The PHIGS Toolkit: A Suite of Integrated Support Software for PHIGS

T.L.J. Howard, W.T. Hewitt, J.G. Williams, S. Larkin
Department of Computer Science / Manchester Computing Centre
University of Manchester
Manchester, M13 9PL
United Kingdom
email: toby@cs.man.ac.uk
phone: +44 61 275 6274
fax: +44 61 275 6236

Abstract

The Programmer's Hierarchical Interactive Graphics System (PHIGS) integrates application modelling with interactive computer graphics. It provides powerful functionality for the interactive manipulation of hierarchical models, but the interface provided to the programmer is at a low level, which leads to a lengthy familiarisation phase, and can make PHIGS difficult to use productively. This paper presents the PHIGS Toolkit – a suite of integrated software tools which provide a powerful support environment for PHIGS programmers, users of PHIGS-based applications, and educators.

1 Introduction: problems with PHIGS

Several years' experience working with research and commercial implementations of PHIGS has provided convincing evidence that, in general, programming with PHIGS is not easy [1]. Nevertheless, this observation does not denigrate the fundamental power and flexibility of the PHIGS model. Instead, it points to the need for providing a more high-level and user-friendly interface to the functionality of PHIGS. This paper identifies several specific problem areas that arise when programming with PHIGS, and describes an approach for solving these problems based on an integrated set of software tools called the PHIGS Toolkit. (In this paper, we assume the reader is familiar with the basic concepts and features of PHIGS, which is taken to include the functionality of PHIGS PLUS. For a full description of PHIGS the reader is referred to [2].)

What are the shortcomings of PHIGS when used to solve real problems? We summarise some important ones below:

- The procedural interface is low-level, which implies writing large amounts of code to support applications. Many functions which programmers soon discover they need (such as INQUIRE NUMBER OF ELEMENTS IN STRUCTURE) are not directly provided and must be implemented by the programmer in terms of existing PHIGS functions.
- All 'named' PHIGS objects, such as structures, labels, and pick identifiers, must be referred to using integers. In many cases, textual names would be far more useful and flexible.
- The traversal of structure networks is an 'atomic' process, and users do not have access to the dynamically changing Traversal State List. This makes debugging of incorrectly constructed structure networks difficult.
- The general purpose viewing model is necessarily complex, requiring a significant amount of understanding before it can be used to generate even simple views. This is a common source of difficulty for novice users.
- Inquiring about data in the Centralised Structure Store (CSS) is awkward. The inquiry functions provided by PHIGS for this purpose return the information in a particularly unhelpful format, as lists of structure identifiers and element numbers. This information will typically need to be 'decoded' to be of use to the application.
- The input tools supported by a conforming PHIGS implementation are low-level. Today, the majority of interactive graphical applications will expect to make extensive use of pull-down menus, and graphical icons for interaction.
- Structure networks are stored with fixed identifiers in archive files. This creates problems when reading archives since the identifiers may clash with ones already in use in the CSS.

We now give an introduction the philosophy behind the design and functionality of the PHIGS Toolkit, and then look at the tools themselves, from a point of view of their use for programmers. While it is not our intention in this paper to describe the specific implementation details of the tools and their underlying support software, we shall point out implementation issues which are of particular interest for PHIGS programmers.

2 An Overview of the PHIGS Toolkit

To use PHIGS effectively, it is necessary to develop a consistent framework in which to work. Given that we wish to improve the interface to PHIGS, where should any improved functionality be located?

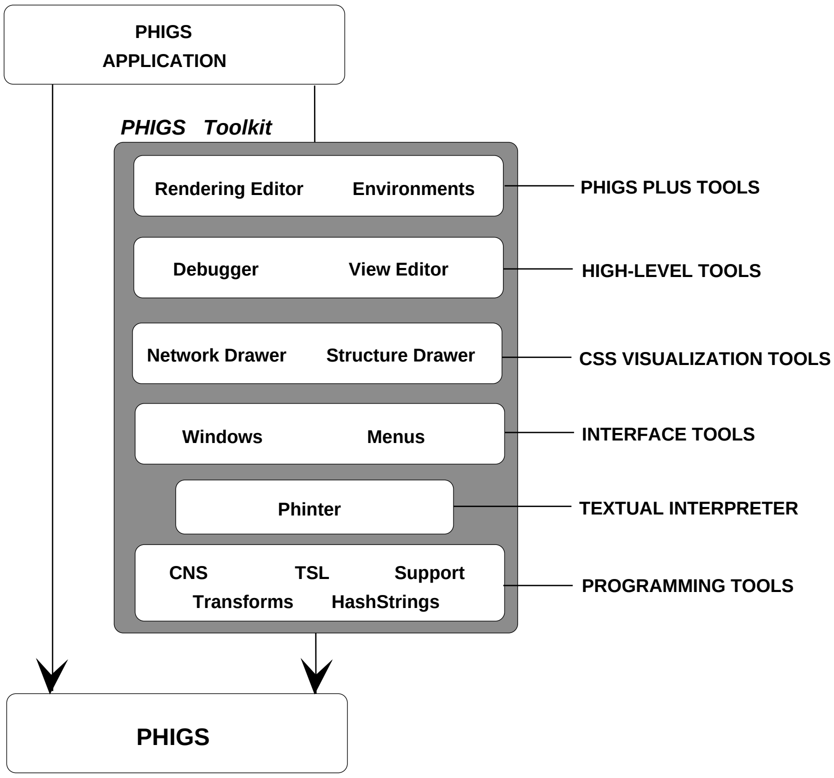

Figure 1: *The components of the PHIGS Toolkit.*

Is it preferable to extend an existing implementation of PHIGS into a 'super-PHIGS', or to work with a self-contained implementation of PHIGS and build interfaces in terms of its functions? If we are to provide 'extensions' which interface cleanly to PHIGS, and which are portable, the latter approach is the most appropriate.

