Porting a Visualization Package from IRIX to NT : what will I get and what will I pay ?

R. van Liere, J. Harkes, J. Kniesmeijer

Software Engineering (SEN)

SEN-R9821 September, 1998

Report SEN-R9821
ISSN 1386-369X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Porting a Visualization Package from IRIX to NT : what will I get and what will I pay ?

Robert van Liere

Jan Harkes
CWI,
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Joris Kniesmeijer
Cap-Gemini,
P.O. Box 2575, 3500 GN, Utrecht, The Netherlands

ABSTRACT

We discuss our experiences in porting a moderately large scientific visualization environment from IRIX to NT 4.0. Two porting strategies have been taken: a port via a POSIX emulation layer and a native NT port. POSIX compliant code can be ported to NT with relatively little effort if the code adheres to general accepted programming principles, such as modularity and encapsulation.

The performance of a modern 3D Wintel machine is quite satisfactory for a variety of scientific desktop tasks. We have compared the performance of a 2 CPU Dell OptiPlex with FireGL 4000 graphics option to various SGI desktop workstations.

1991 Computing Reviews Classification System: I.3.3 [Computer Graphics]: Picture/Image Generation - Display Algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Visible Line/Surface Algorithms;
Keywords and Phrases: Distributed Visualization Architectures.
Note: Work carried out under CWI project SEN-1.3, Interactive Visualization Environments

1. INTRODUCTION.

The transition from low end UNIX desktop workstations to Wintel PCs for technical applications seems inevitable. We discuss our experiences in porting a moderately large scientific visualization environment from IRIX to NT 4.0. Our motivation for doing this work was threefold:

- porting the software from UNIX to NT is a deliverable for an externally funded HPCN project.
- evaluate NT as a platform for developing technical applications.
- compare the performance of state-of-the-art 3D Wintel PCs with SGI desktop offerings.

This paper is organized as follows. First, we give a short overview of the scientific visualization environment and discuss its capabilities. Then, we discuss our experiences in porting the software to NT and give performance comparisons with two SGI desktop offerings. Finally, we discuss some lessons we learned from this work.

2. A QUICK TOUR THROUGH THE SOFTWARE.

Architecture. The scientific visualization environment [1] is a network transparent client/server based architecture, consisting of many general purpose tools that allows a user to define visualizations and data mappings. An overview of the architecture is shown in figure 1. The architecture is centered around a *data manager* that acts as a blackboard for communicating values. Separate processes (*satellites*) connect to the data manager and exchange data with it. The PGO satellite is a general

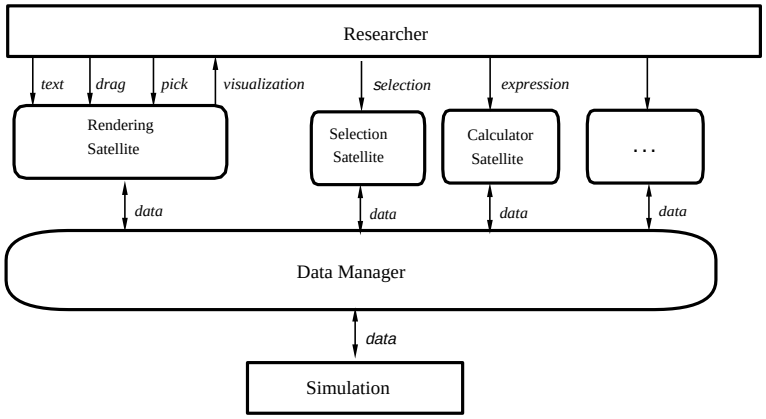

Figure 1: The architecture

purpose rendering tool that allows users to build an interface from graphics objects whose properties are functions of data in the data manager, [2]. Users sketch an interface and bind the geometry and attributes of the graphics objects with data in the data manager. Satellites may drive the interface by mutating the data bound to the graphics objects. Similarly, users may drive the simulation by interacting with graphics objects. Hence, a two-way communication between graphics and data in the simulation can be achieved.

The user may choose to execute a satellite on a remote or a local host. For scientific visualization applications the most natural configuration would be to run the PGO editor on a graphics workstation and the simulation on a remote high performance compute server.

Implementation. The environment provides a number of libraries for satellite development. These libraries interface the visualization software to the features of the underlying operating system or to other external software package. The capabilities provided by the libraries are:

- transport library (`libdm.a`)

  The transport library handles all low level issues concerning: managing satellite connections, communication with the data manager, and data representation.

