Introductory Computer Programming as a Means for Extending Spatial and Temporal Understanding

Mark Burry, School of Architecture and Building, Deakin University, Australia

Sambit Datta, School of Architecture and Building, Deakin University, Australia

Simon Anson, School of Architecture and Building, Deakin University, Australia

Abstract

Should computer programming be taught within schools of architecture?

Incorporating even low-level computer programming within architectural education curricula is a matter of debate but we have found it useful to do so for two reasons: as an introduction or at least a consolidation of the realm of descriptive geometry and in providing an environment for experimenting in morphological time-based change.

Mathematics and descriptive geometry formed a significant proportion of architectural education until the end of the 19th century. This proportion has declined in contemporary curricula, possibly at some cost for despite major advances in automated manufacture, Cartesian measurement is still the principal 'language' with which to describe building for construction purposes. When computer programming is used as a platform for instruction in logic and spatial representation, the waning interest in mathematics as a basis for spatial description can be readdressed using a left-field approach. Students gain insights into topology, Cartesian space and morphology through programmatic form finding, as opposed to through direct manipulation.

In this context, it matters to the architect-programmer how the program operates more than what it does. This paper describes an assignment where students are given a figurative conceptual space comprising the three Cartesian axes with a cube at its centre. Six Phileban solids mark the Cartesian axial limits to the space. Any point in this space represents a hybrid of one, two or three transformations from the central cube towards the various Phileban solids. Students are asked to predict the topological and morphological outcomes of the operations. Through programming, they become aware of morphogenesis and hybridisation. Here we articulate the hypothesis above and report on the outcome from a student group, whose work reveals wider learning opportunities for architecture students in computer programming than conventionally assumed.

1 Reasons for programming computers in architecture schools

The computer has been embraced early by practicing architects and therefore also within the education of architects. Schools still tend to be at the vanguard of experimental computer use, particularly in the area of design. Efficiency gains, especially in office management and streamlining construction documentation are still the usual focus of practices conducting research into computer use to enhance professional opportunities. The use of programming for structured problem solving in design, however, has a long tradition (Knuth, 1968; Weizenbaum, 1976; Winograd, 1986; Mitchell, 1987). Now that courses in CAD (computer-aided drafting, see note 1) are becoming less central to school curricula as successive waves of students enter more comfortable and familiar with the computer in general, there may be two main streams to CAD education. Typically there is one that continues to provide students with basic computer-aided drafting skills. This may be complemented by a possible second stream that looks at new opportunities to inform the design process through the use of the

computer (CAAD - computer-aided architectural design). Going beyond the software designer's interface may not be included in either of these two streams, yet the outcome of an extended education in both CAD and CAAD can be significantly enhanced by doing so (Fox, 1989, Frew, 1989). Programming is a test of understanding with Weizenbaum making the landmark observation that

"One programs not because one understands, but in order to come to understand. Programming is an act of design. To write a program is to legislate the laws for a world one first has to create in imagination"(Weizenbaum, 1976 p 108).

The inclusion of low-level programming into the curriculum is one route to a more thorough understanding and exploitation of the computer's potential. Programmatically, a two-fold approach to finding form is necessary to explore design possibilities. First, one needs to frame the problem, and then one may explore the domain of possibilities that the frame establishes.

2 CAD programming, CAAD programming

Learning to customise software can enhance CAD use. Such useful adaptation aligned to an office management protocol or to an individual's particular preferences is encouraged by nearly all major architectural design and drafting software. Potential productivity gains are quite considerable. The work of specialist CAD managers in larger offices will generally include making the software conform more to the office's pre-CAD methodology in preference to directing the office towards adapting to new and quite different opportunities provided through CAD. Most customisation may involve the incorporation of some logic as a systemic response but it is less likely to involve any mathematics. Customisation of software at school level does not appear to be a major interest but even in schools where programming is taught at all, the majority of educators will probably feel that learning to customise software is a sufficiently risky incursion into this improbable domain for architects.

