1991

Portable 3D Graphics in a Heterogeneous Distributed Environment

Vinod Anupam
Chanderjit L. Bajaj
Malcolm C. Fields
Andrew V. Royappa

Report Number:
91-074
PORTABLE 3D GRAPHICS IN A HETEROGENEOUS DISTRIBUTED ENVIRONMENT

Vinod Anupam
Chandrajit L. Bajaj
Malcolm C. Fields
Andrew V. Royappa

CSD-TR-91-074
October 1991
PORTABLE 3D GRAPHICS IN A HETEROGENEOUS DISTRIBUTED ENVIRONMENT*

Vinod Anupam, Chanderjit L. Bajaj, Malcolm C. Fields, Andrew V. Royappa

Department of Computer Science,
Purdue University,
West Lafayette, IN 47907

September 26, 1991

ABSTRACT

Computer Aided Geometric Design is a young and rapidly growing field that deals with the construction and manipulation of geometric objects. By its very nature it relies heavily on high-performance computer graphics. The SHAstra environment aims to provide distributed, collaborative geometric design across a heterogeneous workstation environment. It was therefore necessary to achieve truly portable computer graphics without suffering the usual loss in performance. The XS suite was designed and built as a solution to this problem, and is now used by all SHAstra applications. We describe our experience with it.

INTRODUCTION

Computer Aided Geometric Design (CAGD) is the study of the construction and manipulation of geometric objects. Loosely speaking, a geometric object is one that has a visual structure with a concise mathematical description. The latter restriction is vital. For instance, a computer image of the Mona Lisa has some visual structure but it lacks a concise mathematical description. Examples of geometric objects are points, lines, planes, cubes, spheres, parabolas, cones, ellipses, polygons, polyhedra, and so on. Such objects can be compactly represented inside a computer. CAGD studies ways of combining these simple components in a step-by-step fashion to produce such complex geometric objects as ship hulls, house frames, car engines, and so on.

Fundamental geometric design operations on geometric solids include operations such as Boolean set operations like intersect, union, difference; operations such as revolution, extrusion, and offsetting; smoothly blending the vertices or edges of a polyhedron, decomposition and triangulation for finite element applications, etc. As an example, see Figure 1 where a piecewise smooth curved object is constructed from a polyhedron, using interpolation and least-squares surface fitting operations.

THE SHAstra PROJECT

Project SHAstra considers the research and development of the next generation of CAGD software environments where multiple users (say, a collaborative engineering design team) interactively create, share, manipulate, simulate and visualize complex geometric designs over a heterogeneous network of workstations and supercomputers. SHAstra is a highly extensible distributed and collaborative geometric design environment [3] consisting of a growing set of individually powerful and interoperable toolkits which support collaborative design sessions.

The GANITII algebraic geometry toolkit [8] manipulates polynomials and power series. It can be used to solve systems of algebraic equations and visualize its multiple solutions. Example applications of this are curve and surface display, curve-curve intersections, surface-surface intersections, curve-surface intersections, global and local parametrizations, inversions, etc.

The VAIDAK medical imaging and model reconstruction toolkit [4] manipulates medical image volume data. It can be used to construct accurate surface and solid models of skeletal and soft tissue structures from CT (Computed tomography), MRI (magnetic resonance imaging) or Laser surface imaging data. See Figure 2. VAIDAK incorporates both heuristic and exact methods of contouring image data, active thresholding, tiling or polygon reconstruction, and curved surface patch reconstruction.

The SHILP solid modeling and display toolkit [1] manipulates curved solid objects with algebraic surfaces. It can be used for the geometric design (creation, editing, etc.) and display of solid models with algebraic surfaces. Curves and surfaces can be represented in both implicit and rational para-
The current functionality of the toolkit includes restricted extrude, revolve and offset operations, edit operations on planar lamina and polyhedral solids, and fleshing of wireframes with interpolating surfaces. Three dimensional grids or meshes superimposed on solid objects can also be generated and are used for finite difference solutions of partial differential equations. For the purpose of finite element computation, algorithms have been implemented which decompose arbitrary polyhedra with holes into convex pieces or tetrahedra.

GANITH provides the computational mathematics infrastructure for SHILP and VAIDAK. SHILP provides the solid model manipulation and display functionality to skeletal structures reconstructed from CT/MRI image data in VAIDAK.

