NARROWING THE GAP BETWEEN CAAD AND COMPUTER PROGRAMMING

A re-examination of the relationship between architects as computer-based designers and software engineers, authors of the CAAD environment.

MARK BURRY
School of Architecture and Building, Deakin University,
Woolstores Campus,1 heringhap Street, Geelong, Victoria ,
Australia 3217
E-mail: mburry@deakin.edu.au

Abstract. As the interfaces between computer users and computer programs become more riendly*, the gap between the architect as designer and the software engineer as strategist increases: the designer is even less inclined to relate to the internal workings of the computer program. One intention of the friendly interface is to allow greater access to relatively sophisticated programs for people to whom the computer is not a natural ally. But with tailor-made software comes the paradox of unintended prescriptive use. This paper discusses the need for design architects to learn basic programming skills in a contemporary context in order to avoid being inadvertently restricted by a software apparent flexibility. An undergraduate course that sets out to familiarise students with the new possibilities is presented here in detail.

1. Closing the Circle or Extending the Helix of CAAD Development?

en years ago, Mitchell, Liggett and Kvan (1987) closed their primer for computer graphics programmers with a statement:

hen it is viewed in a broad cultural perspective, computer graphics seems still to be at a stage, characteristic of infant media, in which the development of technique has far outstripped the ability of artists to understand its true potentials and to use it in a mature way.*

The preface to the book predicted that:

raftsmen (sic) will be as obsolete as scribes*

he sophistication that comes with using the computer in image making, combined with the user working knowledge of programming forms a central premise to the book, a means to enfranchise designers who are otherwise

CAADRIA '97: Proceedings of The Second Conference on Computer Aided Architectural Design Research in Asia. eds. Y-T Liu, J-Y Tsou, and J-H Hou. April 17-19, 1997. National Chioa Tung University, Hsinchu, TAIWAN. pp. 491-498

hobbled by a lack of hand-eye co-ordination.
The intervening decade seems a long time in the history of computer development. While much of the sagacity and many of their predictions are borne out, the evolution of computer use may not have changed at the same rate as the extraordinary pace of hardware development; the maturity in the use of the computer for architectural is yet to come.
For architects at least, The Art of Computer Graphics Programming remains an equally valid resource today as it was when published, but possibly for different reasons. This may be a reasonable (but unfashionable) assertion, of course, but the wider implications of architects learning to manage their relationship with the computer by interacting with its internal workings have other interesting implications: a symbiosis between the emotional human operative and computer-machine, the coolest of logicians. The central question behind this paper is the extent to which architects have been actively exposed to computer programming during the intervening ten years. This is postulated as having been rather less than might be expected. It is always timely to reflect on the value of a greater degree of interaction with the ubiquitous replacement to the drafting machine for designers who wish to go beyond simply operating the medium using the given software interface.

2. nges in Operability

In seeking to provide a more useful and versatile interface, experts and wizards may be making unfortunate assumptions based on the presumed expectations of the nth percentile of users. Software which set out patterns of use, however flexible, are less likely to suit the designer who, by nature, wishes to experiment. This has been recognised by the major CAAD software developers who have obligingly provided the designer with access to the internal workings of CAAD software through several levels of programming language.

f software becomes more intuitive to use, the motivation to learn to program, a characteristic of the early days of CAAD, dissipates. The aims of creativity should be as unbridled as possible by the media being used. A pencil being confronted by a blank piece of paper offers few inhibitions. Regardless of personal disposition, it seems unlikely that the physiognomy of a computer and screen will be as compelling as tactile media, regardless of one ability to design using traditional means. For the highly creative individual trapped behind an untalented pair of hands, the barrier that computers seem to put up in opposition to a more welcoming stance as esign media* still needs to be surmounted. How comfortable will the designer using the computer feel if they compare their modus operandi with that of the talented doodler? For the

one there is an implicit need to subscribe to a rules-based analytical approach to synthesis using the software programmer own strategic view of CAAD. For the other there is an implicit freedom from any such constraints. The antidote to this implied prescription is some degree of programming, either at the level of basic (scripted) customisation, development of iterative approaches to design within the software, or allowing complete access to the working code of the software. Judging how few architects bother even to build word processing macros it would seem that many seem content to accept software as it comes from the programmers. The advisability for the rekindling of any initiative for architects to produce design software for themselves cannot be argued . Many offices, however, continue to see their computers as more efficient drafting machines.

he design potential of the computer can be linked to a programmed design methodology or, more casually, be no more active than the benign acceptance and enjoyment of serendipity and the happy accident which computer use often occasions. There are at least two basic needs for future designers who will work principally on the computer: the need to understand their media by attempting even rudimentary programming and the need to experiment with software that allows a certain degree of parametric variation. This is done in the knowledge that any such move obliges the designer to consider a methodological approach, questionable in today design environment.

