The **Project GeoSim** Graphical User Interface

James M. A. Begole, Clifford A. Shaffer, and Mark Lattanzi

Dept. of Computer Science
Virginia Tech
Blacksburg, Virginia 24061

ABSTRACT

This paper discusses the Graphical User Interface (GUI) and Application Program Interface (API) used by programs developed for **Project GeoSim**. This paper focuses on the problems encountered and the lessons learned during our development effort. Criteria for the GUI/API are explained along with the design decisions made to address those criteria. Of course, the most significant decision was to develop the **GeoSim** GUI/API; we describe why existing systems were unsatisfactory. The problems encountered during the GUI/API development are generally of two types: Graphical User Interface Design and Portability Issues.

1 INTRODUCTION – Project GeoSim

A multidisciplinary effort involving members of the Departments of Geography and Computer Science at Virginia Tech, **Project GeoSim** is developing computer-aided instruction (CAI) modules for teaching introductory geography to college freshmen [1]. At this time, five instructional modules are under development. *IntlPop*, a population demographics tutorial and simulation for countries and regions of the world, has been used in classes since Spring 1993. Our second module, called *MigModel*, lets students study the factors that influence migration between counties in the United States. *MigModel* will be used in class for the first time this Fall. *MentalMaps* is a drill type exercise to test knowledge of the location, size and environment of cities. The fourth module is a simulation of the movement of the radiation cloud generated by the Chernobyl nuclear reactor accident, and is currently in the prototype stage. The fifth module, entitled *Orienteering*, allows the student to develop map reading skills using a simulation of travel through rural countryside.

2 GUI/API CRITERIA

Before **Project GeoSim** began, the following criteria for our GUI/API system were identified as necessary to meet our goals.

Portability. We intend for **Project GeoSim** modules to be used in Geography classes throughout the country. For this to happen, we must write our programs for the computing environments that are available to instructors at most universities and colleges. As a result, our programs must run on a wide variety of computers. The particular platforms that we will support are IBM-compatible machines running MS-DOS, Macintosh computers running MAC OS, and UNIX workstations running X-windows.

Simplicity and Consistency of User Interface. **GeoSim**'s student users are primarily computer novices who will see a **GeoSim** teaching module only once or a few times. Students may, however, use several **GeoSim** modules throughout a geography course. Therefore, we strive for a simple interface while maintaining the greatest possible consistency in the user interface across modules. We hope in this way to ease the students' burden to learn the interface of each module.

Simplicity of Application Program Interface. The three platforms on which **GeoSim** modules run have complex and disparate operating systems. Application developers should not need to know the specific details of these systems. Therefore, our API attempts to hide the specifics of each platform from the application developer.

Flexibility. Module creation includes extensive user testing throughout the development process (formative evaluation). We firmly believe in rapid prototyping and continual revision of the program based on user feedback. Application developers therefore must be able to easily modify the user interface in response to formative evaluation of a module. The API should also be flexible and powerful enough to facilitate the development of applications employing several different instruction techniques such as simulation, tutoring, hands-on, walkthrough, and text.

3 DESIGN DECISIONS

3.1 Project GeoSim GUI/API

Portability. Each of the three target platforms has a well-known GUI development environment already available. However, programs written on any one are not portable to the others. Nor do there exist any well-known and reliable interface packages that run on all three systems targeted for our project. Thus, our need for portability led to the development of our own GUI/API.

The **GeoSim** GUI/API is implemented as a series of C library routines called by the application program. This approach requires us to reimplement only a small core graphics library and integrate our interface with the event handler of the native environment (i.e., MS-DOS, the Macintosh Toolbox or X-windows). No other portions of the API library need be rewritten to port our software to a new environment. The port from MS-DOS to X-Windows required approximately 800 lines of source code from MS-DOS to be translated into approximately 400 lines on X-Windows. Application source code of the **GeoSim** modules using this API library are completely portable, requiring no revision.