There are possibilities for programming in design. Streich presents a thorough discussion in favour of including programming knowledge in the design education of an architect (Streich, 1992). Maeda develops a pedagogy based on "design by numbers" and the understanding of visual design using computational processes and media (Maeda, 1999). Computation and design have not been linked significantly in most contemporary architectural education – the exceptions are notable. Opportunities arising from such an association are not especially obvious. There is, strangely, both more and less reason to consider computation in greater detail following the exit of Modernism as the core focus to design education. A liberating pluralism has replaced the didactic imperatives of the Modern Movement; pluralism sponsors a more liberal approach than the instructed method born of didacticism. But while pluralism might discredit the architectural program (computer or otherwise) as tending towards the 'progrom', equally, the program needs to be explored before being rejected for an imputed unhelpful linearity. This is to say, *it matters how the program operates more than what the program does*. The word 'program' is in fact unhelpful as it implies an attendant methodology, and a route to a known end. As a customising regime for CAD, such a view is both correct and logically defendable. Within the context of CAAD and the design process (see note 2) , however, the challenge is for the computer programmer to devise some 'tools' that are complementary to the designer's particular route to a design without pigeon-holing the designer into a methodology that ends up being substantially controlling. Reviewing the software available for architects, usually marketed as CAD rather than CAAD, reveals some interesting traits. The use of software programs to explore constructed worlds and their relation to craft and abstraction is developed in McCullough (McCullough 1997). The intricate relationships between computers, software and design are well researched (Paul Adler 1992; Winograd 1996).

Office efficiency and productivity are the undisguised focus of volume market software for architects, especially in building representation. A few architects who are atypically able to access a medium that is considerably more costly than conventional architectural software are using software directed towards computer animators in the film industry. There is recognition of greater design potential here. In matters of form finding, such esoteric animation software offers opportunities that are inappropriate within typical CAD software use. The algorithms incorporated within the animation software to facilitate rapid production of sequential frames, each a subtle morphological variant on the previous, provide new opportunities for form finding. The work of Greg Lynn, for instance, clearly demonstrates the potential of animation film software to influence architectural spatial design experimentation (Lynn 1998). The software user in this instance does not need to be a mathematician.

