Individual backlist issues of \( \Delta \) are available for purchase at £22.99. To order and subscribe for 2006 see page 136.
Editorial

Despite D’s long-standing association with the digital in design, this is the first issue with any reference to programming or software in the title. This says much about architecture’s previous relationship with the computer. At the end of the 1990s when I first started editing D, I was often confounded by the gap between design process and text. Why was everyone generating amorphous Maya-inspired blobs, but talking about Derrida?

Programming Cultures hails the way for a new generation of work that comes out of a pure, unadulterated passion for software. For a young designer like CEB Reas, who has been playing with script since childhood, it is an entirely natural creative impulse. For experienced designers like Malcolm McCullough and Greg Lynn, it has come out of decades of pioneering work on computer-aided design. McCullough, one of the first architecture managers for Autodesk, exhorts us that ‘all you need is the will to improvise’; Lynn enthuses ‘I love the moment when I discover some new potential in software.’ This is all constructive play which, as described by Ingeborg M Rooker in her interview with Lynn, adds up to ‘exhaustive exploration’. It is a not insignificant shift, which moves architects away from being mere consumers of software towards becoming knowledgeable adapters, crafters and, ultimately, producers. So much more than a development of architects’ technical skill bases, it is set to have a huge impact on the culture of architecture. Could the deftness with scripting that educators and designers like Mike Silver, the guest-editor of this issue, is encouraging in his students at New York’s Pratt Institute of Architecture become commonplace? Could scripting become the new drawing? The potential aesthetic impact of this way of working is anticipated by Pia Ednie-Brown’s discussion of the new compositional principles spearheaded in the work of Alisa Andrasek of biofishing. For, as Ednie-Brown points out: ‘Working with computational algorithms as primary generative material offers a different bent to, for example, the mathematical ratios of the Renaissance or the flow diagrams of Modernism.’ At such a nascent stage, the ultimate cultural repercussions of a new programming era are only to be guessed at. The implications are that we could be at the brink of an entirely new period of culture and knowledge; if so, could Stephen Wolfram be set to become the next Isaac Newton, and Gehry Technologies’ software be about to eclipse the pattern-books of Palladio?

Helen Castle

Towards a Programming Culture in the Design Arts

In his introduction to this issue of D, guest-editor Mike Silver celebrates ‘the flexible language of commands and logical procedures’ of computers whose creative potential has until now been undervalued in architecture. He explains how the ‘happy accident’ of late 1990s blob architecture is now giving way to a focus on programming and composing new code, which promises ‘to generate new and unprecedented modes of expression’.

Pratt Institute School of Architecture, Carbon-fibre chandelier studio project, autumn 2006
In this studio led by Professor Mike Silver, students’ brief for scripting was applied in a project that developed new software to coordinate the movement of a CNC machine’s rotating bed with the controlled trajectory of its servo-controlled buttons. Here, foam shapes cut on a CNC tumbling foam-cutter were used as moulds for hand-laid carbon-fibre panels.
Chandelier fabrication process using epoxy resin, carbon-fibre and Nomex drapes over a CNC foam-cut mandrel.
You write a few lines of code and suddenly life is better for a hundred million people.

Charles Simonyi, inventor of Microsoft Word

The first architects to use computers were interested primarily in maximising the efficiency of conventional modes of production. Designers working on the World's Fair in 1964 used a primitive calculating machine to build the Unisphere, and around the same time Eero Saarinen engineered the complex reinforced-concrete shells of his TWA terminal using early structural-analysis software. During the 1980s many architectural practices employed the first automated systems for drafting construction documents. As computers became cheaper and more readily available, in the 1990s the buzz surrounding blob architecture embodied design's obsession with digital media at the brink of a new millennium. This trend has now given way to the more urgent problem of process and code. Many of the contributors to this issue have expanded on a programming paradigm originally explored at MIT by the artist and graphic designer John Maeda, whose protégé, CEB Reas is featured in this issue. Not surprisingly, architecture has lagged far behind these pioneers with creativity and experimentation taking a back seat to the priorities of disciplinary continuity, history and function. At MIT, architecture's foray into programming was mostly restricted to the computer replication of known forms and building types exemplified in the work of William Mitchell and Richard Foreman, and the shape grammars of George Szym.

As Greg Lynn has pointed out, these systems were 'merely an extension of a previously delineated and closed set of potential forms whose characteristics can be stated in advance by an ideal mathematician.' Today, programming in architecture has become a much more open process, one that is inspired by the capacity to generate new and unprecedented modes of expression (see Malcolm McMillenough's essay on scripted space). For many architects coding has become the formal and operative focus of building itself.

Universality vs. Difference

In the 1990s, the association between computers and architecture was marked by the proliferation of amorphous, curvilinear forms. But this link is far from necessary or inevitable. It is, in fact, a happy accident precipitated by pre-computational experiments based on 'folding' and the subsequent appropriation of operationally specific modelling software. How, then, do we determine the relationship between computing and architecture? What forms, practices and techniques will seem most relevant if the internal structure of software and the theories that drive its development are changed at the same time? Can we even define computational architecture? The answer to this last question is that we cannot. As Ingeborg Röcker's essay on page 16 shows, computers are universal machines. Unlike classical machines (clocks, steam engines and tin-openers) they can perform a wide variety of tasks without significant changes to their physical design. Through the writing of new programs, very different operations can be executed on a single device.

