Table of Contents

Project Description

1. Prior NSF Support

Until now the PI has received no specific grant aimed at undergraduate education per se from NSF. However, some NSF research funding has some relevance to this proposal, and it will thus be described briefly. It has been recently enhanced with a supplement grant to create a position that provides "Research Experience for Undergraduates" in accordance with NSF Announcement 96-102.

Our research project on "Rapid Prototyping Interface for 3D Solid Parts" (MIP-9632345) studies the interactions between designers and manufacturers of mechanical parts. Our main goal is to link these two different parties with an efficient part description interchange format and a set of checking and verification programs. These tools will streamline the process of submitting parts for fabrication and shorten the time to market for products that depend on the redesign of such parts.

During the last two years, the project has made good progress. A prototype description of the interchange format has been posted on the web, and the crucial modules that parse the language and build suitable data structures for the described CAD models have been implemented and used by several researchers. An interactive web-based design tool and various other utilities, to check the consistency of CAD files or to slice them into many thin layers for some Solid Free-Form fabrication process, also exist in prototype form.

Work on this grant over the last three years has helped the development of an extensive infrastructure of C++ libraries that deal with basic mathematical algorithms and data structures that are frequently used in computer graphics and solid modeling applications. The development of the SLIDE environment described in this proposal has benefited strongly from this infrastructure, both in terms of direct re-use of some of the C++ libraries, and in terms of the general experience gained by Jordan Smith, the key developer of the SLIDE laboratory, as to how one should organize such a software environment. This NSF research project will be

concluded during Fall 1999.

2. Project Overview

We want to make available a modular computer graphics and visualization laboratory that teaches in a "hands-on" manner the construction and operation of a computer graphics pipeline as well as elements of software engineering. The SLIDE laboratory and its renderer are aimed at introductory college-level courses in computer graphics. Experience with such a software environment will benefit students in the scientific, mathematical, and engineering disciplines. Students learn good software engineering skills as well as the ability to create informative visualizations through a modest amount of programming effort, which are invaluable skills in these fields.

Over the last several years we have developed, as part of our introductory graphics course CS 184 at U.C. Berkeley, an interactive programming and visualization environment. The latest and most mature variant is based on SLIDE, a simple yet powerful scene-graph language for interactive and dynamic environments. It’s key developer is Jordan Smith, a third year graduate student, working under the direction of Prof. Carlo Sequin. The core component is a C++ class library that implements the classical rendering pipeline in a modular and extensible way, modeled after the successful "NACHOS" instructional Operating System [2]. In a typical introductory graphics course, the instructor disables one or more of the modules that make up the whole system, and the students in the class then rebuild these modules on their own as 1-2 week assignments. In this hands-on manner, they learn the algorithms and data structures needed for a particular function, such as, hierarchical visibility culling, back-face culling, polygon clipping, 3D transforms, interactive view manipulation, perspective projection, polygon scan conversion, software z(depth)-buffering, lighting, and shading.

We propose to turn our working prototype into a robust and well-documented system that will be freely available on the Internet to academic institutions teaching graphics and visualization courses. The release of the SLIDE system will make available development efforts and experience gained at Berkeley over several years to many instructors across the nation who may

not themselves be experts in computer graphics or who don't have the resources to develop an extensive instructional software environment.

Our system should readily work on any platform that supports C++, OpenGL [3], and Tcl/Tk [4], after it has been compiled for the particular platform. We have used our system in our classes on Windows NT, and on many flavors of UNIX platforms, ranging from LINUX to high-end SGI workstations. However, the whole compilation and distribution environment is something that needs more work and clean-up before this system can be widely distributed. Work that remains to be done for an initial release involves:

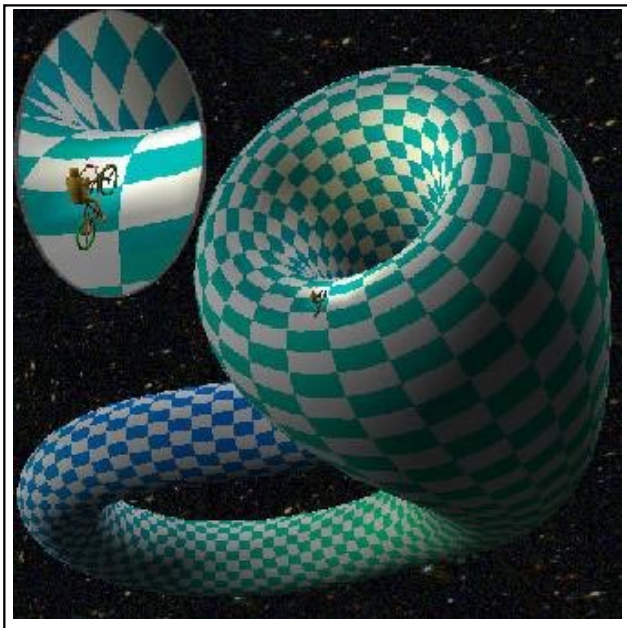
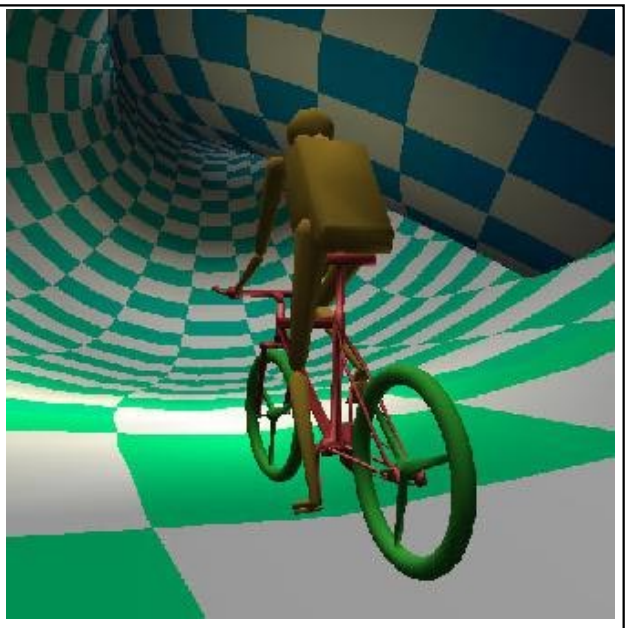
- rewriting of some code to make it more robust
- extensive documentation of how the system works
- more detailed write-ups concerning the weekly assignments
- more examples demonstrating the expressibility of the SLIDE language
- more examples demonstrating the use of the dynamic Tcl/Tk front end
- packaging up the library source code to make it easily down-loadable

So far our system has been used twice in our undergraduate computer graphics course, in Spring of 98 and 99, respectively. We now think that we have the right model and want to obtain feedback from instructors outside our own institution. In the Fall term of 1999, two former students of Prof. Sequin who are now themselves junior professors teaching graphics courses, will use this system at their schools and will help us to address issues of dissemination and maintenance with such a system. Contacts have also been made with a rather diverse set of additional colleges (See Section 7.3) to act as beta test sites for early releases of this software, so that we can better understand the needs of these schools and adjust our software to serve as wide a range of users as possible.

Another aspect of education is making these materials available to a wide audience. We are in the process of building a Web site which will serve as a distribution site for the SLIDE laboratory assignments as well as an informative site about the SLIDE language and about 3D rendering.

3. Goals and Objectives

The SLIDE system comprises an interactive computer graphics and visualization laboratory. It is built around the SLIDE scene-graph language for interactive dynamic environments and its renderer. This software environment contains many utilities that make it easy for students to put together complex scenes and interactive animations, ranging from playful video game scenes to the visualization of scientific data or mathematical objects. One of the final course projects in Spring 1999 presented a model of a bicyclist who could be interactively steered about the surface of a Klein Bottle.