The PHIGS Toolkit is a support library for the application programmer, providing a suite of functions to be used in conjunction with PHIGS. The architecture of the Toolkit is modular, as shown in Figure 1. The lower-level tools are generally quite simple single-purpose procedures, and are designed to help programmers to construct programs more quickly and reliably. The higher-level tools are more powerful, and provide programmers with facilities for visualizing and debugging structure networks. Some of these tools are interactive and have their own user interfaces built using the Toolkit. The following are the components of the PHIGS Toolkit:

- **Programming Tools**. These include the **Transformations Library**, containing functions for constructing and manipulating coordinate transformations; the **HashStrings Library**, which enables text strings to be used in situations where integers would normally be required; the **PHIGS Support Library**, providing utility functions, (such as COPY ELEMENT) which are not directly provided by PHIGS, as well as common sequences of function calls 'bundled up' into single functions; the **Traversal State List Library**, for interactively controlling and inquiring a simulated structure network traversal; and the **Colour Library**, for defining colour values using English words and phrases, and for interchangeably manipulating colours using various colour models.

- **The PHIGS Textual Interpreter (Phinter)**. This is a tool for reading files ('scripts') containing textual calls to PHIGS functions. Phinter may also be used interactively with a PHIGS string device or standard input.

- **CSS Visualization Tools**. These include the **Structure Network Drawer**, for automatically generating diagrams representing the topology of PHIGS structure networks and the **Structure Drawer**, which creates diagrams showing the elements in structures. These diagrams are themselves PHIGS structures, with a documented format.

- **Interface Tools**. These include the **Menus Library**, providing functions for constructing and manipulating menus built using PHIGS structures, and the **Windows Library**, providing functions for displaying and viewing structure networks in windows.
- **High-level Tools**. These are the **PHIGS Debugger**, a powerful tool for interactively simulating the traversal of structure networks. The traversal may be stepped through incrementally and the Traversal State List inquired at any stage; and the **View Editor**, a utility for interactively editing and experimenting with viewing parameters for a scene.
- **PHIGS PLUS Tools**. These are the **Lighting Editor**, a tool for visualizing and interactively editing light source representations; the **Reflectance Properties Editor**, a similar utility which enables interactive editing of reflectance properties elements; and the **Rendering Editor**, a tool which combines the functionality of the Lighting Editor and Reflectance Properties Editor.

The following sections describe the functionality of these tools in more detail, together with examples of their use.

3 The Programming Tools

3.1 The Transformations Library

PHIGS provides a number of utility functions for building transformation matrices. However, these are basic functions, and often programmers need to write many other additional routines to perform more complex transformations. The Transformations Library supplements these utility functions and gives greater support for vector arithmetic and matrix manipulation. These include functions for manipulating homogeneous coordinate transformations, computing vector cross and dot products, vector moduli, shears, rotations about arbitrary axes, matrix inversion, and window-to-viewport type mappings. The transformation functions optionally perform matrix concatenation automatically, removing the need to call the COMPOSE MATRIX function of PHIGS.

3.2 The HashStrings Library

The use of integers for naming PHIGS entities can easily become the source of many application bugs, since managing many integers which may or may not clash is awkward. The HashStrings Library makes the management of named entities much easier through the use of character strings. Strings may be used instead of integers to name structure identifiers, pick identifiers, names for filters, view indices and so on.

Each type of PHIGS entity has its own HashStrings table. The name of the table is arbitrary but all the functions in the Toolkit recognise a standard set of names, for example “structureid”, “labelid” and “colourindex”. The HashStrings library guarantees that each unique string in each table is allocated a unique integer. This scheme provides the basis for the higher-level Toolkit functions such as the Menu and Window Libraries.

PHIGS Toolkit applications are recommended to use the standard HashStrings tables but this is not mandatory. Tables are created with a specified range of integers and numbers will not be allocated outside this range. In order to use the higher-level Toolkit functions most of the standard hashtables must be present but the application may continue with its own method of allocating integers provided that they are outside of the hashtables’ range. This enables existing PHIGS applications to use the Toolkit without changing all their integer names.

3.3 The PHIGS Support Library

There are many low-level functions which PHIGS programmers would find useful but which are not included in the Standard. The intention of the Support Library is to carry on the provision of useful functions where the developers of PHIGS stopped. Examples of these functions are:

- COUNT NUMBER OF ELEMENTS IN STRUCTURE
- COUNT NUMBER OF CHILDREN OF STRUCTURE
- COPY SINGLE ELEMENT
- COPY RANGE OF ELEMENTS
- POST STRUCTURE WITH RELATIVE PRIORITY
- CHANGE POSTED STRUCTURE PRIORITY
- COPY WORKSTATION STATE TABLES
- SEARCH FOR NEXT PICK IDENTIFIER

3.4 The Colour Library

This is an implementation of the Colour Naming Scheme (CNS) method for specifying colours, closely based on the HSL colour model but using English names instead of HSL coordinates [3]. This method is considerably easier to obtain a required colour than by specifying three real numbers. For instance, it is easier to give the string “brown” than the the RGB triple (0.891, 0.685, 0.389). Lightness and saturation may also be given along with the hue name to give constructs such as “very-light vivid red” and “medium greyish blue”. The method integrates well with the HashStrings module because the same string may be used to give the RGB, HSL or HSV triple via the CNS as well as the index in the colour table. The library also contains functions for converting between colour models.

3.5 The Traversal State List Library

The traversal of a structure network is an atomic action, and it is not possible for the programmer to access the Traversal State List (TSL), which only exists while PHIGS is performing a traversal. In situations where, for example, a primitive has not been drawn in the intended colour, or transformations have not given the desired effect, it is often difficult to find out which are the incorrect elements, and in which structures they occur. The TSL library may be used to run a simulated traversal of a network, which is done by inquiring the contents of each element in turn and updating an internal copy of the TSL accordingly. The programmer may call a number of TSL inquiry functions to obtain information such as the current local and global transformation matrices, view index and attributes. The implementation of the traversal simulator is further described in [4]. The TSL library also provides functions for calculating the coordinate range (‘bounding box’) of a structure, which is useful for manipulating and viewing structures.

