THE INITIAL START:
BEGINNING CAADD FOR THE BRAND NEW STUDENT

Michael Kennedy
College of Architecture
University of Kentucky
Lexington, Kentucky 40506

ABSTRACT

Described is a teaching system presently being used during the first five weeks of a first course in Computer Aided Architectural Design and Drafting. (After these five weeks students spend eleven weeks actively using a 2-D drafting package and a 3-D surface modeling package.)

It is the view of the author that a student can obtain much more from her or his first course in CAADD if some fundamental concepts are covered specifically and dramatically, rather than assumed or conveyed by osmosis. On the other hand, one does not want to significantly delay the teaching of the principal objective: how to use a computer as a partner in design and production. The answer to meeting these two divergent objectives is two-fold: (1) careful organization with computer based tutorials, and (2) integration of architectonic lessons during the introduction.

The objectives of the initial five weeks are (1) to demystify computers, (2) teach the fundamental concepts of computer systems relating to hardware (disks, cpu, memory, display), and software (programs, data, files), (3) illustrate programming and program design, and (4) convey the concept of discrete symbol manipulation and its relation to graphics and text.

INTRODUCTION

The University of Kentucky College of Architecture has had a program in Computer Aided Architectural Design and Drafting since the mid sixties -- although then we called it by different but equally uncomfortable names, such as Computer Applications to Architecture, or perhaps Computer Graphics for Architecture. Our objective in those days was to teach students how to build software to aid architects. We, along with other schools -- because it was in the schools that the action was occurring -- contributed to the "monument to ad hocery," that those times produced: writing special purpose programs which later expired, along with the machines on which they were implemented.

I won't delve at length into the changes which have occurred in the last twenty years -- how the light at the end of the tunnel of un-usefulness always seemed closer than it was, until suddenly, like a bullet train, we were out into the day. May it suffice to say that we have undergone a metamorphosis from teaching those who would make tools to teaching those who will use tools. We have tried to make this transition gracefully, remaining true to the belief that students needed to know some fundamental concepts about computers and information, rather than just how to use the magic slate.
A former dean of our acquaintance suggested, without a smile, that perhaps we ought to just put the CAADD equipment out in the hall and let the students learn how to use it on their own. That suggestion was the final prompting for this article.

We believe that students need to know something about information and information handlers -- about how computers do what they do -- before we let them drive off in a CAADD system. We believe this abbreviated form of "computer literacy" needs to be taught as painlessly, compactly, and enjoyably as possible, both so that we can get on to more important things (i.e., "doing architecture") and because most students, at least in our quasi three-dimensional art school, are disinclined to subject themselves to information with abstract, rather than graphical, underpinnings.

Each student who will graduate from our school will be required to have one 3 credit semester course in CAADD. Most students come to this course with considerable non- or misinformation, although the class is usually punctuated with one or two people who know a lot and make a point of it. We once experimented with sending our students over to the computing science department so we could later teach them the material we were interested in having them know; the results were disastrous. Now we have allocated about one third of the introductory course -- the first five weeks -- to the subjects we feel are necessary for the student to make effective use of CAADD equipment and principles. We:

-> Teach about computer storage: memory, hard disks, floppies

-> Warn about the lack of backups (we do this as a chorus; the students get sick of it; they also don't lose as much stuff as they used to)

-> Make the point that a computer is a discrete symbol manipulator, a not-too-bright machine with a finite number of binary states; that all the snazzy stuff it does -- such as graphics and speech generation -- are just functions of great speed, large numbers of small entities, smoke and mirrors.

-> Say that the combination of the fast and accurate, but stupid computer with the slow and sloppy, but smart human is a powerful one; we encourage the student to try to stay in charge.

-> Indicate that programming, editing, "numberizing" and number bashing, libraries, objects, transformations, operating systems, data bases and other non-parallel concepts are all important for making the future with the magic slate go smoothly.

The theory, if that is not too grand a word, underlying this introductory segment of our course is to slightly open several doors, presenting the simple but non-trivial examples of what we want to convey.

THE CHRONOLOGY

We have only ten or eleven lecture hours and slightly fewer laboratory hours to get this material covered so we move quickly. On "day one" we make an assignment. We hand each student a line drawing, such as Figure 1, below, and tell her or him to transmit its essence to another student, randomly chosen, in the class. The difficulty? The student must represent the drawing on paper using only the characters which might be found on a typewriter or computer terminal: no graphics allowed. The second student reads the character-based representation of the drawing and provides a graphical representation which, if both have done their job properly, is identical to the original drawing. The purpose of this exercise is to make the distinction between graphic representation and representation by symbols, and to show that each may be transformed into the
other -- as had better be the case, since computers, under their veneer, will only deal with
discrete symbols.

Figure 1

By the time this assignment is wound up and then unwound, we have been able to arrange two
hours per week for each student in our Computer Aided Design Studio (nicknamed CADS, as in
ZOUNDS). By booting the computer (which we explain does not mean kicking it until it does
what you want) the student brings up a simple, menu-driven "operating system" which covers the
activities the student will engage in during the introductory part of the course.