<table>
<tr>
<td>  </td>
<td>  </td>
</tr>
<tr>
<td>Full view of the Klein bottle with a superimposed close up of the bicyclist</td>
<td>Inside the Klein bottle, the self intersection of the surface is clearly visible</td>
</tr>
</table>

SLIDE has a very simple set of 3D geometric constructs with a powerful instancing mechanism. Its functionality is surprisingly powerful and useful for teaching the fundamentals of 3D Computer Graphics. It also provides the use of a interactive and programmable rendering environment.

With this environment, the students get exposed to a variety of important concepts. They learn to construct computer-based descriptions of complex objects and sophisticated composite scenes in

a well-structured hierarchical way. They learn to define important parameters in their scenes which need to be adjusted interactively and to set up a convenient user interface to control them. They learn how a modern graphics system handles such computer models to display them in a fraction of a second, even though the scene may contain tens of thousands of polygons, many of which may change their shapes or positions in each frame. They learn in detail about some of the key algorithms that need to be performed millions of times every second on the polygons and their edges in the scene, e.g. clipping, back-face culling, scan conversion, depth comparison, and shading. Last but not least, even though this is not the main subject of this laboratory environment, the students learn important lessons about software engineering. They are exposed to a well-constructed set of libraries written in a sound programming style. They gain the skills of familiarizing themselves with an existing system and integrating their own code into it to increase its functionality. These are important skills for programmers on large projects.

There are strong advantages of letting the students work within such a structured framework, as compared to letting them build everything from scratch on their own -- as we did until a couple of years ago. From the beginning, students are exposed to the overall architecture of the graphics pipeline, so they better understand the role of their modules as they implement them. During the development and debugging phase of a particular module they may compare their own work to the operation of a proper compiled module. This helps to reduce any confusions students might have with the written descriptions. Building on top of OpenGL allows for visual debugging of the low level algorithms they are implementing. This also allows the students to step outside of the viewer and explore in detail different aspects of the rendering pipeline. The structured laboratories give the students a fresh start every week in the form of new skeleton code for each assignment that includes the solution to the previous one. This prevents students who fail to finish a single assignment from falling hopelessly behind during the rest of the course.

We are convinced that an application-specific software laboratory like the SLIDE environment described in this proposal has several educational advantages. As the performance of computers and graphics accelerators has increased, it has become possible to run 3D graphics applications on low end personal computers. The SLIDE environment will help to educate this new wide

audience of scientists and engineers who now have this technology available to them. While the next generation of college students has grown up with computers and is very comfortable in their use, they only master the standard packages and user interfaces provided by the typical application programs, ranging from desktop publishing software to simple CAD tools for modeling mechanical parts or architectural scenes. What is much harder to learn for these students, outside a structured environment, is the powerful combination of their programming skills with a well-engineered set of computer graphics rendering utilities. With SLIDE, they are also exposed to good software engineering principles that keep their coding efforts well-structured and re-usable.

4. Detailed Project Plan

4.1 Background

The core graphics course at U.C. Berkeley is CS184, "Foundations of Computer Graphics." This is a 4-unit junior or senior level course that teaches computer science students how things work behind the display screen -- rather than just how to make use of systems such as AutoCAD or API’s such as OpenGL to describe and render pretty objects or interesting composite scenes. We strongly believe in "Learning Through Doing." Thus since 1986, the second year that Prof. Sequin was teaching this graphics course, we have made students build a rendering system from scratch around a very simple, easy-to-parse language to describe polygonal objects and scenes.

The language used in this class had its roots in CIF, the Caltech Intermediate Form [5], an ascii file format for the exchange of VLSI chip designs among academic institutions and semiconductor fabrication houses. This was a simple descriptive format that allowed a designer to specify the geometrical primitives needed to define the layout of the mask levels needed to fabricate integrated circuits. This format became very successful and truly enabled the collaboration of many academic institutions and industrial research laboratories. Layout tools, design rule checkers, analysis and optimization programs were all built around this language at many different institutions and were freely exchanged and shared between them. There can be no doubt that this was one of the accelerating factors in the VLSI CAD revolution of the early

1980’s.

It was an easy task to extend the principles of this language to the description of 3D objects; a third coordinate had to be added to the point specification, and more complex 3D transformations had to be represented. This lead to Berkeley UniGrafix [6], which became the language used for research in graphics and solid modeling at Berkeley for the last two decades.

A simplified version of this language was first used in CS 184 in 1986. It was changed slightly from year to year, to make some enhancements, and to prevent students from simply copying the lab assignments from the previous year; its name also changed frequently:

- 1986 ALFOS -- A Language for Object Specification
- 1987 OFFOS -- Our File Format for Object Specification
- 1990 SDL -- Scene Description Language
- 1994 GLIDE -- Graphics Language for Interactive Dynamic Environments
- 1999 SLIDE -- Scene Language for Interactive Dynamic Environments

The language grew in scope from just representing static polygonal scenes to a rather sophisticated system that can handle interactive dynamic environments efficiently. We now contemplate enhancing it even further to make it possible to dynamically add new, separately compiled objects into a scene without having to stop the renderer.

For several years, the students were asked to build their own rendering systems from scratch over a period of about 12 weeks and then also use it for their final projects in which they tried to compose and render an interesting computer graphics scene. For most of them, this was an intense learning experience, and the students who succeeded clearly took away much useful knowledge -- about graphics, as well as about writing large and complex programming systems. But this course structure clearly posed problems for the weaker students. If the program foundations laid in the first few weeks of the course were inappropriately scoped or implemented in a buggy and brittle way, then the emerging rendering systems built on top of them would crumble around the half-way mark in the course. The bravest students would learn from that experience, restructure their code, and with a little bit of luck catch up again with the main stream

of the course; but many others simply gave up and often even quit the course.

To lessen the chance of failure, we started -- with the era of GLIDE [7] -- to show the students the structure of the complete rendering system that they were supposed to build early in the course, and then let them build that framework one module at a time. This helped, but still left many students stranded in the ruins of their own program pieces. Moreover, GLIDE also had several weaknesses as a general purpose rendering system which made it a cumbersome tool for building interesting dynamic scenes or for learning computer graphics. The most bothersome feature of GLIDE was that the rendering program had to be specially recompiled for each dynamic scene. In effect, this meant that GLIDE was not so much a real rendering system, as it was a 3D application building API. The recompiling aspect created slow user feedback when modeling and modifying dynamic scenes, and made for a tedious and cumbersome development environment.

4.2 The SLIDE Environment

The SLIDE instructional laboratory and software environment improves on GLIDE's framework in the hope of making the students' experiences more positive. SLIDE addresses the students' difficulties in the early programming laboratories by providing a set of well structured laboratories containing C++ skeleton code. This format helps to guide students to learning good solutions to the problems of rendering, and it provides them with a well-designed overall framework into which they can fit their own code. The new structured laboratory assignments take away some of the creative freedom that students have had in past years for the benefit that students can be exposed to more topics and to an overall sound architecture. In the past, students implemented their systems one module at a time without any consideration for future needs. In many cases that resulted in a reenforcement of poor and inefficient data structures, algorithms, and programming practices.

The latest prototype version of our graphics / visualization laboratory has been developed during 1998/99. It was first used in an ad hoc way during Spring 1998, while it was still under heavy development. During Spring semester 1999, it was essentially completed. More documentation,

bug-fixes, and performance improvements were made as the students were using it. In this section we describe some of the key components of this software laboratory environment.

4.3 The SLIDE Language

As stated above, the major goal for the SLIDE language was to keep it simple enough so that it can be implemented by undergraduates in a single semester. At the same time, we wanted it to be powerful enough to describe sophisticated dynamic scenes, so that students can apply what they have learned during the semester to build interesting, interactive final projects. The scheme used to fulfill these two opposing goals has been to limit the scope of the language to a few basic primitives and mechanisms that can be combined to create more interesting tools. This means that the SLIDE language does not have as many ready made features as VRML [8] or Open Inventor [9], even though it is rather similar in its overall approach to describe a hierarchical scene; thus the experience gained with SLIDE is readily applicable to constructing scene models in VRML. (See Section 4.7.2 for a discussion why we did not make VRML the target language for our graphics laboratory.) SLIDE is designed to be a more general programming utility which can be extremely powerful in the hands of programming-oriented users.

