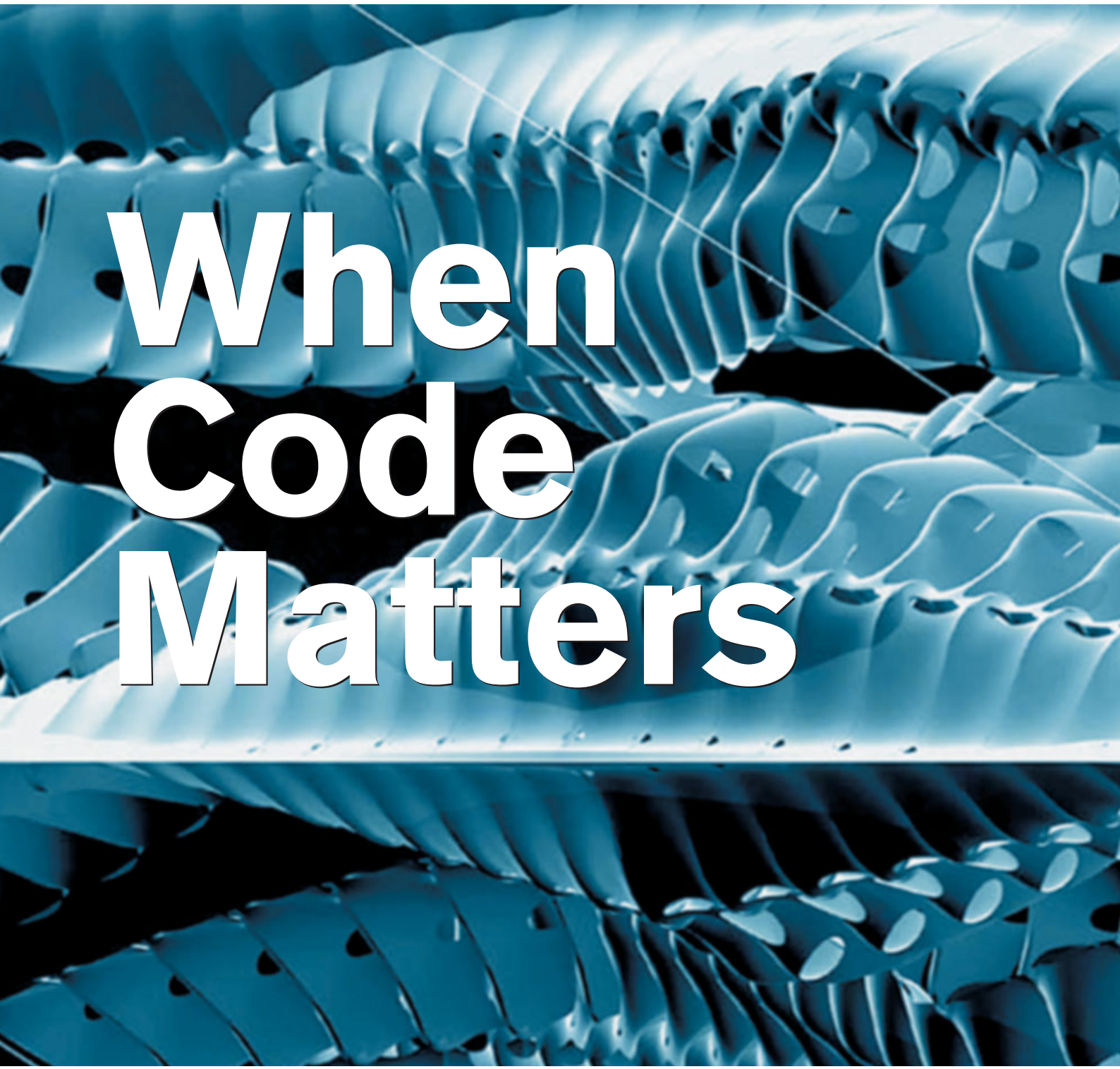
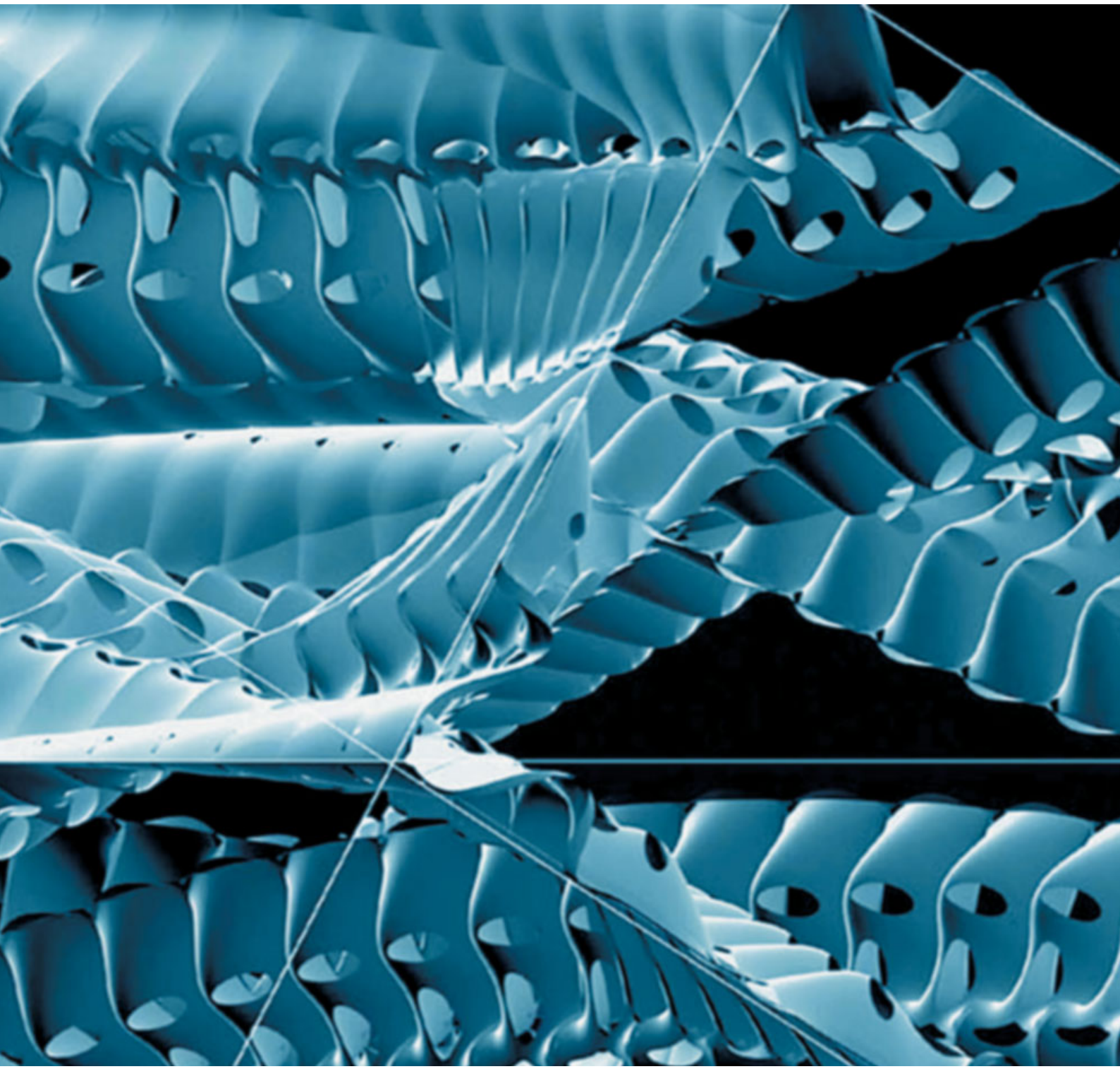


