Geographical Trace Route V1.0

Name: Darren Brown
Date: 20/2/1994
Unit: Computing Project 394

Supervisors: Craig Farrell and Mike Shultze.

Table of Contents

List of Figures

List of Tables

Abstract

Traceroute is a tool that determines the path that a packet would take from the current host to some artibrary destination, cleverly capitalising on a feature of the IP and ICMP protocols to achieve this. It has proved very useful for determining routing inefficiencies and errors.

Output from traceroute however is text and as such is hard to visualise into the logical path taken. The application developed during the course of this project 'Geographical Traceroute V1.0' (or GeoTraceMan to fit in with rest of the NetMan suite of programs) is an attempt to improve the usefulness of Traceroutes results.

What has been implemented is an X windows based application that plots the logical route between the hosts onto a map of the world (Mercator projection). The geographical locations of the hosts in the route are determined using a multi layered resolution scheme.

Preface

Here we go, empty platitudes to people who don't deserve them (kidding). I'd like to thank John Krivitsky for his help with mapping, Greg Barron for Framemaker advice and everyone in the Curtin Computing Science department who answered any one of my many questions. Finally a special thankyou to my supervisors Craig Farrell and Mike Schulze (the ringmasters of the big-top). They both helped out with a combination of advice, abuse and code, were very patient and thankfully don't call me a clown as much as they used to.

1.0 Introduction

In an entity as vast and as complex as the TCP/IP Internet, it is not surprising that things can and do go wrong. A common problem is incorrect routing. Comer (1991) defines routing as "the process of choosing a path over which to send packets" and a router as "any computer making such a choice". The results of bad routing can range from extra delays, to the complete loss of packets. Traceroute (Jacobson 1988) is a tool that helps distinguish if there is a routing problem, and if so at what point the problem is occurring. An outline of how this program achieves this is discussed in the background section of this report.

An unfortunate aspect of Traceroute, is its text output. While the information is useful, it leaves the user to envision the route in terms of real world locations. Graphical images can, if used correctly, convey much more information.

This report describes an application that was developed for the unit Computing Science 453 at Curtin University of Technology called Geographical Traceroute (also called GeoTraceMan or GTM). The program was developed under the UNIX operating system using the 'C' programming language and is X windows based.

2.0 Aims of the Project

There were originally two primary goals for this project:

1) To develop an application that uses the Traceroute program to generate a trace, resolves output host locations into geographical locations and plots the results onto a map of the world. The software developed was to have the following attributes, it would:

- Be X windows based.
- Operate in real time - the route should be plotted as the information from Traceroute arrives.
- Present a map of sufficient detail to give the user an accurate representation of coordinates, yet one of small enough size that the drawing of the map should not incur unreasonable delays.
- Have the ability to offer additional map features, for example rivers and political boundaries.

2) To gain experience in developing a program of size and complexity similar to those in industry.

It was soon realised that before this project could be started, even before decisions regarding its direction could be made, a large amount of knowledge had to be ingested.

Specifically the areas that had to be studied were:

- X11 programming;

- the vogl (Very Ordinary Graphics Library) functions;
- Traceroute, routing, the Internet and its protocols
- mapping and map databases.

As this is an inordinate amount of learning compared to what is normally required for an undergraduate project (basic X Windows programming alone is normally taught in a semester unit) this gaining of knowledge constituted a major goal in itself.

3.0 Development Environment

The development of this project took place on Sun SPARC Stations (IPC and IPX models) running SunOS 4.1.3 and X windows X11R5. RCS (Revision control system) (Tichy 1991), was used to control the source code version updates.

4.0 Background

4.1 The Internet

4.1.1 Concept of an Internet

An internet is a number of connected networks that may consist of many different hardware architectures. Comer's (1991) definition is a large virtual network that is comprised of a number of packet switched networks connected by gateways. Communication between these networks (in fact between the hosts inside the networks) is achieved by the institution of standards known as protocols.

The TCP/IP Internet is the ultimate expression of an internet, spanning the globe and being comprised of many major networks including:

- The United State's Department of Defence (DOD) and National Science Foundation (NSF) networks;
- Australia's AARNET;
- Europe's Janet.

Comer (1991) estimates that by 1995 the Internet will be comprised of some 100000 networks.

4.1.2 Internet Protocols

4.1.2.1 The TCP/IP Suite

In the United States, the Defence Advanced Research Projects Agency (DARPA) funded the research of (Comer 1991) "a set of network standards that specify the details of how computers communicate, as well as a set of conventions for interconnecting networks and routing traffic". This set of standards is used by the Internet, and the protocols used are known as the TCP/IP suite.

This acronym can be decomposed into TCP (Transmission Control Protocol) and IP (Internet Protocol) which are (Comer 1991) “the names of its two main standards.” Those protocols that this project is concerned with are briefly discussed below.

4.1.2.2 IP

The IP datagram is the outside shell of all packets that enter the internet. Packets entering the Internet are treated as independent of each other (connectionless) and delivery is not guaranteed (unreliable).

IP uses a labelling system called an IP address, this unique 32 bit identifier allows the IP datagrams to be routed across the Internet. There are five different classes of IP addresses, three of which are used for individual host identification (A, B and C), one for multicasting (D)(not relevant to this project), and a final class that is not currently utilised (E). The make-up of the class A, B and C addresses is shown in *Figure 1.*

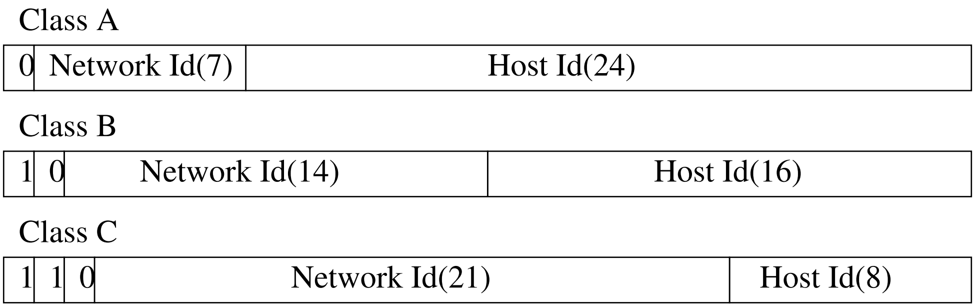

Figure 1: Class A, B, and C IP addresses.

A field called Time to Live (TTL) resides in the header information of the IP protocol, and ensures that datagrams that cannot be delivered are eventually removed from the system. This is done by every host decrementing the TTL value of a datagram when it receives it and then reroutes it. A datagram enters the Internet with a positive TTL value and when this value reaches zero the datagram is discarded. Delays in processing by a particular host cause further reductions in TTL values.

4.1.2.3 UDP

A UDP (User Datagram Protocol) packet is one type that may reside in a IP datagram, another is TCP (mentioned previously). This protocol allows packets to be sent to a specific port on a machine once it has been routed to the destination host by the IP address. Once again this service is connectionless and unreliable. The header information of a UDP datagram contains in its first 32 bits the source and destination port numbers (each 16 bits long).

4.1.2.4 ICMP

ICMP (Internet Control Message Protocol) packets are another type that may be contained in a IP datagram. This protocol (Comer 1991) “allows gateways to send error or control messages to other gateways or hosts”. Comer also says that while ICMP was originally to be sent only by gateways, any host can send an ICMP message to any other host. User programs (super-user only) can receive ICMP messages by opening a raw socket.