The purpose of the SLIDE programming laboratories is for students to get experience implementing the algorithms that make up a real-time rendering pipeline. This hands-on experience should reinforce the concepts behind the design of a 3D rendering pipeline. The goal over the course of a semester is to have students implement a complete rendering pipeline. This rendering pipeline must be somewhat simplified in comparison to a full rendering pipeline like OpenGL. It would not be reasonable to expect students to implement the full OpenGL API or a full VRML renderer in a single semester.

4.4 The SLIDE Renderer

The SLIDE rendering system also has to satisfy two potentially conflicting sets of goals. For instructional purposes it should be modular and easy to understand. As a utility and demonstration system during the final project phase, it should be highly efficient to allow rendering of complex scenes at interactive speeds. Two complementary SLIDE rendering

programs have been developed to meet both these goals. An all-software implementation was designed as an instructional tool for teaching the internal operation of the rendering pipeline. This program is the basis for the SLIDE laboratory assignments. The second SLIDE renderer uses available hardware acceleration of the OpenGL API in order to achieve real-time rendering performance. With this latter renderer, the SLIDE environment is a useful modeling tool.

The software renderer has a very modular and clean architecture. This program is the basis of the SLIDE laboratory assignments. It has been structured so that individual functional modules can be removed from the rendering pipeline without destroying the functionality of the program, and so that a sequence of individual assignments incrementally reconstruct the full rendering system. This program must be clean and well documented since students will spend a lot of time studying its source code and will take it as a model for their own coding efforts.

The utility renderer based on hardware-accelerated OpenGL has different requirements. It should be a useful tool for modeling objects and developing animations, and it should be fast enough to render interesting dynamic scenes in real time. Though it is not designed to be a photo-realistic renderer, it should be useful as a modeling and previewing tool for such scenes. Our solution is to provide SLIDE with an output module that produces RenderMan [10] RIB files which can then be run through a high-quality batch-style rendering program. Last and not least, SLIDE must be extensible, so that it is easy to incorporate additional primitives into it as our expectations grow as to what such a visualization tool should be able to handle.

Some design constraints apply to both of these two rendering programs. Both must be able to run any dynamic SLIDE scene without having to be recompiled. They should be as efficient as possible within their constraints. This implies that they must have mechanisms for culling away unimportant geometry as early as possible in the rendering pass to limit the amount of work the final stages of the rendering process have to do. These goals can be realized with sound strategies concerning the way the scene graph is structured, annotated, and processed. A lot of thinking and effort has gone into formulating these strategies. We believe that they are one of the key assets of the SLIDE system and put it ahead of rendering systems built around VRML.

4.5 The SLIDE Laboratories Overview

The purpose of the SLIDE programming assignments is for students to develop a thorough understanding of computer graphics algorithms and techniques by implementing them. "Learning by doing" forces the students to digest the information presented in class to the point where they can instruct the computer how to apply it. Active learning such as this has a higher chance of having a lasting effect on students than if the students passively listen to lectures without reinforcement.

The architecture of the laboratory assignments breaks the task of implementing the rendering pipeline into smaller, more manageable chunks. They incrementally build on top of each other to incrementally create a complete software solution to a renderer for complex dynamic 3D scenes. A safety net is provided for students who fail to complete a particular assignment by giving students new source code at the beginning of every new assignment. The new framework contains the solution to the previous assignment and additional skeleton code as a guide for the next assignment. Every new assignment starts with a clean slate. This set-up also offers the benefit that students have an opportunity to work with many different partners throughout the semester.

For each assignment, students are given commented skeleton code, a specification of the new features that must be implemented, and on-line notes that give some mathematical background. On-line documentation provided so far is only meant to augment the materials covered in lectures and a text book. But in the future this material could be expanded into a complete on-line introductory graphics text.

The labs are written in C++ using Tcl [4] as the interpreted dynamics language, Tk [4] as the windowing API, and OpenGL [3] as the 3D rendering API. All of these languages and packages are standards that are supported on a variety of operating systems, which means that SLIDE will be completely portable. The primitives of the SLIDE language are implemented in a C++ class hierarchy, which makes management of the scene graph very convenient.

A nice attribute of the students programming environment is that they are implementing an OpenGL-like rendering system with OpenGL as a safety net. This means that in the early assignments, the labs can use OpenGL as a utility to do a lot of the work that the students are not yet ready to deal with. Later as the semester progresses, the labs tend to rely less and less on OpenGL. The use of OpenGL also gives students a carefree way to debug their algorithms visually. For instance, when the students implement a polygon clipping algorithm, the lab is set up so that the students' code is supposed to clip to a rectangle that lies completely inside an OpenGL viewport. If the student's algorithm is incorrect, then it will display spurious pixels outside the clipping rectangle. In the SLIDE laboratory environment, the students are able to view this behavior in real time which is helpful for diagnosing their algorithmic mistakes. If they did not have OpenGL as a foundation, their programs would probably just crash when pixels are written outside the active rendering area. Students would then have to debug their programs either by tediously probing values in a debugger or by printing out text values as the program is running.

Another benefit provided by the structure of the laboratory assignments is that all of the rendering functionality can be interactively enabled or disabled through the options menu. This allows students to isolate algorithms and get a thorough understanding of them. While working on a lab, they can run the executable solution program and isolate the behavior that they are trying to implement, which often helps them to better understand the documentation and concepts.

4.6 The SLIDE Laboratories

Below is a list of the eight incremental assignments associated with the SLIDE rendering pipeline. The first two assignments are meant to get the students familiar with the coding style used throughout the labs, with event loop based programming, with the SLIDE scene language, and with Tcl/Tk. The next two labs teach about scene hierarchies. Some students have difficulties imagining transformations and clipping in 3D at first. For this reason, these two assignments about scene hierarchies are implemented in 2D. Then in the following two assignments the students modify the working 2D renderer into a 3D renderer and add parallel and perspective

projections. The last labs deal with polygon scan conversion and lighting.

"Computer Graphics: Principles and Practices, Second Edition" by Foley, van Dam, Feiner, and Hughes [11] has been used as the text book during the last two offerings of our course. This is an excellent graphics reference containing information about almost all areas of computer graphics. Unfortunately, its structure does not fit well with the sequence of the SLIDE laboratory assignments. Students must bounce back and forth through this 1100 page text to find the information necessary for each assignment. Appendix A contains a list of the text references that provide technical background for each of the programming assignments. This list shows the non-linear traversal of the text necessary for the course. To alleviate this searching, we have made an effort to provide as much theoretic background as possible with each assignment with references to the text for more information. The on-line documentation could eventually be turned into an on-line text book about computer graphics that would be well matched to the SLIDE sequence of assignments.

1. Interactive 2D Polygon Builder

The students build a simple polygon drawing application where the only operations are adding points, closing polygons, and writing out SLIDE descriptions of the drawn polygons. The students learn about windowing API event loops and how call-back functions are used to activate user code. They also learn about designing geometric data structures for describing polygons. Lastly, they learn about the basic geometry constructs of the SLIDE language for their output module.

2. 2D Polygon Editing and Morphing

In this assignment, the students build another 2D modeler which can read in SLIDE descriptions and edit the values of point locations. The students must implement a point location algorithm and maintain the original and current values for all points. The program can either output a static description of the edited geometry, or a dynamic SLIDE file which linearly morphs between the original geometry and the current geometry. While implementing the morphing output, the students learn the basics of the Tcl language and the Tk widget toolkit.