When Code Matters

‘Architecture has been bound and shaped by changing code and constraints throughout its history.’ Ingeborg M Rocker traces, in turn, first the development of calculus into computation and, then, the introduction of computers into architecture. In so doing, she asks what will be the potential effects of computation on the recoding of architecture.



Brandon Williams/Studio Rocker, Expression of Code, 2004
When code matters previously unseen structures begin to emerge. The initiated code may be expressed in different variations ranging from straight walls to twisting columns. At this point performance-based generative systems could, and should, have been applied.

'When God calculates and exercises his thought, the world is created.'

Leibniz,1677

Today, when architects calculate and exercise their thoughts, everything turns into algorithms! Computation,[1] the writing and rewriting of code through simple rules, plays an ever-increasing role in architecture.

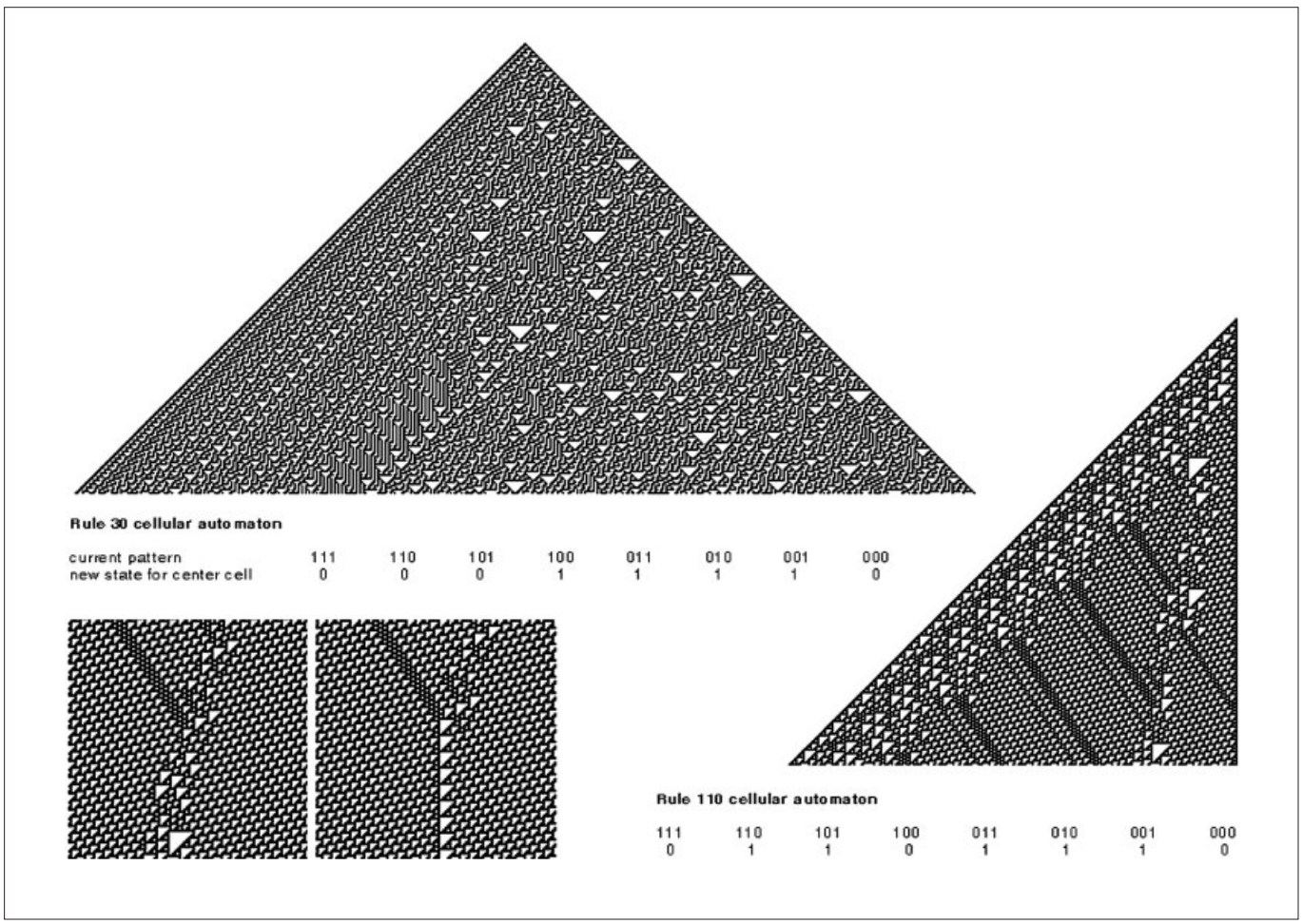
This article explores the role of computation in the discourse and praxis of architecture, and addresses the central question of when code matters: when it gains importance in architecture, and when it, literally, materialises. It looks at historical computational models and concepts – research independent of traditional mathematics and the computer – and hopes to contribute to a critical assessment of architecture based on code.

