Using A GUI Shell to Focus Computer Graphics on Algorithms

Dr. Henry L. Welch, P.E.
Milwaukee School of Engineering

Abstract

Many computer graphics systems, such as the X Windows System, require a steep learning curve and extensive coding before even a single pixel can be activated on the screen. In a single quarter computer graphics class this places an extensive burden on the student who may have to generate a significant block of code that has little or nothing to do with the fundamentals of computer graphics algorithms. At the Milwaukee School of Engineering (MSOE) we have shifted the focus of our junior level computer graphics course for computer engineers from developing a complete application to one that focuses on the basic algorithms of computer graphics. This has been accomplished by supplying the students with a Motif based graphical user interface (GUI) shell which takes care of all of the implementation details, except for the drawing primitives necessary to redraw the screen and maintenance of the user data. This allows the student to develop applications of simple drawing and clipping algorithms while only suffering minimal interference from the overhead of a typical graphics package. This also simplifies compartmentalization of assignments into manageable one or two week units requiring at most a few hundred lines of student code. The results of using this in MSOE's CS321 will be presented.

Introduction

The goal of a computer science or computer engineering program is to provide work ready graduates who are familiar not only with how to write computer programs in various languages, but those familiar with basic algorithms and aspects of computing. Areas of study may often include file management, operating systems, compiler design and computer graphics. Depending upon the emphasis of a program these courses typically appear in either the junior or senior year and vary from required courses to electives. The area of computer graphics has many aspects which make it an ideal candidate for inclusion as a core course in the junior year. Not only does this course rely heavily on basic mathematics and algorithms derived directly from this mathematics, but it also provides the student with the opportunity to develop programs with directly visible results. In addition, many of the algorithms can be compartmentalized, providing a weekly or bi-weekly division of laboratory time.

A disadvantage of using computer graphics at this level is that the initial learning curve and overhead can be quite steep. It takes considerably more code for a program to read input from a mouse or other pointing device and then use that to format and control graphical output than it does to work with simple scanf's and printf's. And while the results are often more satisfying to the student since the programs now appear more modern and realistic, the overhead can be frustrating and prohibitive.

This paper focuses on an approach being tried at MSOE to focus on the positive aspects of computer graphics while attempting to minimize the overhead typical of such systems. In addition a secondary goal is to begin introducing students to the idea of interfacing code between multiple programmers. The background of MSOE's CS321 will be presented along with the overview of a graphical user interface (GUI) shell being used to improve the course. The results of using the shell during the Winter of 1996-97 will also be presented.

History of CS321 – Computer Graphics

A review of the old syllabus for CS321 shows that it follows a fairly traditional approach for computer graphics courses. The course discusses the idea of raster display techniques, reviews some basic mathematics, introduces line and other drawing algorithms, discusses polygons and fill strategies, and then moves into transformation techniques and clipping and eventually covering 3-D objects. Along the way additional information is shared on the ideas of curves, text, color, and shading. This is common approach in such texts as Hearn and Baker[1] and Foley, et al.[2]. This achieves the goal of introducing a number of mathematically well understood algorithms and places sufficient emphasis on how to look for and handle special cases. However, when it comes to programming assignments things tend to look different.

The previous approach to the laboratory in CS321 was having the student prove they were worthy of being a computer scientist or engineer. The first assignment was to develop a double-linked list data structure for storing generic graphical entities. This served the purposes of keeping the students busy until some graphics algorithms could be introduced, reinforced data structures, and provided the students with some utility routines for simplifying the later programming assignments. Then, with a minimum of instruction and preparation the students would be given a simple program for opening and initializing a simple graphics window and then told to write a program that could draw a whole spectrum of graphical objects ranging from points, to wire-frame and filled polygons, to text and arcs. Generally the list included all of the basic graphical entities for the X Windows System with input coming from a well-formatted file. Students were typically allotted less than three weeks to accomplish this and the race was on. During this time the typical student would generate over 2000 lines of code (in the *finished* product alone), yet they learned nothing about algorithms and graphics except how to call basic drawing routines. Subsequent assignments enhanced this initial system to add color and then basic object transformations. Most students, due to poor planning or time pressures, would end up having to rewrite most of their code for each assignment so that by the end of the term an average of 500-1000 lines of code per week were necessary for good laboratory grades. This is not practical and hardly models the normal state of affairs in an industry environment. In addition, many of the algorithms being taught in class were never being coded and those that were often ended up being lost behind the hack jobs and all-nighters.