Choose from the following:
(1) Log in
(2) Turn the printer on
(3) Turn the printer off
(4) Show diskette directory
(5) Show hard disk directory
(6) Execute Kedit (editor)
(7) Backup files on a second diskette
(8) Restore backed-up files
(9) Execute GenPat
(10) Execute Drawbase
(11) Delete a Drawbase drawing
(12) Log out

(The menu may also be easily brought up from DOS. We are using IBM AT's, supercharged
with a lot of memory, "professional" color graphics, digitizers, and sub rosa clock crystals which
brings each machine up to the speed at which it was designed to run.)

After "concretizing" some of the concepts and equipment covered in lecture -- disks, files,
printers, plotters, and so on -- we introduce the students to a sophisticated but easy-to-learn
editor (not a word processor) called Kedit (available from the Mansfield Software Group, PO
Box 532, Storrs, CT 06268). After some introductory playing, to type name, address, school
information, and other biographical data, each student uses the editor to make a list of spaces in
some building. Each space has a name, size, finish code, level, numerical identifier, and so on.
These are typed in tabular form. Kedit has rudimentary sorting capability; the student is asked to
prepare lists of the spaces in several orders: alphabetical by name, by floor within finish code,
and others. In an hour or two, then, students have learned something about editing and files, a
small amount about databases, how such databases might be usefully manipulated, and some of

67
the pitfalls encountered in obtaining information from data.

SYMBOLS IN, GRAPHICS OUT: GenPat

Now the attention of the course returns to the second part of the first assignment, in which the student converted a drawing (Figure 1) from character to graphic representation. In solving this problem, some of the students, on their own, developed some coding techniques (e.g., "R 1.5" means draw a line 1.5 inches to the right). The time is ripe to introduce both the concepts of artificial language and of programming. We do this with an elementary, home grown language called GenPat, a pattern generating language (available from the author in December 1987).

The syntax and semantics of GenPat can be covered in two hours of lecture. It has only six verbs, or commands: FORM, TURN, FLIP, SIZE, DRAW, and NAME. All of its input is character based; virtually all of its output is graphic. Here is a synopsis of the language:

GenPat has two abstract "spaces": Prototype or P-space (memory) and Construction or C-space (visual, either plotter paper or computer monitor). The verbs of FORM, TURN, FLIP, and SIZE create and manipulate objects in the Prototype space. The verbs DRAW and NAME produce line drawings, in color, in the Construction space.

The FORM command allows the student to create, in the memory of the computer, a two-dimensional object made up of lines. (GenPat also has the capability of generating curves but we don't use it in this part of the course.) The object is created in Cartesian two-space and has both a reference point and a name. (Units are either in inches or centimeters; units are set at the time of installation of the GenPat interpreter.)

For example, the command

```
FORM teepee 0 5
ia -1 0
va 0 5
va 1 0
ia 0 5
va 0 2.5
va .5 2.5
```

will generate Figure 2. FORM produces the object named teepee with reference point (to be used later) at x=0 and y=5 (i.e., (0,5)). The object itself consists of an invisible (coded as "i") line to the absolute ("a") point (-1,0), a visible ("v") line drawn to the point (0,5), visible line to (1,0), invisible line to (0,5), and so on. (The line segment codes may also be "i" and "v", allowing lines to be drawn to relative, instead of absolute, points.)
Once an object is defined in the memory (P-space), it may be output using the DRAW command. The DRAW command produces rows and columns of the object on the screen or plotter. The syntax of the DRAW verb is:

```
DRAW color object_name ncols nrows xbeg ybeg xspc yspc
```

where ncols and nrows are the numbers of objects in the horizontal and vertical directions, respectively; xbeg and ybeg are the coordinates on the output where the reference point of the object which will occupy the lower left hand corner is to be placed; and xspc and yspc are the horizontal and vertical spacings between reference points of consecutive objects.

To generate this drawing:

```
DRAW brown teepee 3 2 2 6 3 2.5
```

Figure 2

Figure 3

69
That is, draw in brown 3 columns and 2 rows of the object "teepee" with the reference point of the lower left hand object located at x=2, y=6, and a spacing between consecutive reference points of 3 units horizontally and 2.5 units vertically.

**DRAW commands may be used as often as desired on any named object.**

The **NAME verb** takes the form

```
NAME color 'any text'
```

and simply identifies the drawing by putting 'any text', in the designated color, as a title on the drawing. **NAME** concludes the particular GenPat drawing that is then current.

**DRAW** and **NAME** are the only verbs which result in output -- the only ones which affect the construction space.

The verbs **TURN**, **FLIP**, and **SIZE** operate on the named object in the prototype, or **P-space**.