4 Phinter – A Textual Interpreter for PHIGS

Phinter (*PH*IGS *inter*preter) is a tool which accepts textual representations of calls to PHIGS procedures, and arranges for their execution, just as if they had been compiled as part of a program. Scripts of PHIGS commands may be stored in files and read in at run-time. Commands may also be entered interactively and this gives the user full access to the current PHIGS process.

Phinter can be used to save time by breaking the edit, compile and link cycle of PHIGS application programs. Phinter is not designed for building complete applications but is an ideal tool for small, test programs. For example, experimenting with different text fonts or prompt and echo types normally requires the application to be re-compiled. If a Phinter script is used to set the test value then this can be avoided.

Phinter scripts provide a very useful alternative to archive files for storing and restoring structure networks. A common problem with archives is that of structure identifiers which conflict with those already in the CSS. Phinter uses the HashStrings library for allocating structure identifiers which makes

```
open_structure "square"

polyline 5
0.100000 0.100000
0.100000 0.900000
0.900000 0.900000
0.900000 0.100000
0.100000 0.100000

close_structure

post_structure 1 "square" 0

redraw_all_structures 1 ALWAYS
```

Figure 2: *An example Phinter script to create a structure and post it to a workstation.*

the actual identifiers 'dynamic'. The format of Phinter scripts is much easier to read than that of archives and in addition, allows calls to non-structure building PHIGS functions such as POST STRUCTURE to be stored along with structure definitions. The example script shown in Figure 2 assumes that a workstation is open and creates a structure containing a single polyline. The structure is posted to the workstation and the display is updated.

5 The CSS Visualization Tools

5.1 The Structure Network Drawer

PHIGS structures are used to represent separate logical components of the model, and they may be linked to form hierarchies, using the EXECUTE STRUCTURE element. In general, a structure network has the topology of a directed acyclic graph. This is illustrated in Figure 3, which shows a human body modelled using a network of sixteen structures. This diagram was produced automatically using the Structure Network Drawer, and is itself a structure network. The format of the structures in the network is documented, to enable the diagram to be edited if required. There are three types of topology diagram:

- **Box.** This is the default type and draws a simple box node to represent each structure in the network.
- **Structure.** This type maps the contents of each individual structure to the node position and uses a box if there are no graphical primitives in the structure. This diagram gives an instant picture of the graphical components of a network and is very useful when designing efficient networks.
- **Structure Network.** Here, the contents of each sub-network is displayed in the node positions. This gives an idea of how the final picture is built up at the end of traversal.

Automatic creation of 'neat' diagrams to represent hierarchical data structures has been the subject of much research (see, for example, [5, 6, 7]). The approach used in the Toolkit is reasonably simple, and usually provides acceptable results. For situations where the layout of the diagram could be improved, an interactive diagram tidier is provided, which enables the nodes representing structures to be easily adjusted. More details of the layout algorithm used are given in [8].

5.2 The Structure Content Drawer

Getting a quick idea of the contents of a structure is difficult, because the contents of each element must be inquired from the CSS, and the list of elements scanned and interpreted. The Structure Content