3. 2D Dynamic Hierarchical Scene Renderer

This is one of the more challenging laboratories. The students are provided with the SLIDE parser and data structures. They must implement the matrix representations for 2D transformations, which forces them to carefully review this material. They have to implement the routines Preprocess, PreprocessGraph, Update, UpdateGraph, and Render for 2D hierarchical scenes. Most of the work in this assignment goes into efficiently calculating bounding boxes which are used in later assignments for culling geometry during the rendering pass. The students also use the top-level bounding box to define a maximally fitting, non-distorting viewport mapping.

4. 2D Bounding Box Rejection and Polygon Clipping

This is one of the conceptually simpler laboratories because the work involved is fairly localized in the bounding box class. The students must implement hierarchical view window bounding box culling using the Cohen-Sutherland outcodes. They also implement the Sutherland-Hodgman polygon clipping algorithm.

5. 3D Parallel Viewing and Crystal Ball Interface

This is the first 3D lab, so students tend to have extra difficulties. They must implement 3D transformations. Most of these are straight forward, but the non-commutative 3D rotations always give students trouble the first time around. They also have to deal with a virtual camera description. This implies inverting the camera modeling transformation to create the world-to-camera transformation. Also they must scale the camera frustum to the canonical clipping volume and modify their bounding box culling and clipping algorithms to work in 3D. They must augment their face data structure with a plane equation and perform back face culling by correctly transforming the plane normal into the camera’s coordinate system. These tasks are more manageable because only a simple parallel projection is implemented at this time.

They must also implement a crystal ball user interface, so that the user can click and drag the mouse to rotate and scale an object about the origin as if it were embedded in a crystal track ball.

This exercise really tests their understanding of 3D rotations.

6. 3D Perspective Viewing

In this assignment, the students implement perspective projection. They reimplement many of the algorithms from the previous assignment, so that they will work under perspective projection. The students learn about homogeneous coordinates, and are forced to reason about clipping in 4 dimensions. This assignment does not have a lot of coding, but it does require a lot of geometrical visualization and complex manipulation of matrices. In addition, the students generate a SLIDE file that creates a stereo display.

7. Polygon Scan Conversion and Software Z-Buffering

In this assignment, the student remove their last dependency on OpenGL by implementing their own polygon scan conversion algorithm. This assignment stresses the importance of choosing a filling algorithm that is invariant under rendering order. The students also learn about hidden surface elimination by implementing a software Z-buffering algorithm. Both scan conversion and Z-buffering teach students the valuable lesson of capitalizing on the coherence of geometric data. This is the only lab not designed to run with real time performance.

8. Lighting and Shading

This is the last of the software pipeline assignments. The students implement the Phong lighting model and both flat and Gouraud polygon shading algorithms. The types of lights supported in SLIDE are ambient, directional, point, and spot lights. The students must augment their data structures to store vertex colors as well as normals. These normals may be specified in the SLIDE description, but if they are not they should be computed by a weighted average of the normals of the faces that are adjacent to the vertex.

With this enhanced geometric data structure, the students are now ready to add lighting to their rendering algorithm. They must deal with multiple lights transformed by separate modeling transformations. The lab is set up, so that all points are transformed to a light’s local coordinate

system before the Phong lighting equations are calculated. Flat shaded polygons compute a single lighting value at their centroid and uniformly apply that color to the entire polygon. Gouraud shaded polygons compute a different color value at each vertex and then apply Gouraud color interpolation while scan converting. A number of different algorithms must be updated to make Gouraud shading work correctly. The polygon clipping algorithm must be updated to interpolate color values as well as vertex positions. The scan conversion code must be updated to efficiently interpolate the color values across scan lines.

4.7 Course Projects: Using the SLIDE System

After the students have completed the software rendering pipeline, it is time for them to learn about more advanced rendering techniques. The students switch from being implementers of the rendering pipeline to users of the OpenGL accelerated SLIDE renderer. Using the SLIDE language and renderer the students learn about procedural modeling, ray tracing, radiosity, and animation.

9. Procedural Modeling and Photo-realistic Rendering

In this assignment students learn about procedural modeling and photo-realistic rendering techniques from a user's point of view. Procedural model generators can be embedded directly in the SLIDE language through the mechanism of Tcl bindings to the SLIDE parser. These Tcl commands can be called from within SLIDE `tclinit` blocks to generate SLIDE scene geometry. Tcl is the procedural control language. This procedural mechanism is very powerful. Users are able to describe complicated fractal geometries in a few lines of Tcl code. A flattened SLIDE description of the same geometry might take up multiple megabytes of storage.

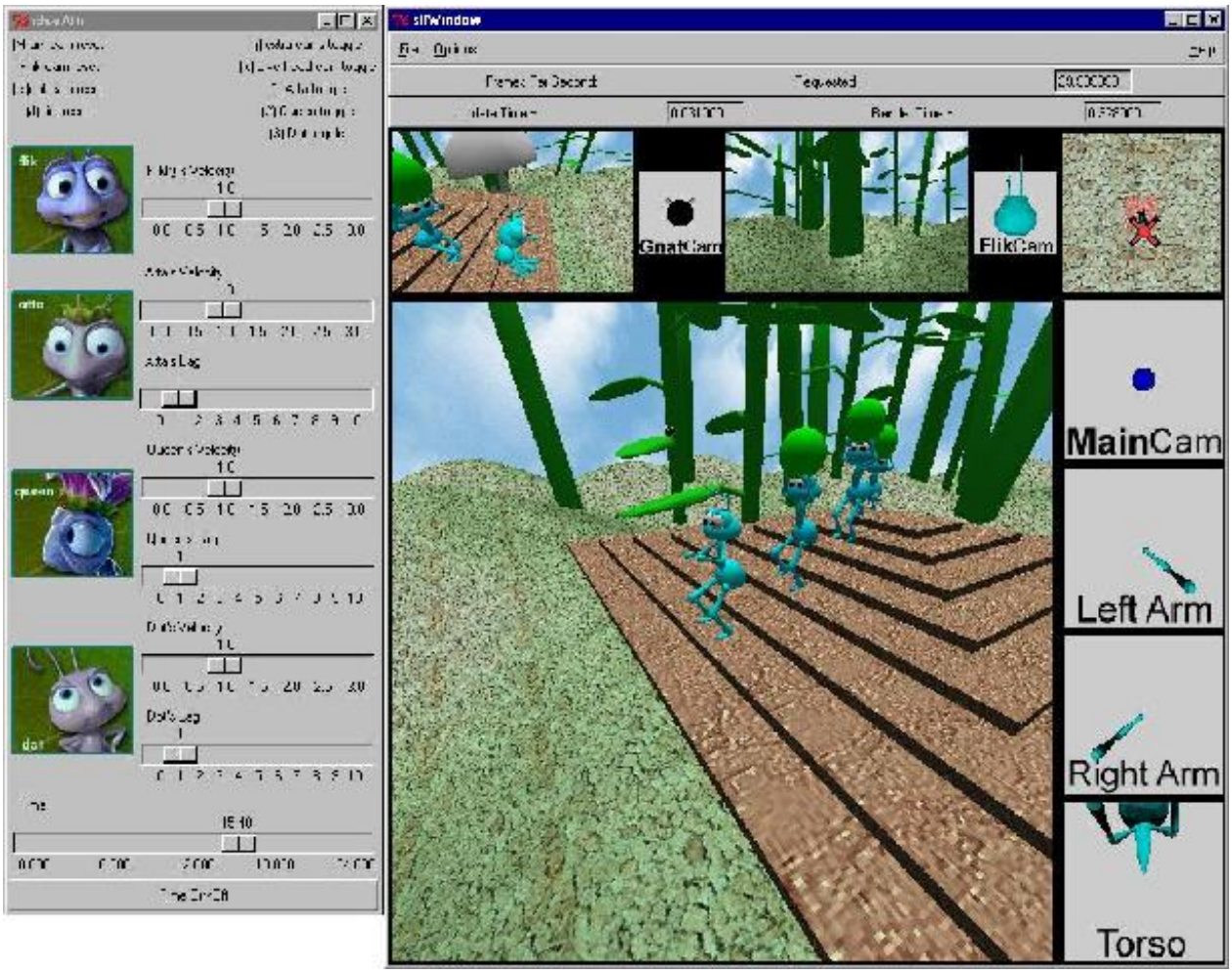
Students now use the hardware-accelerated SLIDE renderer to view their objects and assemble an interesting scene in real time. The SLIDE renderer does not do ray tracing or radiosity, but it serves well as an interactive previewer before calling one of the compatible batch style rendering processes. Once the students have created the scene they want, they can use the SLIDE renderer's RIB output module to create input for a RenderMan [10] compatible renderer. Using a renderer

such as BMRT [12] or PRMan [13], the students can create photo-realistic images.

