Computer Graphics and Image Processing Laboratory for Undergraduate Instruction.

Dr. Dennis Mikkelson
University of Wisconsin-Stout

The ILI funded laboratory and curriculum development project described in this paper provided UNIX workstations for undergraduate courses in computer graphics and image processing at the University of Wisconsin-Stout. Prior to this project, the computer graphics course was taught on 80286 class personal computers using an implementation of the Graphical Kernel System (GKS) for PASCAL. This project allowed the computer graphics course to move to a C/Unix environment using the Programmer's Hierarchical Interactive Graphics System (PHIGS). Some basic software tools were developed to facilitate the use of PHIGS in an instructional setting. PHIGS in the C/Unix environment has provided an excellent environment for teaching computer graphics. The image processing course has also benefited greatly from the move to the workstation environment. An image processing "workbench" to provide a highly interactive environment for students to develop and test image processing algorithms is under development.

1.0 INTRODUCTION

The Department of Mathematics, Statistics and Computer Science at UW-Stout has offered a course in computer graphics for the last ten years and has offered a course in image processing for the last six years. During this period of time the hardware/software environments available for computer graphics and image processing have evolved rapidly. While the environments have changed rapidly, a solid understanding of the fundamentals of computer graphics and image processing remains the primary goal for these courses.

A major upgrade in the hardware/software available for these courses was provided by the NSF/ILI project described in this paper. The NSF/ILI project provided five DEC 3000 ALPHA/AXP workstations for the graphics and image processing courses. The required PHIGS software was provided by Digital Equipment Corporation under their College Software Library Grant program. This paper gives some considerations for using PHIGS in an introductory computer graphics course and how the project changed the graphics course. The effect of the project on the image processing course is described more briefly, since software development for the image processing

course is still in progress.

2.0 USING PHIGS IN AN INTRODUCTORY COMPUTER GRAPHICS COURSE

One fundamental decision to be made in the design of a computer graphics course is the graphics environment to use for student exercises. Some of the many possibilities are listed below:

- Develop the entire environment from scratch during the course. That is, using the most basic graphics operation, PutPixel, implement the graphics output primitives and use the resulting set of primitives for all student exercises.

- Use a commonly available set of graphics primitives such as the BGI graphics available with Borland's compilers on PCs.

- Use a graphics system designed for teaching graphics such as the Simple Raster Graphics Package (SRGP), or Simple PHIGS (SPHIGS) system from Foley and Van Dam.

- Use a standard graphics system such as the Graphical Kernel System (GKS), Programmer's Hierarchical Graphics System (PHIGS) or OpenGL.

Each of these possibilities has advantages and disadvantages. During the first two years that the graphics course was offered, it used the first approach. The primary disadvantages to this approach are the following:

- Too much of the course is spent developing the graphics primitives, leaving little time for the students to construct exciting graphics programs.

- Graphical input capabilities are quite limited. Implementing such input devices as pop-up choice devices, pop-up valuators and pop-up string input devices would take much more time than is available in such a course.

- The time spent developing and using such a "home-grown" graphics system may seem to be wasted when students leave the course. Specifically, the students will not see such a system again when they begin their career or graduate school.

In an effort to address these concerns as completely as possible, it was decided to use a standard graphics system. In order to do this as quickly as possible on the hardware available at the time, a version of GKS was implemented, carefully following the PASCAL language binding specification. This was eventually developed to a level 0b GKS implementation. This locally developed version of GKS provided a quite effective environment for teaching computer graphics.

However, as time progressed, the graphics course also had to progress. Specifically, students frequently ended up working in a C/Unix environment upon graduation, so it would be beneficial to provide them with more experience in that environment. Also, PHIGS is a newer 3D standard that provides a better environment for constructing interactive 3D programs. Finally, entry level UNIX workstations would provide significantly more computational power for animation and interactive 3D graphics programs than the 80286 based PCs in use at the time. The equipment provided by this NSF/ILI project made the change to PHIGS in a UNIX/C environment possible.

PHIGS provides a powerful environment for developing interactive 3D graphics programs. Some of the strong points of the PHIGS environment include:

- Graphical models are defined and stored in "structures" containing graphical primitives, primitive attributes, modelling transformations, etc. Animation of a model is then easily accomplished by editing a structure and changing the modelling transformations.