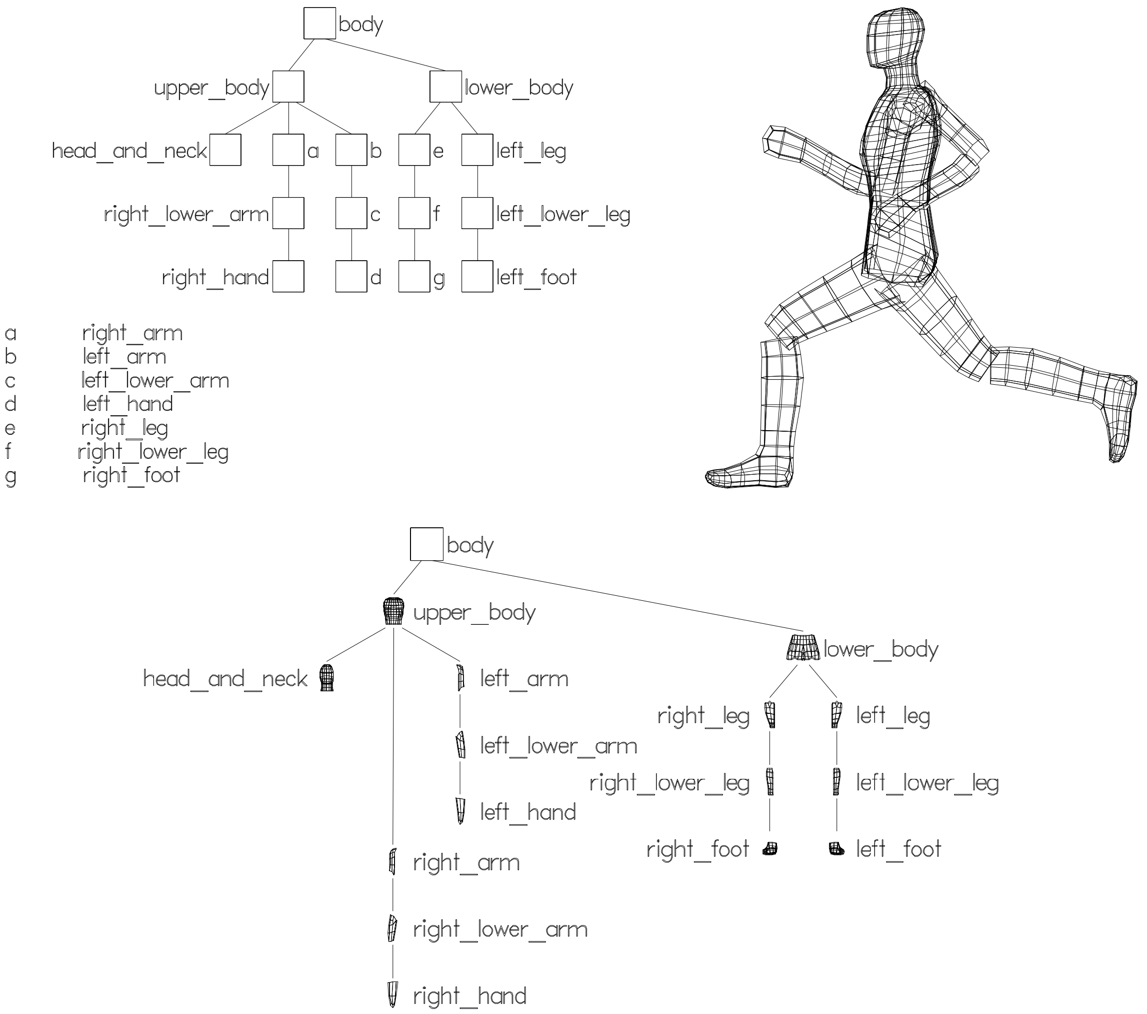

Figure 3: *Topology diagrams of a human body network. The top-left diagram is a default Box topology and below it is a Structure topology of the same model.*

Drawer performs a similar function to the Network Drawer, and generates a diagram of the structure contents. Figure 4 shows a map of Great Britain and a Structure Content diagram displaying the element types. The diagram is drawn as a table of elements starting with the first element at the top. Each item in the table contains the type of the element and alongside is the element number. The arrow may be set to point at any element and is used in the PHIGS Debugger to mark the current stage of traversal.

6 Interface Tools

6.1 PHIGS Input Devices

PHIGS has six types of logical input device and although they all share functionality across implementations, the ways in which they are realised are often quite different. They may all use different physical devices, such as a mouse or dialbox, and different style graphics for valuator slider bars and string input boxes. This often leads to a different ‘look and feel’ for a given application on different PHIGS platforms.

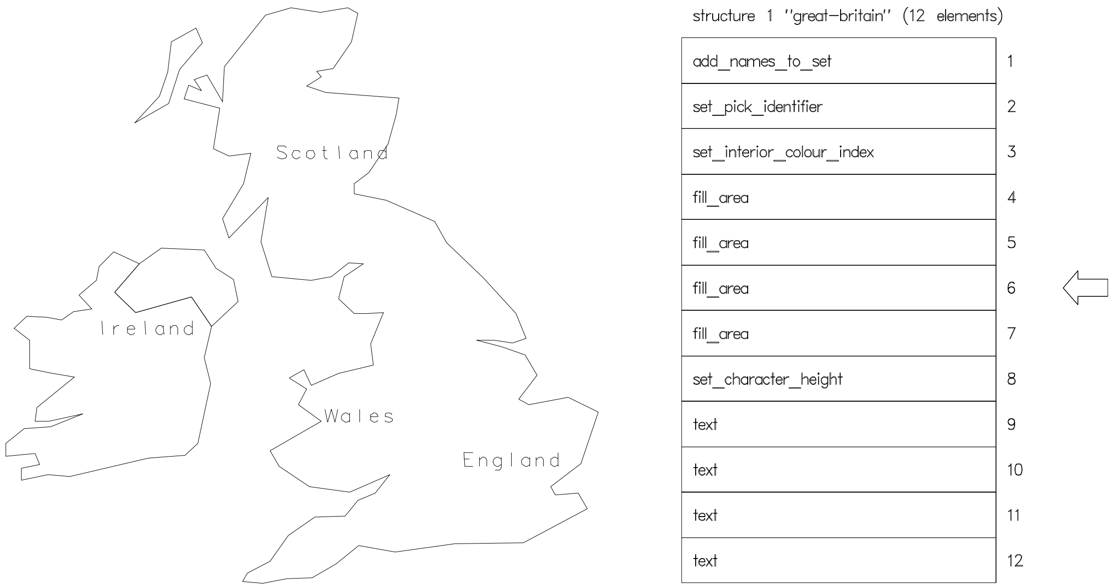

Figure 4: *A PHIGS structure and its Structure Content diagram.*

6.2 The Windows Library

Today many applications use multiple on-screen windows for viewing scenes, displaying textual information and for interaction. Most PHIGS applications implement the logical input/output PHIGS workstation as a single X window. Management of the workstation is then handled by the current window manager. A PHIGS application with multiple windows therefore requires multiple workstations, and this can create a major headache for the programmer. Each PHIGS workstation has its own set of view, lighting, colour and attribute bundle tables, and to share information between the windows requires careful copying of these tables.

The Windows Library provides the application with multiple windows inside a single PHIGS workstation window. Each window has a banner, frame and viewing area and may be iconised. Figure 5 shows a Structure Window viewing a model, and a Terminal Window displaying textual information. Each window is implemented as a PHIGS structure and is allocated a unique view representation. There are four types of window:

- **Structure.** This is the default type and may be used for viewing any PHIGS structures. A virtual camera interface is implemented for this window type which enables the programmer to specify views using simplified parameters such as camera position and point of interest.
- **Topology.** This type is intended for viewing Topology diagrams and has functions for zooming and panning.
- **Content.** This is similar to the Topology window but is used for viewing Structure Content diagrams. A scrolling function is supplied.
- **Terminal.** This type is intended for displaying textual messages and is automatically scrolled to display the most recently written text.