There are a number of ICMP messages that may be sent, but there are two that are particularly important to this project:

- Destination Unreachable - This message is sent for a number of different circumstances that mean that an IP datagram cannot be delivered. Traceroute uses a particular subtype of this message called Port Unreachable. This refers to a datagram reaching the right host and finding that it cannot contact the specified port.

- Time Exceeded for a Datagram - Sent to the source host of a IP datagram when an intermediate gateway decreases the datagrams TTL value to zero or less and discards it.

4.2 Traceroute

Traceroute is a program that is used to discover the route that is taken from a particular IP address, to another. It was written by Van Jacobsen with the idea of the algorithm attributed to Steve Deering. Traceroute depends on some of the features of the protocols that were described above to get the route information. A series of IP datagrams are sent containing UDP packets addressed to a port number that isn’t normally used. Below is a basic version of Traceroute’s algorithm that only holds true for the very simple case (i.e. if there are no other error conditions).

```
for time = 0 to maximum hosts {
	for packet = 0 to number_of_packets_to_send {
		Send a IP datagram (TTL = time), containing UDP addressed to unused socket.
		if no reply in a given time
			notify timeout
		elseif ICMP Time Exceeded
			return host name
		else if ICMP Destination Unreachable (Port Unreachable) {
			return host name
			at end of route.
		}
		else
			error
	}
}
```

As can be seen, Traceroute relies on ICMP replies to the UDP packets that are sent: Time Exceeded messages from intermediate hosts; Destination Unreachable (Port Unreachable) from the destination host. Unfortunately there are many other errors that can occur, as Rose (1991) says "not all implementations of even IP, ICMP and UDP operate correctly". ICMP messages themselves are not even guaranteed to arrive as they are just another packet encapsulated in an unreliable IP datagram.

Due to the accessing of ICMP replies, as stated above the program must open a raw socket. This requires that either the user is root (or is effectively root through chmod, setuid) whilst executing the program.

4.3 The CIA World Map Databank (CWMDII)

4.3.1 Description

This database is publicly available from the United States Government and consists of 5719617 Cobol records which describe the Earth's coastlines, rivers and political boundaries. In its original form it requires 130 Mbytes of storage space. Fortunately a compressed version has been created by Brian Reid (at Digital Equipment Corporation) which only occupies 15 Mbytes (Dellinger 1989), and it is this version that has been utilised for the project.

4.3.2 Composition of the Compressed Database

Three levels of segregation are used to distinguish a vector from its peers (vector refers to a connected set of points). Firstly they are broken down into continents, then secondly grouped by their feature class. Each continent occupies a separate directory and each feature class within that continent a separate file. So there are five continent directories each containing three files of the form:

- <Continent_Name>.cil - coastlines, islands etc;
- <Continent_Name>.riv - rivers, lakes etc;
- <Continent_Name>.pby - political boundaries.

The exception is North America which has an extra file that details its interstate borders called namer.bdy.

Within each file there is another criteria called rank that gives the vector's level of importance. In all cases a vector of rank one is most prominent, with prominence decreasing as rank increases. For example the rankings of the vectors in the coastline files are (Dellinger 1989):

Table 1: Coastline feature rankings

Rank	Feature Description
01	Coast, islands and lakes that appear on all maps
02	Additional major islands and lakes
03	Intermediate islands and lakes
04	Minor islands and lakes

Table 1: Coastline feature rankings

Rank	Feature Description
06	Intermittent major lakes
07	Intermittent minor lakes
08	Reefs
09	Salt pans -- major
10	Salt pans -- minor
13	Ice Shelves -- major
14	Ice Shelves -- minor
15	Glaciers

4.3.3 Justification for the Database Choice

A vector based map was not the only option available. Another avenue examined were bitmaps. The advantages of such a map are that it could be drawn very quickly and would occupy very little space both in storage and in memory. However as the trace points were required to correlate with the map coordinates, it was decided that a vector based map would be more accurate. Also governing this decision was the knowledge that a future version of the application may have different levels of magnification and alternate projections as features. Both of these would be much simpler to implement with a vector based map. In fact the zoom function was implemented during the course of this project, and the method is discussed later in this report.

Other public domain (important criteria) map databases were considered, but none had the complexity and ranking arrangements of CWMDII. Although it is more detailed than was originally considered necessary, it was chosen to allow for future developments. Another advantage was that its points were in northings and eastings which made coordinate overlaying very simple.

4.4 Mfil

This reasonably simple program extracts map files from CWMDII and was written by Ken Chin-Purcell of the Minnesota Supercomputer Center (Chin-Purcell 1992). Mfil translates the data from CWMDII into a very easy format on which to do further processing, the way the map is stored in memory and written out to a file is given below.

```
typedef struct MapData {
    longnseg;
    long*nstroke;
    long**line;
} MapData;
```

Nseg represents the total number of vectors that have been extracted, the next element of the structure is nseg longs that give the number of segments in each vector. Lastly the double pointer variable line contains the actual points themselves in degree seconds. These are stored in monotonic x y format stroke within segment. Mfil also features command line options, one of which specifies the maximum rank of the vectors to be extracted and two others that give the top left and bottom right clipping coordinates.

For this project, mfil was used to mutate the map data into an easy to understand format that was used as input for further processing.

4.5 X Windows

To quote Nye (1992) “The X Window System (or simply X) is a hardware and operating system independent windowing system.” This independence means that software written utilising X Windows is portable to many different architectures. Another advantage is that X is public domain, developed by MIT and Digital Equipment Corporation. The combination of these two factors has made X the standard for graphical workstation based applications.

A detailed explanation of the X windows system is not within the scope of this report, O’Reilly and associates publish a series of volumes on the subject that the interested reader is referred to. The implementation section of this report will explain how X was utilised in the application.

4.6 Vogl

The Vogl (Very Ordinary Graphics Library) was developed by Eric H. Echidna (serious?) at the University of Melbourne and is public domain (although the authors do make a “send beer” plea!). It is a library of ‘C’ subroutines that has been designed to be compatible with Silicon Graphics GL library, and was utilised in this project for drawing lines and text to the main display widget. Xlib calls could have been used instead but this library greatly simplified the procedure.

To allow vogl to link with X, a special call has to be made - vo_xt_window. This call sets the passed window id as the output device for vogl. Other parameters to this call include the size of the window. If these dimensions do change vogl can be notified by the use of the function vo_xt_window_size.

4.7 The Netman Project

This application is just one of many that are being developed at Curtin University covering different facets of network management and administration. Other tools include etherman, interman and packetman, all of which are available by anonymous ftp from marsh.cs.curtin.edu.au.

5.0 Implementation: Geographical Traceroute V1.0

5.1 Evolution of the Final Map Data Sets

5.1.1 Extraction from CWDII

In order to create the final maps that were used in the application, a number of procedures had to be carried out. Initially mfil (Chin-Purcell 1992) was used to extract the vectors from CWMDII. A problem was encountered initially because the byte ordering of the database was different to that of the host on which the extraction took place (SUN SPARC Station IPX). This was solved by bit manipulation of the integers and long integers as they were read in.

As was explained above, the database is segmented into files by continent and topographical feature. Thus a number of output files were created. To ensure that all of the vectors were extracted the clipping boundary parameters were disabled leaving only screening by vector rank active.