  `libdm.a` uses UCB sockets to connect satellites to and exchange control information with the data manager. In addition, sockets and, when possible, shared memory are used for data movement. A simple XDR like mechanism ensures valid data representations among heterogeneous hosts.

- graphics library (`libdraw.a`)

The PGO editor uses a number of high level primitives for rendering. These high level primitives are layered on top of lower level primitives supported by underlying graphics packages.

The 3D version of the PGO editor uses IrixGL or OpenGL as the underlying graphics packages. In addition, the 2D version has been ported to X11.

- widget library (`libui.a`)

  A simple library is defined to create panels of widgets (labels, buttons, sliders, drawing areas, etc).

  `libui.a` has been implemented to use Motif or XForms.

- miscellaneous (`libmisc.a`)

  `libmisc.a` is a library that emulates POSIX functions. Examples of this functionality are accurate timers and sleep. This library should be empty if POSIX is supported, but can contain substantial amounts of code if POSIX is not supported.

Code statistics. Table 1 gives an indication of the source tree measured in the number of files and lines of C code. The numbers are given for the four enabling libraries, the data manager and the PGO

	files	lines
`libdm.a`	19	2471
`libdraw.a`	10	6416
`libui.a`	2	1333
`libmisc.a`	1	0
`data manager`	14	4416
`PGO`	93	29267

Table 1: Number of files and lines of IRIX source code

editor. Other satellites and higher level libraries are not given in this overview.

The PGO and data manager are layered on top of the four enabling libraries. Given a reasonable ANSI compliant c-compiler, we expect that the data manager and pgo satellites to compile, link and execute without problems.

3. PORT.

Two porting strategies have been taken: a port via the UNIX emulation layer and a native port to NT. We use the SGI IRIX version of the environment as the base implementation and compare the porting effort of the other ports.

The next three paragraphs discuss some technical details of the port.

IRIX implementation. The IRIX version uses reliable, two-way connection based byte stream (i.e SOCK_STREAM) sockets of type AF_INET and AF_UNIX and SysV shared memory for `libdm.a`. OpenGL is used to implement `libdraw.a`. XForms is used to implement `libui.a`. POSIX provides the functionality needed by `libmisc.a`.

Porting using U/WIN. The U/WIN package provides a POSIX emulation layer on top of NT. [1] The goal was to change as little as possible of the original UNIX code. The second column of table 2 shows how many lines of code were changed in each module in the system.

A few lines of code were changed in `libdm.a` since NT does not support the AF_UNIX type of socket. The changes in `data manager` are due to the way signal handlers are defined and used in IRIX (which is slightly different than in POSIX). The changes in `libdraw.a` and `PGO` are due some to SGI specific math routines which are not available in U/WIN.

Native port to NT. The goal of the native NT was to obtain the highest performance. The third column of table 2 shows how many lines of code were changed in each module.

The changes in `libdm.a` are mostly due to NTs lack of shared memory. `libmisc.a` emulates a number of POSIX functions related to timers and also the scatter and gather I/O routines `readv` and `writev`. Changes in `data manager` were mostly due to different control paths due to the lack of shared memory. The changes in `libdraw.a` and `PGO` are due some to SGI specific math routines which are not available in U/WIN. Large parts of `libui.a` needed to be rewritten, since the NT widget set is incompatible with XForms.

Porting statistics. Table 2 gives the number of lines changed for the U/WIN and NT version of the source code. The table shows that the effort of porting via U/WIN is less than the native

	U/WIN	NT
`libdm.a`	179	546
`libdraw.a`	5	5
`libui.a`	1745	1745
`libmisc.a`	0	101
`data manager`	13	103
`PGO`	8	12

Table 2: Number of lines of code changed

port. An exception is `libui.a` which is implemented using the WIN32 API which was completely rewritten. [2]

4. PERFORMANCE.

Two specific applications were used to benchmark the performance.

1. *Bouncing balls in 2D.* This configuration consists of a simulation, the PGO editor and the data manager running on the same machine. The simulation calculates new balls positions and velocities in a 2D field. In addition, an attraction field is calculated. A number of parameters can steer these calculations: number of balls, ball radius, attraction forces, damping factor, etc. After each step the simulation dumps the state into the data manager, and checks if parameters

[1] See appendix A for a overview of U/WIN.

