CMU-ITC-85-065

Using Andrew for Development of Educational Applications

David Trowbridge

Center for Design
of Educational Computing
Carnegie-Mellon University
June 2, 1985

Andrew is the computing environment under development by the Information Technology Center, a joint project of CMU and IBM. Andrew runs on a campus wide network of high function workstations. The two major components of Andrew are a file system and a user interface. The file system permits users to access their files from any workstation on campus. The user interface includes a window manager, a base editor, a rudimentary data base package, and tools for development of educational applications, as described in this note. At present, Andrew has been implemented on six different displays and three different workstations. Initial deployment of Andrew has been on Sun workstations.

1. Concepts

The Andrew window manager supports user-controllable multiple display windows. Applications programs (referred to as "clients") may be displayed in one or more windows. Typically, each window corresponds to a separate process.

Rather than controlling the screen directly, client programs communicate solely with the window manager. The window manager mediates the interaction between the user and the client. Keyboard input and mouse clicks go to the window manager which dispatches them to the proper client program. The client in turn processes the input and tells the window manager how to update the display. Client programs need not be running on the same workstation as the window manager that controls the display.

Since the size and shape of windows are ultimately set by the user, client programs should usually be designed to generate a reasonable display over a wide range of sizes and aspect ratios. The client program may suggest to the window manager the minimum and maximum dimensions it wants, but the screen "real estate" finally allocated for each display is the result of a "negotiation" between the client program and the user. If the allocated window is unusable by the client, the client should generate a request to the user to reshape the window to more acceptable dimensions, but it cannot guarantee that this request will be satisfied. In some cases, it may be desirable for clients of the window manager to generate a fixed size image. In such cases the image is clipped to fit the available window.

This discipline forces client programs to be more flexible and cooperative, which is important in a multi-process environment. It also anticipates a time when the system will be moved to many different kinds of hardware, trying to minimize the resulting disruption to client programs by hiding device dependencies in the window manager.

The system makes heavy use of a three-button mouse. The middle button is reserved for menus and for moving window boundaries. It is used to drag boundaries between windows, or to enlarge, reshape or hide windows via a pop-up menu selection. When the cursor is in a title bar, a menu for window manager options pops up; when the cursor is inside a window, the items on the menu depend upon information from the client program. The other buttons on the mouse may be used directly by client programs; for example, to draw lines or to position a text caret.

2. Underlying Software

If desired, client programs in Andrew may be written using the window manager alone. The window manager subroutine library has functions for defining and selecting fonts (wm_DefineFont and wm_SelectFont), for placing a string of characters at arbitrary pixel coordinates (wm_DrawString), for line drawing (wm_MoveTo and wm_DrawTo) and area fill (wm_FillTrapezoid). While all the basic tools for displaying text and graphics are available through function calls to the window manager, higher level tools are also available to aid in the creation of more sophisticated programs. Among these are the layout manager, the base editor and the grits subroutine libraries. The layout manager and base editor may be viewed as residing on top of the window manager; that is, while they depend upon and use the functions of the window manager, they package and extend window manager routines into higher level, more complex routines. Grits in turn uses base editor facilities and provides enhancements.

The layout manager partitions real estate within a window using "layouts," separate rectangles (either tiled or overlapped) which may be used for different purposes. For instance, one layout might be used for a document with a scroll bar, another for data base retrieval, another for animation. Each layout has associated with it procedures for redrawing it, updating it (an update is a partial redraw), handling mouse clicks within it, and displaying menus specific to the layout. Layouts may be cleared and redrawn independently.

Programs which require layout structures within windows or which make use of documents should use the base editor subroutine library. The base editor supports cutting, pasting and copying text within a document and across documents (including to and from documents in different windows), scrolling, and on-screen formatting of text into various typesetting styles such as italics, bold, and special fonts. Documents usually have a scroll bar on the left which shows what portion of the document is currently visible in the window. It displays a cursor for repositioning the document. Standard menus include items for saving documents, searching for strings, and changing text styles (new text styles appear immediately on the screen). Client programs may provide their own items for pop-up menus. In layouts which do not contain documents, client programs provide routines to the window manager for handling mouse clicks, for performing partial updates of the display when necessary, and for doing complete redraws whenever the window size is changed by the user.

Grits is a data base facility available at both a command-language level and as a subroutine library. It provides functions for storing, organizing, retrieving and displaying user data. The electronic mail and bulletin board programs use grits facilities, as does an online faculty/staff phone directory.