To display this map data, a very simple routine was written (from ideas taken from mfil's documentation) that simply reversed mfil's write procedure to read the data into memory and then used vogl to display the vectors. This routine was written to take a number of command line arguments, each being an extracted map name.

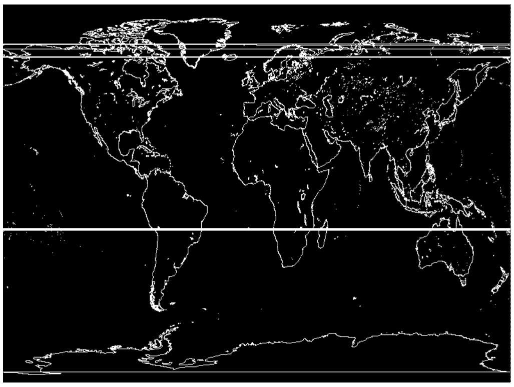

Figure 2: Original extracted map data

Clearly it can be seen that this map (*Figure 2.)* is of very high quality but there were some major problems.

1) Vectors that are in reality joined on the globe, but are at separate ends of the two dimensional map became horizontal lines transversing the display.

2) The density of data is much higher than what was required, with many vector segments mapping to the same pixels. A collary to this is that the map files were far too large, in total they occupied over 13.5 Mbytes of disk space just for the coastline files.

3) Another problem, but one that cannot be gleaned from examination of the diagram, is related to the second problem being an unacceptably large drawing time. On a Sun SPARCstation IPX with the map data contained in the local file system, the reading and drawing procedures together took 2:34.44 minutes.

The first problem is one that could have been easily solved, a heuristic could have been used to cut the offending vectors in the appropriate places. Unfortunately if the breaking of vectors was done during preprocessing, then when the map was rotated the problem would have resurfaced. Otherwise, if it was done dynamically during execution time, extra overheads would have been introduced (every pair of joined points requiring a test). Fortunately the problem was solved by a manipulation of the data that is discussed in a forthcoming section. The second and third problems are linked and could be solved by a reduction in the dataset size.

5.1.2 Initial Thinning Attempt: First and Last

At this stage, a trial was conducted based on using only the first and last points of each connected set of segments. The dataset produced was an improvement because it was very small and the production of the map was virtually instantaneous. Also at this point there was only one output data set for each feature class, as the vectors from all of the continents were merged. Conversely a horizontal line still crossed the map (*Figure 3.*) and the result was of extremely low quality, giving only a very vague representation of the Earth at best.

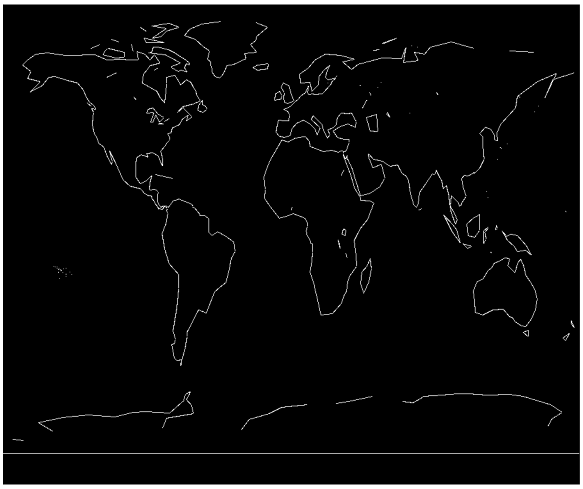

Figure 3: First and last points only

5.1.3 Thinning by Threshold

Next a different method of screening the vectors was trialed, and it was this method that was eventually utilised to create the final maps. This method involved thinning the dataset by testing each points distance from its predeccesor. If the point was over the threshold distance away it was included and it became the current point otherwise the point was ignored. The procedure being repeated for each remaining point in the vector. The diagram below (Figure 4.) gives an example of a vector before and after the thresholding procedure.

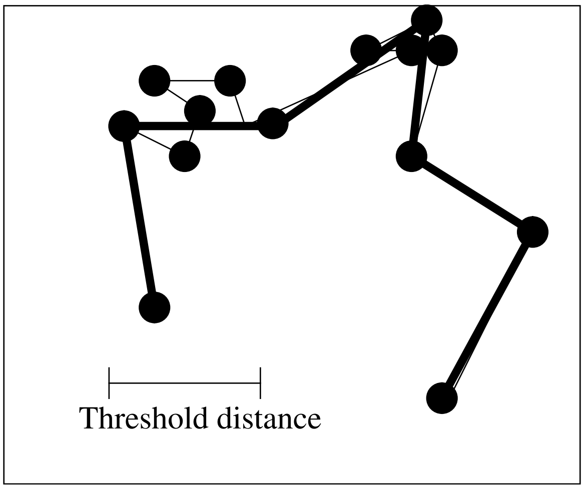

Figure 4: Example of threshold thinning

5.1.4 Segmentation of Map Into Regions

In order to reduce the amount of disk space and memory that the map occupied, it was segmented into a number of squares which were called regions. There were a number of reasons why this was considered beneficial:

- Horizontal lines that had crossed the map previously would be removed, as all vectors would be cut on region boundaries.
- Points inside regions would be represented by a combination of the region x,y indexes and the offsets inside the region themselves. This meant that each point occupied far less space. For example if the maximum region size was limited to a square of resolution 256 x 256 then each x, y pair would occupy two bytes rather sixteen.
- Clipping operations would be carried out on region boundaries. In situations where only part of the map is displayed, only those regions that are all or partially visible would be drawn. This greatly sped up the drawing of small areas of data intense maps.

5.1.5 Mappart

A program was finally developed called mappart, that used the region segmentation and threshold thinning techniques that are described above. This program processed the output maps from the mfil program and created the maps that are used by GeoTraceMan. Files that resulted from this program consisted of the regions themselves (in raster fashion) preceded by the number of regions in each dimension (side_size) and the resolution of these regions (region_range). Both of these last values being represented by integers. The regions themselves were of the following form:

```
typedef struct Region_Struct {
        int nvect;
        int *nseg;
        unsigned char **line;
}Region_Struct;
```

There is a great deal of similarity between this structure, and the one used in mfil. The main difference is that mfil's structure represents the whole map, whereas each region structure only represents a small portion. Another difference is that the sizes of the elements in the structure are smaller. Two unsigned chars are used instead of two long integers to represent a point. The number of vectors and the number of segments within those vectors are represented by integers rather than long integers.

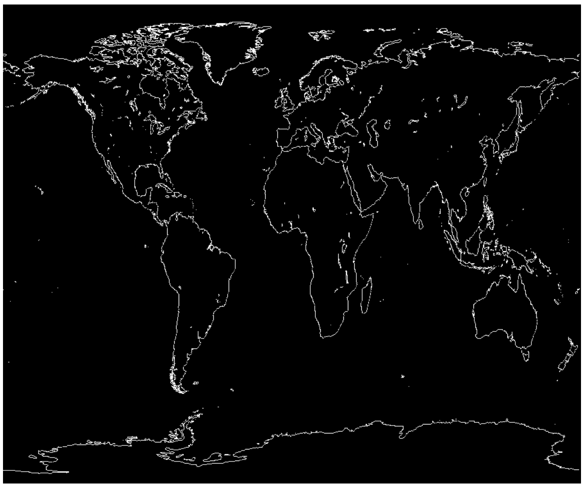

Figure 5: Current data set

An example is map is shown in *Figure 5.* This particular map had a side size of 4 (total of 16 regions) and a region range of 256. A threshold value of 1 was used., this number being relative to the new coordinate system which is 0 - 1023 in both dimensions. The new map was of equivalent quality to the original, did not have any horizontal lines crossing it and was drawn almost instantaneously. Disk space was also saved as the new coastline representation only occupied 76608 bytes rather than the 13.5 Mbytes quoted previously.