The SHASTRA environment has been designed to promote distributed problem solving by providing a rich set of interoperable tools in a user friendly setting. As more tools are integrated into the environment, it will be possible to perform highly sophisticated scientific manipulations. In the setting of available functionality, human skeleton modeling is a good example of distributed problem solving. CT/MRI data is input to the medical imaging system to produce a polyhedral solid in VAIDAK. This is passed on to SHILP which calls upon an instance of GANITH to interpolate the polyhedral surface and produce compact curved surface solid models of the skeleton (similar to Figure 1). Also see Figure 3 where a three dimensional model of a human head of Figure 2 was reconstructed in VAIDAK and communicated over to SHILP.

This scenario can be used to construct an application for interactive design of artificial implants using SHILP to make calls to GANITH for smoothing operations. The collaborative layer allows multi-user creation, manipulation and visualization of geometric models. It provides a powerful problem solving virtual machine supporting applications which exploit batch and interactive parallelism. One major goal of project SHASTRA is the development of an environment suitable for a group of geometric designers, medical practitioners, surgeons, and material scientists involved in artificial limb design and manufacture. A primary requirement is that the environment be able to exist on top of a heterogeneous mix of hardware platforms.

PORTABLE, HIGH PERFORMANCE 3D GRAPHICS

Each of the CAGD applications described utilizes two very different areas of computer graphics. First, each operates in a windowing environment, namely X11, and hence provides a sophisticated user interface. The user interfaces are based on MIT's Athena Widget Toolkit, and present the familiar "look-and-feel" based on buttons, menus, scrollbars, etc. The second area of computer graphics used is the smoothly shaded display of three-dimensional curved objects, and this is what will be addressed in the rest of this work.

Once a (three-dimensional) object has been constructed in some fashion out of simple constituents, it is necessary to display it accurately, for any position of a hypothetical viewer. A sizable portion of this accurate display is hidden-surface removal: parts of the object that are obscured by other parts, and hence are not visible to the viewer, must never be displayed (see Figures).

Furthermore, a complete geometric design system will also provide tools to display a scene composed of several objects in a visually appealing fashion. For instance, the color and position of various light sources, whether they are beams (i.e. spotlights) or omnidirectional, the level of ambient lighting, etc.; the color, shininess, and material type (e.g. "green plastic") of each surface in the scene, etc., are instances of such tools.

A successful CAGD system facilitates the design and composition of a complex geometric object, and also provides ways for designers to visualize their work in a manner that leads to productive modifications and improvements in their design.

The visualization of complex geometric objects in the manner described above is a computer-intensive task, requiring hundreds of millions of floating-point operations per second to approach real-time demands. To solve this problem a new generation of computer graphics workstations have appeared. These workstations contain special-purpose hardware for the visualization operations described above, and make possible the real-time display and animation of complex, smoothly-shaded scenes.

Each workstation vendor provides a custom software library to access their hardware. Each such library implements a particular graphics paradigm, i.e. it has its own notion of viewing model (how the viewer looks at a scene), lighting model (how lights affect a scene), etc. These paradigms are similar but not isomorphic.

Thus at one point each of the three SHASTRA applications had one or more software versions, one per workstation type. Since the graphics paradigms of each workstation were different, the code differences were not even purely syntactic, and because of the ubiquity of graphics operations throughout these programs, conditional compilation directives were not a viable solution. Because each SHASTRA application is a moderately complex software system in its own right (totaling about 100,000 lines of code), the overall software management problem soon became unacceptable.

Therefore we addressed the problem of designing a three-dimensional computer graphics library to be used by all the SHASTRA applications. Such a library would mean that each application used the same paradigm for three-dimensional computer graphics, encouraging software exchange and transparency. In addition to providing the usual low-level graphics facilities, this library would satisfy the following conditions:

1. It would eliminate the need to maintain a separate software version of each application for each workstation type, i.e. it would facilitate source-level portability.

2. When a workstation provides specialized computer graphics hardware, the library would utilize this hardware whenever possible.
Figure 1: A Polyhedron Smoothed in SHILP by Remote Calls to GANITH

Figure 2: Human Anatomy Models Reconstructed from CT/MRI Images in VAIDAK
3. It would be possible to link this library with a client application of the X window system, so each SHASTRA application could run as an X client, providing the familiar widget-based user interface.