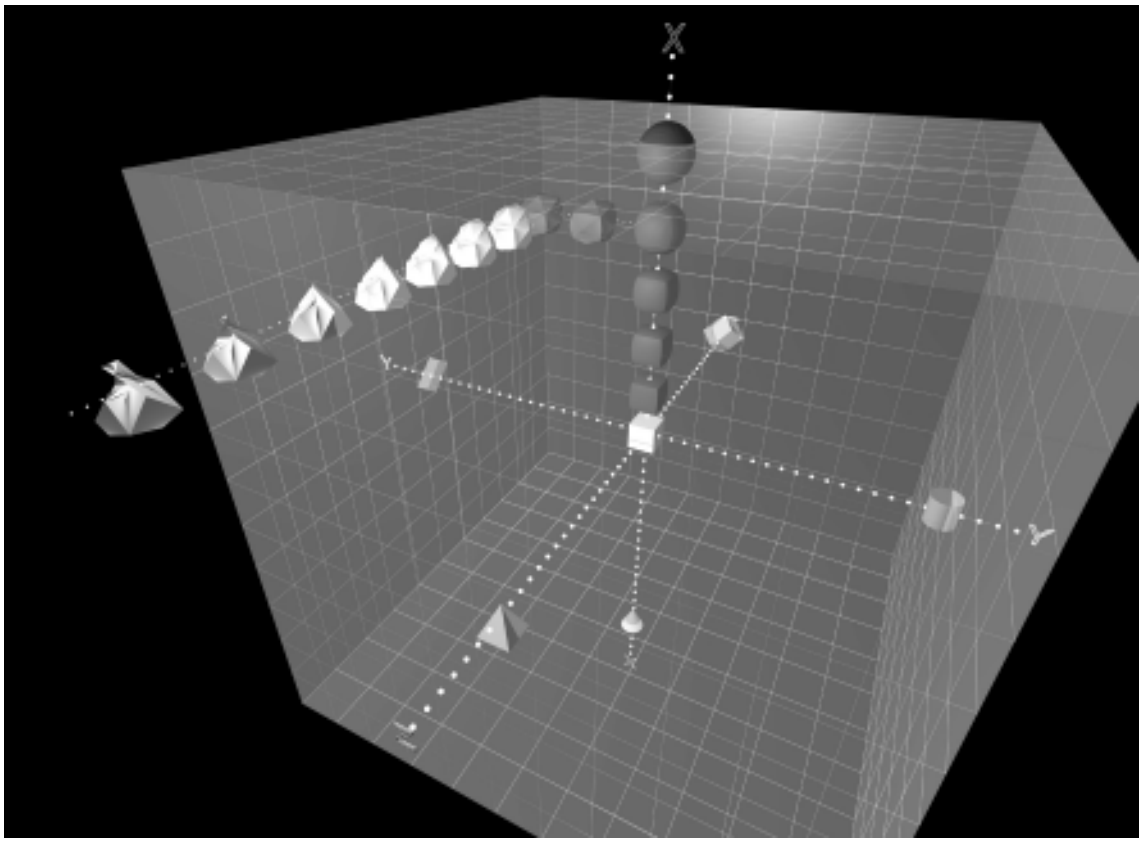

Figure 1. 'Our World'. The programming assignment required that a central form (in this case a cube) undergo a series of transformations. As an interactive program, the user can take the original form and steer it partly towards a sphere (in the example above), then partly towards a wedge, and finishing on a path beyond the condition of a pyramid.

It is unlikely that animation software engineers can anticipate all the needs for the designer working in a pluralist environment. *"If the designers themselves knew something of programming would they not be better equipped to commission software more closely aligned to their needs?"* This is the central question articulated in this paper as it reports on the outcome from a student group encountering low-level programming for the first time. The opportunities for learning through computer programming revealed by the work go beyond learning opportunities elsewhere in the curriculum. In the next section we sketch how Weizenbaum's "world" is developed along with the "legislation" of its functioning through a process of programmatic form-finding.

3 'Our World'

The universe of discourse, our world, is represented by a figurative spatial environment (figure 1). At the centre of 'our world' there is the form of a cube. The domain has been named in order to assist the conceptual understanding of a space used to marshal ideas rather than represent occupancy. A given number of morphological variants are located by the boundaries of this 3D space. They range between the initiating cube at the centre and six other Phileban forms that are located at each of the two ends of the three Cartesian axes, thus setting the limits of Our World (operation '1'). On the axial planes within this space at any other intermediate positions, there are hybrids of two transformations (operations '1' and '2'). Hybrids from three sequential transformations (operations '1', '2' and '3') are associated with the remaining Cartesian locations within the space. This construct exists simply to aid recognition of the possibilities to sponsor morphological change. The operation 1->2->3, is a statement of three sequential transformations on the original cube. Used as an interactive program, the user has taken the original form partly towards a sphere, then partly towards a wedge, and finished on a path beyond a pyramid.

Predictions of the topological and morphological outcome of the three operations are sought from the students. The purpose of the exercise is for participants to gain further insights into topology, Cartesian space, morphology, morphogenesis, and hybridisation. The participants also become very conscious of the power of this level of computer - user interaction through programming, as opposed to simple one-off and direct manipulation. They now experience a completely different relationship with the computer, as designers. It is a relationship redolent with opportunity for experimentation and innovative form seeking even at this rather trivial level (in terms of actual programming).

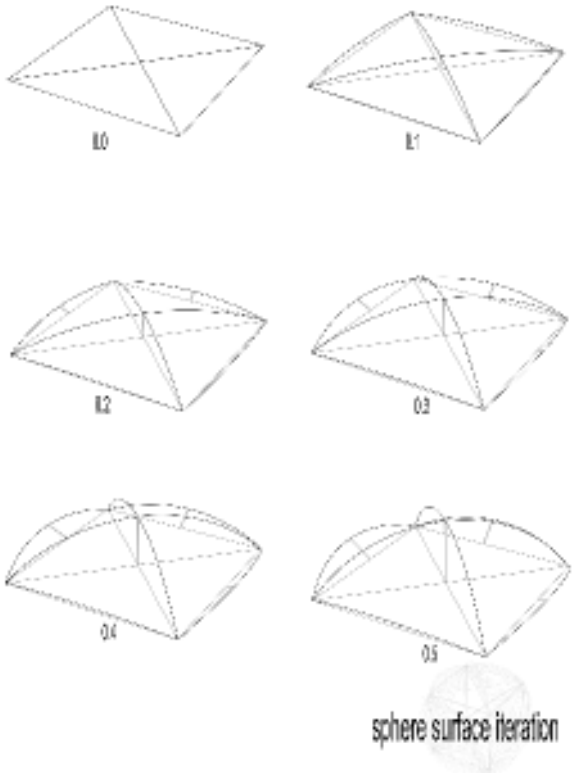

Figure 2. Sequence of changes to the twenty-four surfaces that describe both the original cube and the sphere showing the cube tending towards the sphere.