Hiding platform specific I/O from the application developer ensures portability. A disadvantage, however, is that we cannot use the interface development environments and tools native to the platforms. Additionally, users familiar with a platform's standard interface (such as the Macintosh Desktop) will not find expected features. **GeoSim** module users, however, are typically computer novices with no such expectations.

Graphical User Interface. To address the need for simplicity and consistency in the user interface among the **GeoSim** modules, we desired a mouse-driven Graphical User Interface (GUI) incorporating direct manipulation.

Project GeoSim modules conform to a basic "look and feel" standard we have developed (see Figure 1). For example, the main window of each **GeoSim** module has a menu bar along the top. A message window at the bottom displays context sensitive help. Colors are used consistently among modules.

Flexibility. To allow for easy changes to the user interface, a separate *interface file* is used to store the details of interface elements. This text file provides a central location for all interface details, such as size and placement of windows, menus and buttons, as well as the text for labels and help messages. The API handles storage and retrieval of interface items, each of which is given a name by which an application developer may access them. Appendix 1 contains a sample *interface file*.

These user interface elements may be changed without recompiling the application program. Adding to the flexibility of this file, colors within the file may be *aliased*. Only the alias needs to be changed in order to change the color of all elements using that alias.

There are some disadvantages to this approach in terms of power and efficiency for the application developer. Some details of the interface elements are unavailable to the application. Also, the developer is restricted to the interface objects provided by the API. Finally, there is some (minor) overhead to store and retrieve interface items. However, compared to the overhead imposed by systems such as X-Windows, the Macintosh Toolbox, or Microsoft Windows, our system is very efficient.

Several instructional modules have been developed with the **GeoSim** GUI/API, illustrating its flexibility