**TURN object_name degrees**

rotates the named object the designated number of degrees clockwise around its reference point in the **P-space**. This transformed object could then be **DRAWn**, or further transformed. For example, if "teepee" were as in Figure 2 in the Prototype space, and the command

```
TURN teepee 90
```

were given, "teepee" would be transformed to

![Figure 4](image)

**FLIP object_name axis**

produces the "mirror image" of the object with the axis of symmetry being either "horizontal" or "vertical". To continue with the transformation of "teepee" we might say

```
FLIP teepee horizontal
```

to produce:
To stretch or shrink objects we use

```
SIZE object name xmul ymul
```

where xmul and ymul are the multipliers of the x coordinates and the y coordinates of the object, respectively. The reference point is transformed as well. To shrink the "teepee" along its longer -- now horizontal -- dimension from 5 units to 2, while leaving its other dimension unchanged we could say:

```
SIZE teepee 0.4 1.0
```

to produce

in the Prototype space.

TURN, SIZE, and FLIP represent the complete set of linear transformations which may be performed on two dimensional objects.

A GenPat PROGRAM

The first GenPat assignment for the students is to produce the drawing that the student pair did by hand in the first week (Figure 1). The program would look something like this:

```
/* Form a square */
/* (Note use of relative, instead of absolute, line segment codes */

FORM quad 0 0
ia 0 0
```
vr 1 0
vr 0 1
vr -1 0
vr 0 -1

*/ Draw six quads */
*/ (The "color" contrast makes white on a black screen or black
on white paper) */

DRAW contrast quad 2 3 1 1 2 2

*/ Make a teepee 5 units high by 2 units wide. */

FORM teepee 0 5
ia -1 0
va 0 5
va 1 0
ia 0 3
va 0 2.5
va .5 2.5

*/ Put big teepee in place */
*/ (Since there is only one object, the spacing can be anything) */

DRAW contrast teepee 1 1 6 6 0 0

*/ Shrink teepee down to 2 units, turn it sideways, and draw it */

SIZE teepee 1 0.4
TURN teepee 270
DRAW brown teepee 1 1 7 5 0 0

*/ Flip it over and draw again */

FLIP teepee horizontal
DRAW green teepee 1 1 7 2 0 0

*/ Make and draw the border */

SIZE quad 9 7
DRAW contrast quad 1 1 0 0 0 0

*/ Finish it */

NAME contrast ' Assignment Number One '

Students can make quite nice drawings with this simple language. Actually, of course, the
language has the power to make any drawing which is composed of lines. Here is some student
work generated in previous semesters:
LESSONS & SUMMARY

The lessons which can be derived from the GenPat session are several:

-> The student learns something about programming and about the concept of an artificial language. GenPat has no decision making elements, no loops, no variables; it is admittedly short on many elements of programming. But the bones of the beast are there.

-> The student creates a file with an editor, modifies that file to produce the results desired. Backups are shown to be important. The file is read by the GenPat processor (we avoid subtleties such as "interpreter" and "compiler") and output corresponding to the input is produced. If the input is significantly changed and reprocessed, then different output is produced.

-> The student generates and works with a library of elements which may be used "as is" or modified at will.

-> The idea of a computer as a discrete symbol manipulator is reinforced; the concept of the computer as something magical is reduced.

Some architectonic lessons can be given as well. For example, the sequence of drawings (Figures 15 through 21) was produced by trivial parameter changes. The ideas of absolute and relative coordinates is introduced. Also, something can be made of the concept of a prototype space and a construction space. We also point out the balance between economy in building the prototype and economy in building the product. The concept could be called "sensible modularity" and illustrated this way:

Suppose we wished to construct nine one-unit square boxes in a regular, square pattern. With GenPat one could construct them all with the FORM command using a plethora of segment codes and a simple DRAW command. Another possibility is to use the simplest FORM command (FORM a line 0 0; ia 0 0; va 1 0) and use TURN and DRAW to do the construction. Other possibilities exist (an "ell" shape, for example) but the most sensible way is to make a square with FORM and to use DRAW to produce the end product. What real world application has this concept? Many systems (whether in construction or design) will have some costs associated with an early part of the process (examples: (1) building a module off site, or (2) drawing a detail) and other costs with the remainder of the process (e.g., (1) transporting the module to the site and putting it in place, or (2) referencing the detail drawing).

The final phase of our five week introduction is to have the student begin work with a 2-D drafting and database package called Drawbase (available from 3ok Systems, Cambridge, MA) which uses graphic input from a tablet for commands and coordinates. Students spend four weeks on this system and then learn the Sigma Design 3-D surface modeling system ARRIS (available from Sigma Design, Inc., Englewood, CO).

We find students who complete the five week introduction have an excellent grasp of many of the concepts frequently foreign to architecture students; certainly they have a better understanding of the foundations of CAADD than our students of a year or two ago who simply began learning a CAADD as the only element in the course. We believe that, despite the five week later start on CAADD proper, at the end of the semester students will have more familiarity with our two CAADD systems than previously, will have produced as much -- and will comprehend much more.