- Viewing a graphical model or collection of models is separate from the modelling process. One or more views, based on such things as the position of an observer in world coordinates, which direction is up, etc., can be specified and saved in the workstation state table. A "fly by" sequence of views of a graphical model is then easily accomplished by redefining the view.

- A collection of logical input devices including PICK, LOCATOR, CHOICE, STRING and VALUATOR are available. While not perfect, these logical devices can be used and programmed in a portable way. In particular, it is not necessary to go outside of PHIGS and use an underlying window system to get input such as locations, values selected by a graphical slider and choices from pop-up menus. This is a significant advantage, since teaching the details of programming in a windowing environment such as the X-Window System would take too much time in a one semester course on computer graphics.
- PHIGS (at least in it's newer form, PHIGS+) provides good support for rendering 3D primitives together with lighting and shading.

While PHIGS is a powerful graphics programming environment, it is not easy for beginning graphics programmers to use. Some of the difficulties for beginning graphics programmers come from the flexibility of PHIGS and resulting complications. Several situations in which this is troublesome are specifying views, initializing input devices and specifying colors. In these cases, some simple higher level utilities can isolate the beginning graphics programmer from a mass of details, and help focus on the graphical concepts.

2.1 VIEWING

The PHIGS transformation pipeline involves five coordinate systems as shown below:

Modelling Stage			Viewing Stage			Workstation Transformation		
Modelling Coordinates	->	World Coordinates	->	View Reference Coordinates	->	Normalized Projection Coordinates	->	Device Coordinates

At the modelling stage, objects are constructed in any convenient coordinate system (modelling coordinates) and mapped to a common world coordinate system using a composite modelling transformation. This stage constructs the scene that can then be viewed in different ways from different positions. The viewing stage maps the scene to a coordinate system centered at the point the observer is looking at (the View Reference Point) and ultimately maps the scene to all or part of the 3D unit cube known as Normalized Projection Coordinates (NPC). The mapping is made in such a way that a parallel projection of the NPC unit cube parallel to the z-axis will produce the desired view. One or more views of scenes can be constructed in NPC in this way. The workstation transformation is an aspect ratio preserving transformation that can be thought of as copying all or part of NPC to the display surface. Typically, the workstation transformation can be left with its default value which ultimately produces a parallel projection of the NPC unit cube onto a square window on the workstation display.

A "view" in PHIGS is constructed using three PHIGS functions:

```
peval_view_ori_matrix3(     const Ppoint3           *vrp,
                            const Pvec3             *vpn,
                            const Pvec3             *vup,
                            Pint                    *error_ind,
                            Pmatrix3                matrix     )
```

This constructs the view orientation matrix that maps World Coordinates to View Reference Coordinates. The programmer must specify the View Reference Point (the point where the observer is looking), the View Up Vector (which direction is up for the observer) and the View Plane Normal (a vector perpendicular to the virtual viewing screen).

```
peval_view_map_matrix3(     const Pview_map3    *mapping,
                            Pint                *error_ind,
                            Pmatrix3            matrix      )
```

This constructs the view mapping matrix that maps View Reference Coordinates to Normalized Projection Coordinates. The programmer must specify front and back clipping planes, a "window" on the virtual viewing screen, parallel or perspective projection and the region in Normalized Projection Coordinates that the specified region in View Reference Coordinates will be mapped to.

```
pset_view_rep3(     Pint                wsid,
                    Pint                view_index,
                    const   Pview_rep3  *rep        )
```

This records the view orientation matrix and view mapping matrix together with clipping information in the workstation state table. The programmer must specify a clipping region in Normalized Projection Coordinates and flags to indicate which of the clipping boundaries are to be used.