6.3 Menus and Rotators

Menus are a fundamental requirement of interactive graphical applications and most PHIGS applications use the choice input device. This often limits the menus to a list of character strings and makes

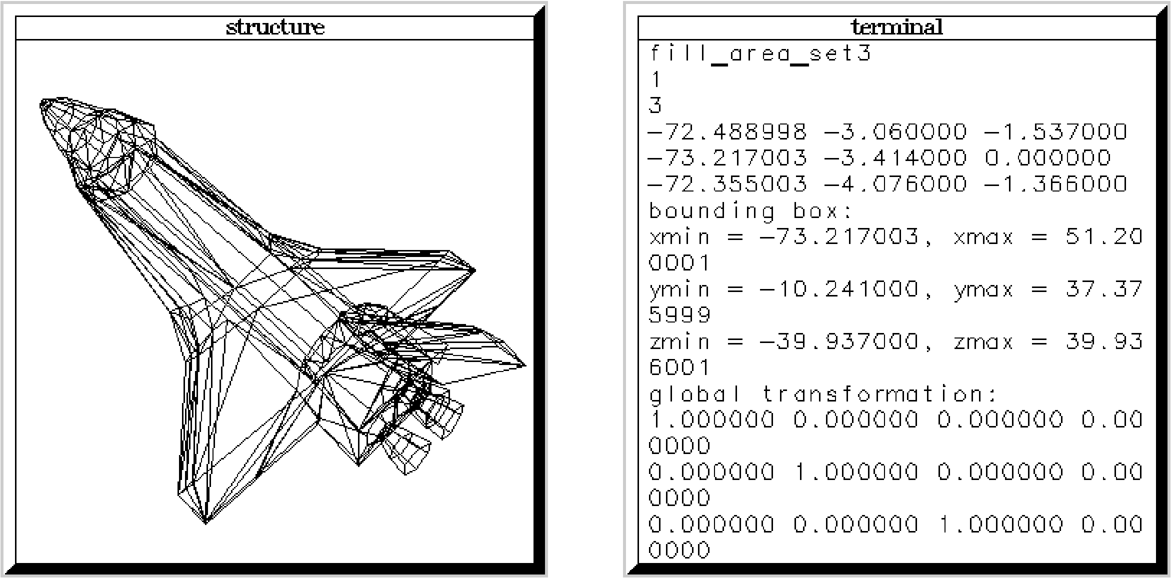

Figure 5: *Structure and Terminal windows. The Structure window (left) shows a perspective view of a 3D model using the virtual camera and the Terminal window (right) displays text which has been automatically broken across lines and scrolled.*

use of implementation-dependent features of PHIGS. The Toolkit menus provide an implementation-independent method of creating menus which may contain character strings and structures. There are three types of menu, as shown in Figure 6, and each one is realised as a structure:

- **Box.** Each menu item may contain a character string or a structure.
- **User.** No restrictions are made on the contents of menu items, and every item is a structure supplied by the application.
- **Rotator.** This is a special form of User Menu and comprises an arrangement of four arrows. Rotators are a useful interaction technique for specifying real values such as rotation angles, and may be used in place of standard PHIGS valuators.

7 The High-level Tools

7.1 The PHIGS Debugger

There are natural analogies between debugging the traversal of structure networks, and debugging the execution of programs written in an imperative language such as C. In each case, there is the notion of an 'execution' which proceeds sequentially. The PHIGS Debugger offers similar functionality to traditional language debuggers, with operations to step to the next element, set breakpoints and tracepoints, and deposit and inquire values in elements. The user may interactively step through the traversal process, element by element, and observe how the picture is built up and how the TSL changes. The Debugger has a graphical user interface, as shown in Figure 7, built using the Toolkit interface tools. Windows display the topology of the network being debugged, the contents of the current structure, the structure network itself at the current stage of traversal, and a terminal window for textual information. For full details of the functionality and implementation of the Debugger, see [4].

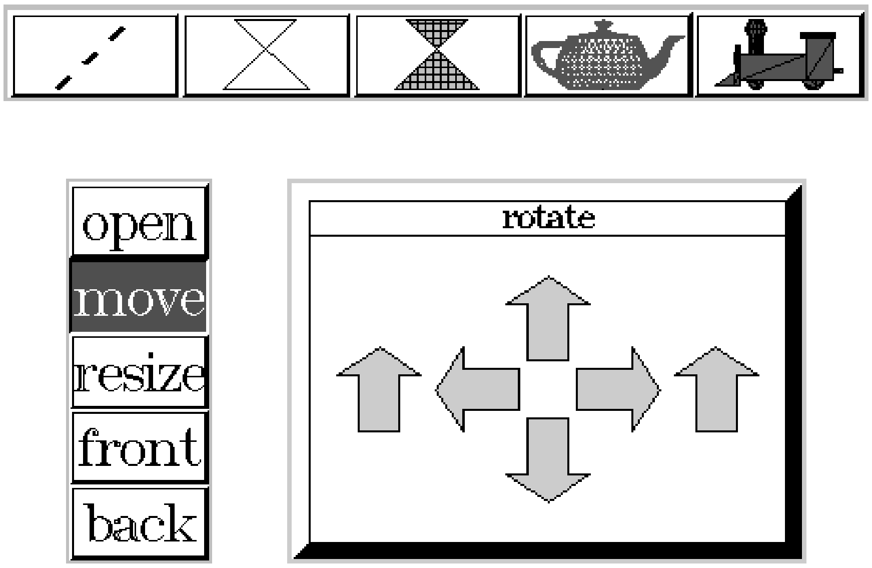

Figure 6: *Box Menus and 3D Rotator. The top menu has a menu path of* RIGHT *and each item contains a PHIGS structure. The bottom-left menu has a* DOWN *path and has text item 2 highlighted.*

7.2 The View Editor

The View Editor is an interactive tool which may be used to experiment with different viewing parameters. It enables the user to observe how changing each of the many view parameters affects the final view. The editor has proved to be a valuable teaching aid and also helps experienced programmers to quickly establish the required viewing parameters for a scene.

The View Editor is shown in use in Figure 8 and has a window for each stage in the viewing pipeline, from World Coordinates to Device Coordinates. Each window displays the scene along with other structures representing the viewing parameters. The World Coordinates window contains the WC and VRC coordinate axes. The PRP, view window and front, back and view planes are shown in the VRC window. The NPC window displays the unit cube and the projection viewport. All these parameters may be changed interactively using rotators or standard PHIGS string input. The virtual camera may be used in each window, apart from the DC window, to give a view of the model independent from the view currently being edited.