10. Final Project: Interactive Visualizition and Simulation

The final projects are designed to be open ended. The students are asked to create interesting interactive scenes using SLIDE. They get a chance to fully exercise the SLIDE renderer and their imaginations in an enjoyable way. The subject matter of the visualization is up to the students. In the past, students have created projects ranging from interesting preprogrammed animations to fully interactive video games. Importance is placed on animation and user interaction. The new availability of Tk widgets from within a SLIDE scene description has added a lot to the quality of the interfaces of these projects in recent years.

As an example of what is possible, in Spring 1999 one group of two students created a simulation modeled after the movie "It’s a Bug’s Life." The scene includes four ants which use inverse kinematics to position their legs as they walk along a terrain made up of a sinusoidal function and a pyramid of steps. The ants can be directed where to go with a mouse-based interface. The ants follow the leader to the new location as long as they do not collide with any blades of grass along the way. This group made good use of capability in SLIDE to create multiple views of the scene (from the point of view of different ants) as well as multiple interactive user interfaces. They also used Tcl to create a separate window with a bank of Tk sliders for controlling the ants’ motion parameters.



4.8 Other SLIDE Utilities

The C++ framework of the SLIDE renderer is a strong foundation for building other utilities. Currently, the renderer has added output modules for creating scene graph visualizations and descriptions of solids for manufacturing. In the future, this program could be expanded to create an interactive 3D modeling tool -- possibly in an immersive virtual reality setting.

4.8.1 Integrated Scene Graph Visualization

A future version of the SLIDE renderer may allow dynamically changing scene graph topologies. This will make it highly desirable to view a conceptual graph of the scene while viewing the 3D scene. With such an interface the user will then be able to make edits through the abstract graph interface or the 3D interface. Currently we can output a scene graph description in the dot

language [14], which can then be viewed in the dotty graph visualization program. This is useful to students for debugging scene hierarchies and optimizing scene performance. In the future, we would like to make use of the Tcl/Tk bindings available for dotty, so that scene graph visualization can be integrated into the SLIDE viewer.

4.8.2 Link to Solid Free-Form Fabrication

Just recently we have added a special output module to the SLIDE modeler that will output triangulated surfaces in the .STL format, which is the de facto standard for sending part descriptions to manufacturing shops with Solid Free-form Fabrication (SFF) equipment such as, Stereolithography (SLA), Selective Laser Sintering (SLS), or Fused Deposition Modeling (FDM). This allows users of the SLIDE system to quickly and painlessly build physical models or prototypes of any parts that they may design in this environment. We plan to run an experimental course that will use the SLIDE software in that mode in Spring of 2000.

4.9 Technical Discussion

As we contemplate to release such a system nation-wide, it is time for some introspection. Do we have the right model for such a lab course in computer graphics? Are we still on the right track and heading in the right direction with our plans for further enhancements? How does the emergence of VRML [8] and Java 3D [15] affect our own development?

We did indeed ask those questions in 1998 when we started the development of the SLIDE environment, its many associated libraries, and the latest version of the laboratory infrastructure for our graphics course. Why did we introduce Tcl/Tk into the system? Why did we not switch to VRML at that point?

4.9.1 Tcl and Tk

Tcl and Tk combine to form a powerful utility for creating applications with Graphical User Interfaces (GUI). The Tool Command Language (Tcl) is a string-based scripting language. In Tcl, any type of data can be represented as a string. Tcl is useful for many reasons. Tcl itself is a

complete programming language. The SLIDE viewer embeds a Tcl interpreter that allows SLIDE to use Tcl as the scripting language for dynamic scenes. The Tcl language is a standard interface to Tk, a powerful and easy-to-use widget tool kit for creating GUI's. Tk has been implemented on many operating systems, which makes it a system-independent windowing API. Tcl's interface to Tk was the largest motivating factor for using Tcl for SLIDE.

Tcl is also useful for gluing different components of an application together. Tcl has a convenient interface to C and can be used to glue C code to GUI's or to other C code. A program can make Tcl calls from C, or it can make C calls from Tcl. An application can export some of its C functions to Tcl through a binding mechanism. Tcl bindings to C functions in applications may also be created by dynamically loading C shared library binaries that create function and variable exports in their initialization. The SLIDE system has only just begun to explore the possibilities that the Tcl framework provides.

4.7.2 SLIDE versus VRML

The SLIDE language had evolved over a duration of 12 years of teaching computer graphics. Its roots go back to Berkeley UniGrafix [6]. Paul Strauss who played a major role in the initial development of the Berkeley Unigrafix system, later also played a major role in the definition of the Inventor language at SGI, and the similarities in structure are very clear. Inventor in turn was a strong influence on VRML which is now a standard for modeling 3D dynamic scenes on the Web. As is clear from their origins, all these languages have many fundamental concepts in common.

The GLIDE/SLIDE language, through its predecessors used in our graphics courses, had reached some level of maturity, when many issues concerning VRML were still hotly debated. For our instructional use, it seemed desirable to stay with the simpler and cleaner SLIDE language, which was entirely under our control. During the last two years, as we started to build up the capabilities of SLIDE and ventured to build ever more complex scene graphs with dynamic objects, while struggling to preserve the efficiency of the rendering process, we noticed that VRML has some serious deficiencies in this respect. SLIDE's handling of a scene hierarchy that mixes static and

dynamic branches, and which has multiple changing lights and multiple cameras, is much cleaner than VRML's approach. Thus it now seemed even more important to teach the students the cleanest possible concept and show them the best way to efficiently handle such mixed dynamic and interactive environments. Another advantage of SLIDE over VRML is the ability to easily create GUI's within scene description files using Tcl and Tk. Tk provides many useful widgets and easy mechanisms for combining them to create custom UI's. With this mechanism, it is very easy to create a custom procedural modeler where the user can change parameters interactively. To do the same thing in VRML would take much more work.

On the other hand, VRML browsers have more built in facilities for navigating a scene and interacting with objects in that scene in specific ways. Thus the set of necessary primitives for a VRML browser is much greater than the requirements for a SLIDE renderer. SLIDE does not provide as many off-the-shelf utilities, instead it provides a small set of powerful, general mechanisms. With a little ingenuity scene creators can use SLIDE to make anything they could make with VRML. The limited scope of SLIDE makes building a renderer for it easier.

4.7.3 What about Java 3D?

With the emergence of the Java 3D API, we also have to ask ourselves why we should not tailor our didactic system to this evolving standard. The main reason why we have not done it yet, is again a historical one: When we started our latest development phase, Java 3D was just a dream.

Java 3D is similar to Open Inventor or the SLIDE class hierarchy. Java 3D is a scene graph API implemented as Java classes. Java 3D had not yet been released when the SLIDE system was started in 1998. Future implementations of SLIDE may be tailored towards Java 3D instead of OpenGL.

There is certainly the possibility to retarget our system to the Java 3D interface -- even though it seems prudent to wait a while until the Java 3D environment has stabilized and "the dust has settled."