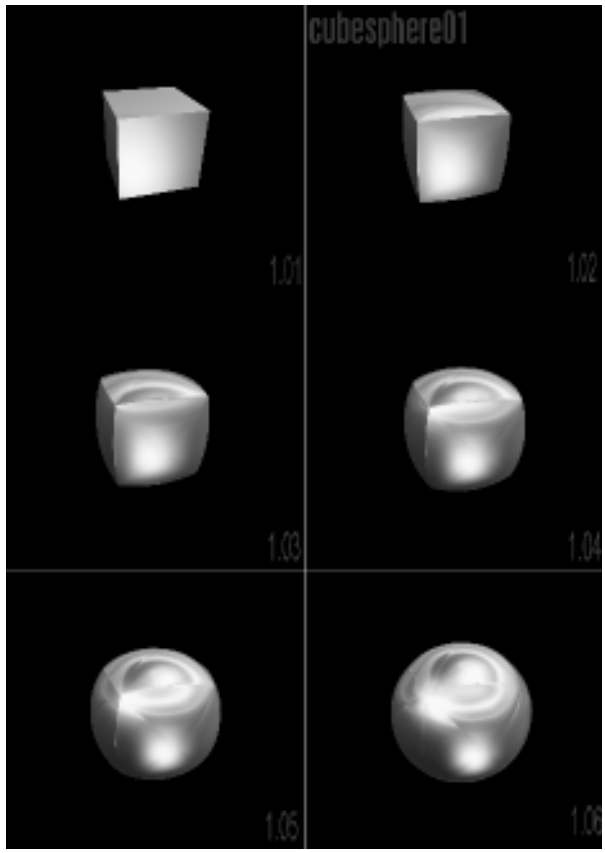

Figure 3. Sequence of changes to the twenty-four surfaces that describe both the original cube and the sphere showing the cube tending towards the sphere. The imperfections to the surface of the sphere are deficiencies in the surface algorithms in the chosen software.

4 Same topology, different morphology

Figure 2 shows the iterative route from cube to sphere. All the students undertake this first planning exercise. They have to devise a collective topological approach within given software in order to effect a smooth transformation from one to the other. All subsequent transformations are allocated as individual problem solving tasks (for example student 1 transforms a cube into a pyramid, student 2 works towards a cone, student 3...). So that the individual algorithms can be combined and the route 1->2->3 (->...) can be any of forty-eight unique combinations, all individual outputs must be compatible. Each transformation must occur iteratively, the number of iterations being at the discretion of the programme user. Each iteration or morphological change event outputs a file used finally as one of a series of frames in animation.

The highest common denominator in this process is the cube. It has the most parameters: six faces versus the single spherical surface etc. Conceptually a common understanding is required. Points and lines have names conforming to an agreed shared notation to ensure that each algorithm operates with common symbols and is therefore interoperative. The face of the cube is divided into four triangles (figure 2): these triangles are essentially the foundation class for the project. The program needs to convert the linear curves that bound the triangular cube face components into non-linear curves - in the case of the sphere these curves will be sector arcs. Also, in the case of the sphere, the points at the corners of the cube correspond with their equivalents for the sphere: they are coincident. Therefore a point that describes a mid-point for each curve moves taking the curve with it with the result that an associated surface transforms from a plane to a sector of a sphere. The result is shown in figure 3, and an equivalent transformation from cube to cylinder is shown in figures 4 and 5.