8 PHIGS PLUS Tools

Recent work has been concerned with designing tools for use with PHIGS PLUS. This provides the facilities for lighting and shading and for additional PHIGS output primitives. The **Environment Library** is intended to make rendering of PHIGS structures simpler using a similar method to that of PHIGS Toolkit windows. An environment is itself a PHIGS structure and contains elements which control which light sources are active, which shading method and reflectance model is used, depth cueing, face culling and face distinguishing. For rendering, structures are posted to an environment which results in them being executed from the environment structure.

The **Lighting Editor** may be used to interactively create and edit all types of light source and is useful for experimenting with different lighting effects. The reflectance properties of an object determine if it reflects light as if it were made of metal, plastic or some other material and may be set interactively using the **Reflectance Properties Editor**. Both editors are combined in the **Rendering Editor** and all are implemented in a similar style to the Debugger and View Editor. This gives a consistent look and feel to the Toolkit across editors and PHIGS implementations. The Lighting Editor has a Preview

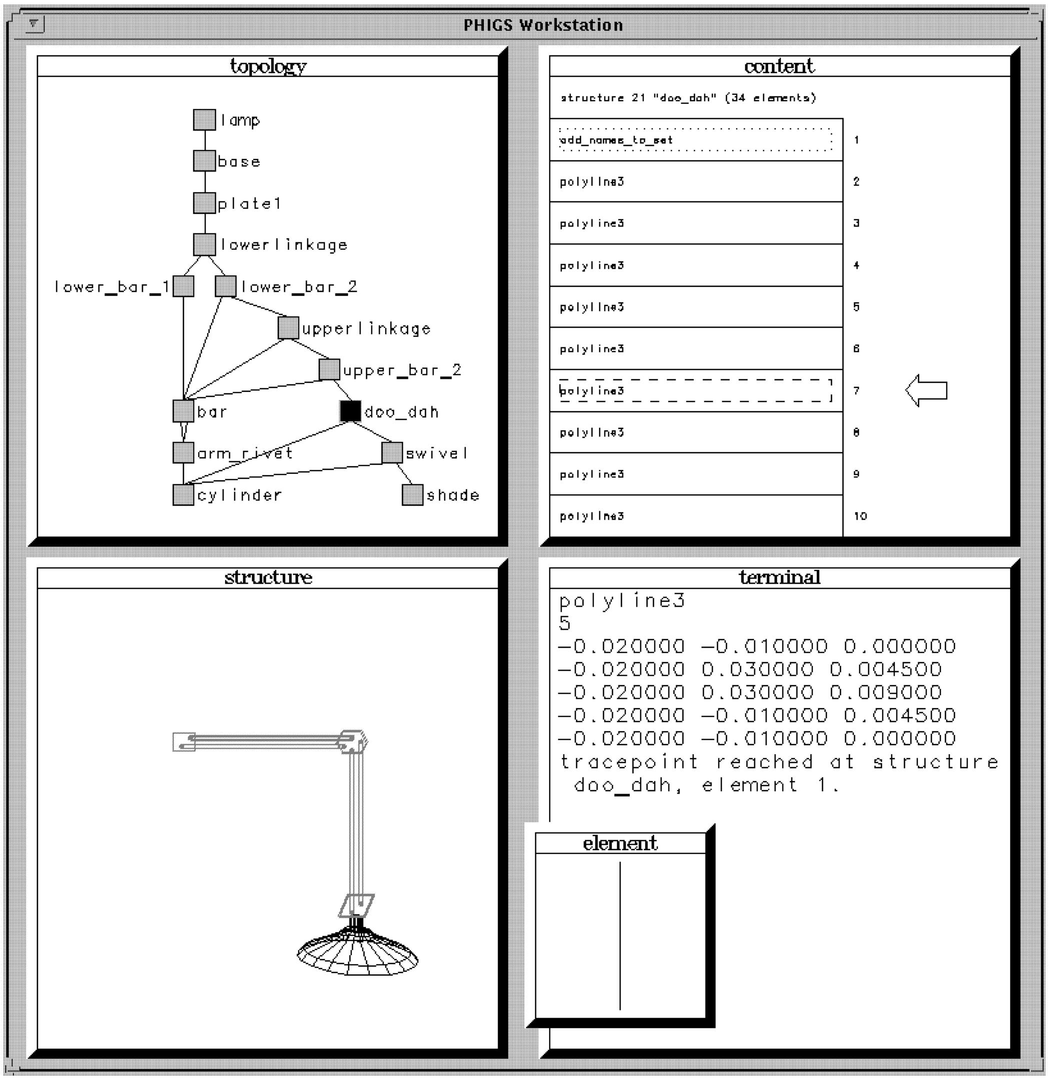

Figure 7: *The PHIGS Debugger. The top-left window shows the network being debugged with the structure currently being traversed highlighted in black. The top-right window is a diagram of the structure, with the current element being traversed highlighted in the dashed box. The bottom-left window contains the result of the traversal, and all textual information is displayed in the bottom-right terminal window.*

Window for displaying the scene in wireframe with graphical representations of the active lights. The fully rendered scene is displayed in the Render Window, and both windows are shown in Figure 9. The Attributes Window is used in all three editors and displays a diagram of the current lighting or depth cueing attributes. From left to right the windows in Figure 10 illustrate ambient and directional light, positional light, spot lights and depth cueing.