4.7.4 Timing of the Release

What is the right time to "freeze" such a system and package it up for nation-wide distribution? Looking back over the last five years, we definitely have the feeling that we have a mature concept on our hands. The feedback from the students and the quality of their projects in the last two years and particularly in Spring 1999, confirms our beliefs. Also during the last year, we have received much interest from colleagues at other institutions to which we have talked on the phone or at graphics conferences, and who would like to try such a laboratory as part of their own courses. Thus this seems to be the right time to invest a consolidating effort that will properly clean-up, document, and package this material, so that it can be handed off to other instructors for their own use.

A major aspect of this packaging effort is to make yet another pass at a further modularization of the system. Many capabilities that now are firmly integrated into the system, such as parameterized cylinders and tori, will be taken out and compiled as separate libraries that can then be dynamically linked to the main modeling and visualization system. This new structure will make the system much more flexible and more easily extensible. New capabilities that we or other users may think of in the years ahead, can simply be added in the form of additional dynamically linked libraries.

5. Experience and Capability of the PI’s

The principal investigator, Carlo Sequin, is a professor of EECS and has been teaching graphics and solid modeling courses, both at the undergraduate and graduate level, for fifteen years.

From 1977 till 1981, as an instructor of the first VLSI courses at Berkeley and as a consultant at Xerox PARC, he worked closely with Lynn Conway, Carver Mead, and Bob Sproull on defining levels of abstraction for the VLSI design process. In particular, he was directly involved in the design of CIF, the Caltech Intermediate Format for LSI layout Descriptions [5], which became soon one of the de facto interchange standards for exchanging integrated circuit chip information among universities, industries, and chip manufacturing services such as MOSIS [16]. Later that

language evolved into todays Electronic Design Interchange Format (EDIF), which is the dominant format used to interchange design data between CAD systems [17].

From 1980 till 1982, Sequin together with Paul Strauss developed the Berkeley UniGrafix language, a logical extension of CIF to three dimensions. Many of the concepts in this language were later carried by Strauss to SGI and found their re-application in Inventor, which in turn strongly influenced the definition of VRML. At Berkeley, the UniGrafix language has been used for almost two decades in research and instruction in geometric modeling. It has also served as a model for a sequence of closely related languages employed in the introductory graphics courses at Berkeley, culminating in 1994 with GLIDE, a ‘‘Graphics Language for Interactive Dynamic Environments,’’ that permits effective descriptions for scenes with time-dependent shapes and constellations with which the user may interact in real time. The SLIDE language discussed in this proposal is slightly enhanced variant of GLIDE, but implemented in a much more modular and efficient way by Jordan Smith.

Jordan Smith is a third year graduate student, working under the direction of Prof. Carlo Sequin towards a Ph.D. in the area of geometric compilation for effective rendering of complex dynamic scene graphs. He will be the team leader and senior programmer in the proposed effort to further modularize, package for distribution, and document the SLIDE software environment. His understanding of the relevant software engineering issues have clearly been demonstrated by his work on SLIDE and on a similar system supporting the development of SIF, a Solid Interchange Format for rapid prototyping of free-form mechanical parts. Smith spent the summers of 1994-96 as an intern at Microsoft, where he gained valuable software experience in an industrial setting.

A second graduate student with suitable skills will be added to the team in the coming academic year. The projects should also provide good opportunities for undergraduate students who want to gain some research/project experience.

6. Evaluation Plan

We have observed a drastic increase in the success rate in this graphics laboratory during the two semesters that we have used the new SLIDE environment. A larger fraction of students stick with the course and complete all the assignments, and the quality and sophistication of the final course projects has grown substantially. There is some, but more tenuous, evidence from the results on the exams that the students also gain a better understanding of the core issues and algorithms for which they have completed a laboratory assignment, possibly because they spend a larger fraction of their time working on the actual algorithms and less time on repairing their own brittle program frameworks. In Spring 1999, we also took an anonymous survey [Appendix B] of the students in which we polled them about their views of the system. Their feedback was valuable, and overall very positive. In future offerings of the course we will continue to do these surveys with more refined questionnaires.

Clearly the most important feedback will be coming from users outside our own department. We will encourage the instructors at the other test sites to perform similar surveys, and we will provide them with our survey forms over the web. A subset of the instructors at other institutions, with preference to those with no former ties to our department, will form a panel that will be asked on a yearly basis to make an objective statement about the success and impact of the system and to provide suggestions for ways to improve it. These reviews will be made available in the progress reports to NSF.

It will be much harder to test the long-term impact of instruction using the SLIDE system on the actual understanding and skill levels of the students completing the course. Informal evidence will amass from the quality and sophistication of the projects that the student can complete. Interviews might be done with a sample of students about three years after they have completes school to collect their assessment of what benefits this particular form of "hands-on" instruction might have had for them. More ideas are clearly needed, and we will be open to suggestions.

7. Dissemination of Results

7.1 Deliverables

We will make our SLIDE system available on the Internet, so that other universities as well as individual users will have access to it. There will be three main components of SLIDE that users will want to access.

- The SLIDE language and rendering system is ideal for modeling 3D objects and scenes and building 3D applications. A growing library of dynamically linked plug-ins will gradually enhance its modeling and visualization capabilities.
- The SLIDE pipeline assignments present a structured opportunity for learning how 3D rendering works. The specifications for each assignment and the skeleton code from which students start will be on-line. These modular assignments are appropriate for classroom instructions as well as for individual self-paced study by motivated individuals.
- The SLIDE documentation which will start out as a manual to the pipeline, and gradually evolve into the equivalent of an on-line graphics text book.

These three components are currently available at "http://www.cs.berkeley.edu/~ug/slide" [1] but to make it easier to find we contemplate to establish a special site "http://www.slide.org". The possibility exists to also take the general technical material about computer graphics and complex scene graphs and make it into a paper book, published by a commercial book publisher. We would only do that if we get strong feedback that there is a need or desire for also having the information in hard copy.

7.2 Technical Assistance

No matter how well-designed the system will be, new users will require some hand-holding to become familiar and productive with the system, and prompt assistance when things go wrong. This is one of the reasons why we need at least two graduate students on the project -- to provide back-up and redundancy in the technical assistance service. To keep this overhead load on the

graduate students reasonable, most of the assistance will be provided via e-mail which will be processed in batch-mode once a day. Full-time telephone assistance during working hours appears too expensive to provide.

7.3 Audience and Clients

Our audience and potential clients are the instructors of graphics, modeling, visualization, and perhaps software engineering courses throughout the nation. Former students and other colleagues with whom the PI interacts at conferences or over the internet are interested and more than willing to become test sites. We plan to try the system pretty much as it is today in the Fall term of 1999, with two former students of our group, Tom Funkhouser at Princeton, and Seth Teller at MIT, both now assistant professors in the field of computer graphics. The long-term personal contact between PI and these first "clients" will ease the resolution of problems that are likely to crop up when such a system is first moved to another school.

Contacts are being made with a wide and diverse range of other colleges, including some local ones, i.e., Mills College in Oakland, CA; San Jose State, CA; as well as colleges in other states, including North Carolina State University, Howard University, Smith, and Wellesley. Experiments at these schools should be run during year one of our grant, i.e. in Spring 2000. Early feedback from these schools will allow us to better understand their needs so that we can adjust our software to serve as wide a range of users as possible. During the second year, the software environment should be robust and the distribution mechanism should be well established so that any interested party can readily download the SLIDE environment from our Web site.

Individual users will probably be most interested in using the SLIDE modeling and visualization system as a turn-key product. The range of possible applications includes modeling of 3D objects and mathematical entities, mechanical part design through possible submission for prototyping with SFF technologies, or interactive creation of complex 3D animations. These scenes and animation frames can then be sent via the RIB interface to a batch-mode rendering system which, given enough compute cycles, can produce photo-realistic still pictures or movie clips. Of course,

there is the potential use of the system in a self-paced self-study mode for strongly motivated individuals such as middle-aged professional people who want to brush up their graphics programming skills, or currently unemployed people who are preparing for a second career path.