From Traditional Mathematics to Computation

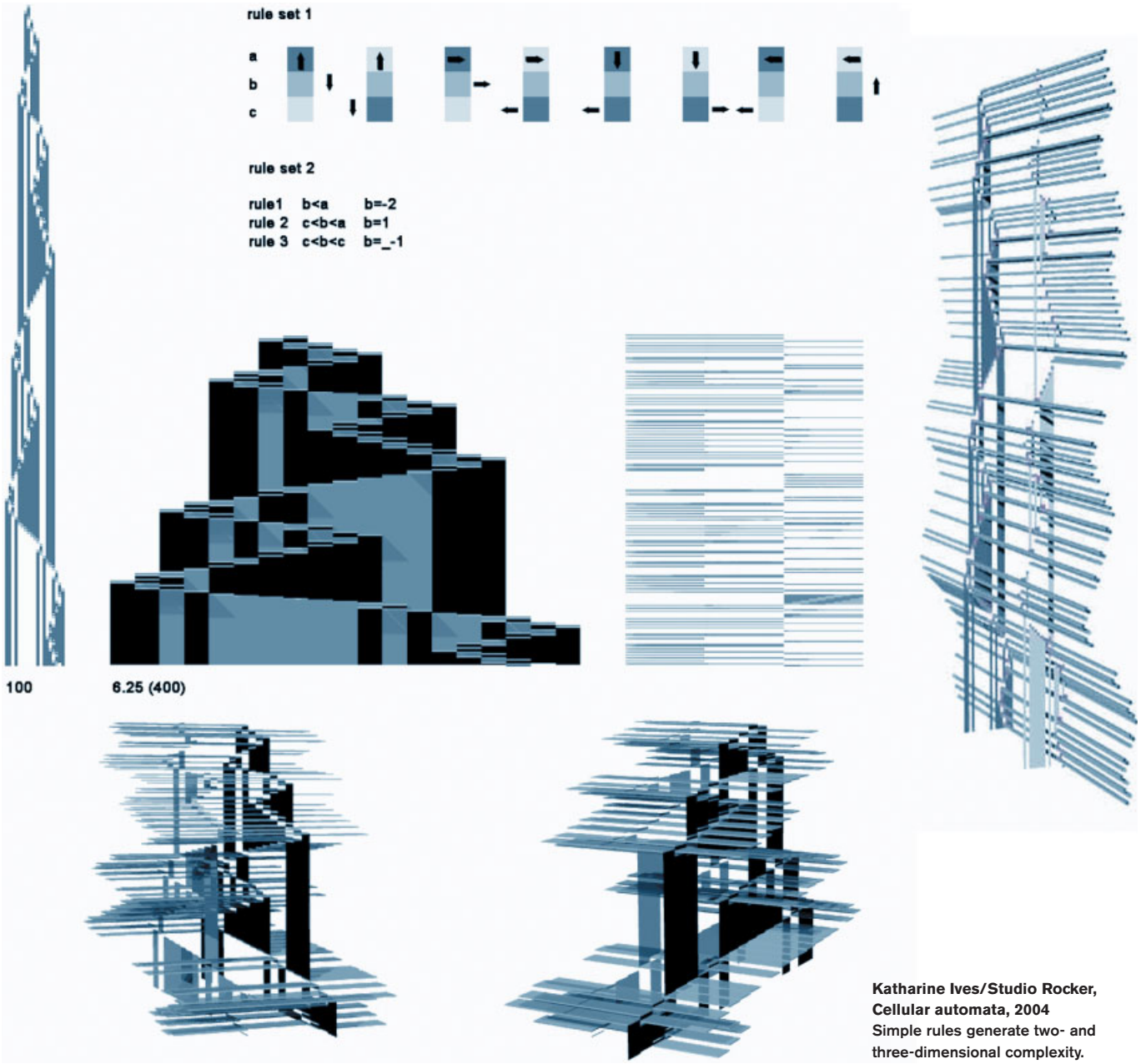
With the introduction of calculus[2] in the late 17th century, abstract rules became increasingly successful in describing and explaining natural phenomena. An abstract mathematical framework then developed from which scientific conclusions could be drawn without direct reference to physical reality. Eventually, physical phenomena became reproducible. Calculus could be applied to a broad range of scientific problems as long they were describable using mathematical equations. Nevertheless, many problems – in particular those of complexity – remained unaddressed.

In the early 1980s, the mathematician Stephen Wolfram began to explore problems of complexity in computer experiments based on cellular automata,[3] and 20 years on his *A New Kind of Science*[4] hoped to initiate a further transformation of the sciences – comparable to that which resulted from the introduction of calculus – this time based on computation.

Wolfram criticised the traditional sciences for relying mainly on mathematical formalism and predictions that often proved incapable of anticipating the complex behaviour of a system. His research suggested that computation, rather than



Stephen Wolfram, Cellular automata, 2002
Here, simple rules generate complexity. Images from Stephen Wolfram, *A New Kind of Science*, Wolfram Media (Champaign, IL), 2002.



Katharine Ives/Studio Rocker, Cellular automata, 2004
Simple rules generate two- and three-dimensional complexity.

mathematics, was the proper means of realising[5] complex natural and artificial processes. Counter to the prevailing belief that complexity is either the product of complex processes or complex formal arrangements, Wolfram suggested that simple rules can produce great complexities. In computation – rather than traditional mathematics – simple rules were sufficient to generate, approximate and explain complex phenomena:

'There are many systems in nature that show highly complex behavior. But there are also many systems, which rather show simple behavior – most often either complete uniformity, or repetition. ... Programs are very much the same: some show highly complex behavior, while others show rather only simple behavior. Traditional intuition might have made one assume that there must be a direct correspondence between the complexity of observed behavior and the complexity of underlying rules. But one of [my] central discoveries ... is that in fact there is not. For even programs with some of the very simplest possible rules yield highly complex behavior.'[6]

Using computer experiments, Wolfram not only explored complex phenomena, but also their self-organisation over time. Such phenomena and their behaviours could not be anticipated prior to their computation. Only by successive, step-by-step computations is the system of complexities realised and thus becomes realisable. Consequently, only actual computational experiments – rather than a priori determined models of classical mathematics – are sufficient to capture both complex phenomena and their self-organisation.

Wolfram calls his computational methods 'a new kind of science', suggesting a dramatic shift in the sciences away from the mathematical to the computational. Computation will, as Wolfram expects, change our intuition regarding simplicity and complexity.