5.2 The X Interface

5.2.1 General Layout

The basic shell for the X section of the code was taken from the application *etherman* (Schulze 1992) which is part of the Netman Project mentioned earlier. However extensive changes had to be made as the two programs are quite different in purpose. What remained common however was the method by which the widgets were instantiated and realised, as well as the menuing system.

Appendix B is a table that describes GeoTraceMan's main widget heirachy. Careful choices had to be made as to which widget classes were utilised for each purpose, especially in terms of constraint widgets because each class of constraint widget has different rules for managing its children. Form class widgets were chosen to hold each of the main sections of the display, as they allow their children to set XawChain type resource values. These resources give the child the means to inform their parent of the size they must have. A breakdown of the graphical interface and the functions of each of the menus and buttons is detailed in the *Appendix A* Manual page. *Figure 6* below shows a overview image of the interface.

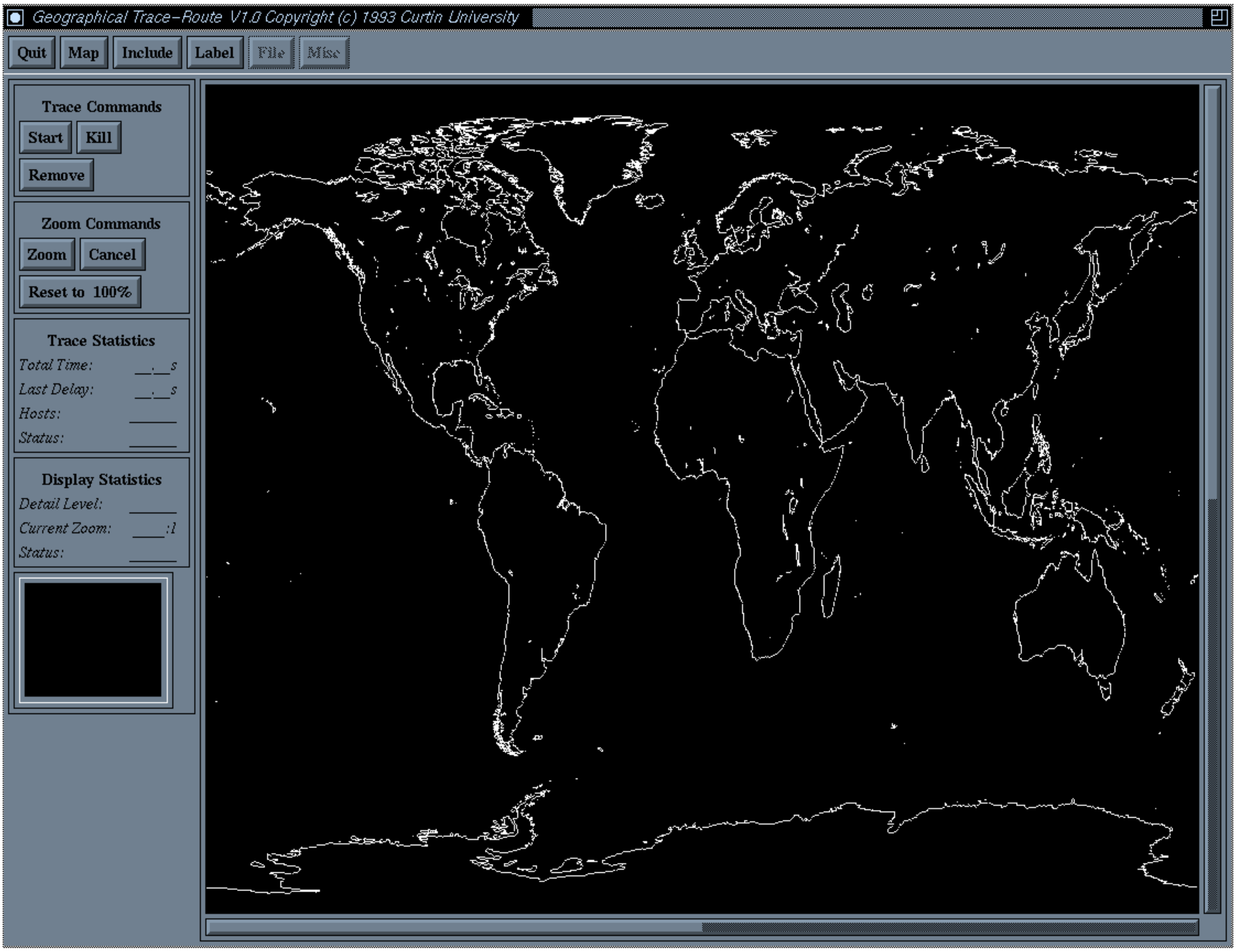

Figure 6: The GeoTraceMan interface

5.3 Map Functionality

5.3.1 Map Structures

To handle the large number of variables that dealt with the current state of the map being displayed, a structure was built to contain them. This structure was made global throughout the program as nearly every function needed to access it. Also special X functions called actions that were used cannot receive any application data as parameters. In order for these to access the map state it had to be made global.

Firstly a small structure contains the minimum and maximum visible coordinates and regions for a single dimension.

```
typedef struct Dimension_Attributes {
     long min_vis;
     long max_vis;
     int min_vis_reg;
     int max_vis_reg;
} Dimension_Attributes;
```

Two of these structures are built into the larger Map_State_Struct, which also contains information on the current maps that are in memory, as well as other information about the map options and configuration. Built into this structure is the ability to specify the current map detail level. GeoTraceMan allows multiple levels of map detail that are swapped in and out of memory as the magnification level is altered.

```
typedef struct Map_State_Struct{
     int detail; /* Level of current map detail */
     int side_size; /* Side size in regions of current map */
     int num_regions; /* Just the side size squared, but it doesn't take up much
room*/
     int region_range; /* Range per region */
     long max_range;/* Range over map */
     int  x_reg_offset; /* The number of regions the globe has been rotated - or +
*/
     unsigned char x_offset; /* The offset within the region that the globe has
been */
     float rotation_fraction; /* The percentage the map is to be rotated */
     float zoom; /* A value indicating how far the map has been zoomed in on */
     ZoomState_Type zoom_state;
     Boolean include_pdy; /* Include political boundaries */
     Boolean include_riv; /* Include rivers ? */
     Boolean include_cit; /* Include cities */
     Dimension_Attributes x;
     Dimension_Attributes y;
} Map_State_Struct;
```

5.3.2 Displaying the Map

There are a number of user events that require a change in the map being displayed:

- Exposure: A portion or all of the main display widget (canvas) has become visible when previously it was obscured. This event also happens at the applications initiation.
- Reconfiguration: A change in size as a result of the entire window pane being resized.
- Rotation: The world can be rotated either left or right.
- Zoom: The magnification level is changed.
- Include: Include options such as rivers, political boundaries and cities can be toggled.
- Labels: Trace host labels can consist of any combination of IP name, IP address, and the

hosts position in the route.