In general, programming with the base editor involves writing some initialization code and supplying specific handler routines for displaying pop-up menus, updating part of the window, redrawing the entire window, processing mouse hits, and responding to keystrokes. Typically, the last control structure in the main() block is an infinite loop:

```
while(TRUE)
    Interact();
```

Each time Interact is called it processes one "event": an input, a redraw request, or the firing of a timer.

3. Tools for Writing Client Programs

In addition to the subroutine libraries of the window manager, base editor, layout manager and grits, a few other tools are available to assist in the development of educational applications. Several of these are under development.

A Graphics Layout Organizer (glo) enables the designer to organize layouts on the screen and to revise them. Using the mouse to point, the author partitions the window into sub-windows ("layouts"). When the author is satisfied that the physical layout of the window is correct, a "prototype" of the program is created by selecting the menu item, "create prototype."

Those layouts which will be used for displaying documents are assigned functions from the base editor. Other layouts may be defined arbitrarily by specifying a HitProc (a procedure for processing mouse button presses), a RedrawProc (responsible for redrawing the display upon receiving a redraw request from the window manager), an AddMenuProc (for displaying pop-up menus), and an InitProc (for initialization). One then creates a file containing the C code for these procedures, and compiles it with a header file generated by the glo editor and the glo subroutine library. The result is an executable program whose layouts are editable using the glo editor.

Several drawing editors are under development. One, dubbed the Frame-oriented Animation Drawing (fad) editor, allows line drawing of "frames" using the mouse and animated interpolation of images between frames. The animation gradually transforms vertices in the first frame into corresponding vertices in the second frame. The fad editor is closely associated with glo, and can be used to generate animation within a glo window.

Another graphics editor, currently known as "banzai," is constraint based. The user defines points using the mouse and selects constraints (e.g., parallel to, congruent to, vertical or horizontal) to be applied to a collection of selected points. Figures are stored as ASCII files consisting of control structures and statements in the banzai programming language. They may be modified using either the banzai interactive graphics editor or by editing the source text file. Complex figures may be constructed by assembling routines which call on more primitive figures. An interface between client programs and the banzai subroutine library is currently under development.

Recently, a GKS (Graphics Kemal System) graphics package has been implemented in the Andrew environment. Applications programs may call routines for displaying and transforming a wide range of two-dimensional graphics images.

The standard suite of programs in BSD 4.2 UNIX are all available in Andrew. These include some utilities which are particularly germane to the creation of educational applications. For instance, **awk**, a pattern scanning and processing language is useful for creating and improving dialogs. An author may specify in a text file the patterns he would like to search for in a student's input string, the logic for branching on the analysis of that input, and the messages to display in response to the student. The awk program may be called from a window manager/base editor program created using the glo editor. Thus an author may specify a first draft for a dialog in an awk script, create an executable program using glo, and test the program with human subjects. As revisions are required in the logic of the dialog, the author may edit the awk script, re-execute the program without recompilation and test the result. Turnaround time for revisions may be reduced to minutes. Programs may be improved on the basis of testing with students in the target population immediately after the first draft is ready.

The UNIX utilities **lex** and **yacc** also contain powerful tools for analysis of student input. For instance, they can be used to build a general purpose expression parser, so that students may enter expressions in standard algebraic format, and programs can generate graphs or other output representing the mathematical or physical content of the expressions.

Exploration of these various tools has begun with the creation of several small educational applications, each of which exploit one or more of the capabilities described above.