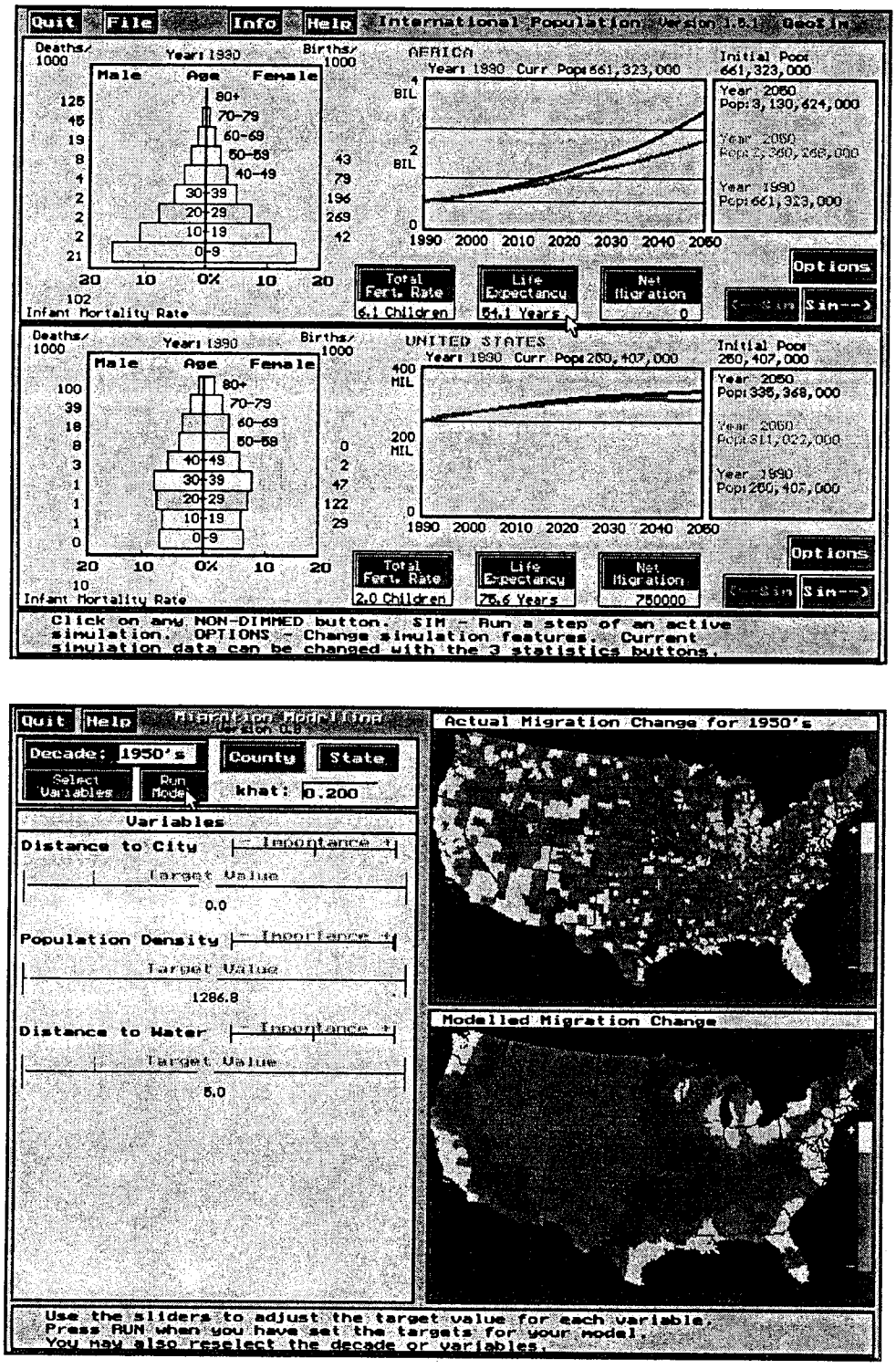

Figure 1: Two of **GeoSim**'s education modules: *IntlPop* and *MigModel*.

and power. While we are currently developing modules for Geography classes, the GUI/API is not restricted to this subject. In fact, applications other than CAI could use the **GeoSim** GUI/API.

Simplicity of Application Program Interface. Our GUI is not intended as a complete operating system, nor as a complete windowing system. **GeoSim** modules contain one static background window over which the application developer controls creation and removal of other static windows. The user is not able to resize or move these windows. Certain other standard features, such as automatic clipping of graphical objects to a window, are not supported.

Following the idea that less is more, we have tried to keep the number of features provided by the **GeoSim** API small. In part, this is motivated by our having developed the initial version for MS-DOS. Programmers who wish to develop some self discipline with respect to the use of computer resources (especially main memory) would do well to spend some time programming under MS-DOS.

The following are the major **GeoSim** interface elements.

Windows are rectangular regions of the screen within which other elements may be grouped.

Push Buttons perform a function when pressed with a mouse. Their scope is local to the window in which they are declared.

Drag Rectangles (or "dragrects") are rectangular regions of the screen within which a user may perform some direct manipulation. Dragrects can handle several events such as character input, mouse motion, and mouse "drags." Their scope is also local.

Fields are (typically small) rectangular regions in which output is displayed. These allow the location of graphical objects such as labels or pictures to be defined in the *interface file* rather than hard-coding these locations within the program. The scope of a field is global to all windows in the application. However, the coordinates of a field are relative to the current window at run-time. In this way, the same field definition may be used in more than one window.

Menus are lists of functions from which a user may select. Like push buttons, a developer may disable a menu item or make it invisible. Check-list menus, with which a user may toggle program parameters, are also available.

Independent Functions are functions that are not part of any specific interface element. The developer may declare an independent function to be called each time the API iterates its event loop. Such functions may be activated and deactivated by the program as needed.

The API library currently includes functions to support higher level interface objects such as lists and a multipage text display system. These elements may be combined to create more specialized controls. For example, a scrollable list may be made by combining the basic list, push button, and dragrect elements.

3.2 Primary Development Platform – MS-DOS

MS-DOS was chosen as the primary development platform for several reasons. First, our collaborators in the Geography department have an MS-DOS based computer laboratory which is used by their students running **GeoSim** modules. Second, our developers were most comfortable with this environment. Third, the bulk of our potential users outside of Virginia Tech will require MS-DOS based programs. MS-DOS has turned out to be a fortunate choice as the initial development platform, since it is easier to have the X-Windows and Macintosh environments emulate the capabilities of MS-DOS than the converse. The API has recently been ported to the X-Window System V11 R5. An earlier version of the API has been ported to Macintosh System 6.