3. puting Programming within Education

Architects learn basic structural analysis in order to relate more effectively with the structural engineer. In the same way they can learn basic programming in order to relate more effectively with the machine they use and the software devised to complement the expression of their design intentions. Whilst the designer might be the least inclined to learn basic programming skills, they are the most likely to benefit. When a designer is confronted with a situation of their own devising such that programming support is required, either they must do the work themselves, or be sufficiently expert to brief a programmer to do the work for them.

3.1 CAAD PROGRAMMING AS COURSEWORK

As part of the Computer Applications course at Victoria University of Wellington, New Zealand , architecture students are required to undertake an assignment using AutoLISP. 50 students completed the course successfully in 1996. The project has been developed during the last four years (example: http://www.ab.deakin.edu.au/research/cal/caadria97/).

he Computer Applications course is an elective representing one sixth of the

1996 year grading for each candidate from fourth and fifth years. The programming assignment was worth approximately a third of the course grades, 5.5% of their grades for the year. The programming task complemented animation and word processing macro scripting assignments.

he students work individually and are required to write a routine invoked from within AutoCAD that asks a few pertinent questions about the principal dimensions and characteristics (parameters) for a window before inserting it into a plan view of a wall, trimming all lines in conflict. The students are also invited to consider alternative routines that will enable a window to be drawn in section or elevation to whatever configuration the user has given. The handout suggests that if the routines can draw these three conventional orthogonal 2D representations, then why not consider a routine which will draw a complete window in 3D? The students come from a variety of CAAD competency backgrounds; how they tackle the assignment accords to their prior experience in AutoCAD, if any. In this way, assessment can be tuned to thebindividuals* backgrounds although lack of previous experience in AutoCAD has not proven to be a major disadvantage.

3.2 CORE OBJECTIVES

This assignment provides the opportunity for students to demonstrate their ability to:

nderstand and develop the skills required in using AutoCAD's top level (as opposed to internal) programming tool (AutoLISP);

evelop a programme of proven benefit in a typical drawing office situation.

3.3 TASK

The task is set out as follows:

The task in this assignment is to produce a series of automated routines that describe and insert a window into a wall, the routine having been configured interactively by the user. This will be done in four separate stages:

insertion of window as a plan into 2D drafted walls of a small building;

insertion of window as a section into 2D drafted walls of a small building;

insertion of window as a elevation into 2D drafted elevations of a small building;

insertion of window as a 3D model into 3D model of a small building.

mplicit within this group of tasks is the opportunity for the student with initiative (and confidence) to proceed immediately to the 3D task and extract the required 2D sections from the 3D model.

3.4 OUTCOME

The hard-copy for both exercises is submitted modelled on the script contained in a brief introduction to AutoLISP techniques provided via the web (example: HYPERLINK http://www.ab.deakin.edu.au/research/cal/caadria97/)
http://www.ab.deakin.edu.au/research/cal/caadria97/) . They are required to provide laboured commented-out explanations of the purpose of each line of code (using a different font for clarity). All routines are submitted over the web for rigorous testing by the course co-ordinator.

Assessment is made on the basis of the functionality versus the complexity of each routine. The ideal result is a set of extremely simple routines that are highly functional. Each routine is tested in front of the class by a novice without assistance from the designer, other than through text-based pointers within the routine user interface. The routines should be readily familiarised by the user and fully operational. If the user defines parameters that are not physically possible they should be advised of this during and as part of the operation of the routine (a window too wide for a wall for instance).

The routines are measured for their efficiency measured as the length of time that they take to operate and the economy of means within the programming itself. Assessment is also made on the basis of the difficulty rating of the proposed routine and the effectiveness of the solution. Over-ambitious projects that do not yield results fare relatively poorly.

But is this caad or merely cad?

It is quite clear that the task described above is not an assignment that leads to highly theoretical design work; rather, it is unashamedly pragmatic in intention. In design terms it is somewhere between CAAD and CAD.

When this course was first run four years previous to the example given, there was a greater degree of flexibility in the choice of outcome with a commensurate level of ensuing dilemma. How could a routine which drew plants in 3D (taking into account wind direction), for instance, be compared with another which modeled staircases in 3D constraining the design to the relevant building codes? It was agreed thereafter to set a common project base from within which students could demonstrate lateral and, subject to interpretation, design ability. Those students who were capable of wringing-out a 3D version (always solid model based from within which sections could be obtained without difficulty) had, in effect, produced an iterative and interactive esign tool*. When sufficiently robust, fully modeled windows (that is, they include the glazing and the putty) can be inserted at will, in seconds, by proceeding window by window around the perimeter of a building,. Examples are shown as figures;

http://www.ab.deakin.edu.au/research/cal/caadria97/figures.