These functions use parameters constructed from the following data types:

```
                                /* enumerated types for projection          */
                                /* types and clipping flags             */
typedef enum {
        PTYPE_PARAL,
        PTYPE_PERSPECT
} Pproj_type;
```

```
typedef enum {
        PIND_NO_CLIP,
        PIND_CLIP
} Pclip_ind;
```

```
                                /* limit structures for describing          */
                                /* 2D and 3D "boxes"                        */
typedef struct {
        Pfloat          x_min;
        Pfloat          x_max;
        Pfloat          y_min;
        Pfloat          y_max;
} Plimit;

typedef struct {
        Pfloat          x_min;
        Pfloat          x_max;
        Pfloat          y_min;
        Pfloat          y_max;
        Pfloat          z_min;
        Pfloat          z_max;
} Plimit3;

                                /* 3D point and vector structures */
typedef struct {
        Pfloat          x;
        Pfloat          y;
        Pfloat          z;
} Ppoint3;

typedef struct {
        Pfloat          delta_x;
        Pfloat          delta_y;
        Pfloat          delta_z;
} Pvec3;

                                /* 4X4 matrix type for transformations      */
typedef Pfloat Pmatrix3[4][4];

                                /* special structures for                   */
                                /* peval_view_map_matrix3 and pset_view_rep3 */
typedef struct {
        Plimit          win;
        Plimit3         proj_vp;
        Pproj_type      proj_type;
        Ppoint3         proj_ref_point;
```

```
        Pfloat          view_plane;
        Pfloat          back_plane;
        Pfloat          front_plane;
} Pview_map3;
```

```
typedef struct {
        Pmatrix3        ori_matrix;
        Pmatrix3        map_matrix;
        Plimit3         clip_limit;
        Pclip_ind       xy_clip;
        Pclip_ind       back_clip;
        Pclip_ind       front_clip;
} Pview_rep3;
```

While this is a very flexible way to specify views and the programmer has control over all aspects of the view, this is more flexibility than is typically needed. Beginning graphics programmers tend to get lost in the details of providing useful values for all of the fields of all of the structures used.

For most graphics programs this is more flexibility than is needed. A somewhat higher level easier to use routine with less flexibility saves the beginning PHIGS programmer a lot of time. In most cases, one would like to specify the view more concisely. That is, it is much more natural to specify basic information about the observer, type of projection and where the projected scene is to appear. The other parameters required by the above PHIGS functions can then either be calculated from the specified information, or provided with usable default values. This is easily accomplished by constructing a simple utility routine such as:

```
void BuildView(     Pint            wsid,
                    Pint            view_index,
                    Ppoint3         vrp,
                    Ppoint3         cop,
                    Pvec3           vuv,
                    Pfloat                  view_angle,
                    Pproj_type      projection_type,
                    Plimit3         NPC_viewport    );
```

The first two input parameters are the workstation ID and view_index being set. In addition, information about the observer is provided by the parameters vrp, cop and vuv specifying the View Reference Point, View Up Vector, and the Center of Projection (the position of the observer). The view_angle parameter specifies the angle subtended by the virtual screen from the point of view of the observer. The virtual screen is assumed to be square and centered around the View Reference Point. The projection_type parameter selects a parallel or perspective projection. Finally, the NPC_viewport selects where the transformed scene will appear in Normalized Projection Coordinates.

The BuildView function described above is quite simple to use for beginning graphics

programmers. In addition it makes it very easy to carry out common operations. A "fly by" animation sequence can be easily accomplished merely by repeatedly calling BuildView passing in different Center of Projection points along the observer's path and updating the workstation. Similarly, zooming in/out can be accomplished by changing the view_angle. The virtual observer can look at different positions merely by changing the View Reference Point.

2.2 INPUT DEVICES

Similar difficulties arise when dealing with the logical input devices. While all of the devices can be used quite easily in their default states, some must be initialized to be used in a meaningful way. For example, in order to be useful, the pop-up choice device must be initialized with a list of character strings representing menu choices. To be really useful, pop-up valuators need to have minimum and maximum values and labels specified appropriately. Finally, in-order to use different prompt and echo types such as rubber-band lines, rubber-band boxes and cross-hair cursors with a locator device, it too must be initialized.

Unfortunately, initializing the input devices is often quite cumbersome. The device must first be set to request mode, initialized and then set to the desired mode (sample, request or event). The most troublesome part of this is the initialization itself. The first problem one encounters is that the position of the input device must be specified in device coordinates. The particular device coordinates used in an implementation can vary, so it is typically necessary first to use an inquire function to find out the range of device coordinates. The initialization functions have some reasonably straightforward parameters, followed by a data record that contains many of the low level details for the initialization. These data records can be very elaborate. For example, the data record for initializing the locator is a C struct that takes more than a page to list. This structure uses other structures that take another page to list. It is very difficult to work through these structures and initialize the device from scratch without a detailed knowledge of PHIGS.