Universal Machines

Wolfram's research in the field of computation builds on earlier models such as Alan Turing's Machine (1936) and Aristid Lindenmayer's L-systems (1968). The main features of such models are explained below, as they are increasingly gaining relevance in contemporary discussions on algorithmic architecture.

The Turing Dimension[7]

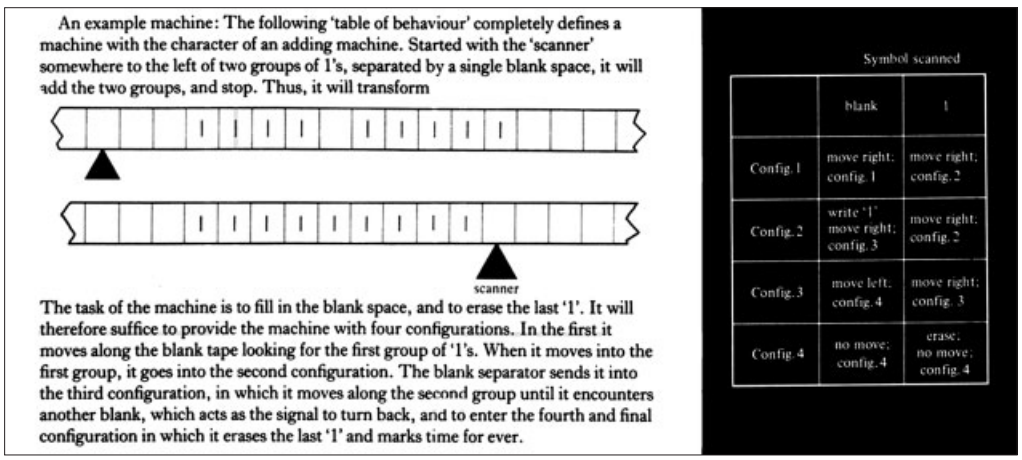
In 1936, the model for the Turing Machine, developed by English mathematician Alan Turing, laid the theoretical foundation for computing.[8] Turing did not envision his machine as a practical computing technology, but rather as a thought experiment that could provide – independent of the formalism of traditional mathematics – a precise definition of a mechanical procedure – an algorithm.[9] In principle, the machine could be physically assembled out of a few components: a table of contents that held the rules for its operations, a reading and writing head that operated along the lines of those rules, writing 0s or 1s – the code – on a presumably infinitely long tape. The moving head followed three modes of operation: writing, scanning, erasing.

Though basic, this design was sufficient for Turing to prove that a certain set of mathematical problems were fundamentally noncomputable. Nevertheless, his machine, based on a finite set of rules and symbols, was able to compute every problem outside of this set. Turing consequently considered his machine universal in the sense that it could simulate all other (calculable) machines. It had the unprecedented ability to emulate divergent and multivalent processes.

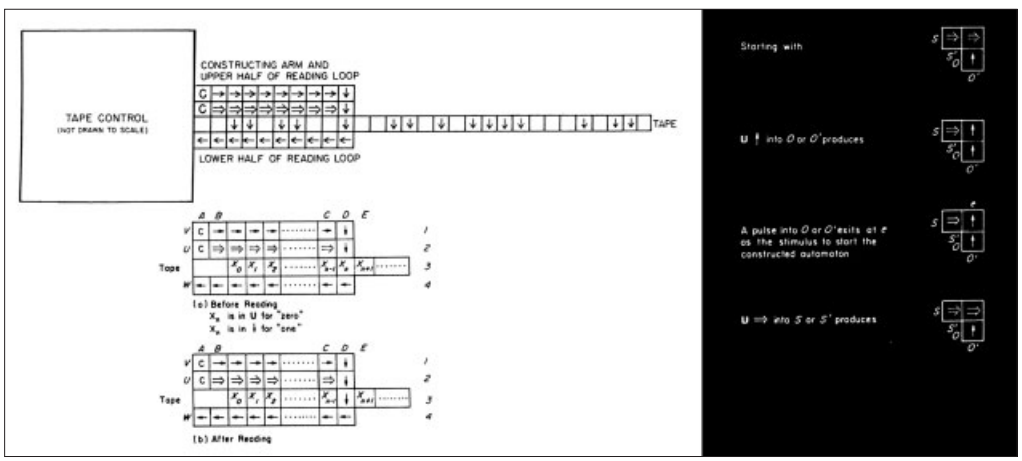
Inspired partly by the Turing Machine model, in the 1940s John von Neumann, a member of the Manhattan Project Team, began work on self-replicating systems at the Los Alamos National Laboratory.[10]

'Von Neumann's initial design was founded upon the notion of one robot building another robot. This design is known as the kinematic model. As he developed this design, von Neumann came to realise the great difficulty of building a self-replicating robot, and of the great cost in providing the robot with a "sea of parts" from which to build its replicant.'[11]

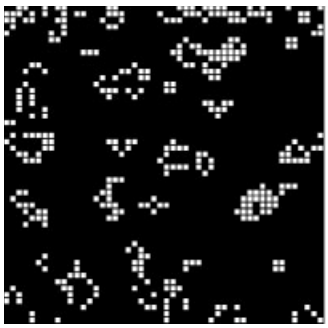
At the suggestion of his colleague Stanislaw Ulam, who at the time was studying the growth of crystals using simple two-dimensional lattice networks as his abstract models, von



Alan Turing, Turing Machine, 1936
The Turing Machine consists of a head reading and writing on a tape according to a set of rules. The operation of the machine depends on the rules provided – and thus changes as the rules change. It is for this reason that the machine can emulate the operation of various machines. Images from Andrew Hodges, *Alan Turing: The Enigma*, Simon and Schuster (New York), 1983, pp 98–9.



John von Neumann, Universal Copier and Constructor, 1940s
Von Neumann's hypothetical machine for self-replication proved that theoretically a cellular automaton consisting of orthogonal cells could make endless copies of itself. Images from John von Neumann, *Essays on Cellular Automata*, ed Arthur W Burks, University of Illinois Press (Urbana, IL), 1970, pp 37, 41.



John Conway, Game of Life, 1970
The image shows Kevin Lindsey's implementation of Conway's Game of Life using JavaScript and SVG. The rules (algorithms) of the game are very simple: if a black cell has two or three black neighbours, it stays black; if a white cell has three black neighbours, it becomes black. In all other cases, the cell stays or becomes white.