Exposure can be handled by a simple call to swapbuffers (a vogl function) which simply exchanges the front and back buffers that vogl has set up for video memory. All other events require that the map is redrawn, a general function written for this purpose called update_display. All the event handling function needs to do is to change the appropriate values in the Map_State_Struct, and call update_display. Include state changes are a special case, as the display only needs to be redrawn if a option is toggled to false.

Zooming was implemented by allowing the user to draw a bounding box on the display widget, and then transforming the given pixel references to the map coordinate system (relative to the current zoom level).Vogl makes drawing maps of different scales over the same area very easy, the function ortho2 sets a particular range over the x and y dimensions. When there is a configure event then the same range can be set over the new window area. In the case of a zoom then the selected area is stretched over the full display by the same call with appropriate parameters.

Multiple levels of map detail were implemented in order to keep the map display times within user tolerance levels. When the zoom level goes beyond certain points the function swap_maps frees the area of memory held by the current data sets and reads in the new ones. By cutting the data into spatial squares, denser maps were rendered more quickly because only those regions that were displayed in part or full were drawn. In the past X would have clipped all of the vectors that fell outside the boundaries greatly increasing the display time. Different levels of machines are catered for, as the swap thresholds are user configurable. If the user has a very slow machine they may prefer to keep the lower density level maps at higher magnifications.

The diagram below (*Figure 7.*) shows how this minimal approach to displaying regions works. In the example a map of four regions square contains an object (bold, non-dashed). The dotted bold square shows the area that is currently being displayed. In this situation only those regions shaded would be drawn.

Extra files that have the same structure as the coastline data contain the river and political boundary vectors. These files are swapped in and out of memory whenever coastline data is exchanged. Displaying them simply requires extra calls to the same display procedure already used. Cities reside in a file called cities.loc, and this include option simply puts all of the cities at the locations specified in the file. The file format is *city_name northing easting*.

Two variables represent the total rotational offset in the Map_State_Struct, one indicating the number of regions and the other the offset past that region. The user can select a rotation either to the left or right, the default rotation is currently 36 degrees. This presented a problem, when the whole map was displayed, as the current rotational offset didn’t have to fall on a region boundary. To solve this, columns of regions that were split were drawn twice with different x offsets.

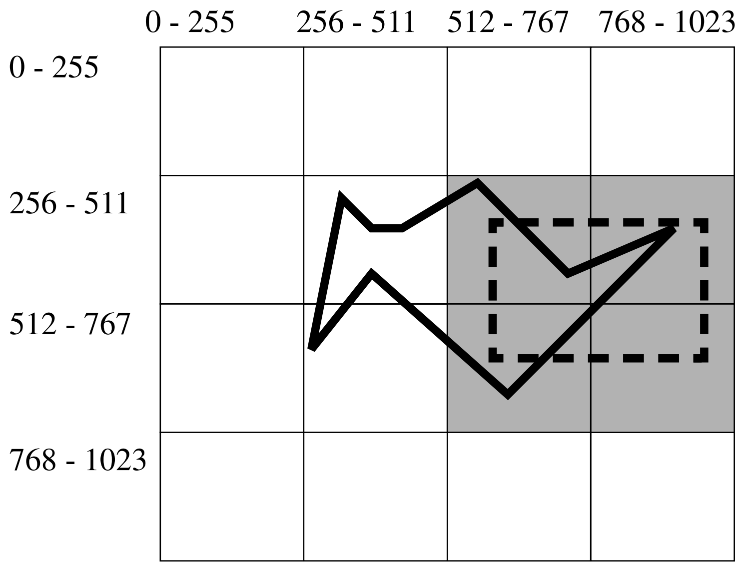

Figure 7: Partial map display

5.4 Trace Functionality

5.4.1 Trace Structures

For the same reasons given for the Map_State_Structure, the trace variables are also bound up into a large global structure. Traces_Struct in turn is constructed of a number of smaller structures. The base struct is called Host_Struct, each one containing the location, IP name and address of a host plus two boolean variables. Boolean resolved is an indication of whether the host has been found using the resolution scheme. Last_at_site indicates if the host is the last in the route at that geographical location.

```
typedef struct Host_Struct{
     char ipnum[Host_ID_Len];/* Hosts IP num as a string */
     char ipname[Host_ID_Len];/* Hosts IP name */
     Boolean resolved;/* Could the host be resolved or not */
     Boolean last_at_site;/* Is the host the last at the site */
     float northing; /* Hosts longitude position in degree-seconds */
     float easting;  /* Hosts longitude position in degree-seconds */
} Host_Struct;
```

The Trace_Stuct is designed to contain the hosts of a single trace plus variables relating to that trace. Hosts are stored as an array, the destination host being stored separately.

```
typedef struct Trace_Struct{
     int numhosts;   /* Number of gateways in the current trace */
     Boolean label_ipnum;/* What to label the hosts when they are displayed */
     Boolean label_ipname;
     Boolean label_position;
     int colour;     /* Colour to draw the current trace in */
     struct Host_Struct dest_host;/* Destination host in the trace */
     struct Host_Struct *hosts;/* Hosts in the trace */
```

```
        struct Trace_Struct *next_trace;/* Pointer to the next trace */
} Trace_Struct;
```

Currently the application only allows one trace at a time, however the option has been built in to extend this with the Traces_Struct below. It is this structure that is global, a trace is accessed through the pointer traces.

```
typedef struct Traces_Status_Struct{
        Boolean tracing;/* Is a route currently  being traced */
        int numtrace;/* How many traces are there */
        int tracepid;/* Process id for the current trace */
        int tracesd;/* Socket descriptor that is the connection to the trace process 
*/
        Trace_Struct *traces;/* Trace Structure pointer */
        Boolean visible[Max_Trace]; /* Which of the traces are to be displayed */
} Traces_Status_Struct;
```

5.4.2 Execution of Traceroute

Traceroute is spawned by use of the ‘C’ function fork, creating an exact copy of the current process. This new child process receives a return value of zero from the fork call, and it executes the Traceroute code. Back in the original process (parent) the child’s process id is returned from fork so this process continues to execute GeoTraceMan. In order for the two processes to communicate, a pipe between them had to be established. Before the fork call, the pipe is created returning two file descriptors. Traceroute’s code was modified to allow it to accept a descriptor as a parameter, its other parameter being the destination hosts IP address/name. The parent process closes the input file descriptor and the process running Traceroute the output one, leaving a one way channel between them.

Because the route is displayed as it is returned from Traceroute, the file descriptor in the parent must be continually checked for incoming bytes. At the same time the application must process user events. The X toolkit function XtAppAddWorkProc runs the passed function on a periodic basis until the function returns true. A ‘work procedure’ was added that checked the pipe for host names and if one existed ran the resolution and display schemes. Initially problems occurred because the reads to the file descriptor were blocking (waiting for input). A call to fcntl configured this file descriptor as non-blocking.