3.3 Generic Database Reader

For many of our modeling exercises, significant amounts of data must be collected and organized. One of the incidental benefits of our project to the education community is the creation and publication of these databases. To make the most of this, the data files

should be human as well as machine readable. An ASCII-based data formatting scheme has been selected, and a set of utility functions for manipulating files in this format has been created. An additional benefit of this approach is that it was easily extended to support other project needs, such as module configuration files. Aside from image data, all database and support files in the **GeoSim** System conform to our standard database format. We are presently designing a series of tools to automatically convert raw data files to our format. Such data files include the Census Bureau's City County Data Book.

4 PROBLEMS ENCOUNTERED AND THEIR RESOLUTIONS

The problems we have encountered are generally of two types: Graphical User Interface Capabilities and Portability Issues. The categories overlap each other somewhat, especially when a portability issue or a limitation of one of the platforms has driven a decision behind a GUI capability.

4.1 GUI/API Design

Developing our own GUI/API, we wanted to provide an application developer with as much capability as he or she might require, balanced by a desire to keep the support libraries small and simple. At the same time, the usability of the GUI was paramount.

API Library Remains Open. Each application module requires different levels of control of the interface elements. Therefore, we have left the API open to extension and enhancement as we develop application modules with it. As a result, the API library has grown larger and more complex than we had initially intended.

GUI Mouse Controls. We have provided support for basic mouse control of the user interface. Users press a button on the screen or select an item in a menu by positioning the screen cursor over the button or item and *clicking* the mouse button. Additionally, users may manipulate an item by *dragging* it.

A problem arises in distinguishing between a *click* and a *drag* when the user presses a mouse button. Initially, our API considered it a *drag* if the user moved the mouse or if a certain amount of time passed before release. Otherwise, it was considered a *click*. This algorithm distinguishes a drag from a "flying click" in which a user quickly clicks the mouse button while the cursor is moving across the screen. However, we are currently reimplementing this part of the API to consider it a *drag* if and only if the mouse has moved. This change was prompted by the difficulty of deciding what the proper duration was to consider a mouse press as a *drag*. Different users hold the button down for different lengths of time. Additionally, we found during formative evaluation that novice users generally do not use "flying clicks."

We have not yet needed the capability of *double-clicking* an item, nor the use of more than one button on a mouse. The consensus within our group is that *double-clicking* is a bad user interface.

Linking Function names to Function Pointers. GUI objects such as buttons must often be associated with C functions that perform some action when the button is pressed. The function name can be specified in the interface description file. However, there must be some mechanism for attaching the name of the function as specified in the interface file to a function pointer stored with the C structure representing the GUI object. This is done using an array of structures initialized within the C program. Each element of the array has the name of the object and the associated function pointer. This approach requires recompilation if either the logical or physical name of the function changes, as well as some coordination between the interface file and the C program. However, the names rarely change in practice, and this has been a satisfactory approach.

Inefficiency Due to Abstraction. Due to the different graphics conventions on the three target platforms, access to basic graphical functions such as line drawing must go through a **GeoSim** API function call. The call to the **GeoSim** API function is then translated to the graphics routine native to the target platform. This extra layer causes some delay, but ensures portability (see Figure 2). Also, at the core level, functions such as setpixel and drawline can be optimized in assembly for each platform.

In addition, interface items are stored in an array of

structures containing item type, location, function, etc. This data structure must be searched to find the appropriate function to call at each event. Since there are few interface items, however, such searches are generally very quick. Latency from the time a button is pressed to when its function is invoked is not noticeable.

Overlapping Buttons. In some cases it is useful to be able to enclose smaller buttons within a large dragrect. For instance, the large enclosing dragrect may process some input, such as key presses, that the smaller buttons do not process (see Figure 3). A Dragrect may enclose other dragrects as well as buttons.