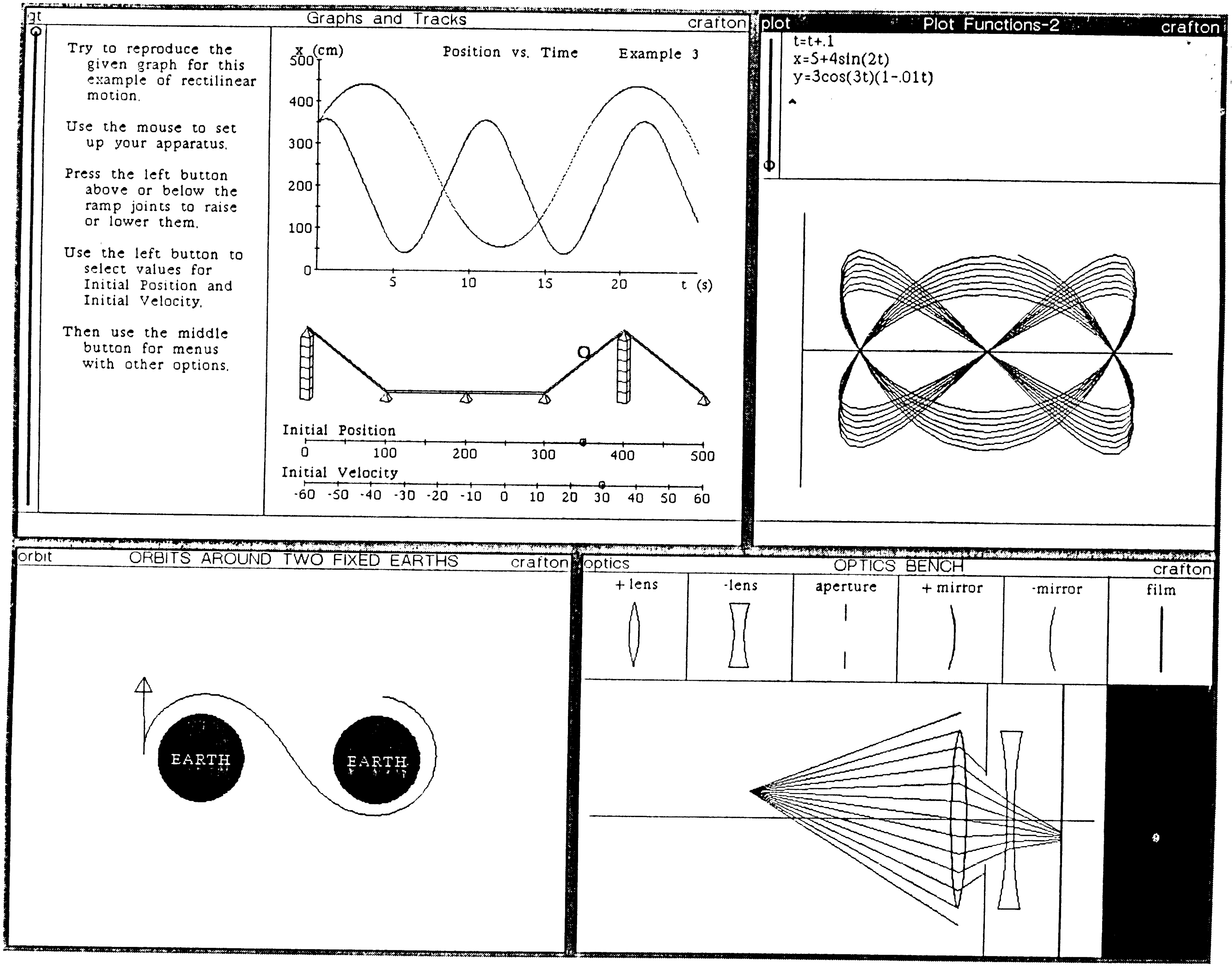
"Graphs and Tracks" was developed using the window manager and the base editor. It is designed to help students interpret graphs of position, velocity and acceleration, using animation of a ball rolling along an adjustable arrangement of sloping tracks, with simultaneous graph plotting.

"Orbit" allows students to experiment with trajectories of a mass subject to the gravitational attraction of two fixed "Earths." The student indicates the initial position and initial velocity vector using the mouse, and then observes the resulting orbit in two dimensions. Even without a floating point co-processor, such calculations are fast enough on the Sun 120 to produce interesting complex orbits within a few seconds.

"Optics" simulates an optics bench on which the student may place lenses and mirrors (convex and concave), an aperture and a plate of "film." Using the mouse, the student selects an origination point for a spray of rays which emanate from it, pass through the optical elements and strike the film. A face-on view of the film illustrates a facsimile of the image which would have been formed from the point source. Again, computation speed is sufficient to display a bundle of rays passing through several elements in a fraction of a second.

"Graph" is a program which parses algebraic expressions and draws graphs from them. It uses lex and yacc to parse the input and compile tokens into interpretable code at runtime.

"Vgraphs" teaches through questioning. The student is asked to give verbal interpretations of simple graphs of velocity versus time. The program analyzes the input and responds with kudos, help, or additional questions. It illustrates the use of awk for pattern matching of open-ended string input.



Try to reproduce the given graph for this example of rectilinear motion.

Use the mouse to set up your apparatus.

Press the left button above or below the ramp joints to raise or lower them.

Use the left button to select values for Initial Position and Initial Velocity.

Then use the middle button for menus with other options.

```
t=t+.1
x=5+4sin(2t)
y=3cos(3t)(1-.01t)
```