The GUI Shell

Last winter I had the difficult task of taking over CS321 in the middle of the term. During the subsequent course evaluation process two improvements were suggested repeatedly by students. 1) Would it not be possible for the laboratory assignments to be more GUI based? and 2) The programming assignments were too long. I agreed with both of these assessments and decided to build a simple GUI shell to facilitate the overhead of the graphical system and to redevelop the laboratory assignments so that they focused on graphical algorithms and not extreme amounts of code. Figure 1 shows a screen dump of the Motif-based GUI shell designed for CS321 which is loosely based on the application in [3]. It consists of the four basic input areas typical of most GUIs and is designed so that unused features in the GUI can be ignored.

The menu bar features a number of simple commands for basic file management (Open, Save, and Save As) some simple editing commands (Reset, Erase, Zoom In and Zoom Out) and a simple Help system for which the student provides some code. The button panel to the right of the interface duplicates the editing commands as well as the Help and Quit options. This is to introduce the student to the idea of buttons and menus as input devices. At the bottom is a command line into which a user can type simple commands for adding as yet unspecified features. The upward pointing arrow allows the user to access recently typed commands. The main body of the GUI is a scroll-bar controlled drawing window that is sensitive to not only exposure events, but to the mouse based events common in an event driven graphical system.

The student is provided with the source and header files for using the system. The primary file contains all of the "system" routines that support the GUI shell. This includes management of the basic shell itself, handling the dialogs, and supporting call backs for the student implemented aspects of the shell. This file contains two types of routines those that handle information behind the scenes and those that are utility routines to support the user. This is designed to mimic the information hiding found in software libraries and object-oriented systems. Generally any system variable or graphical hook can be obtained from calling one of the utility functions. This file is provided in both object and source (for reference) forms.