While much of the complexity may be justified on the basis of flexibility (for example, the rubber-band box for the locator may be hollow or solid with different fill colors and patterns) this flexibility comes at a cost and is much more than is needed for an introductory graphics class. A higher level routine such as:

```
void InitLocatorDevice3(    Pint        wsid,
                            Pint        device,
                            Pop_mode    mode,
                            Pint        loc_pet,
                            Pint        view_ind,
                            Ppoint3     init_pos )
```

can be provided by the instructor. This allows the student to specify the crucial information needed to initialize the locator and uses reasonable defaults for the other values needed. Here wsid is the workstation ID, device specifies the device number to use (typically 1). The input mode, request, sample or event as well as the prompt and echo type (rubber-band line, cross-hair, etc.) are specified. Finally, the initial position for the locator is specified by the 3D world coordinate point init_pos and the viewing transformation that should be used when PHIGS

transforms it to the display surface.

Similar routines have been implemented and used for the other input devices. These routines allow the beginning graphics programmer to easily get graphical input for interactive 3D graphics programs.

2.3 COLORS

As a final example of complexity in PHIGS that can be easily avoided with simple utility routines, consider the problem of setting colors in PHIGS. PHIGS allows colors to be specified using an index into a color table, or by specifying components in one of several color spaces, rgb, hls, etc. This leads to approximately a page of C structures to work through to specify a color. The resulting code directly using the PHIGS types is not actually that cumbersome. For example, the statements below set the interior color for subsequent fill area primitives:

```
Pgcolor color;

color.colr_type = PCOLR_RGB;
color.colr_value.colr_rep.rgb.red   = 0.5;
color.colr_value.colr_rep.rgb.green = 0.5;
color.colr_value.colr_rep.rgb.blue  = 0.5;
pset_int_colr( &color );
```

However, it is convenient to hide some of these details in a higher level routine to construct a Pgcolor structure in a specific color space, such as RGB.

3.0 EFFECT OF PROJECT ON GRAPHICS COURSE

With a collection of utility routines to hide some of the low level details of PHIGS, PHIGS becomes an excellent environment for teaching the 3D aspects of an introductory computer graphics course. In order to get a good understanding of the basic concepts in computer graphics it is still necessary to do some introductory exercises using the most basic graphics primitive, PutPixel. Currently three introductory programming exercises are given. In these exercises the student fills in values in a 2D frame buffer that is displayed by an X-Windows/Motif program provided by the instructor. These exercises are assigned very early in the course as the corresponding 2D graphics concepts are introduced.

Introductory 2D Programming exercises:

1. Draw a fractal, eg. the Mandelbrot set. This gives the student some initial experience with mapping points from the 2D plane to the corresponding row and column

in a simulated raster display.

2. Implement some basic 2D primitives, such as lines and circles using Bresenham's algorithms and draw a picture using them on the simulated raster display.

3. Implement the basic 2D transformations and use them together with the 2D primitives from #2 to produce a simple animation.

At this point the students are generally ready for PHIGS. As PHIGS and 3D graphics concepts are being introduced, several very simple PHIGS exercises are assigned to gain some initial familiarity with PHIGS. Eventually, as the corresponding concepts are covered in class, more involved examples and exercises are given. Finally, the students design and implement a larger interactive 3D graphics program in groups of two or three students.

PHIGS Programming exercises:

4. Using PHIGS 2D primitives, draw a picture.

5. Draw four simultaneous views (top, front, side and oblique) of a 3D wire frame aircraft.

6. Implement a menu driven program to interact with a simple scene consisting of an object over a grid. Operations include rotating the object about axes parallel to the x, y or z axis using valuators, zooming in or out using a valuator and changing the view reference point and center of projection using the locator device.

7. Design and implement an 3D graphics program allowing the user to interact with objects in a scene. Solid objects are rendered as solids using lighting and shading. The particular project is chosen by the group of students, with instructor suggestions and approval.

The projects chosen, designed and implemented by the students have been quite varied and many of them have had excellent results. Some of the student projects done during the last two years include:

- Display a lighted/shaded view of a mountainous region. Allow the user to change the point of view. Change the position of the sun according to the time of day. Display names of lakes when the user picks a lake with the pointer.

- Simulate the Tacoma Narrows bridge collapse. Allow the user to fly over the bridge along various paths.

- Program the motions of an acrobat. (The acrobat model is not physically based, but by careful selection of motions fairly realistic looking movement can be generated.) The acrobat's path is determined by selecting several points through which the acrobat will pass. Various motions such as squat, tuck, rotate and twist can be specified as occurring at or between points.