Neumann developed the first two-dimensional self-replicating automaton, called the 'Universal Copier and Constructor' (UCC), which at once seemed to reference and at the same time extend Turing's Machine. Here, the process of self-replication was described through simple rules: 'The result of the construction is a copy of the universal constructor together with an input tape which contains its own description, which then can go on to construct a copy of itself, together with a copy of its own description, and so on indefinitely.'[12]

In the open system of the UCC, the machines operated upon one another, constantly modifying each other's configurations (code) and operations (rules): 'The machines were made sustainable by modifying themselves within the inter-textual context of other Universal Turing Machines.'[13]

The system's most important function was the self-replication of its process, which resulted in a successive iteration of each system-cell's state, manifesting itself in evolving and dissolving patterns. Patterns became the visual indicator of an otherwise invisible algorithmic operation. The interactions of the different machines thus opened up a new dimension, the Turing Dimension. This differs from the spatial dimensions commonly used in architecture as it is an operational dimension, where one programmatic dimension is linked with another. Small local changes of the Turing Dimension may resonate in global changes to the entire system.

In 1970, another type of cellular automaton was popularised as the Game of Life through an article in *Scientific American*.[14] The two-state, two-dimensional cellular automaton developed by the British mathematician John Conway operated according to carefully chosen rules.[15] It consisted of two-dimensional fields of cells, with the state of each system-cell determined through the state of its neighbours. All cells are directly or indirectly related to each other, rendering visible, via changes in colour, the process of computation. The player-less game was solely determined by its initial state (code) and rules (algorithms). Similar to the Universal Turing Machine, the game computed anything that was computable algorithmically. Despite its simplicity, the system achieved an impressive diversity of behaviour, fluctuating between apparent randomness and order. The changing patterns directly reflected how the machine's operation writes and rewrites its code over time. Thus with Conway's Game of Life, a new field of research on cellular automata was born.[16]

L-systems

Cellular automata were tested across the sciences. In 1968, the theoretical biologist and botanist Aristid Lindenmayer devised – based on Chomsky's grammars – L-systems for modelling the growth of plants.[17] L-systems consist of four elements: a starting point, a set of rules or syntax, constants and variables. Diversity can be achieved by specifying varying starting points and different growth times.[18] L-systems grow by writing and rewriting the code, and expression of the code depends on the graphical command selected. They are still used to model plant systems, and to explore the capabilities of computation.

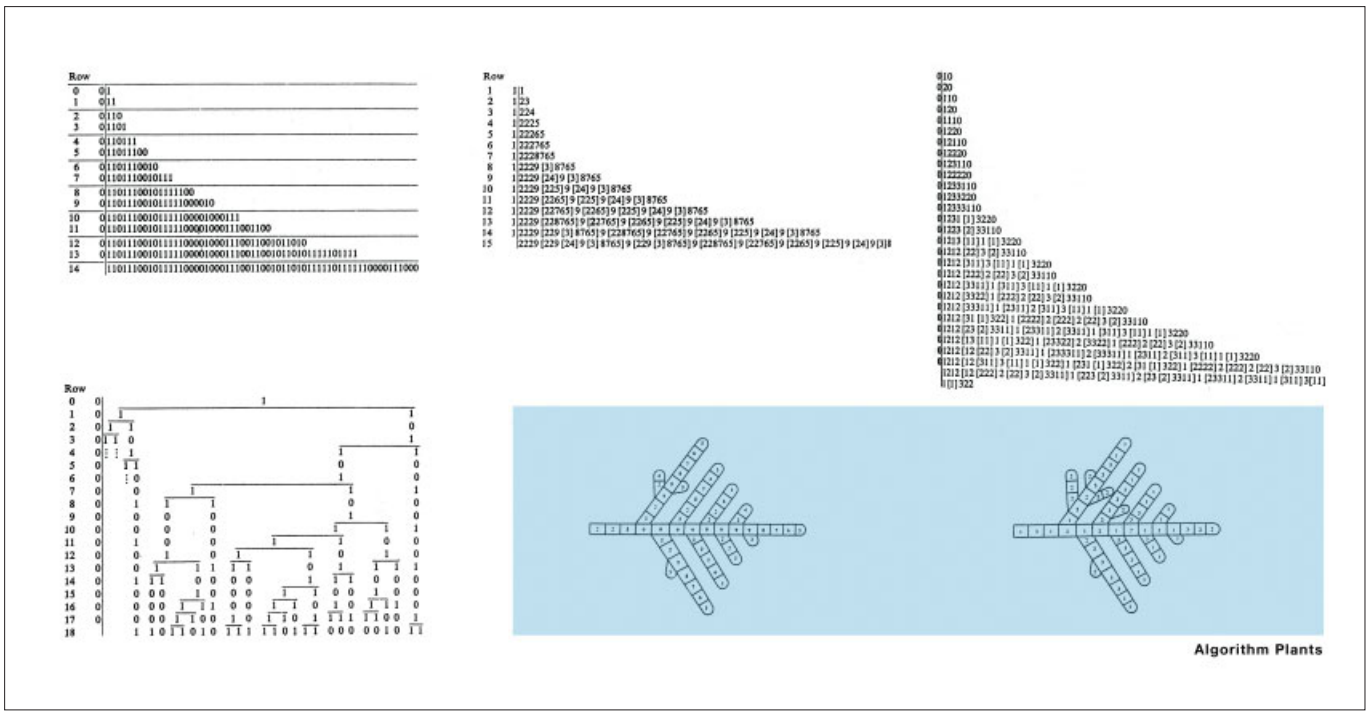
Regardless of which of the models of computation described above is applied, anything that surfaces on the screen is visualised and materialised by one and the same: the digital medium, the alteration of 0s and 1s.

