THE BUILDING OF ALGORITHMIC THOUGHT

THE BUILDING OF ALGORITHMIC THOUGHT

THE BUILDING OF ALGORITHMIC THOUGHT

THE BUILDING OF ALGORITHMIC THOUGHT
Computation is redefining the practice of architecture. Architects are developing digital tools that create opportunities in design process, fabrication and construction. Using built architectural projects, this issue of 3 provides insight into emerging design approaches that use computation as a design method. While this issue aims for a balance of ideas, scales of work, and types of practices, it is not a cross-section through all of architecture, but rather a presentation of state-of-the-art uses of computation.

But what do we mean by computation? Most architects now use computers, but usually to simply digitise existing procedures with entities or processes that are preconceived in the mind of the designer. For example, architects use the computer as a virtual drafting board making it easier to edit, copy and increase the precision of drawings. This mode of working has been termed 'computerisation'.

'Computation', on the other hand, allows designers to extend their abilities to deal with highly complex situations. Sean Ahlquist and Achim Menges define computation as 'the processing of information and interactions between elements which constitute a specific environment; it provides a framework for negotiating and influencing the interrelation of datasets of information, with the capacity to generate complex order, form, and structure. In this issue of 3, the term 'computation' means the use of the computer to process information through an understood model which can be expressed as an algorithm. This then allows the exploration of new ideas: computation augments the intellect of the designer and increases capability to solve complex problems.

As illustrated here with new building projects by some of the world's most forward-thinking practices, computation also has the potential to provide inspiration and go beyond the intellect of the designer, like other techniques of architectural design, through the generation of unexpected results. For example, when an architect writes a computer program to solve a design problem, further options can then be explored through modifications to the program – sketching by algorithm. An algorithm is a particular set of instructions, and for these instructions to be understood by the computer they must be written in a language the computer can understand, a code.

Architects designing many of the projects featured in this 3 have used scripting languages such as RhinoScript or Visual Basic for Applications (VBA) in Bentley Systems’ MicroStation™ to write programs to customise their design environments in their existing architectural design software. While other computer languages are used, and indeed have been used since architects have first used computers (see Daniel Cardoso Llach's 'Algorithmic Tectonics' on pp 16–21), the power and availability of these scripting languages, and now also Robert McNeel & Associates' Grasshopper® visual programming language, have propelled the increased usage of computation in practice. Algorithmic thinking means taking on an interpretive role to understand the results of the generating code, knowing how to modify the code to explore new options, and speculating on further design potentials. We are moving from an era where architects use software to one where they create software.
A second approach to integrating computational design expertise into the design process is to have a consultancy of computational designers. These practices tend to be technical and specialised, either in engineering or software development. They can be hired by architecture firms who can therefore take advantage of computational techniques without actually having to internalise the expertise. Examples include Buro Happold SMART Solutions (pp 62–5), Knippers Helbig Advanced Engineering (pp 74–81) and Gehry Technologies (pp 36–41).

Computation can also be fully integrated into the practice and the actual design process. In these firms there is no separation between design intent and computational technique, and computation is used in a natural or unconscious way. Examples are MOS (pp 96–103) and Facit Homes (pp 88–91).

The fourth approach relates to an emerging model of hybrid software engineers/architects. Empowered by advances in scripting interfaces and knowledge of computer programming, these designers are actively creating their own design software. While these small offices have not yet built many projects, they are, for their size, very relevant to architectural practice as a whole. Examples included here are: David Rutten (Grasshopper®/Galapagos – pp 132–5), Daniel Piker (Kangaroo – pp 136–7), Andrew O Payne and Jason Kelly Johnson (Firefly – pp 144–7), Giulio Piacentino (WeaverBird – pp 140–1), Thomas Grabner and Ursula Frick of [uto] (GECO™ – pp 142–3), Arthur van der Harten (Pachyderm Acoustical Simulation – pp 138–9).

But do all architecture firms need to have an internal specialist group to develop computational approaches? Perhaps not, as networks, both digital and social, now allow for the access to knowledge generated by others. Events such as Smartgeometry, and online forums such as the Grasshopper community, allow designers to gain knowledge of digital tools and codes, workflows and algorithms that they can then use or adapt to their own design. Computation and the use of the computer facilitate the sharing of codes, tools and ideas. This accumulation of ideas is one of the ways in which we can refer to a building of algorithmic thought.

Computational designers construct 3-D models and create design tools, but their expertise goes beyond these tasks. They generate and explore architectural spaces and concepts through the writing and modifying of algorithms that relate to element placement, element configuration, and the relationships between elements. In this important way, their role also goes beyond simply making digital tools for designers. Critically, the making of these custom tools takes place within the design process, and becomes integral to the design itself. These two points are key in understanding the possibilities of computational design and the role of the designer in architectural practice. For computational techniques to be useful, they must be flexible – they must adapt to the constantly changing parameters of architectural design. The design environment, of which the architect is now part author, must be flexible and have the ability to accommodate change (see Brady Peters and Daniel Davis’s ‘Design Ecosystems’ on pp 124–31).

The structure of architectural firms is changing in response to the work of computational designers. Looking at the offices represented in this issue of 3, there are four ways in which these designers are organised: the internal specialist group, the external specialist consultancy, the computationally aware and integrated practice, and the lone software developer/designer.