Regardless of the mathematical simplicity, there are many unfamiliar aspects for architects in this

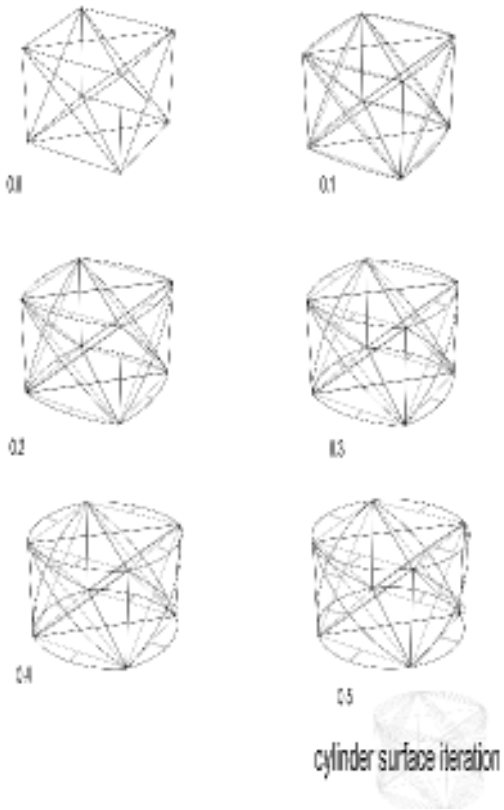

Figure 4. Sequence of changes to the twenty-four surfaces that describe both the original cube and the cylinder (see figure 1). This figure shows the cube tending towards the cylinder. The imperfections to the surface of the cylinder are deficiencies in the surface algorithms in the chosen software.

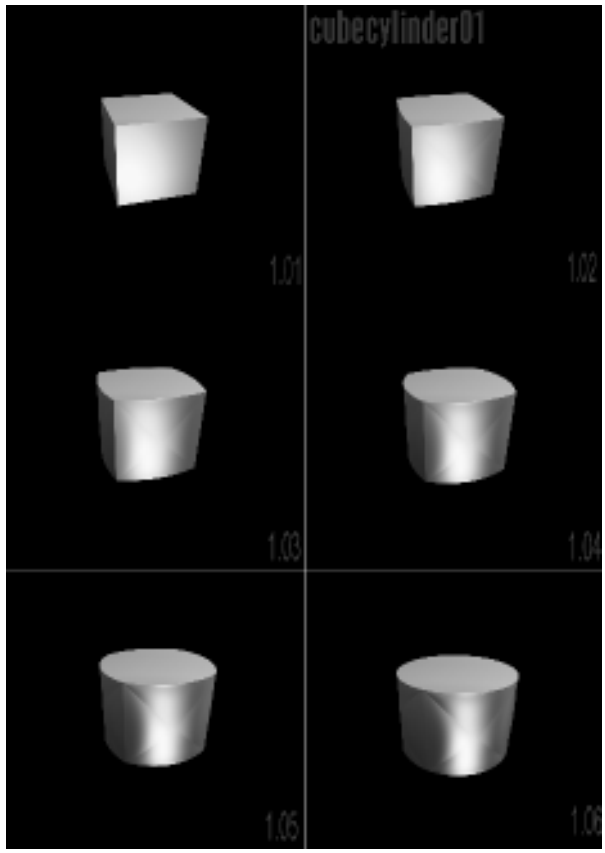

Figure 5. Sequence of changes to the twenty-four surfaces that describe both the original cube and the cylinder (see figure 1). This figure shows the cube tending towards the cylinder. The imperfections to the surface of the cylinder are deficiencies in the surface algorithms in the chosen software.

procedure. It serves to kindle interest and subsequently more complex problems can be approached with confidence. Reducing a cube to a pyramid, for instance, leaves the problem of what to do with the top face that reduces to nothing while the change from a cube to a cone has even more philosophical and practical difficulties. Here the algorithm reduces the top face of the cube to a point that is the apex to the cone (as for the pyramid). As its area diminishes, the base transforms from a square to a circle. The intermediate iterations show a rounding square at the base with a perfect reducing square at top until the apex of the cone is reached. This is an aesthetic problem, and the programmer is obliged to include a proportional system to compensate for an otherwise inconvenient visual effect

5 Stepping outside 'our world'

The students have thus learned the fundamentals of programming. If they had no real grasp of Cartesian space before the project (and it is surprising how many of our students do not) they do so after completing this project. They have also learnt about the complexities of formal problem solving using both logic and mathematics. In the realm of aesthetics, however, they are challenged with quite unfamiliar circumstances: semiautomatic design. Most students can predict the effects of 1->2 transformations, if not mentally they are at least able to derive the outcome through sketching. Predicting the results of the third transformation step seems to be especially demanding, and the outcome from a fourth transformation step impossible to visualise it seems (figure 6). This represents an unusual dilemma. The computer can often produce interesting and unpredictable forms through error, which are unrepeatable and probably unable to be used in making buildings. With this programming assignment students are able to act as agents for curious yet computationally (but not intellectually) predictable forms that are repeatable, and therefore of potential use in architectural design. Where matters move into fundamentally new territory is stepping outside the Phileban territory we have called 'our world'. Here the possibilities are quite extraordinary (figures 7 and 8). Through producing more iterations than proportional steps, forms of self-intersecting yet topologically