Compositional changes can be effected efficiently by deleting any particular item and reconfiguring the layout using the routine. In this way, and in

common with other proprietary software, a generic 3D modeling software can be tailored to a level of interactive design tool. The difference between proprietary packages and the user-programmed adapted software is the greater opportunity to tune the routine to an individual modus operandi and expression.

3.6 TIMELINE CONSIDERATIONS

The outcome from the course work has always exceeded expectation, and each year has improved on the previous. There was no clear indication on how long students actually spent on the assignment but is was clear that once the initial negativity to the idea of rogramming* had dissolved, the assignment represented a worthwhile challenge to students. In many cases, those with no previous programming experience did unexpectedly well. From the instructors* point of view, the greatest revelation was that in New Zealand at least, many students came to university from high school with basic programming experience. It was not such a tall order, after all, to offer students the chance to experiment with graphically orientated computer programming.

Having moved the assignment to a new country to a school with no tradition in CAD (or CAAD) programming, a six week summer research scholarship was offered to a second year student with only the most basic 2D knowledge of AutoCAD. The exercise was to introduce the project to colleagues and obtain data in control conditions to ascertain how long this level of programming actually requires. In this worked case study, the student not only succeeded to the level of robust routine within the time frame , he also produced the basic material for a framed and independently scrollable hypertext based tutorial. The revised tutorial will include each of the principal AutoLISP expressions which can call up an appropriate worked example using the web. This work will form the basis of senior computer electives in due course.

4. Concluding observations

It is not such a tall order to introduce architecture students to computer programming. This commentary concludes with some observations about user programming compared with high-end proprietary software which include opportunities to work with parametric variation.

4.1 PARAMETRIC VARIATION AND ASSOCIATIVE GEOMETRY

The term parametric variation or design is used broadly to describe a range of techniques where a design is described by a set of relationships rather than explicit geometry such as a line going from * to * where a and b are

immutably defined co-ordinates. If entities have dimensions and relationships that can be reassessed and changed using a graphical interface, opportunities for iterative design experimentation are made possible without the need for user programming. Such tools exist but are relatively expensive and surprisingly constraining.

The limitations were mooted at the time when parametric variation first emerged within proprietary software (Miller, 1990). Indeed, its potential may be restricted to design development rather than conceptual design (Burry, 1996). The problem with any sequential programming for design is an imbedded assumption that the design process itself is sequential. If a square is invoked early in the design process as the appropriate generating geometry, only to be reconsidered at a later stage in favour of, say, a circle, all that came after the original square is lost. Artificial Intelligence, as such, has not reached a level of being capable of anticipating what the designer would have done at each stage where the original decision making involving the square was compromised by the later introduction of the circle as generator.

4.2 PARAMETRIC VARIABILITY VERSUS USER PROGRAMMING

With each new version of lower-end software, a greater degree of parametric variation is included. AutoCAD, for instance, introduced rip points*. These allow the nodes of hapes* to be moved, controlled by eye with the mouse or through changing co-ordinate values. Useful to a degree, parametric revision represents only a fraction of the capability of parametric modules supplied with high-end packages. High-end packages, however, represent investments of $50,000 + for a typical mechanical engineering package involving a single operator at any given time. This is beyond the needs of most institutions and practices who are unlikely to work at this level, especially as the construction industry is geared to the lowest common denominator in terms of performance (simple forms are cheaper than the more complex).

The student exercise described above is, in fact, a parametric design tool, subject to the same limitations as proprietary modules, perhaps, but capable of being revised at source rather than simply being a factor of implementation. It is beyond the scope of this paper to report on the further experimental possibilities that programming provides. Once an architectural designer has reached the level of competence described above, they are in a better position to work creatively in tandem with the computer. They will have assumed an unusual degree of symbiosis with the software engineer rather than being merely reliant on a generic CAAD solution.

Acknowledgments

The work described is part of the Computer Applications elective run at Victoria University of Wellington, New Zealand. The author acknowledges the role of his former colleagues in formulating the project described above.

References

Burry, M.C. (1996). Parametric Design and the Sagrada Fam瑿ia. Architectural Research Quarterly (ARQ) Issue 5. London, UK. June 1996

Miller, F.C. (1990). Form Processing Workshop: Architectural Design and Solid Modelling at MIT. In (eds) McCullough, M., Mitchell, W.J. and Purcell, P. The Electronic Design Studio. pp441-455. The MIT Press, Cambridge, Massachusetts.

Mitchell, W.J., Liggett, R.S., Kvan, T., (1987). The Art of Computer Graphics Programming. Van Nostrand Reinhold Company, New York