Fonts. In most commercial GUIs, the font subsystem is a major cause of "software bloat." So far we have not found a need for more than the most basic fonts. Therefore, our API provides two simple fonts: SMALL and LARGE. On our primary development environment, MS-DOS, these fonts are simple 8x8 pixel characters. The X-Windows and Macintosh environments provide ample fonts from which to select appropriate substitutes.

Backing Store. To allow windows to be drawn and removed quickly, we desired some form of backing store for overdrawn screen areas. For the application programmer, a stacking back store would be easiest to use. However, storing screen images requires significant amounts of memory. In the MS-DOS environment, memory limits discourage more than one level of backing store. Secondary memory is not suitable for use as backing store, since its disk I/O time would not allow us to replace a stored screen area quicker than simply redrawing the whole screen. Therefore, we have provided backing store for only one window at a time. While so far this has proven satisfactory, this is an area of the API that we hope to enhance.

Window Relative vs. Absolute Coordinates. Generally, an application sends output within a window on the screen. The program wants to write to a window, regardless of where that window is on the screen. The **GeoSim** drawing functions, therefore, take window relative coordinates as opposed to absolute screen coordinates. However, the API event handler sends button relative coordinates to a dragrect's function, because these are more useful to the application developer. If the application needs to find the window relative coordinates of such an event, it can retrieve the window relative origin of that button.

Log Files. The API provides two log files for the application: an *error* log and an *output* log. Both the API and application may log errors to the *error* log. The application alone writes to the output log, which may be used to record a user's session in a learning module for grading purposes.

Active vs. Inactive Windows. A user may only manipulate the controls of *active* windows. The most recently drawn window is considered active by default. Additionally, the application developer may set windows active or inactive as desired. He or she is responsible for guarding against overlapping active windows or other problems.

Active vs. Inactive vs. Invisible Buttons and Menu Items. Buttons and menu items may also be either active or inactive. In the inactive state, the label of a button or menu item is *dimmed*. In this way, its presence is observable, but its inaccessibility is clear. Inactive buttons and menu items may also be made invisible, if the application programmer does not want them to be seen.

Bitmap Images. In MS-DOS, storing images by bitplanes allows faster drawing than storing pixel values one after the other. However, bitplanes offer no such advantage on the other platforms. Additionally, finding the value for a specific pixel in the former format is more difficult than in the latter.

Memory limits of MS-DOS only allow a few images to be kept in memory at run time unless they are compressed. On any machine, disk I/O time encourages small images. Therefore, the **GeoSim** API provides support for Run Length Encoding (RLE) compressed images. Several pixel resolutions are supported to provide optimal compression and flexibility.

If an image is to be in memory, the application developer has complete control and responsibility for the memory block used by an image. This approach allows

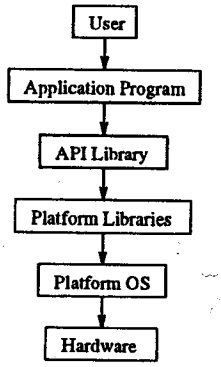

Figure 2: Abstraction layers of **Project GeoSim** modules from User to Hardware.