5.4.3 Resolving Host Locations

Host resolution has been implemented using a cache file approach. A text file called site.loc contains entries of the form:

```
IP_Address IP_Name Northing[ns] Easting[ew]
```

This file is parsed from top to bottom until either the site is found or the host is resolved. Whilst being rather simplistic, this resolution scheme does pick up a remarkable number of hosts. However if the number of entries in the cache file was to multiply exponentially, the resolution scheme would become very slow. Also it is not necessary to have every host on every network, a smarter system would resolve on a network rather than a host basis. A multi-level approach to resolution has been formulated and it will be included in the next version of GeoTraceMan. For details please see section 6.3 Improvement of the Resolution Scheme.

5.4.4 Displaying the Trace

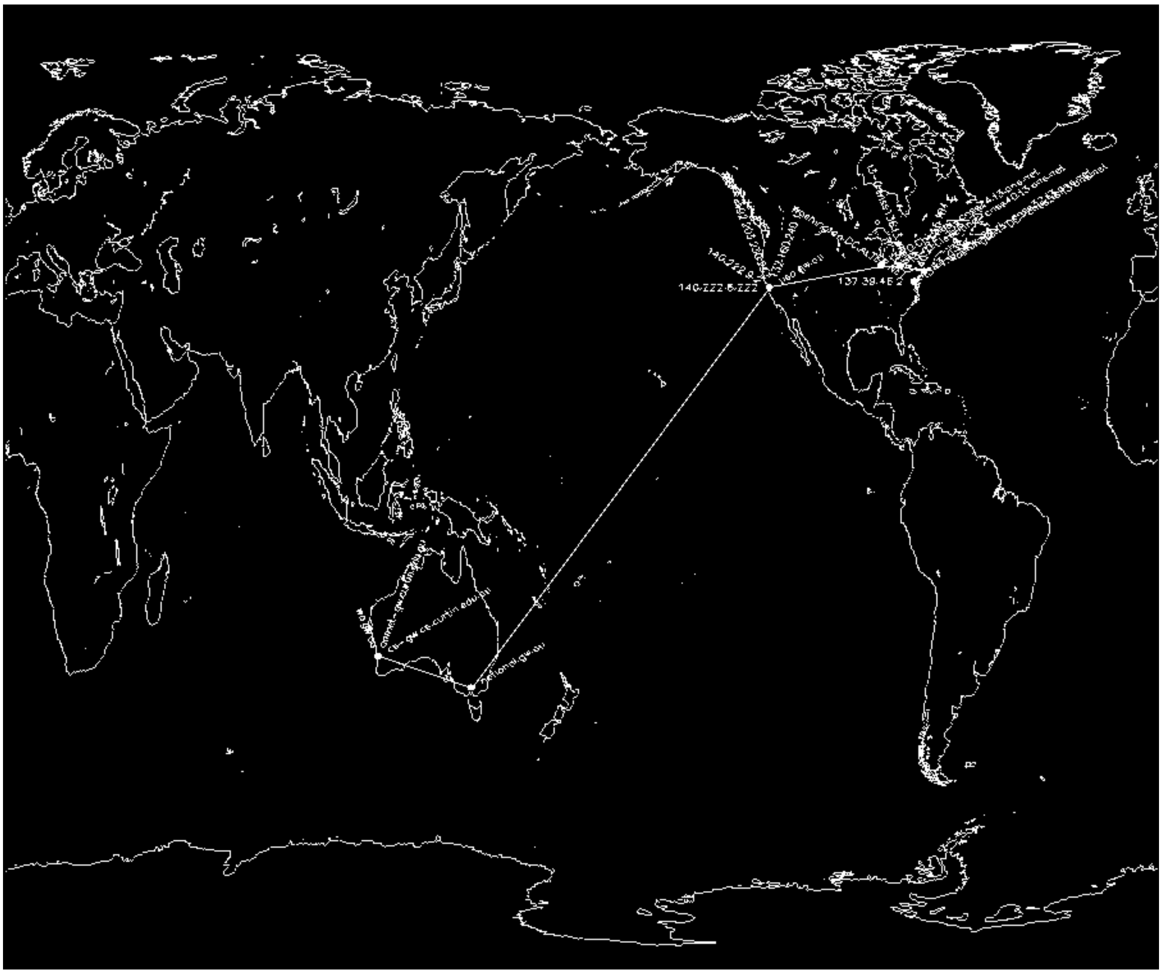

Figure 8: An example trace

Once the resolution scheme has attempted to resolve the host, it is displayed to the canvas. If the host could not be resolved it is simply placed at the same location as its predecessor. It would be hoped that the location of the first host in the route would be known, but if it isn't it is located at the center of the map. Whilst the location of the host is the same as the one before, the current host is drawn on the same point with its label at an ever increasing angle. When the current host is at a different location a link is drawn between the two sites and the text rotational angle is reset to zero. Also the previous host has it's last_at_site variable set to true. This continues until the route is complete. An example of a complete trace from cain.cs.curtin.edu.au to ftp.uu.net is shown in *Figure 8.* Note that the world position is different from that in *Figure 7.* due to rotation.

In the event of the map being redrawn, the trace must also be redisplayed. The procedure to do this is similar to that used to display one host. At each site the number of hosts is calculated and from this the text rotational angles. After each site is drawn a link is drawn to the next unless the current host is equal to the number of hosts in the route. In all cases the rotational angle of zero is not used to allow for a city name. *Figure 9.* shows part of the trace information that is displayed in the previous figure. A zoom operation has magnified the east coast of North America. At this level of magnification the current map consists of 256 x 256 regions each of resolution 256.