The most common approach is to have computational designers working in internal specialist groups largely separate from the design teams. These groups act as internal consultancies and designers integrate with the design process to varying degrees depending on the needs of the project. They exist in practices such as Foster + Partners (pp 22–35), Herzog & de Meuron (pp 56–61), Grimshaw (pp 104–9), Aedas|R+D (pp 42–7), UNStudio (pp 82–7) and Skidmore, Owings & Merrill (SOM) (pp 48–55).
When architects have a sufficient understanding of algorithmic concepts, when we no longer need to discuss the digital as something different, then computation can become a true method of design for architecture.
Computing Performance and Simulation

Architects are increasingly experimenting with computation to simulate building performance, to incorporate performance analysis and knowledge about material, tectonics and parameters of production machinery in their design drawings. These new custom digital tools allow for performance feedback at various stages of an architectural project, creating new design opportunities. Using these tools, structural, material or environmental performance can become a fundamental parameter in the creation of architectural form (see Sawako Kaijima, Roland Bouffanais, Karen Willcox and Suresh Naidu’s ‘Computational Fluid Dynamics for Architectural Design’ on pp 118–23, and Clemens Preisinger’s ‘Linking Structure and Parametric Geometry’ on pp 110–13). The development of computational simulation tools can create more responsive designs, allowing architects to explore new design options and to analyse architectural decisions during the design process.

Stan Allen suggests that meaning in architecture is constructed as an encounter between architecture and the public. If so, then it does not matter what design tool or technique was used in the design of a building; however, the choice of tools does have an impact on the design. Throughout history, the work of an architect has been linked to the use of drawing as a design tool. Like drawing, architects working with computers and with computation still work through a medium of representation. However, with its increasing simulation capabilities, the computer lets architects predict, model and simulate the encounter between architecture and the public using more accurate and sophisticated methods. In this way, computation makes possible not only the simulation and communication of the constructional aspects of a building, but also the experience and the creation of meaning.
In architectural practice, computation not only works, but has become necessary, to build the largest projects in the world. Given the complexities of form and the compressed timescales of construction today, groups such as Foster + Partners' Specialist Modelling Group (SMG) have become essential aspects in the construction of many projects. As Mouzhan Majidi has said: `This hasn't simply transformed what we can design – it's had a huge impact on how we build.' One example of the impact of computational design is in component design. Unlike in Modernism, where the design effort often went into the perfection of a single detail, the computational approach currently tends to be the development of parametric families of components and in the requisite control of data. Here, what is relevant is the relationship between the parts, and the management of this change in response to local performance requirements. As new design tools are developed to link the virtual design environment with the physical environment, architectural designers will increasingly have the capacity to explore building systems and building environments (see Andrew O Payne and Jason Kelly Johnson, pp 144–7) This could lead to a future where an architect’s digital model could continue to be relevant during the occupation of the building, where feedback between users, building and environment is updated in the digital model and reflected in changes in the building and its performance.

Computational design linked to computationally driven manufacturing requires a new interpretation of the design and construction process (see Jan Knippers's `From Model Thinking to Process Design' on pp 74–81). This invention of new techniques and technologies has, and will continue to cause shifts in our discipline's definition and boundaries. Jan Knippers notes that this is the point for significant innovation. Similarly, Dennis Shelden writes that these computational tools and techniques will even more significantly affect the processes of design and delivery, the definition of the discipline of architecture, and the connection of the work to us and to society (see his 'Networked Space' on pp 36–41).
COMPUTATION AS AN INTEGRATED ART FORM

Computational designers are more than just creators of complex 3-D models or the developers of digital tools – they distil the underlying logic of architecture and create new environments in which to explore designs and simulate performance, both physical and experiential. Their roles and skills are currently evolving with technology and the needs of the particular project and practice. While there are many ways in which they integrate with practice – as lone guns or integrated designers – it is clear that computation enables new ways of thinking. Through computation, the digital architectural design environment has both the ability to construct complex models of buildings and give performance feedback on these models. In a similar way that the pen or pencil can be used to either draw building details or create conceptual sketches for buildings, computational tools can be used to increase efficiency and allow for better communication, as well as for conceptual sketching of algorithmic concepts.

When computation is integrated as an intuitive and natural way to design, then perhaps collections of essays like this will no longer be necessary. However, we are not yet at this point; these concepts must be tested in practice through designing and building. The results must be communicated and reflected upon. Architecture is currently experiencing a shift from the drawing to the algorithm as the method of capturing and communicating designs. This computational way of working augments the designer’s intellect and allows us to capture not only the complexity of how to build a project, but also the multitude of parameters that are instrumental in a building’s formation. When architects have a sufficient understanding of algorithmic concepts, when we no longer need to discuss the digital as something different, then computation can become a true method of design for architecture.

Hugh Whitehead, the former head of the Foster + Partners SMG, observes that ‘at present scripters tend to be of the “lone gun” mentality and are justifiably proud of their firepower, usually developed through many late nights of obsessive concentration. There is the danger that if the celebration of skills is allowed to obscure and divert from the real design objectives, then scripting degenerates to become an isolated craft rather than developing into an integrated art form.’ This issue of promotes the idea of computation in architecture as an integrated art form.

Notes
12. Mark Burry, op cit, p 252.