With the completion of such a library, we envisioned an environment in which there was precisely one software version for each SHASTRA application; the applications would continue to be X clients as before, providing a standardized "look-and-feel;" and finally, when an application is compiled and run on workstation W, all advanced graphics performed by the application will use custom graphics accelerators provided by W (if any).

We were aware of the PEX (PHIGS Extensions to X) project, which has similar intentions on a larger scale XXX; however, PEX was not available to us then (and has not been released at the time of writing). Since we did not wish to wait until PEX was made available by each workstation vendor, we proceeded to build and test our own solution to this problem.

Our solution was to create a suite of libraries, called the XS graphics libraries. (XS is an acronym for the somewhat vacuous name of "eXperimental System"). The libraries access system-dependent graphics facilities in a uniform, system-independent manner. Each system supported is represented by a single library in the suite. All libraries in the suite implement the same graphics paradigm and present the same function-call interface. In this way, an application program can maintain source-level portability across several systems by simply linking with appropriate members of the XS suite.

We will next describe the design of the XS. The technical aspects of XS are described in the XS report [2] and will not be addressed here in any detail. Instead we will concentrate on the design process and some of the problems encountered along the way.

DESIGNING THE XS SUITE

We now describe the design of XS (see Figure 4).

The XS Philosophy

The design of XS was driven by the following simple principle: an application should have the same appearance and behavior on all workstation types. That is, the XS application programmer can work on a single software version of the application, knowing it will have a particular "look and feel" that will not change from workstation to workstation. This philosophy of a standardized look-and-feel is a central theme behind the various user-interface toolkits now available; it is applied here in the context of high-performance computer graphics.

It was apparent from the start that this goal was idealistic, because of the differences in the graphics paradigms among workstations. For instance, one vendor may not supply a certain feature that is supplied by all others. Taken to an extreme, a simpler example is that an application using the color "red" may not look the same on different workstations, due to hardware differences among color monitors. To fully satisfy this goal would require a collaboration between workstation vendors; it was therefore necessary to lower our sights.

One approach considered was to have XS provide only those features that were implemented on all the target workstations. This would indeed lead to a reasonable approximation of the principle above. However, using the intersection of workstation capacities would limit XS to the capacities of the least powerful workstation, so this idea was soon abandoned. We settled instead on this two-fold philosophy:

1. XS would provide an idealistic graphics paradigm that was sufficiently general for our needs.

2. Each library in the XS suite would implement the whole paradigm; however, if a particular workstation did not
Figure 4: The XS library architecture
As an example, the XS paradigm provides a way to display a smoothly shaded polygon. On a workstation with shading hardware, an XS application would draw the polygon with its interior smoothly shaded; on a workstation without such hardware, the corresponding function might simply draw the outline of the polygon, or it might attempt to perform the shading computation in software.

In this way an XS application could maintain one software version and yet run on a gamut of workstations. Of course, a low-end workstation would only be able to approximate a high-end one, but in practice we have found this very useful. Since there are fewer high-end workstations available, a user can perform a large amount of preparatory visualization on an inexpensive workstation such as a Sun-3/50 (which has no provisions for three-dimensional graphics) and then transfer the work to a graphics processor such as a Silicon Graphics 4D/25 for a final, accurate display. This may reveal flaws or suggest improvements which the user may carry out on the 4D/25, or the user may go back to the Sun 3/50 and repeat the cycle.

In passing we mention that one special library of the XS suite supports a "pure X" workstation (i.e. a workstation without any 3D graphics support capable of running an X server). An XS application linked with this library could run on any X server, even on a dumb X display terminal (note: at this time it will work only on a color X display, but the change to allow black and white displays in the current model is not very complicated).

Implementing The Graphics Paradigm

The three-dimensional graphics paradigm was designed at the outset with simplicity as a guiding notion. No complicated data structures were to be used for specifying graphics primitives, and only one data type was allowed for coordinate specification (most vendors allow several). XS would provide for the creation and destruction of any number of windows which would support the usual low level three-dimensional graphics output primitives. Some simple constraints could be laid on these windows, such as disabling resize operations by the user, or enforcing a certain aspect ratio. In addition, a simple input model based on callback procedures was supported, to handle user-generated events. An application could register a callback procedure for three kinds of events: mouse click/motion inside a window, mouse entry/exit from a window, and change in window size or location. For the motion/click callback a provision was added to enable or disable the motion part, because some workstations generate a large number of motion events even when the mouse is moved a small distance.