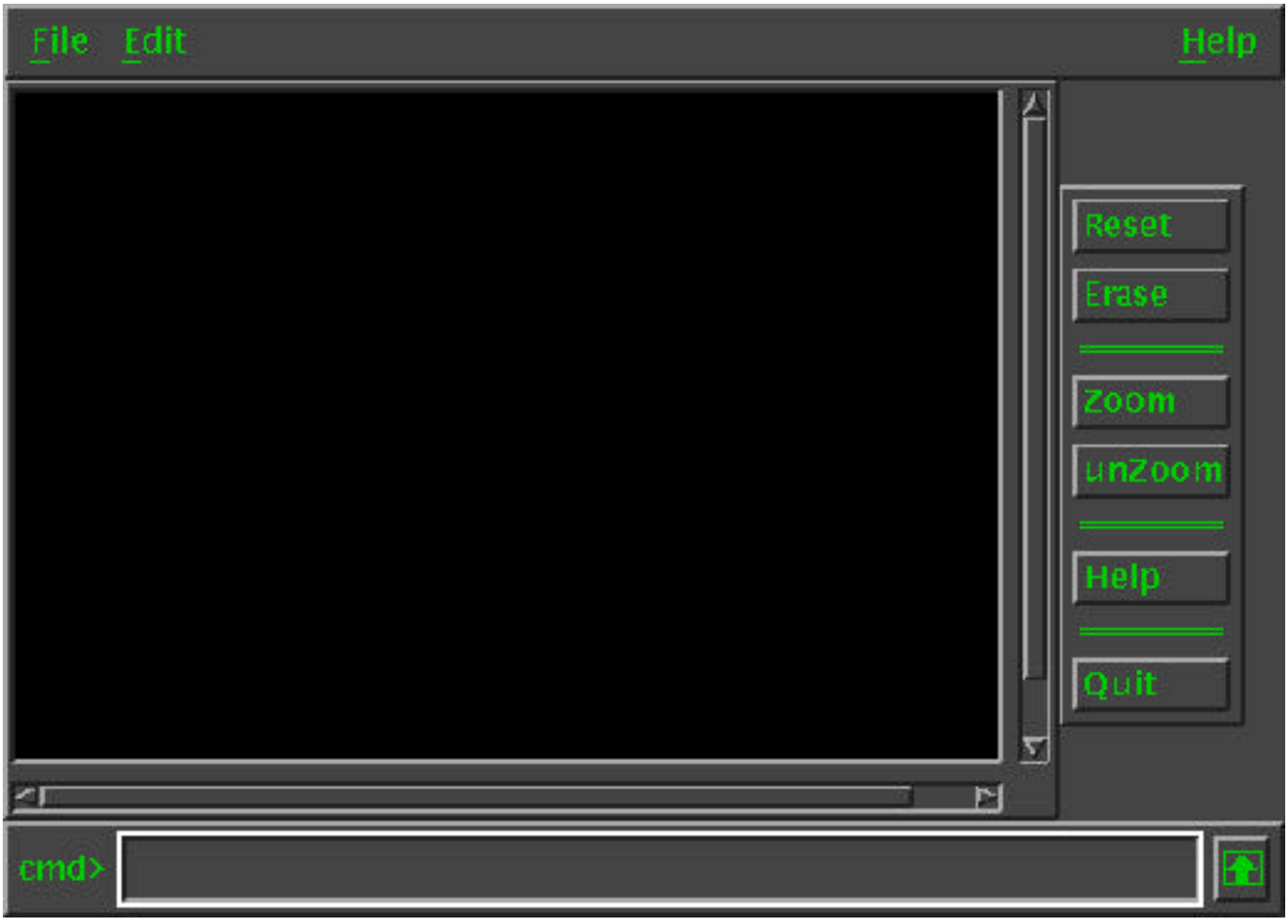

Figure 1 – The Motif-based GUI Shell for CS321

Additional source files contain the bare-bones of the code for the GUI shell. This contains the main program which loads the shell and all of the call back routines necessary to implement full functionality. Among the call back routines included are those for reading and writing files, handling all the buttons, processing the command line, responding to the help request, and handling the drawing window events such as mouse button presses and exposures.

A First Graphics Program

The first graphical programming assignment in CS321 is rather straight-forward and consists of four basic objectives. The first is to integrate the data structure from an earlier lab into the shell to facilitate the storage and retrieval of graphical entities. This is possible by simply treating the code as a software library. Second a simple command line parser needs to be developed that allows a user to specify which pixels in the drawing window to activate and third to implement the simple help call back for describing the syntax of the command line. The final objective involves handling events from the drawing window itself.

Since most of the drawing window events can be ignored this final objective consists of handling button presses on the mouse and exposure events. Button presses are fairly straight-forward and involve accessing the system generated event structure and adding the point specified to the list of those to draw. The exposure events are more complicated in that when the first occurs (after window creation) a graphical context for drawing must be created and then after that extract and display all of the points recorded to the window. A final requirement will be to free all dynamically allocated memory upon exit to foster good house-keeping skills.

Additional Assignments

A number of additional assignments have been developed and their the results of using them appear in the table below.

Assignment	Time	Avg. # of lines
Drawing lines (e.g. Bresenham's Algorithm) or circles and color	1 week	250
Drawing polygons and 2-D transformations	2 weeks	430
Polygon filling - **NOT ASSIGNED**	N/A	N/A
3-D polygons with transformations	2 weeks	300
Button in the drawing window	1 week	100

Conclusions

Feedback from students in CS321 this past Winter term has been favorable. After an initial period of confusion (caused by writing a program for which they were not the sole author) the majority of students adapted quite well to the new approach. Many students also requested a number of added features so that they could provide functionality beyond the minimum. As the code counts indicate the size of the assignments is more manageable and most students reported fewer frustrations with the graphics library. The use of the shell also promoted better programming practices since the programming focus is on more manageable units. The students also learned how to program in an event driven environment with the problems of sharing information between interrupt/event servicing routines and an asynchronous input stream.

References

[1] Hearn, D. and Baker, M. P., *Computer Graphics*, Prentice Hall, 1997.

[2] Foley, et al., *Computer Graphics: Principles and Practice*, Addison Wesley, 1996.

[3} Sebern, M., *Building OSF/Motif Applications: A Practical Introduction*, Prentice Hall, 1994.

Biography

Dr. Henry L. Welch is an Associate Professor of Electrical Engineering and Computer Science at the Milwaukee School of Engineering. He earned his Ph.D. in Computer and Systems Engineering from Rensselaer Polytechnic University in 1990. His primary teaching areas are in digital circuits, microprocessor systems, and advanced computer engineering topics such as computer graphics and fuzzy logic.