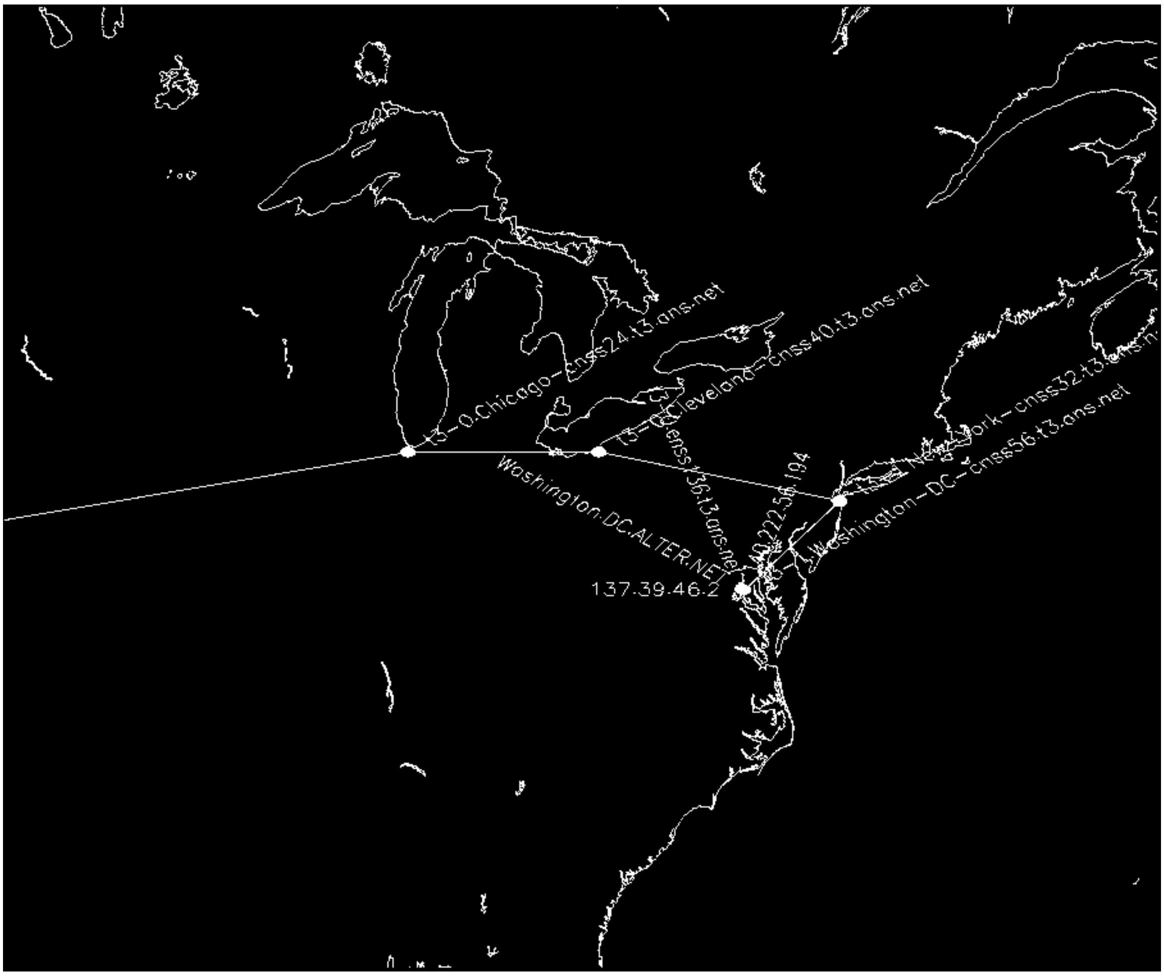

Figure 9: A magnified view of the east coast of North America

6.0 Future Possibilities: Additional Features and Improvements

6.1 Knowing When to Stop

With any program, it is always difficult to know when to stop coding. This project was (unfortunately) no exception. As many facilities and improvements that were added, there is probably just as many again that could be included. Some examples and improvement areas are given below.

6.2 General Improvement of the X Interface

While the GUI (graphical user interface) to GTM is quite clean and handles events such as exposure and resizing properly, more work could go into making it ‘just right’. Perfection is always subjective, and a lot of time can quite happily be wasted trying to find the right combination of resource values for the widgets. To me while the current setup is workable, there are some adjustments that I would like to make, including sizes of buttons etc.

6.3 Improvement of the Resolution Scheme

Resolution of hosts is currently quite successful even though it is a simple system. A number of other sources for host-location mappings exist. The DNS (Domain Name System) which (Rose 1991) "primarily provides mappings between host names and network addresses" could be utilised as its servers have the ability to hold northing and easting locations. Obviously this would be the last level in the scheme, as querying remote hosts would be quite slow. Unfortunately this facility of DNS is not widely known, so it is not used at many sites. Another possible source of data is the UUCP (UNIX to UNIX Copy Program) Map. This database is used to determine mail routes, and contains information (including geographical location) about many sites. If a method could be found of converting this database into IP address - location mappings, it would resolve almost every site.

An easier scheme to implement is a multi-tiered cache file system, similar to the single cache file currently used. This system was suggested by Craig Farrell, and would consist of the following three layers:

1) Local knowledge cache: Special local information could be included in this file such as hosts that are part of a network, but reside in a different location.

2) Known networks cache: Simply determining the location of the network gives a very good indication, as most networks are in one geographical location.

3) Known backbone hosts cache: For those hosts that cannot be resolved, at least if the backbone host can be resolved a general indication of the hosts real location can be gained.

6.4 Postscript Dump

One feature that would be very useful is the postscipt dump option. There are always going to be cases when the user would prefer a hard copy of the output. Trials were carried using a application to capture parts of the video memory, and then writing the output as postscript. Unfortunately the results were quite poor as information was lost through the conversion from vectors and characters to pixels and then to postscript.

6.5 Multiple Traces

Built into the code is the capacity to allow multiple traces to be handled at once. This could be quite a useful feature if combined with the trace file I/O option. For example, traces captured at different times (perhaps months apart) could be compared to see if there has been any changes in the routes. The number of traces is limited by the defined constant MaxTrace. This declaration can be found in the file *geotraceman.h* and is currently set to one.

A problem that would have to be solved is trace links and site labels being duplicated. Already with one trace there is some obscurement of names occurring when site locations are close together but not on the same point.

6.6 Alternate Map Projections

There is a plethora of different map projections, and each uses mathematics to somehow present the Earth on a flat piece of paper. For some application a particular map projection may be more applicable than another, as some are for displaying the whole planet, while others better represent small parts of the Earths surface. Perhaps different projections could be used at different levels of magnification. This avenue was not explored, as the author is not well versed in this area. Another possibility is a globe representation, this would be very impressive visually but would perhaps be quite slow. It would also require substantial knowledge of three dimensional graphics.

6.7 Panning Widget

Half implemented is a device that allows the user to pan across the map once they have zoomed in. The basis of this is the widget class called *Panner*. This widget is perfect for this application as it allows scrolling in two dimensions. Its function is to allow a foreground square to be selected with the mouse and moved across a background window usually holding a pixmap. This square would be resized as the magnification of the map is altered, giving the user a very good idea of what portion of the world they are currently viewing. Problems occurred using vogl routines to access two windows, so this feature was not fully implemented. For this case Xlib graphics calls may have to be used instead.

6.8 Trace, Map Statistics and Trace Delay Information

When the interface was discussed, mention was made of the statistics displays that are implemented but at present not working. A minimal amount of work could return real time statistics such as the total trace time and the number of hosts in the route. Provision has also been made for map statistics such as the current level of magnification and map detail.Status information for both the trace and map could also be added so as the user is always kept up to date rather than waiting for something to happen.

Another addition that could be made is the return of delay information from Traceroute. Delays could be drawn on the links between sites and on the labels of those hosts in the sites. This would require modification of the code linking Traceroute and GeoTraceMan, and perhaps a simple protocol connecting the two would be appropriate.

7.0 Conclusion

This project has produced an application (Geographical Traceroute V1.0) that meets all of the fairly open stipulations that were set at the start of the project. In addition features, such as zooming, that were not part of the original specification, have been implemented. As a by-product, a general purpose program called mappart has been written. It produces diskspace and memory efficient maps from the CIA World Bank II map database. Obviously there is still work to be done on GeoTraceMan, as described by the previous section which consisted of a number of additions and improvements that could be made. However the application has all of the basic functionality and is a good expandible base for future work.

I learnt a number of lessons from this project in areas including programming, time management and research. If I was to be set a project of similar complexity again, I would probably do everything differently but would also be able to do it faster and better. Thus the second aim has also been achieved.

8.0 References

[1] Black, U. D. (1992) TCP/IP and Related Protocols. McGraw-Hill: New York.

[2] Chin-Purcell, K. (1992) *Mfil* source code. Graphics, AHPCRC, Minnesota Supercomputer Ctr. Minnesota.

[3] Comer, D. E. (1991) Internetworking with TCP/IP. Volume 1; Principles, Protocols and Architectures. Prentice-Hall: London.

[4] Dellinger, J. (1989) README from the *domap* source files. Stanford University.

[5] Frame Technology Corporation. (1992) The *FrameMaker* Desktop Publisher. Frame Technology Corporation. San Jose: California.