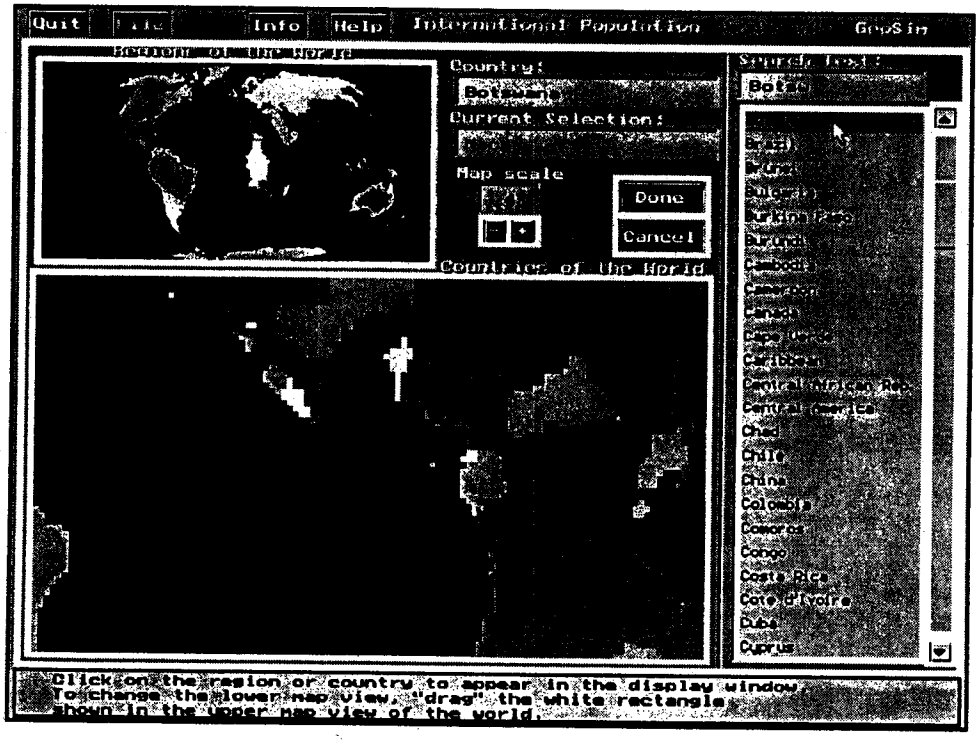

Figure 3: The largest enclosing rectangle outlines a hidden dragrect that handles keyboard input while the smaller rectangles outline buttons and dragrects that perform specific functions.

the application developer to swap images as needed. Additionally, the API allows an image to be drawn directly from a file without being first loaded into memory, if desired.

4.2 Portability Issues

The concern for portability has driven several design decisions. Many portability problems are solved by masking platform dependencies with **GeoSim** API functions. Other portability problems are not so easily solved.

Detecting Events. To port our API library to the three platforms, our event handler needs to be integrated with the native platform's event handler. For MS-DOS, our event loop polls the hardware for events. In X-Windows and the Macintosh, the event queue is read for events.

Graphics Resolutions. The **GeoSim** API uses a 640x480 pixel portion of the screen, which is the entire screen on MS-DOS (VGA) and most Macintosh machines. Whereas MS-DOS applications control the entire screen, on the Macintosh the top of the screen is taken by a standard menu bar. To obtain 640x480 pixels we eliminate this bar from the Macintosh screen (sorry, Apple).

Currently we only require 16 colors for any module. However, we anticipate future need for more than 16 but no more than 256 colors. Although this is not a problem on the Macintosh or X-Windows platforms, there is no standard among the many manufacturers of MS-DOS video components that provides 640x480 pixels with 256 colors. Despite this, we are able to use most of the major video hardware that does support this resolution with public domain graphics drivers. The API initialization routine tests for the presence of unsupported hardware and, if detected, falls back to a standard resolution of 640x480 pixels with 16 colors.

Time Resolution. The three platforms have varying standards for measuring time. Resolutions below one second are not consistent among the platforms. Whereas MS-DOS provides a standard resolution of 1/18.2 second, UNIX machines often offer 1/60 second as the standard resolution. While the timer of an MS-DOS machine may be forced to provide finer resolution, machine performance suffers as the CPU spends more cycles handling timer events. Additionally, such non-standard timing may lead to unanticipated problems for any programs running concurrently with **GeoSim** modules (such as MS-Windows).

To mask the different standards, the API includes a function that translates a platform's time standard to milliseconds. For example, one clock tick is approximately 55 milliseconds under MS-DOS and approximately 16.67 milliseconds under ULTRIX running on a DECstation 5000/133. While some rounding error is incurred, care is taken not to accumulate such errors.

Assembly Level Programming. For speed, some I/O functions of the API were written in assembly language under MS-DOS. While these are certainly nonportable functions, they are easily re-written in C on the other platforms.

Naming Conventions. File names follow the MS-DOS convention, since the other platforms allow this format.