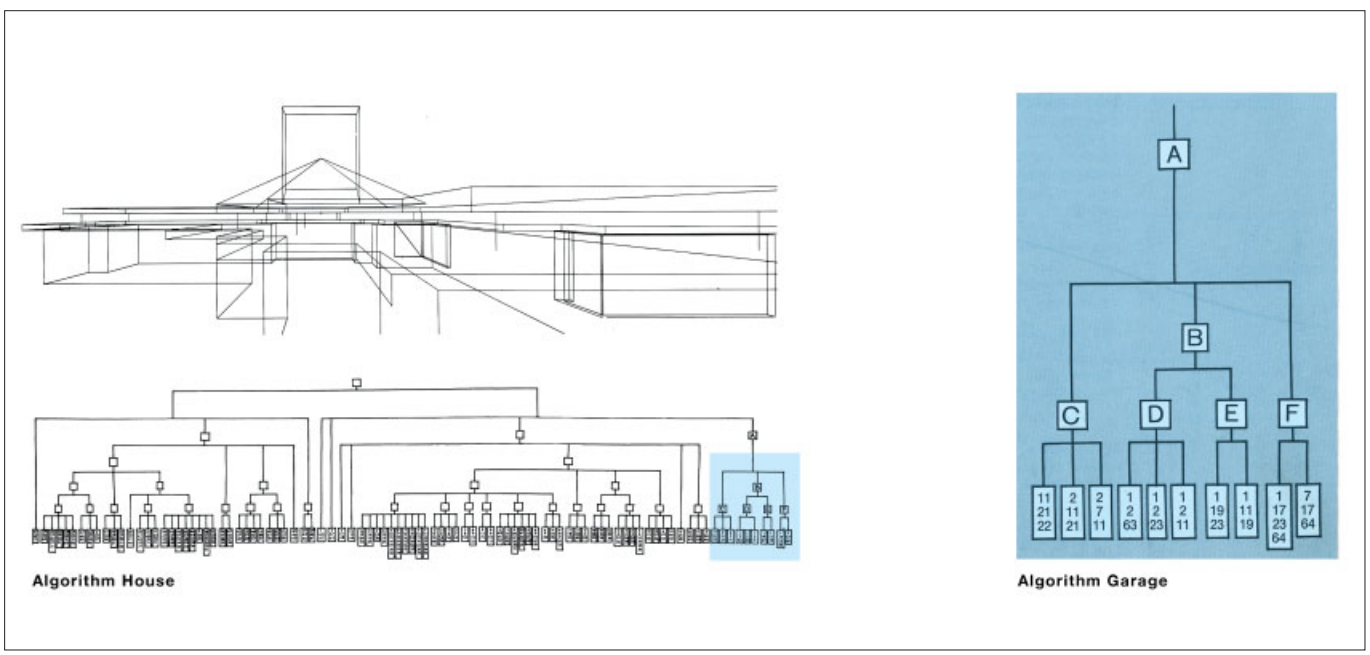
Code in Architecture

Turning the focus of this discussion on computation to architecture, and to the question of when code matters in architecture, it is necessary to look at the role of codes in the past and how they have changed since the introduction of the digital medium in architecture. The use of code has a long tradition dating back to the Latin term *codex* that refers to documents formed originally from wooden tablets. The Romans used such documents to distribute a system of principles and rules throughout their empire. In architecture, systems and rules have dominated, and still dominate, all stages of architectural production, in the form of drawing and design conventions. Throughout its history, architecture has been bound and shaped by changing codes and constraints, and neither architecture nor its media (from pencil drawings to physical models, computer renderings to the built projects) will ever be free of codes. Architecture is, and always has been, coded.

The arrival of computers extended understanding of code in architecture. Code was now understood as a set of instructions written in a programming language. It stood for 'source code', a series of statements written in some human-readable computer programming language, or 'machine code', instruction patterns of bits (0s and 1s) corresponding to different machine commands.

Writing code, writing a set of instructions, with the aim of generating architecture, forced the architect of the past to formalise both the design process and the design. In contrast to many scientific procedures, which could quite easily be expressed through mathematical equations, the often intuitive and even unconscious use of design rules could not. Perhaps it was for this reason that previously many algorithms were invented and implemented in planning,





Aristid Lindenmayer, L-system generating algorithmic plants, 1968
Top: Lindenmayer studied mathematical models of cellular interactions in development and their simple and branching elements. Images from: Aristid Lindenmayer, 'Mathematical models of cellular interactions in development: I. Filaments with one-sided inputs', *Journal for Theoretical Biology*, Vol 18, 1968, figs 1 and 2 (p 286), figs 4 and 5 (p 310), and figs 6 and 7 (p 312).

Allen Bernholtz and Edward Bierstone, Algorithmic de- and recomposition of design problem, 1967
Above: Bernholtz, of Harvard University, and Bierstone, of the University of Toronto, adopted Christopher Alexander's and Marvin L Manheimer's computer program HIDECS3 (1963) for hierarchical de- and recomposition to first decompose a complex design problem into its sub-problems and then to recompose a solution. The marked area addresses the design of the garage for which 31 out of 72 possible 'misfit factors' were held characteristic and needed to be excluded in order to arrive at the most appropriate design. Images from Martin Krampen and Peter Seitz (eds), *Design and Planning 2: Computers in Design and Communication*, Hastings House, (New York), 1967, pp 47, 51.

rather than in architectural design. An exception to this was Christopher Alexander's HIDECS3 (1963), a program that decomposed algorithmically complex design tasks into simple sub-problems in order to then recompose them to the 'fittest design'. However, in the end Alexander's algorithms served only to force architectural design into an overly structured rational corset.

The Computational Turn

Regardless of the absurdity of the planning mania of the past, architects are now once again devoted to code, this time in the form of scripting algorithms. While previously architects were obsessed with the reduction of complexity through algorithms, today they are invested in exploring complexities based on the generative power of algorithms and computation.

