core TkEMA package works on Solaris and NT platforms. (NT binaries for both a TkEMA enabled “tclsh” and “wish” are supplied) The package has been updated from the release described in “Tcl Me EMA” [Nolan 1998]. Some of the mapping extensions mentioned in this paper have, as of this writing, not been compiled on NT.

REFERENCES

Mil3. 1998. *Chapter EMA: External Model Access*, Mil3, Inc., Washington, D.C.

Nolan, Ted, 1998, “Tcl Me EMA,” *Proc. OPNETWORK ’98*, Washington, D.C. (May).

Ousterhout, J. 1994. *Tcl and the Tk Toolkit*, Addison-Wesley, Reading, Massachusetts.

Welch, B. 1997. *Practical Programming in Tcl and Tk: 2/e*, Prentice Hall PTR, Upper Saddle River, New Jersey.