Memory Issues. MS-DOS uses a segmented memory scheme which requires that memory blocks be confined to 64 Kilobytes. Additionally, MS-DOS does not provide virtual memory. Applications are restricted to less than 640 Kilobytes of run-time memory. While these are not uncomfortable restrictions, they illustrate another reason why we are happy with our selection of MS-DOS as the primary development platform. If we had not had such restrictions in mind while developing the API on a machine providing a flat (vs. segmented) memory model and virtual memory, we would have encountered trouble when porting to MS-DOS, which does not provide such memory management features. To support MS-DOS, memory frugality is required by both the GUI/API and applications.

Mouse Cursor. MS-DOS requires that the mouse cursor be *hidden* (that is, removed from the screen) before anything is drawn over the cursor. The API's graphic routines are therefore bracketed by mouse-off/mouse-on calls. Having no such requirement under X-Windows, the mouse-off and mouse-on functions do nothing.

5 CONCLUSIONS AND FUTURE WORK

The portability criterion drove our decision to develop our own GUI/API. The other criteria, consistency, simplicity, and flexibility, were addressed during the development of that GUI/API. The **GeoSim** GUI enforces consistency among the user interfaces of our modules. While the API is larger than initially hoped for (about 110K under MS-DOS for the executable code), it is nevertheless relatively easy to use. The use of a text *interface file* provides the application developer with the flexibility he or she needs to respond quickly to formative evaluation. While a graphical user interface development environment is all the rage these days, having that added to our system would speed up our program development by only a trivial amount.

We encountered several problems during the development process. These generally fall in one of two categories: Graphical User Interface Design, and Portability Issues. Formative evaluations of the first **GeoSim** module, *IntlPop*, have satisfied us that our interface is usable by our target users. The portability problems have been satisfactorily resolved. The API has recently been ported to X-Windows. We believe that the choice of MS-DOS as the primary development platform simplified our porting efforts, since it is easier to have the X-Windows and Macintosh environments emulate the capabilities of MS-DOS than the converse.

In the future, we hope to enhance some areas of the GUI/API such as the backing store system and the API's event handler. Evaluation of the User Interface continues and we expect to change and improve it in response to those evaluations.

IntlPop is available via anonymous ftp from geosim.cs.vt.edu (128.173.6.111). *MigModel* will be introduced in classrooms this Fall and several more modules are currently under development.

ABOUT THE AUTHORS

James Begole ("Bo") received his B.S. in Mathematical Sciences/Computer Science at Virginia Commonwealth University in 1992. He is currently pursuing his M.S. in Computer Science at Virginia Tech. Mr. Begole works as a research assistant for **Project GeoSim**. His research interests include Computer Aided Instruction/Computer Based Training, and user interface design and development. He may be reached via electronic mail at **begolej@cs.vt.edu**

Dr. Clifford A. Shaffer received his B.S. in 1980, his M.S. in 1982 and his Ph.D. in 1986, all from the Computer Science Department at the University of Maryland. He joined the Department of Computer Science at Virginia Tech in 1987, where he is presently an Associate Professor. Dr. Shaffer's research interests are primarily in the area of data structures and algorithms, particularly for applications such as Geographic Information Systems and Computer Graphics. Dr. Shaffer is also actively engaged in research projects to develop Computer Aided Education materials, such as **Project GeoSim** and the Educational Infrastructure project in the Department of Computer Science at Virginia Tech. He may be reached via electronic mail at **shaffer@vtopus.cs.vt.edu**.

Mark Lattanzi received his B.S. in Electrical Engineering at Virginia Tech in 1987 with a concentration in digital design. He received his M.S. in Computer Science in 1989 from Virginia Tech under Dr. Shaffer. His primary fields of interest include computer graphics and spatial data structures. Mr. Lattanzi's industry experience includes working for Hewlett Packard Company in Fort Collins, Colorado in the Graphics Technology Division. He developed low level graphics device drivers for HP personal computers and workstations. In 1991, he returned to Virginia Tech to pursue a Ph.D. in Computer Science under Dr. Sallie Henry. His field of research focuses on software metrics, the object oriented paradigm, and software reusability. He may be reached via electronic mail at **lattanzi@cs.vt.edu**

REFERENCES

[1] L.W. Carstensen, C.A. Shaffer, R.W. Morrill and E.A. Fox, **GeoSim**: A GIS-based simulation laboratory for introductory geography, to appear in *Journal of Geography*.