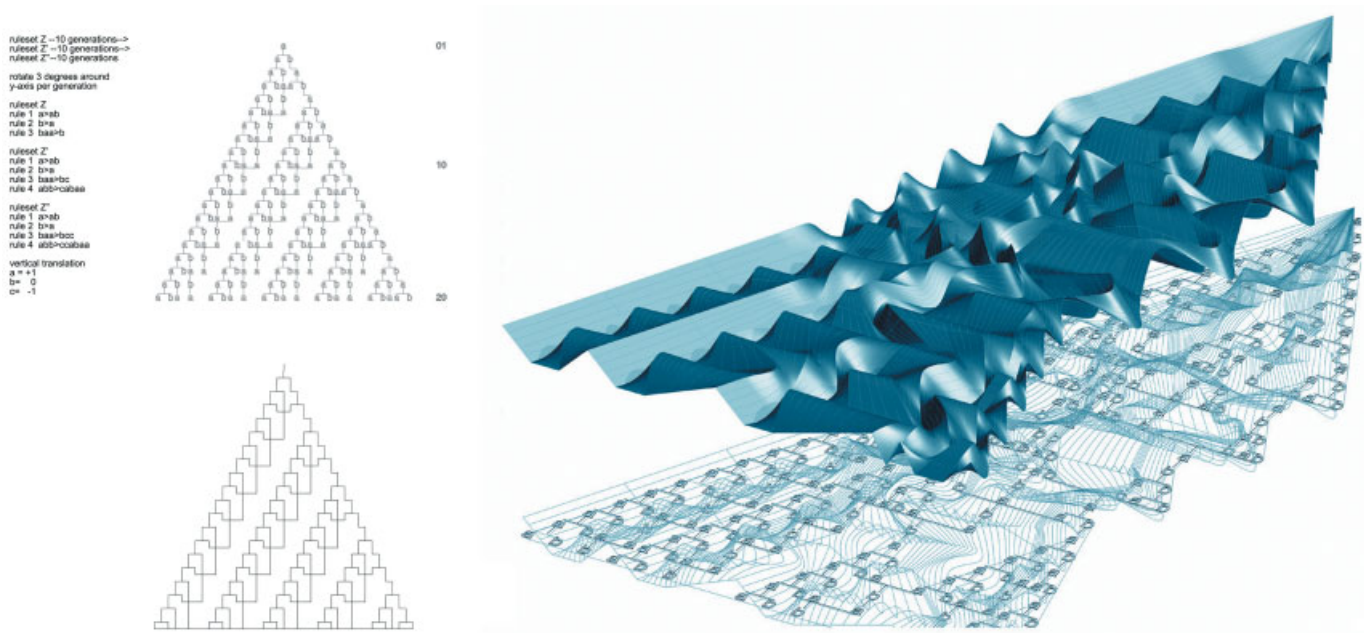
We are now witnessing a 'computational turn' – a timely turn that countermands the reduction of architectural praxis to the mindless perfection of modelling and rendering techniques.

For many of the architects featured in this issue, the prepackaged design environments with their inscribed limitations – regardless of all the rhetoric associated with them – never really addressed the genuine operations of the digital medium: 'The dominant mode of utilizing computers in architecture today is that of computerization.'[19]

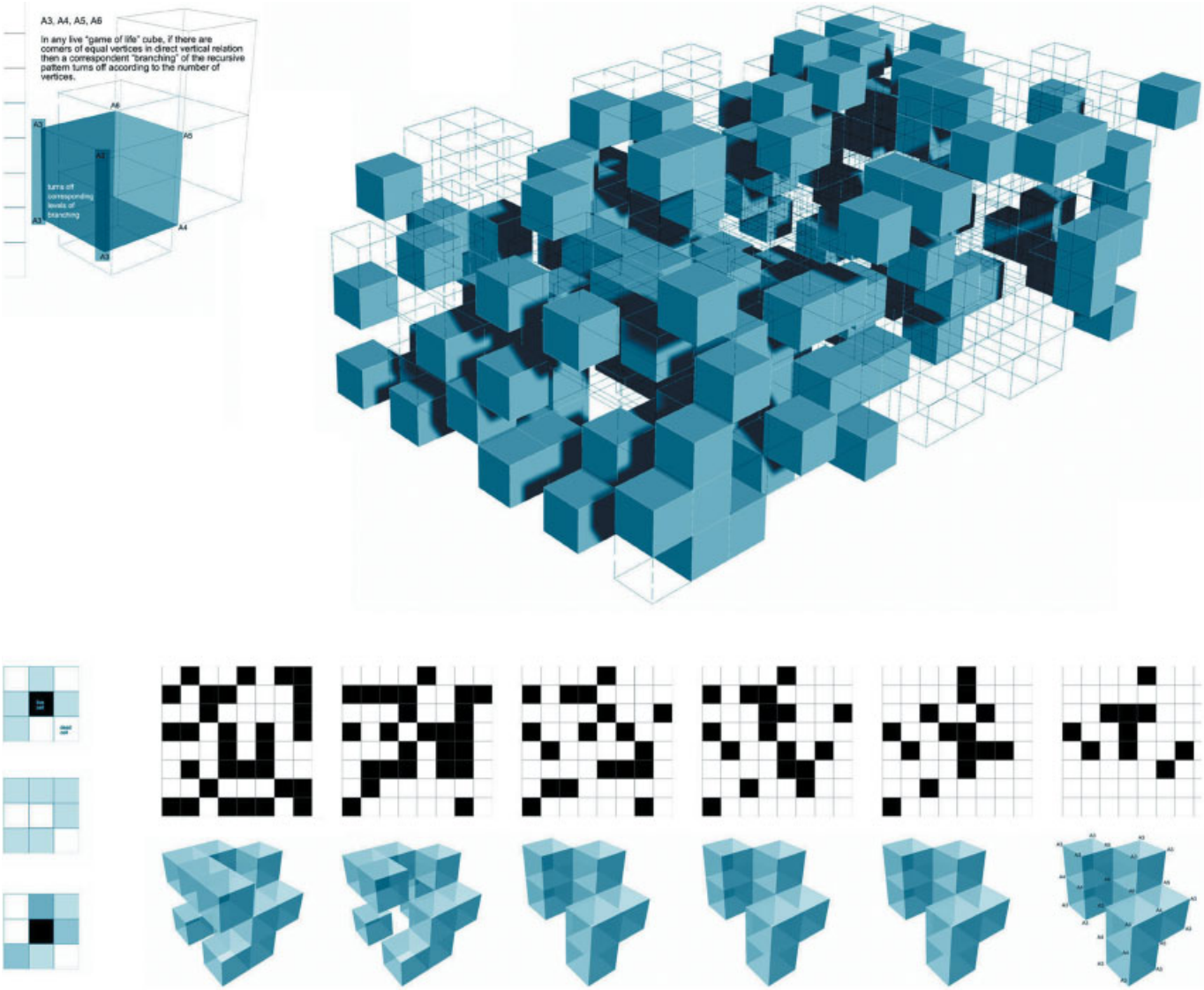
Most architects now use computers and interactive software programs as exploratory tools. All their work is informed by, and thus dependent on the software they are using, which inscribes its logic, perhaps even unnoticed, onto their everyday routines. Such users of software packages have little or no knowledge of the algorithms powering the programs they employ. Most of the interactivity is reduced to a manipulation of displayed forms on the screen, neglecting the underlying mathematical calculations behind them. All of this – even though implemented on computers – has little to do with the logics of computation.