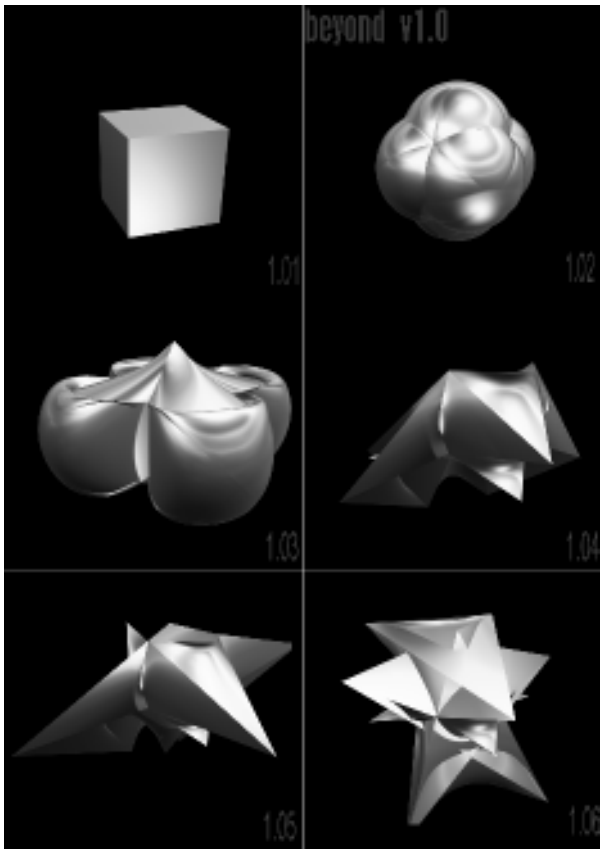

Figure 6. Sequence of changes to the twenty-four surfaces that describe both the original cube and a hybrid form (see figure 1 steps '1' '2' and '3'). This figure shows the cube tending towards first a sphere (partway) then partway towards a pyramid and finally a rhomboid.

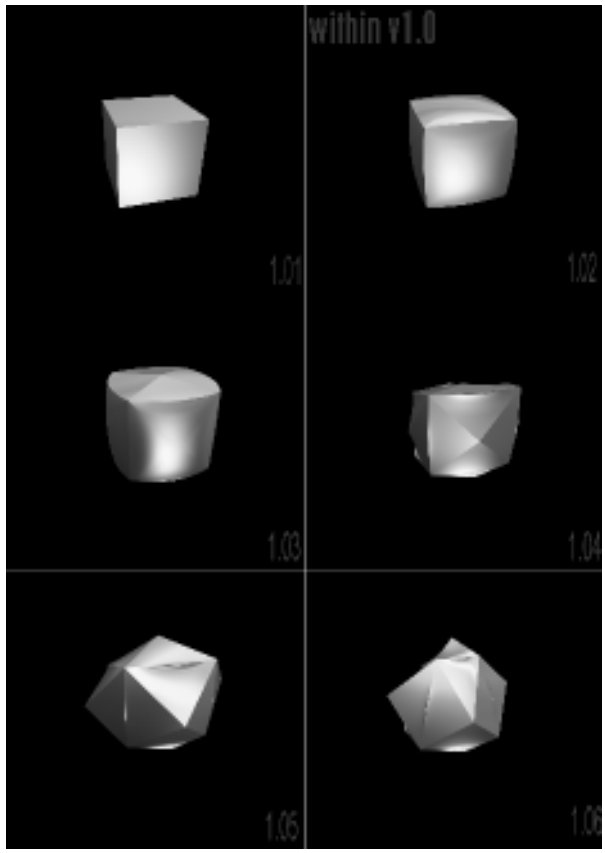

Figure 7. Too much hybrid vigour. The figure shows a sequence of changes that go beyond the boundaries to 'our world' (figure 1). The original cube has developed into a hybrid form (see figure 1 steps '1' '2' and '3') that distorts the original Phileban objects to guided but conceptually unpredictable outcomes.

equivalent descriptions challenge the designer to find a practical use for their new programming skills.