Where the information theorist defines universality through the production of difference, for architects it holds exactly the opposite meaning. The stripped-down and monolithic style of international Modernism was thought to be universal for all places and times. Messianic 'universal space' was constrained by a homogeneous and infinitely extended grid, very much unlike the complex, convoluted matrix of interconnected microvitricity that constitutes today's digital networks. The universality of a programmable computer is therefore measured by its degrees of freedom. What makes it unique, as a device, is the flexible language of commands and logical procedures that can instantly transform it from one function to another. With existing software, designers are often forced to conduct experimental work using fixed protocols originally developed to solve the visualization problems faced by aircraft designers and Hollywood filmmakers. Even in these disciplines, staff programmers are busy creating new codes for challenges that are beyond the scope of a particular product. Not surprisingly, this has also become true for the early advocates of software appropriation in architecture. Faced with increasingly complex commissions and a growing practice, many designers have turned to the in-house creation of proprietary code. Here, one immediately thinks of Dennis Shelden's redevelopment of CATIA at Gehry Systems, and Greg Lynn's collaboration with Microstation.

Both examples suggest the possibility of freeing computer-aided design (CAD) from any one particular concept or form rooted in any one particular software. If indeed codes can be changed to fit specific needs or developed from scratch, then there can be no such thing as a 'computational architecture'; only possible architectures actualised by new programs. These programs can be both simple, generative codes that
produce complexity (see, for example, the articles by Stephen Wolfram and Karl Chu), or complex programs that help make difficult tasks easy. Programs have become pervasive mainly because their instructions can be applied to so many diverse problems (for example, one piece of software in the stealth bomber actually counteracts the plane's poor aerodynamics and prevents it falling from the sky even when the pilot deliberately initiates a stall). New systems of feedback and control, database-management protocols, genetic design algorithms and a whole universe of scripting systems now drive everything from a simple electric toothbrush to the Internet.

Building Programs and Programs for Building

The ability to craft tools that address both the practical challenges of building design and the human capacity to imagine new forms is a fairly recent development. As specific programming languages become less mysterious and easier to master, 'home-made' software will most likely become a familiar part of design culture. This move by the design community to take an active role in the production of code transcends the limitations of prefabricated software tools.

To a great extent this shift is becoming a necessity. Large, unanticipated gaps in the computer-aided design and construction process can only be bridged by the creation of special software. Many of these tools cannot be anticipated in advance by outside developers since the most imaginative projects begin with ideas that exceed ordinary expectations. In order to connect different production processes, materials and fabrication devices, new tools are required. In fact, as architect Harosh Lahavni has demonstrated in his collaboration with Milgoffbink (see his interview with John Lobell), unexpected functions for standard digital-fabrication equipment can be created simply by writing code.

In academia, students and teachers have been increasingly drawn to the possibilities of proprietary software development. For many architects programming has become the new drawing. In the carbon-fibre chandelier project designed for the lobby of Pratt Institute's new School of Architecture, complex EPS moulds for hand-laid composites were produced using a large-scale computer-numerically controlled (CNC) foam-cutter. In order to precisely construct the necessary 3D shapes, new software was developed to coordinate the movement of the machine's rotating bed with the controlled trajectory of its servo-controlled hotwire. The ability to link disparate material practices (foam-cutting and advanced composite construction) would not have been possible without the new scripts.

With these examples in mind, Programming Cultures explores the power of code by encouraging artists and architects to become more involved in the creation of home-made, task-specific tools. Through individual labour or by collaborating with skilled developers, designers can harness the power of universality. In this way universality supports differences in opposition to the sterile homogeneity commonly associated with rigid protocols and fixed procedures. What computation must now serve (as the work of Ivan Douglas aptly demonstrates) are the founding concepts, intuitions and desires that can only emerge from a varied and creative practice. Rather than working through algorithms imposed on the architect by an external agent or product philosophy, intuition and desire motivate artistic production while encouraging the spread of individual creativity.

As the projects in this book suggest, it is only a question of time before software development becomes an integral part of the building design process. Certainly, this will be an unprecedented moment in the history of architecture.

Thank you to Evan Douglas and Dean Thomas Hannahan at the Pratt Institute School of Architecture for their support of the Universal Machines symposium held during the spring of 2005, which provided the original impetus for this issue. Malcolm McCalough, John Lobell and Yeong Pung Cho were especially instrumental in provoking questions and inspiring discussion. Programming Cultures: Architecture, Art and Science in the Age of Software Development would not have been possible if it were not for the support provided by Helen Castle, the staff of John Wiley & Sons and a generous grant from the Graham Foundation.

Notes
1 Stew Loo, Go Ti, Parsons Books (New York), 2001, p. 2.
3 Where Wolfram defines 'code' as a generative abstraction that emulates pattern formation at a multitude of spatial dimensions within the universe (analytic) for Chu 'codes' are used for the construction of possible worlds (synthetic).