[6] Jacobson, V. (1989) The *traceroute(8)* manual page.

[7] Kiethley, K. S. (1993) The 3D Athena Widgets (*Xaw3d*). Jet Propulsion Laboratories, NASA.

[8] Postel, J. (1981) Internet Control Message Protocol. Request for comments 792.

[9] Rose, M. T. (1991) The Simple Book: An Introduction to Management of TCP/IP-Based Internets. Prentice-Hall: Englewood Cliffs, New Jersey.

[10] Schulze, M. (1992) *Etherman* v1.0 source code. Department of Computer Science, Curtin University of Technology: Perth (Western Australia).

[11] Schulze, M., Benko, G. and C. Farrell (1993) Homebrew Network Monitoring: A Prelude to Network Management. Technical Report No. 9-1993, Department of Computer Science, Curtin University of Technology: Perth (Western Australia).

[12] Tichy, W. F. (1991) RCS — A System for Version Control. Department of Computer Sciences, Purdue University: West Lafayette (Indiana).

Appendix A — Manual Page

NAME

geotraceman

SYNOPSIS

geotraceman [**–xrm resource**]

DESCRIPTION

GeoTraceMan or (Geographical Traceroute) is an X based utility which uses the *Traceroute(8)* program to generate host locations, and then plots these locations onto a map of the Earth (Mercator projection). Due to *Traceroute's* need to open a raw socket, *GeoTraceMan* can only be used by root for route analysis (or by using setuid). All users can make use of the mapping functions.

OPTIONS

–xrm resource
Use this resource.

GRAPHICAL DISPLAY

GeoTraceman's display can be divided into a number of key areas:

- Menu Options
- Trace Command Buttons
- Map Command Buttons
- Statistics Boxes
- Panner and Scrollbars
- Main Display

Menu Options:

Quit - Causes the application to exit nicely

Map - Has three menu entries, the first two rotate the map left and right respectively and the last resets all of the map options

Include - Toggles the map include options: Rivers, Political Boundaries and Cities.

Label - Toggles the trace host labels. The label can be a combination of the hosts IP address, IP name and numerical position in the route.

File - This option is not implemented.

Miscellaneous - This menu is also not currently in use. It is envisioned that it will contain a postscipt option.

Trace Command Buttons:

Start - Spawns *Traceroute(8)*

Stop - Sends a signal to *Traceroute(8)* to stop the trace.

Remove - Deletes the trace from the display and frees the memory held by it.

Map Command Buttons:

Zoom - Processes the zoom on the current bounding box specified.

Cancel - Removes the bounding box from the display.

Reset to 100% - Resets zoom level to one.

Statistics Boxes: Currently not fully implemented.

Panners and Scrollbars: A choice has not yet been made as to which will be used in the application, so they are both included but not functional whilst in the development stage.

Main Display - This is where the world map and traces are displayed. A bounding box can be drawn by pressing the left mouse button within the display and releasing it once the area required is enclosed in a rectangle.

FILES

./site.loc	*Mappings of host locations to IP names and addresses*
./cities.loc	*Names of cities and their geographical locations*
*./maps/gtm_*.**	Datafiles containing map data.MapsUser ID to User name mappings.
./GeoTraceMan.ad	Applications default file for *geotraceman.* This file is not required, but it may be used for customisation of *geotraceman's* setup with an: *xrdb -merge GeoTraceMan.ad*.

SUPPORTING LIBRARIES

X11 and Xtoolkit Libraries by MIT X Consortium
Xaw3d Library by Kaleb Keithley, NASA Jet Propulsion Labs
EzMenu Library by MIT X consortium, Dirk Grunwald and George Ferguson

SEE ALSO

traceroute(8), interman(1), etherman(1), packetman(1), X(1)

BUGS

This application is currently under development, there are a number of problems which are not worth going into here as many will be fixed by the time it is released.

Yes there are bugs send any reports to **netman@cs.curtin.edu.au**

Appendix B

Table 2: GeoTraceMan main widget hierachy.

Parent Hierachy	Widget Type	Name
.	Paned	mainPane
mainPane	Box	menuBox
mainPane.menuBox	Command	quitButton
mainPane.menuBox	MenuButton	mapmenuButton
mainPane.menuBox.mapmenuButton	EzMenu	mapMenu
mainPane.menuBox	MenuButton	includemenuButton
mainPane.menuBox.includemenuButton	EzMenu	includeMenu
mainPane.menuBox	MenuButton	labelmenuButton
mainPane.menuBox.labelmenuButton	EzMenu	labelMenu
mainPane.menuBox	MenuButton	filemenuButton
mainPane.menuBox.filemenuButton	EzMenu	fileMenu
mainPane.menuBox	MenuButton	miscmenuButton
mainPane.menuBox.miscmenuButton	EzMenu	miscMenu
mainPane	Form	bigForm
mainPane.bigForm	Form	iconForm
mainPane.bigForm.iconForm	Form	traceForm
mainPane.bigForm.iconForm.traceForm	Label	traceLabel
mainPane.bigForm.iconForm.traceForm	Command	starttraceButton
mainPane.bigForm.iconForm.traceForm	Command	killtraceButton
mainPane.bigForm.iconForm.traceForm	Command	removetraceButton
mainPane.bigForm.iconForm	Form	mapForm
mainPane.bigForm.iconForm.mapForm	Label	mapLabel
mainPane.bigForm.iconForm.mapForm	Command	zoomButton
mainPane.bigForm.iconForm.mapForm	Command	cancelzoomButton
mainPane.bigForm.iconForm.mapForm	Command	resetzoomButton

Table 2: GeoTraceMan main widget hierachy.

Parent Hierachy	Widget Type	Name
mainPane.bigForm.iconForm	Form	tstatsForm
mainPane.bigForm.iconForm.tstatsForm	Label	tstatsLabel
mainPane.bigForm.iconForm.tstatsForm	Label	totaltimeLabel
mainPane.bigForm.iconForm.tstatsForm	Label	lasthostLabel
mainPane.bigForm.iconForm.tstatsForm	Label	lasthostnLabel
mainPane.bigForm.iconForm.tstatsForm	Label	numhostsLabel
mainPane.bigForm.iconForm.tstatsForm	Label	numhostsnLabel
mainPane.bigForm.iconForm.tstatsForm	Label	tstatusLabel
mainPane.bigForm.iconForm.tstatsForm	Label	tstatusnLabel
mainPane.bigForm.iconForm	Form	mstatsForm
mainPane.bigForm.iconForm.mstatsForm	Label	mstatsLabel
mainPane.bigForm.iconForm.mstatsForm	Label	detailLabel
mainPane.bigForm.iconForm.mstatsForm	Label	detailnLabel
mainPane.bigForm.iconForm.mstatsForm	Label	currentzoomLabel
mainPane.bigForm.iconForm.mstatsForm	Label	currentnzoomLabel
mainPane.bigForm.iconForm.mstatsForm	Label	mstatusLabel
mainPane.bigForm.iconForm.mstatsForm	Label	mstatusnLabel
mainPane.bigForm.iconForm	Form	pannerForm
mainPane.bigForm	Form	canvasForm
mainPane.bigForm.canvasForm	Simple	drawingCanvas
mainPane.bigForm.canvasForm	Scrollbar	xpansb
mainPane.bigForm.canvasForm	Scrollbar	ypansb