9 Conclusions

It is the policy of the PHIGS Toolkit project to support as many PHIGS implementations as possible, although since the publication of the standard ISO PHIGS C binding we have restricted support to implementations which comply with this standard. To date, the PHIGS Toolkit has been ported to the following C PHIGS programming environments:

- SunPHIGS on SunOS (Fortran support also provided)

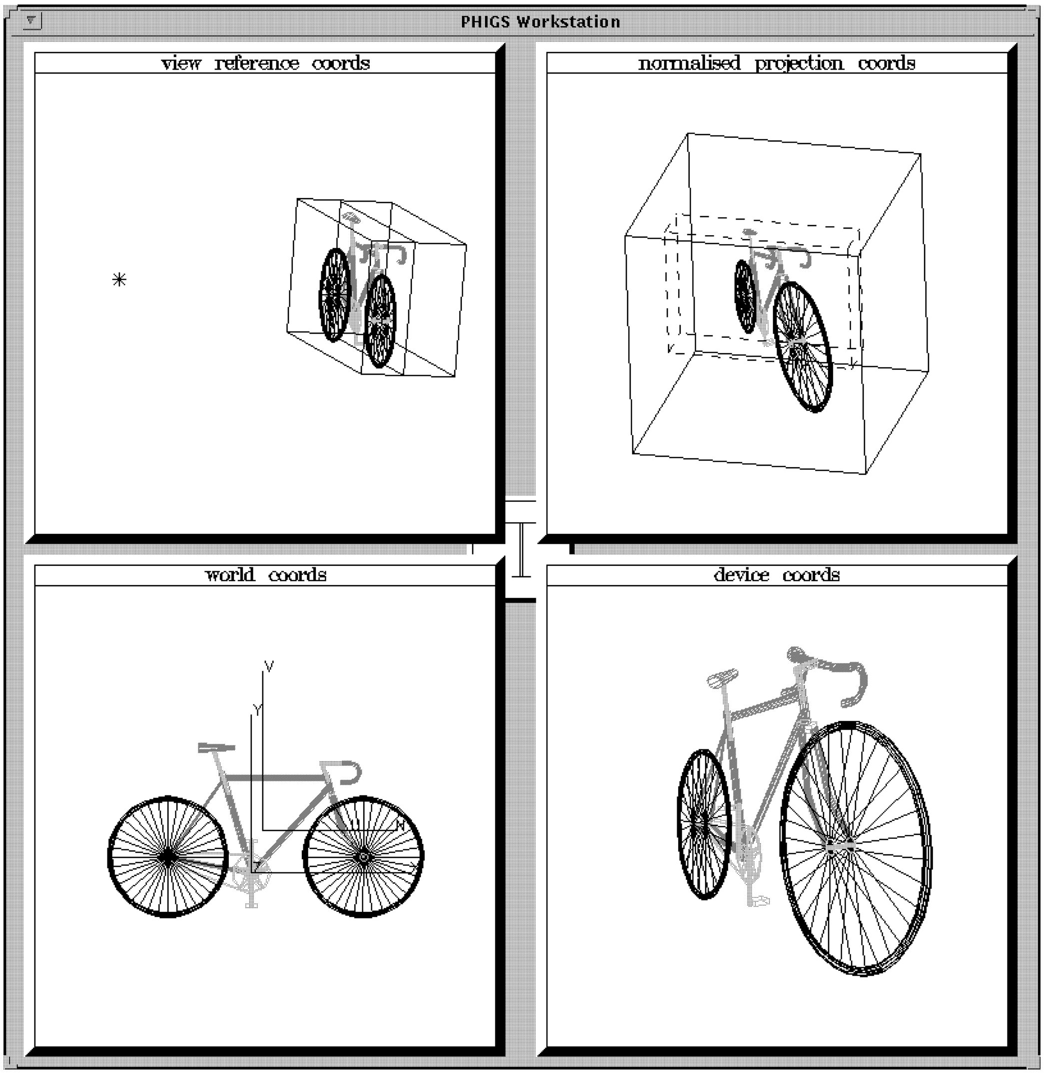

Figure 8: *The View Editor. The bottom-left window shows the model in World Coordinates with the XYZ and UVN axes. The top-left window displays the PRP as an asterisk and the view window and view planes are drawn around the model space. The dashed box in the top-right window represents the projection viewport and is surrounded by the unit cube. The Device Coordinates window at the bottom-right shows the model as it would appear normally in the complete workstation window.*

- HP PHIGS on HP-UX
- graPHIGS on AIX
- PEX-SI on SunOS
- DEC PHIGS on VAX/VMS (PTK 2.0 only; Fortran support also provided)

We have used the Toolkit extensively for research and teaching at Manchester, and several other UK academic institutions are using the Toolkit. Once the fundamental PHIGS concepts have been grasped, students have been able to produce effective user-interfaces to their applications and the virtual camera, in particular, has helped many get over the stumbling block of the viewing pipeline. One unusual application of the Toolkit was in a project to generate a model of Rodin's sculpture *The Thinker* for an advertising poster [9].

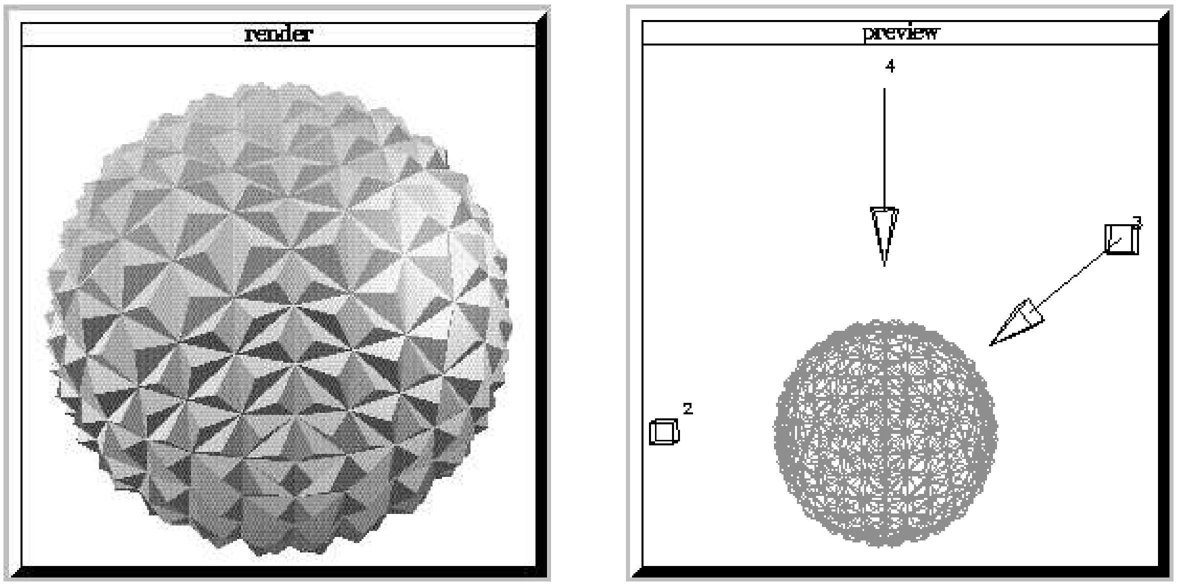

Figure 9: *The Lighting Editor, showing the Render Window (left) and Preview Window (right). The arrows in the Preview Window represent active light sources and the numbers indicate their indices in the state table.*