References

Fox, C. W. (1989). Integrating computing into an Architectural Undergraduate program. In *The Electronic Design Studio: Architectural Knowledge and Media in the Computer Era*. Malcolm McCullough, W.J. Mitchell and P. Purcell (eds).The MIT press**.** 377-386.

Frew, R. S. (1989). The organisation of CAD teaching in design schools. In *The Electronic Design Studio: Architectural Knowledge and Media in the Computer Era*. Malcolm McCullough, W.J. Mitchell and P. Purcell (eds).The MIT press**.** 387-392.

Knuth, D. E. (1968). *The art of computer programming*. Addison-Wesley Publishing Company. Reading, Mass.

Lynn, G. (1998). *Animate Form*. Princeton Architectural Press, New Jersey.

Maeda, J. (1999). *Design by Numbers*. MIT Press, Cambridge.

McCullough, M. (1997). *Abstracting Craft : The Practiced Digital Hand*. The MIT Press, Cambridge.

Mitchell, W. J. (1987). *The art of computer graphics programming: a structured introduction for architects and designers*. Van Nostrand Reinhold, New York

Streich, B. (1992). Should We Integrate Programming Knowledge into the Architect's CAAD-Education? In *CAAD Instruction : The New Teaching of an Architect* eCAADe Conference

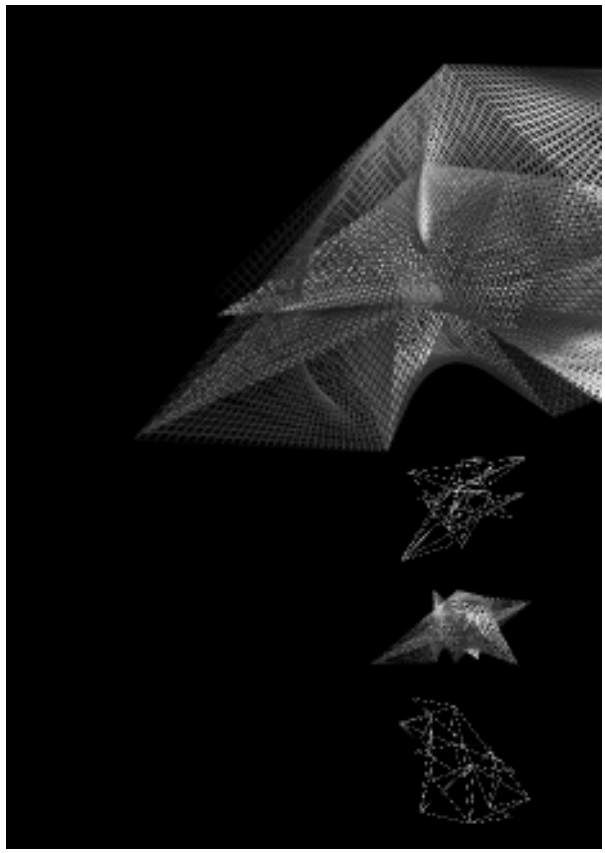

Figure 8. Too much hybrid vigour. The designer can experiment with the algorithm and seek formal responses to given architectural problems. The debate centres on whether they have encapsulated a 'design process', or merely produced an effective 'tool' to aid compositional strategies.

Proceedings, Barcelona (Spain). 399-406.

Weizenbaum, J. (c1976). *Computer power and human reason: from judgment to calculation.* W. H. Freeman, San Francisco.

Winograd, T. (c1986). *Understanding computers and cognition: a new foundation for design.* Ablex Pub. Corp., Norwood, N.J.

Winograd, T. (1996). *Bringing design to software.* ACM Press, New York, N.Y.

Notes

1 The acronym CAD is often misinterpreted as 'Computer-aided design' where the 'D' stands more accurately for drafting, not design (representation, not synthesis).

2 The design process is presumed not to be a prescriptive situation but something unique to the designer. The semantic difficulty comes from the use of the word 'process', process implies something that can be defined, and automated.