7.4 Dissemination Efforts

So far we have not advertised the SLIDE system, since the system in its current form is not ready for widespread dissemination, and we are currently lacking the support structure to handle distribution and technical assistance. As these shortcomings are corrected -- through the requested NSF grant -- we will step up our advertising efforts. We will make our site more visible on the Web, and we will make presentations at education conferences as well as graphics conferences and workshops.

8. Equipment Needs

We would like to make SLIDE available on as many popular platforms as possible. The major platforms that we wish to support are Intel processors running WindowsNT or Linux, Unix workstations (especially SGI), and Machintosh. To make sure that we have code that compiles on each of the supported platforms, we will need to have easy access to one of each of these platforms. To keep performance expectations reasonable, we would like to obtain the state of the art of the low end lines of machines. These machines are indicative of the performance that home users will be able to afford in about two years.

8.1 Unix

Our research environment is primarily Unix based. We mainly use SGI workstations. We have several workstations which can be used for this development effort including an Octane and a couple of O2’s.

8.2 Intel

We currently have access to 3 year old Pentium Pro Intel machines which are quickly becoming outdated. We are beginning to experience hardware compatibility problems with new 3D graphics accelerator cards. We would like to purchase an SGI 540 Visual Work Station with Dual Pentium III Xeon CPU's. This system integrates the high-performance, low-cost Intel processors with SGI graphics accelerator hardware. This may be the graphics workstation of the future. If this system is not obtainable, then we would seek to get a high end Intel machine with third party 3D graphics acceleration.

8.3 Machintosh

Our group does not own any Machintosh equipment. In addition, all Apple equipment in the Graphics/Vision/GUI groups of the CS Division is at least 5 years old and is running old and nearly obsolete operating systems. One modern PowerPC G3 computer with sufficient memory and disk space must be purchased if we want to support an Apple version of the SLIDE system.

8.4 SLIDE Web Server

We plan to use the reliable and professionally maintained web server in the CS Division to distribute the SLIDE system. We will, however, need to purchase dedicated, routinely backed-up disk space (about 4GB) total to store the software, documentation, and examples for distribution.

8.5 Matching Funds

We plan to approach Intel, SGI, and Apple to see whether they will sell us one of these workstations at 50% of the typical discounted costs for educational institutions so as to fulfill the equipment matching requirements. If this is not possible we will draw the matching funds from other industrial donations.

References Cited

1. Smith, J., *SLIDE Scene Language for Interactive and Dynamic Environments*, UC Berkeley, 1999
   Homepage: http://www.cs.berkeley.edu/~ug/slide
   Language Doc: http://www.cs.berkeley.edu/~ug/slide/docs/slide
2. Christopher, W., Procter, S., Anderson, T., *The Nachos instructional operating system*, IN: USENIX Association. Proceedings of the Winter 1993 USENIX Conference. (USENIX Association. Proceedings of the Winter 1993 USENIX Conference, San Diego, CA, USA, 25-29 Jan. 1993). Berkeley, CA, USA: USENIX Assoc, 1993. p. 481-9.
   http://http.cs.berkeley.edu/~tea/nachos
3. Woo, M., Neider, J., Davis, T., *OpenGL Programming Guide Second Edition*, Silicon Graphics, Inc., 1997
   http://www.sgi.com/software/opengl
4. Ousterhout, J. K., *Tcl and the Tk Toolkit*, Addison Wesley Publishing Company, 1994.
   http://www.scriptics.com
5. Mead, C. and Conway, L., "The Caltech Intermediate Form for LSI Layout Description," *Introduction to VLSI Systems*, p. 115-127, Reading, MA: Addison-Wesley, 1980
6. Sequin, C., Strauss, P., *UniGrafix*, 20th Design Automation Conference Proc., 300 pp 374-381, June 28, 1983.
7. Sreekanth, A., *GLIDE 3.0 A Graphics Language for Interactive and Dynamic Environments*, UC Berkeley Masters Report, 1998
8. Hartman, J., Wernecke, J., *The VRML 2.0 Handbook*, Silicon Graphics, Inc., 1996.
   http://cosmosoftware.com/developer/handbook/
9. Wernecke, J., *The Inventor Mentor*, Silicon Graphics, Inc., 1994.
   http://www.sgi.com/developers/technology/graphics/inventor.html
10. Upstill, S., *The RenderMan Companion*, Pixar, 1990.
    http://www.pixar.com/products/renderman/toolkit/Toolkit/index.html
11. Foley, van Dam, Feiner, Hughes, *Computer Graphics: Principles and Practices, Second*

Edition, Addison-Wesley Publishing Company, Inc., 1990

12. Gritz, L., *BMRT: Blue Moon Rendering Tools*, 1994. http://www.bmrt.org/
13. *PhotoRealistic RenderMan*, Pixar, 1998. http://www.pixar.com/products/renderman/toolkit/Toolkit/rnotes-3.8.html
14. *dot*, Bell Labs/Lucent Graphviz Tools. http://www.research.att.com/sw/tools/graphviz/
15. *Java 3D*, Sun Microsystems. http://www.sun.com/desktop/java3d
16. *MOSIS*, Homepage of MOS Implementation Service. http://www.mosis.org/New/menu-main.html
17. *EDIF*, Homepage of Electronic Design Interchange Format. http://www.edif.org/about.html

Appendices

A. SLIDE Laboratory Assignments Text References

"Computer Graphics: Principles and Practices, Second Edition" by Foley, van Dam, Feiner, and Hughes [11] has been used as the text book during the last two offerings of our course. The sequence of the SLIDE laboratory assignments make reference to sections of this book in a rather non-linear sequence. The following list orders these references by the SLIDE assignment which first makes use of them:

1. **Interactive 2D Polygon Builder**
    - Geometry Construction Techniques, Section 8.3.2, p. 382-385
    - Sampling vs. Event-Driven Processing, Section 2.2.3, p. 42-48
2. **2D Polygon Editing and Morphing**
    - Pick Correlation, Section 2.2.6, p. 48-50
    - Dynamic Manipulation, Section 8.3.3, p. 386-388
    - Interpolation, Section 21.1.3, p. 1060-1064
3. **2D Dynamic Hierarchical Scene Renderer**
    - Mathematics for Computer Graphics, Appendix A, p. 1083-1111
    - Standard Rendering Pipeline, Section 18.3, p. 866-873
    - Geometric Transformations and Homogeneous Matrix Representation, Chapter 5, p. 201-226
    - Geometrical and Hierarchical Modeling, Section 7.1, p. 286
    - Extents and Bounding Volumes, Section 15.2.3, p. 660-663
4. **2D Bounding Box Rejection and Polygon Clipping**
    - Visibility Hierarchy, Section 15.2.6, p. 665
    - Elision and LOD, Section 7.13, p. 340-341
    - Cohen-Sutherland Outcodes, Section 3.12.3, p. 113-117
    - Sutherland-Hodgman Polygon Clipping, Section 3.14, p. 124-127

5. **3D Parallel Viewing and Crystal Ball Interface**
    - Projections and View Orientation Matrix, Section 6.5, p. 258-271
    - Plane Equation, Section 11.1.3, p. 476-477
    - Back Face Culling, Section 15.2.4, p. 663-664
    - Transforming Planes, Section 5.6, p. 216-217
    - Crystal Ball, Section 8.2.6, p. 376-387
6. **3D Perspective Viewing**
    - 4D Polygon Clipping, Section 6.5.3, p. 271-278
    - Stereopsis, Section 14.7, p. 616-617
7. **Polygon Scan Conversion and Software Z-Buffer**
    - Coherence, Section 15.2.1, p. 657
    - Polygon Filling, Section 3.6, p. 92-99
    - Filling Rules, Section 19.2.8, p. 964-965
    - Z-Buffer, Section 15.4, p. 668-672
    - Scan-line Algorithms, Section 15.6, p. 680-685
8. **Lighting and Shading**
    - Local Illumination Pipelines, Section 16.14.1, p. 806-809
    - Illumination and Shading, Chapter 16, p. 721-741
    - Diffuse or Lambertian Illumination, Section 16.1.2, p. 723-727
    - Specular or Phong Illumination, Section 16.1.4, p. 728-731
    - Gouraud Shading and Average Vertex Normals for Polyhedrons, Section 16.2.4, p. 736-737
9. **Procedural Modeling and Photo-realistic Rendering**
    - Representing Curves and Surfaces, Chapter 11, p. 471-529
    - Primitive Instancing and Sweeps, Section 12.3, p. 539-541
    - Procedural Models, Sections 20.2-20.4, p. 1018-1030
    - Global Illumination Pipelines, Section 16.14.2, p. 809
    - Visible-Surface Ray Tracing, Section 15.10, p. 701-715