- Model the interior of a building. Allow the user to walk through the building, look in various directions, open doors and turn lights on/off by pressing a light switch on a wall.

4.0 EFFECT OF PROJECT ON IMAGE PROCESSING COURSE

Prior to this project, the image processing course was restricted to very small (128 X 128) gray-scale images since they were the largest that could be efficiently dealt with on an 80286 class PC with only 1 Meg of RAM. Even with such small images using Fourier Transforms to demonstrate frequency domain processing took several minutes for each operation.

The equipment provided by the project changed this significantly. The class now routinely deals with a variety of image sizes from 128x128 through 1024x1024 and occasionally larger. Some of the images now used include the images from the Voyager missions, distributed on CD by the National Space Sciences Data Center. Additional images include remote sensing images, a large chest x-ray and images of students scanned in locally. The computational power available on the workstations has opened up a world of possibilities.

Some software has been developed to support the image processing course on the workstations. The software includes a simple image viewer, MImage, that supports zooming to see the effect of processing algorithms on individual pixels, pixel value read-back for debugging and analysis purposes, and a histogram display. A second program FFTDemo displays four images simultaneously: an original image, its 2D Fourier transform, a filtered version of its 2D Fourier transform and the inverse transform of the filtered version. The students are required to implement the 2D FFT using 1D FFTs and implement various standard filters: ideal high pass and low pass, Butterworth high pass and low pass and notch filters. When the student has implemented these portions, the FFTDemo program allows the student to experiment with the filters on different images, interactively selecting the filter and cutoff frequency.

A more comprehensive image processing workbench that will allow the student to interactively control various parameters to their algorithms and display before and after results simultaneously is being developed.

5.0 OTHER USES IN DEPARTMENT

The equipment purchased by this project provided the first access to UNIX systems for courses in our department. Consequently, in addition to using the workstations for the graphics and image processing courses, they have also been used by our department in the following ways:

- The Systems Programming course was run on the workstations for the last two years.

- A topics course on X/Motif programming was offered during the spring semester of the 1994-95 academic year.

- Several groups of students have done projects on the workstations for their Math Models class.

- Some work on a faculty and student project with the Intense Pulsed Neutron Source of Argonne National Laboratory was done on the workstations.

The five workstations have been able to meet these additional needs within our department for several reasons. First, since they are connected to the campus network, they are accessible from virtually anywhere. Consequently, students not requiring graphics could work on their programs remotely. Also, LINUX has been installed on several PCs in the department. While the LINUX PCs do not have an implementation of PHIGS available, they do make excellent X-Terminals and expand the number of X capable machines.

6.0 CONCLUSIONS

While PHIGS is a powerful and sometimes complex 3D graphics programming system, its use in an introductory graphics class is practical given some higher level convenience routines to hide distracting lower level details. The use of PHIGS allows the student to construct more exciting graphics programs than would be possible using a lower level system.

Overall, this project has been very beneficial. It has allowed us to update the computer graphics and image processing courses. By facilitating the construction of more exciting graphics programs and the use of larger data sets and more significant images it has stimulated student interest. The PHIGS utilities and image processing "workbench" are freely available via anonymous FTP from dmikk.mscs.uwstout.edu.

7.0 ACKNOWLEDGEMENT

The author gratefully acknowledges the support of following organizations and programs:

The National Science Foundation provided funding for the purchase of five UNIX

workstations through Instrumentation and Laboratory Improvement Grant No. DUE-9351943.
Digital Equipment Corporation provided PHIGS and other software through the College Software Library Grant program.

The University of Wisconsin - Stout provided matching funds and the academic environment in which the work was performed.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the above named organizations.

[1] Computer Graphics-Programmer's Hierarchical Interactive Graphics System (PHIGS) Functional Description, ANSI X3.144-1988, American National Standards Institute, New York, 1988

[2] Computer Graphics: Principles and Practice, 2nd Ed., J. Foley, A. van Dam, S. Feiner, J. Hughes, Addison-Wesley, 1990

DENNIS MIKKELSON is a professor in the Department of Mathematics, Statistics and Computer Science at the University of Wisconsin-Stout. He has also been involved in developing software for visualizing neutron diffraction data at the Intense Pulsed Neutron Source Division of Argonne National Laboratory. His interests include scientific visualization, computer graphics and image processing.