As an illustration of the difficulties encountered in implementing this paradigm on top of a specific graphics substrate, we discuss the first library of XS, which was for the SGI 4D class workstations, on top of SGI's graphics library (GL). Mapping the XS output primitives to GL primitives was easy (the output part of the XS paradigm bore some inevitable similarities to GL due of our familiarity with it), but the input model presented serious problems. This was because the SGI machines run in dual-server mode: at all times there is a native server and an X server. The native server is an implementation of NeWS [11] and dual-server mode allowed SGI users to run either NeWS or X applications on the same display. However, X server-only mode was not supported. In GL, all graphics windows are created and handled by the NeWS server, thus this library had to deal with input events from two distinct servers! The first version of the XS main loop simply performed a non-blocking check on each server connection for events. If an X event was present, it was given to an an X Athena Widget Toolkit routine for processing; if a NeWS event was present, XS handled it, calling any registered callback procedures if necessary. However, this tight loop caused an obvious degradation of performance. Since both X and NeWS provided a way to access the UNIX file descriptors corresponding to the server connections, it was possible to use the UNIX system call select() to perform a blocking check on both server connections simultaneously [10]. Thus the body of the XS main loop will only be entered if some event (X or NeWS) is in the queue, eliminating the busy wait.

Another illustration is given by the XS library for pure X workstations. This was based on the Athena Widget Toolkit and each XS window was simply a widget of the "simple" class, contained inside another widget of the "topLevelShell" class. The shell widget allowed the XS window to interact with a window manager. The window of the simple widget was used as a canvas for drawing, using the usual X library routines for two-dimensional graphics. Of course, this version had to implement in software the viewing, perspective and modeling transformations, etc. that are performed in hardware on the SGI's.

The XS paradigm allows a small amount of customization: instead of specifying a general model for lighting and surface material properties, it was decided that each library would simply provide a fixed set of "good" shading models which it would store internally. An application program can access the table of names of these shading models, e.g. "green plastic," "gold" and so on. It can then allow the user to choose among any of these shades, if shading is requested. For now this simple approach has proven sufficient for our purposes.

Drawbacks And Extensions

An application running XS will naturally not be as efficient as one that uses a certain vendor's custom graphics directly. However, because of the simplicity of the graphics paradigm, there is not a large overhead in translating data from the XS format to the format required by a specific graphics substrate. The bulk of the time is still spent performing the actual graphics operations.

A problem that is yet unsolved is that of color management. The XS paradigm allows one to set the "current drawing color" to any (red, green, blue) triple. This works well on workstations with 24 bitplanes of direct color, but poorly on 8-bit worksta-
tions with a single hardware color map. At present the pure X library, which is usually used on the latter type of workstation, simply allocates a small table of colors for itself, and uses them until they run out. Future color requests use the "closest" color in the table. This way an application can build a color map for itself. However, certain applications which need many different colors (e.g. VAIDAK, which displays CT/MRI data containing many shades of grey) may not always work satisfactorily on such workstations. Some further study of adaptive color management and the possible applications of color quantization is needed.

A drawback that cannot be resolved in the current scheme is that an XS application must generally use the display connected to the host it runs on: i.e., it will not possess the ability to export its display to any X server, as normal X clients are capable of doing. Applications using the pure X library are of course exempt from this restriction: they are true X clients in every way.

Other than expanding the number of libraries in the XS suite, some possible future extensions would be to add a simple model for lighting and shading, and to add provisions for managing groups of graphics primitives.

CONCLUSIONS
The XS suite includes libraries for the SGI 4D workstations, pure X workstations, and HP 97000 workstations. In addition, a "stand alone" XS, independent of X, has been developed as a simple three-dimensional graphics package for the IBM PC, and parts of GANITII have been successfully ported to the IBM PC using this version of XS.

All three SHAstra applications now use XS. Hence they can be used on any of the workstation types referred to above, and only need to maintain a single version of their software. When PEX becomes an accepted and widely available standard, and the SHAstra applications are ported to use PEX, the effort required will be much smaller than what was required before.

It is therefore evident that a fairly modest programming effort has paid large dividends in code standardization and clarity, and has greatly simplified the implementation of three-dimensional computer graphics in the SHAstra environment.

REFERENCES
10. Sun Microsystems Inc. (1990), "Network Programming Guide".