- Recursive Ray Tracing, Section 16.12, p. 776-793
- Radiosity Methods, Section 16.12, p. 793-804
- Aliasing and Antialiasing, Section 14.10, p. 617-646

10. Final Project: Interactive Visualizition and Simulation

- Animation, Chapter 21, p. 1057-1080

B. SLIDE Student Survey

At the end of the Spring 1999 semester of CS184: Foundations of Computer Graphics, 44 students filled out surveys about the SLIDE graphics language, rendering system, and set of programming laboratories. The following shows their responses in a condensed form.

1. The things I liked best about SLIDE and the programming labs are:

Category	Repeats	Student Comment
Fun / Rewarding		
	8	Cool and Fun Assignments
	8	Visual aspect allowed to view results of efforts
Educational		
	9	Focused lab assignments closely followed lecture material
	1	"see formulas and concepts from class plug directly into code"
	5	Progressive or incremental learning of Rendering Pipeline
	5	Hands on approach / Practice of concepts
	2	Cohesive overall picture -- get to see total package from day one
	6	Learned a lot about the Rendering Pipeline
	1	Challenging
	2	Good interactions with TA’s and fellow students
Ease of Use		
	7	SLIDE language simple yet powerful

	4	SLIDE objects, instances, and groups made it easy to build simple scene hierarchies
	1	Primitive objects
	4	Tcl procedural object generation more convenient than flat data files
	5	Renderer program was easy to use
	6	On line documentation was good
	2	Good example .slf files for labs
Interactivity		
	3	Renderer runs with real time interactivity
	1	Crystal Ball UI
	6	Tk widgets useful for controlling object parameters
Labs' Source Code		
	11	Extensible and Modular C++ framework was easy to code in
	2	Learned about software engineering and good coding style
	1	Liked the Hungarian variable naming convention
	9	Fresh start for each lab assignment
	5	Areas of modification are well commented in the code
	2	Portability of code (can work at home)
	1	OpenGL is useful for debugging
	1	Fast to compile
	1	Much of the tedious coding done in advance

2. The things that should be redesigned or fixed are:

Category	Repeats	Student Comment	Response
Documentation			

	11	More documentation (i.e. Assignment 3)	
	4	More examples	
	1	Better OpenGL documentation	
	1	Better Tcl/Tk documentation	
	2	Teach a Tcl/Tk programming primer in the beginning of the semester	
	1	Better explanation of Scene Hierarchy	
	1	Assignment 3 is too hard	
	1	Better presentation of relevance of pure linear algebra	
	8	Matrix Transformations are hard	
Source Code			
	15	More comments in the code	More commenting of fields and procedure requirements will be added, but a balance must be set so that the students will be forced to reason out the problems themselves instead of simply following a list of instructions.
	4	Comment all flags better	

	2	Pre- and Post-condition comments for all Procedures	
	2	Some method names are counter intuitive (Set which returns reference which can be assigned a value)	
	2	Interface to Vector and Matrix package is unintuitive	
	2	Remove all Hungarian notation	Hungarian naming convention gives a short hand for type information. This is a valuable form of commenting and will not be removed.
Portability			
	5	Precompiled binaries and libraries for more platforms	We were only supporting the UC Berkeley instructional machines this semester, although the software is portable to any system which supports OpenGL and Tcl/Tk. In the future, SLIDE will be supported for arbitrary computers over the web.
	2	Dependence on environment variables and initialization scripts (window.tcl)	This needs to be addressed for packaging the system for download
SLIDE Language			

	1	True mirroring transformations	
	1	Switch cameras without totally modifying transformations	This is fully supported by having multiple render statements for the same viewport. Need to document this feature better.
	1	Give SLIDE viewports Tk path names for the bind command to use	As we move away from the SLIDE render statement controlling input more to having it all done in Tcl, we need to address this.
	1	Annoying that SLF_MOUSE_X and SLF_MOUSE_Y only update with mouse button down	Again need to revisit issues of data input to the system.
SLIDE Renderer			
	5	Memory leaks when reloading files	There are issues having to do with multiple windows which we need to redesign to fix this problem.
	3	Too slow	Profiling needs to be done, but it does work reasonably fast for reasonably sized scenes.
	1	Default for cylinder Z-slices seems to be 8, should be 1	
	1	Two sided lighting for SLF_HOLLOW	

	1	LOD flags don't work	It is a known bug that SLF_EDGES and SLF_BOUND will not work on static trees under OpenGL because the OpenGL display list mechanism will not allow it.
	3	Lights and Surface Reflectivity	There are problems having to do with translating SLIDE values into OpenGL light descriptions, but there may be implementation bugs on top of that which still need to be fixed.
	1	Point and Spot Lights do not illuminate much	This is a result of the parameters chosen by the student for the Phong lighting model.
	2	Error messages easier to read by sending to a file	Redirect output of SLIDE renderer to a file.
	5	Better error messages especially for Tcl blocks	
	1	Better debugging of .slf embedded Tcl code	
	1	Better error checking of data, viewer sometimes assert fails on bad values	It is difficult to do full checking when the .slf file is allowed to write any random value it wants. I am sure there are bugs to fix of this kind.

Based on the comments collected in this survey of CS184 students in Spring 1999, the consensus is that SLIDE is a beneficial tool for learning Computer Graphics. The following is a summary of the positive and negative aspects of SLIDE as told by the students.

The students had many positive comments about SLIDE. They were motivated to keep up with the pace of the assignments because the subject matter is fun and the results of the assignments

are rewarding and interactive. The labs have well defined descriptions and keep pace with the materials presented in lecture. The labs incrementally build up a complete software rendering pipeline. Taken as a whole, the labs form a well structured, cohesive system built in C++ on top of OpenGL and Tcl/Tk. Each lab starts with the solution to the previous lab and incrementally adds new rendering functionality. This fresh start policy was a welcome safety net for students, because it prevented them for being penalized all semester for any mistakes made in early assignments.

The SLIDE language is simple enough such that students could readily learn and apply it to build scenes. At the same time, it is powerful and flexible enough for students to build interesting, interactive 3D projects with it.

The negative comments of the students will be taken into account in refining the SLIDE system. Many students complained that the on line documentation was confusing and incomplete at times. A concerted effort was made through the course of the semester to provide enough information for the students to learn and complete their assignments, but more work needs to be done to refine the documentation and make it easy to reference. There is a common complaint that there were not enough comments in the provided skeleton code. The suggestions to write pre- and post-conditions for all the procedures and to describe all the flags better will be taken into consideration on the next iteration of the assignments. However, it is important not to over-comment or else the students will end up just following a list of instructions instead of reasoning out their own solutions. It is important to make sure the students become self sufficient programmers even though they are working within a frame work. This is a difficult balance to strike.

Many students complained that the work load was too heavy. Others complained that they were so bogged down by coding and implementation that they did not feel that they were learning the concepts. Based on these comments, it appears that we need to make a greater effort at providing conceptual materials in an easily accessible, streamlined manner. We also need to do a better job at motivating the students to digest the material before they start coding. We believe that if we could get students to follow these practices, then they would find that the actual implementation

time would be greatly reduced.