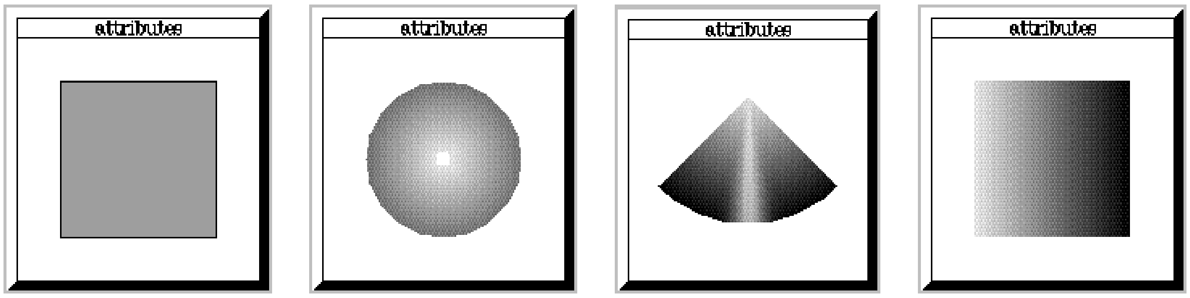

Figure 10: *Four instances of the Attributes Window applied to ambient and directional, positional, and spot lights and depth cueing.*

Although developing the Toolkit has made us extremely aware of the deficiencies and idiosyncrasies of PHIGS, it is still our firm belief that PHIGS is a powerful system for performing general-purpose modelling with computer graphics.

Acknowledgements

The PHIGS Toolkit has been under development since 1988, arising from work on the *KRT*^{3} project supported by SERC grant GR/D/20533. Subsequent work was funded by a Digital Equipment Corporation grant, and by the UK Advisory Group on Computer Graphics. Following original work by the authors and Karen Wyrwas, many of our colleagues have made valuable contributions, and it is our pleasure to be able to thank: Roger Hubbold, Tony Arnold, David Yip, Alan Murta, Daf Tregear, Mike Keeley, Rachel Lau, Andy Grant, Chris Lilley, Fenqiang Lin, Neil Gatenby, Ian Whittaker, Martin Preston, Mary McDerby, Julian Gallop, Eric Thomas and Anne Mumford. Thanks also to our beta testers Ken Brodlie, Colin Pinks, I. Haneef, Howard Parish, David Johnston, Nick Hill and Gurm Bacchus, and to the many students who have been willing guinea pigs.

Appendix: Obtaining the PHIGS Toolkit

The PHIGS Toolkit is freely available for non-commercial use. The software distribution includes full source code, example programs and documentation, and is available by from HENSA (Higher Education National Software Archive) at the University of Kent and from the University of Manchester. The authors will be pleased to supply full details on request, from `phigstoolkit@cs.man.ac.uk`.

References

[1] T.L.J. Howard. Evaluating PHIGS for CAD and general graphics applications. *Computer Aided Design*, 23(4):244–251, May 1991.

[2] T.L.J. Howard, W.T. Hewitt, R.J. Hubbold, and K.M. Wyrwas. *A Practical Introduction to PHIGS and PHIGS PLUS*. Addison-Wesley, Wokingham, UK, 1991.

[3] T. Berk, L. Brownston, and A. Kaufman. A new color-naming system for graphics languages. *IEEE Computer Graphics and Applications*, 2(3):37–44, May 1982.

[4] T.L.J. Howard, W.T. Hewitt, and S. Larkin. An interactive debugger for PHIGS. In C.E. Vandoni and D.A. Duce, editors, *Proceedings Eurographics '90*, pages 177–193, Amsterdam, 1990. Eurographics, North-Holland.

[5] L.A. Rowe, M. Davies, E. Messinger, C. Meyer, C. Spirakis, and A. Tuan. A browser for directed graphs. *Software, Practice and Experience*, 17(1):61–76, January 1987.

[6] G. Robins. The ISI Grapher: a portable tool for displaying graphs pictorially. Technical report, Intelligent Systems Division, Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, California, CA 90292–6695, USA, 1987.

[7] E.R. Gasner, S.C. North, and K.P. Vo. Dag – a program that draws directed graphs. Technical report, AT&T Bell Laboratories, 1988.

[8] T.L.J. Howard. *TopDraw*: A structure network visualiser for PHIGS. In *Proceedings Eurographics UK Conference*, pages 107–124, 1990.

[9] J.G. Williams, A.D. Murta, and T.L.J. Howard. Modelling Rodin's Thinker: A case study combining PHIGS and ray-tracing. Project report, University of Manchester, 1993.

[10] T.L.J. Howard. An annotated PHIGS bibliography. *Computer Graphics Forum*, 8:262–265, 1989.

[11] T.L.J. Howard. A shareable centralised database for KRT^3 – a hierarchical graphics system based on PHIGS. In G. Maréchal, editor, *Proceedings Eurographics '87*, pages 465–480, Amsterdam, 1987. Eurographics, North-Holland.

[12] T.L.J. Howard, W.T. Hewitt, J.G. Williams, and S. Larkin. *The PHIGS Toolkit User Manual*. Project Report ptk/100, University of Manchester, 1991.