1 Geotest.inf

The following is a sample *interface file.*

```
# Interface file --- Geotest
# Geotest is an application to test the API on the 3 platforms.

# The following is the syntax of elements in this file.
# WINDOW ( title x y wd ht fg bg bdrcolor bordersize active?
#          backstore? )
# POPUP ( title x y efg dfg bg border cancel_on? )
# BUTTON ( title |label| x y wd ht efg dfg bg function border?
#          active? )
# DRAGRECT ( title x y wd ht bg function border active? )
# LABEL ( x y color |text| <backgroundcolor> )
# PLABEL ( active function |text| )
# FIELD ( title x y wd ht color )
# Colors under PALETTE == ( colorname red green blue <index> )
# ALIAS ( aliasname colorname )
# BUTTONSTUFF ( bordercolor diagonals highlite_TopLeft
#               highlite_BottomRight )

# the first 7 colors are standard GeoSim interface colors
# -- do NOT change
PALETTE
( black      0   0   0 )  #  Never change this
( dgray     50  45  45 )  #  disabled popup text
( mgray     30  30  55 )  #  disabled button text
( lgray     53  53  53 )  #  display window background
( bluegray   0  50  50 )  #  background, highlight bar
( lblue     10  10  55 )  #  button color
( peach     63  47  50 )  #  msg window / popup bckgrnd
( lcyan      0  50  50 ) #  the following colors can change
  #  except white (15)
( mcyan      0  35  35 )
( gray      25  25  25 )
( myellow   40  40   0 )
( lyellow   60  60   0 )
( magenta   30   0  30 )
( dgreen     0  30   0 )
( pasgreen  30  50  30 )
( white     63  63  63 )  #  Never change this (mouse cursor color
                          #                      - MS-DOS)
END  #PALETTE

#  Aliases for colors for ease of modification
ALIAS ( rootbg bluegray )
```

```
ALIAS ( rootfg black )
ALIAS ( bbg lblue )
ALIAS ( befg white )
ALIAS ( bdfg mgray )
ALIAS ( popefg black )
ALIAS ( popdfg dgray )
ALIAS ( popbg peach )
ALIAS ( popbdr black )
ALIAS ( wbdr black )

HIGHLIGHT ( bluegray )
SPEEDUP ( 100 )
BUTTONSTUFF ( black 0 white black )
CANCEL ( befg bdfg bbg mycancel )
FUNCTION ( intro 0 intro )

POPUP ( Quit 9 23 popefg popdfg popbg popbdr 0 )
  PLABEL ( 1 doquit |Quit| )

WINDOW ( root 0 0 640 480 white rootbg wbdr 1 1 0 )
  BUTTON ( Quit |Quit| 4 3 44 18 befg bdfg bbg @quit 1 1 )
  LABEL ( 250 15 white |Geosim Application Test| black )
  FIELD ( VERSION 450 3 60 19 rootfg )    # location of program version

WINDOW ( Help 115 23 380 384 black peach wbdr 1 1 0 )
  FIELD ( hello1 480 18 121 16 lyellow )
  FIELD ( hello2 480 18 121 16 white )
  DRAGRECT ( textinput 2 2 376 380 peach handledragrect 0 1 )
  BUTTON ( testbutton |Again| 40 350 80 20 befg bdfg bbg button 1 1 )
  FIELD ( textfield 220 18 121 16 dgray )
  FIELD ( textoutput 219 40 123 370 black )
  RECT ( 218 16 125 20 dgray 1 fill )
  RECT ( 218 16 125 20 wbdr 1 hollow )
  LINE ( 218 34 342 34 2 white )     # bottom
  LINE ( 341 34 341 16 2 white )     # right
```