[2] Recent versions of Tcl/TK have been ported to NT. The number of changed lines in `libui.a` would be substantially less if it was implemented by Tcl/TK.

have been changed. After each step, the PGO reads in the state and redraws the scene. The PGO dumps new parameter values if the user interacts with the graphics objects.

The amount of data movement moving from simulation to data manager to PGO is approximately 200 Kbytes per frame. For 2D rendering, each frame consists of a 64x64 mesh, a number of circles and boxes. In total this amounts to roughly 45K triangles per frame.

2. *Bouncing balls in 3D.* This configuration consists of a simulation, the PGO editor and the data manager running on the same machine. The simulation calculates new balls positions and velocities in a 3D field. A number of parameters can steer these calculations: number of balls, ball radius, damping factor, etc. After each step the simulation dumps the state into the data manager, and checks if parameters have been changed. After each step, the PGO reads in the state and redraws the scene. The PGO dumps new parameter values if the user interacts with the graphics objects.

   The amount of data movement moving from simulation to data manager to PGO is approximately 20 Kbytes per frame. For 3D rendering the scene is simple; each frame consists of a number of circles and boxes. Lighting and z-buffering is on. Shadows are also calculated and drawn.

Table 3 shows the performance of the different configurations. Performance comments:

	balls2D	balls3D
`Dell OptiPlex U/WIN`	14	16
`Dell OptiPlex NT`	14	17
`Indigo2 MaxImpact`	15	30
`Indigo2 HiImpact`	12	20
`O2`	11	12

Table 3: Frames/sec of two bouncing balls applications.

- The Dell Optiplex started shipping in 1996. The FireGL 4000 card started shipping in 1997. The Indigo2 started shipping in June 1995. The O2 shipped in October 1996.
- These applications demonstrate a mixture of computation, data movement and graphics. The simulation does more computation and generates more data in the 2D case.
- The NT version uses sockets and not shared memory for data transport. However, the performance numbers seem to incdicate that these applications are not data movement bound.
- Performance results indicate that the additional overhead of the U/WIN layer is negligible.
- Stand Alone 3D benchmarks (eg viewperf) indicate that the FireGL 4000 is very competitive with the HiImpact. See the appendix for an overview of viewperf results measured on FireGL 4000 and MaxImpact, HiImpact and O2.

5. BITS AND PIECES.

In this section we list some miscellaneous topics that have greatly influenced the porting effort:

- modular software design.

  A modular approach was used to design the software. All operating systems dependent code has been encapsulated in low level libraries; higher level libraries only build upon these low level libraries.

- vendor feature independence.

  The software does not use SGI specific features. Instead, whenever possible, only standardized APIs (POSIX, OpenGL, Motif) were used.

  For example, SGI offers a rich set of additional libraries (eg libdmedia.a, libgutil.a), tools (eg. pca, pfa) and extensions (eg. gl∗EXT, libSgm.a) which are not taken advantage of.

- NT semantics may clash with POSIX.

  In many cases the semantics of the NT APIs differ from POSIX semantics. An example is the UCB socket library. The `socket` routine returns a file descriptor which can be used in `select`, `read` and `write` system calls. In NT, the `socket` routine does not return a file descriptor. Instead, an object of an opaque `SOCKET` type is returned, which can be in `select` but not in `read` and `write` routines.

- development environment.

  The traditional UNIX development environment consists of a set of independent development tools (ed, cc, adb, make, ar, ls, nm, grep) which users can combine in a command line interface to perform higher level tasks. The Visual C/C++ environment provides an integrated environment which offers roughly the same functionality through a graphical user interface.

  Ironically, experienced UNIX users find the Visual C/C++ environment awkward, whereas Visual C/C++ experts find the command driven interfaces awkward.

6. CONCLUSIONS.

We have discussed the effort involved in porting a moderately large scientific visualization environment from IRIX to NT4.0. Two porting strategies have been taken: a port via an UNIX emulation layer and a native port to NT. Using the U/WIN package, we have demonstrated that POSIX compliant code can be ported to NT with relatively little effort if the code adheres to general accepted programming principles, such as modularity and encapsulation. Your milage may vary enormously.

The performance of todays 3D PC is sufficient for a variety of scientific desktop tasks. We have compared the performance of a 2 CPU Dell OptiPlex with SGI's R4400 Indigo2 Max and HiImpact and O2 workstations. For 3D graphics, the performance of the OptiPlex with FireGL 4000 is competitive with an SGI R4400 Indigo2 HiImpact.

It is important to note that price and performance is only a small part of the equation when evaluating technical desktop seats. In this paper we have not discussed issues such as completeness of the programming development environment, scalability, robustness, security, and administration.

References

1. R. van Liere, J.D. Mulder, and J.J. van Wijk. Computational steering. *Future Generation Computer Systems*, 12(5):441–450, April 1997.
2. J.D. Mulder and J.J. van Wijk. 3D computational steering with parametrized geometric objects. In G.M. Nielson and D. Silver, editors, *Visualization '95 (Proceedings of the 1995 Visualization Conference)*, pages 304–311, 1995.

1. A QUICK TOUR THROUGH U/WIN.

The U/WIN package provides a mechanism for building and running UNIX applications on Windows NT with few, if any, changes. The U/WIN package contains the following three elements:

- libraries that provide the UNIX application programming interface.
- include files and development tools such as cc, yacc, lex and make.
- Korn Shell and over 160 utilities such as ls, sed, tar, stty, etc.

A cc command is provided to compile and link programs for U/WIN. The cc command calls the MicroSoft Visual C/C++ compiler to perform the actual compilation and linking. For example, the familiar UNIX command:

```
cc -c -I/d/robertl/include foo.c
```

is executed as:

```
/C/MSDEV/bin/cl.exe -nologo -W3 -G3 -D_X86_=1
-D__STDC__=1 -D_POSIX_ -Dunix '-Id:\robertl\include'
'-FIC:\usr\include\astwin32.h' '-IC:\usr\include'
'-D_STD_INCLUDE_DIR=c:\MSDEV\include' '-Ic:\MSDEV\include'
-c -Fofoo.o foo.c
```

The U/WIN package provides the following functionality: process and control management, file descriptor semantics, UNIX signal semantics, terminal interface for consoles, sockets and serial lines, use of mouse for console windows, UCB sockets based on WINSOCK, pathname mapping from UNIX to NT, UNIX naming conventions, mapping to and from UNIX ids/permissions to NT permissions, memory mapping and shared memory, System V IPC, runtime linking of dynamically linked libraries, error mapping from NT to UNIX, i-node numbers and link counts, symbolic links, fifo's, setuid programs, telnet deamon.

Visit http://www.research.att.com/sw/tools/uwin for more information on U/WIN.

2. HARDWARE SPECIFICATIONS.

Dell

```
Dell Optiplex
2x PentiumPro @ 200 Mhz
128 MB
4 GB SCSI
PCI
FireGL 4000
```

Indigo2

```
; uname -a
IRIX borneo 6.2 03131015 IP22
; hinv
1 250 MHZ IP22 Processor
FPU: MIPS R4000 Floating Point Coprocessor Revision: 0.0
CPU: MIPS R4400 Processor Chip Revision: 6.0
Main memory size: 128 Mbytes
Graphics board: Maximum Impact
```

```
; uname -a
IRIX sumatra  6.2 03131015 IP22
; hinv
1 250 MHZ IP22 Processor
FPU: MIPS R4000 Floating Point Coprocessor Revision: 0.0
CPU: MIPS R4400 Processor Chip Revision: 6.0
Main memory size: 128 Mbytes
Graphics board: High Impact
```

O2

```
; uname -a
IRIX sloep 6.3 12161207 IP32
; hinv
1 180 MHZ IP32 Processor
FPU: MIPS R5000 Floating Point Coprocessor Revision: 1.0
CPU: MIPS R5000 Processor Chip Revision: 1.0
Main memory size: 64 Mbytes
CRM graphics installed
```

3. VIEWPERF CDRS RESULTS.

Viewperf is a portable OpenGL performance benchmark program. The OpenGL Performance Characterization (OPC) project group has endorsed Viewperf as its first OpenGL benchmark. The OPC project group maintains a single source code version of the Viewperf code that is available to the public.

Currently, there are five standard OPC viewsets. We report only the numbers of the CDRS set for the four machines we have tested. See the previous appendix for specific hardware configurations.

Viewperf adopts a weighted geometric mean as the single composite metric for each viewset. The results of seven tests form a single composite metric.

	1	2	3	4	5	6	7	Mean
FireGL 4000	35.8	20.8	21.1	14.2	11.6	14.9	35.8	
MaxImpact	59.5	29.9	29.9	20.0	29.9	29.7	59.6	
HiImpact	36.1	24.1	18.0	12.0	14.4	17.5	36.1	
O2	19.9	11.7	9.9	7.4	6.6	8.5	19.9	

Table 4: CDRS viewset results