For architects and artists like Karl Chu, Kostas Terzidis, George Liaropoulos-Legendre, Mike Silver and CEB Reas, scripting is the means to develop their own design tools and environments. According to Kostas Terzidis: 'By using scripting languages designers can ... transcend the factory set limitations of current 3-D software. Algorithmic design does not eradicate differences but incorporates both computational complexity and creative use of computers.'[20]

In writing algorithms, design intentions or parameters become encoded. Terzidis and Chu base most of their explorations on simple, clearly defined rules capable of computing a priori indeterminable complexities. As such, alternative modes of architectural production are developed: 'Unlikely computerization and digitization, the extraction of algorithmic processes is an act of high-level abstraction. ... Algorithmic structures represent abstract patterns that are not necessarily associated with experience or perception. ... In



Brandon Williams/Studio Rocker, Recursions, 2004
Recursive procedures that repeat indefinitely are reading and writing code according to preset rules: line by line, generation by generation. Hereby, each generation impacts the next generation and consequently all following ones. Patterns of code appear and disappear.



Brandon Williams/Studio Rocker, 3-D Game of Life, 2004
Cellular automata and the Game of Life became the architect's basis for experimentation. The moment a cell turns active, the project code is realised, and thus becomes realisable.

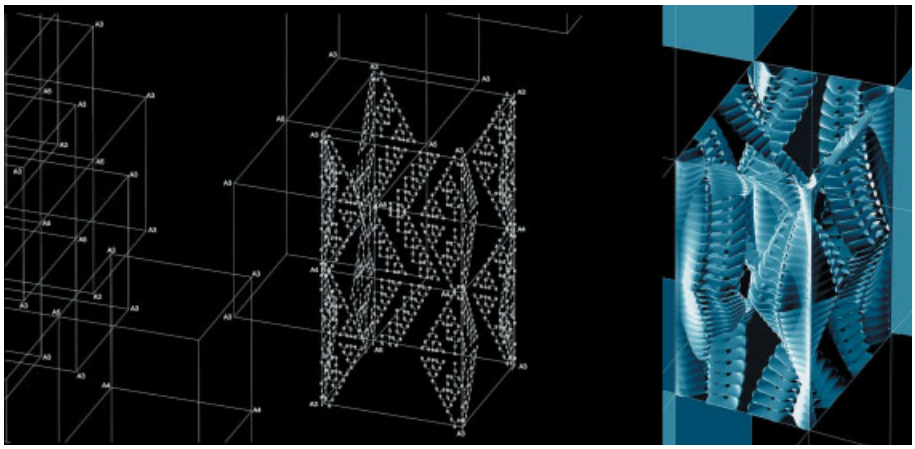
this sense algorithmic processes become a vehicle for exploration that extends beyond the limits of perception.'[21]

The computer is no longer used as a tool for representation, but as a medium to conduct computations. Architecture emerges as a trace of algorithmic operations. Surprisingly enough, algorithms – deterministic in their form and abstract in their operations – challenge both design conventions and, perhaps even more surprisingly, some of our basic intuitions.

For the supporters of algorithmic architecture, a new field of explorations has opened up, one that aims at a better understanding and exploration of computation's genuine processes and their potential for the production of architecture. They are fascinated by how complex architectures emerge from simple rules and models like the Turing Machine and the cellular automaton, in the moment of their computation, as exemplified here by some of the work produced by Studio Rocker[22] in spring 2004.

The promises and limits of such explorations are diverse. The use of genetic algorithms has been traditionally driven by the desire to generate novel forms yet without a deeper consideration of their physical actualisation and performance. Most, if not all, explorations into algorithmic process- and form-generation generally neglect the structural and material integrity of architecture. Nevertheless, some projects do present a significant development within the production and discussion of design, and many have recast the authorial role of the architect, upsetting the guardians of the humanist school.

The projects presented in this issue attempt to counter the neglect referred to above by exploring specific links between



Brandon Williams/Studio Rocker, Expression of code, 2004
Quite different to the Turing Machine, which only uses a one-dimensional tape, Brandon Williams's design is a two-dimensional surface. Modes of transposition determine how the abstract code, consisting of As and Bs, realises and thus becomes realisable as surface and structure. Obviously, the chosen mode of transposing code into its expression is just one of many possibles. Any code's expression is thus always just one of an infinite set of possible realisations. We just have realised the incompleteness of realisation.

computation and material praxis. They therefore represent an important development within the production and discourse of design. The question remains open as to whether the turn to computation will reconfigure architecture to such an extent that a new kind of architecture will emerge.

Coding a Strategy for Alteration

This article presents code as a technical and discursive construct. Code is hereby not considered a normalising restriction of architecture, but rather as a site where a recoding of architecture may occur. The strategy of coding and recoding embraces the free play of symbolic code, as it impacts on architecture's traditional coding systems and their standardised, prototypical material realisation. Computation allows for the emergence of form and space independent of such traditional constraints, and thus allows us to arrive at alternative formal and spatial conceptions, which decode and, at the same time, recode architecture.

Here code matters. Δ

Notes

1 Emphasis in this article is given to computation. The logic of computation, not the literal use of computers, is relevant for the argument.
2 The word 'calculus' originated from the development of mathematics: the early Greeks used pebbles arranged in patterns to learn arithmetic and geometry, and the Latin word for 'pebble' is *calculus*, a diminutive of *calx* (genitive *calcis*) meaning 'limestone'.
3 Stephen Wolfram, *Cellular Automata and Complexity: Collected Papers*, Addison-Wesley (Reading, MA), 1994.
4 Stephen Wolfram, *A New Kind of Science*, Wolfram Media (Champaign, IL), 2002.
5 'Realisation' is in this context used in the double sense of the word, suggesting something that becomes real and graspable, intelligible.
6 Wolfram, *A New Kind of Science*, p 351.
7 A term I owe to Karl Chu. For further reference see Karl Chu, *Turing Dimension*, X Kavya (Los Angeles, CA), 1999.
8 Alan M Turing, 'On computable numbers, with an application to the Entscheidungsproblem 1936–37', in *Proceedings of the London Mathematical Society*, Ser 2, 42, pp 230–65.
9 The word 'algorithm' etymologically derives from the name of the 9th-century Persian mathematician Abu Abdullah Muhammad bin Musa al-Khwarizmi. The word 'algorism' originally referred only to the rules of performing arithmetic using Hindu-Arabic numerals, but evolved via the European-Latin translation of al-Khwarizmi's name into 'algorithm' by the 18th century. The word came to include all definite procedures for solving problems or performing tasks. An algorithm is a finite set of well-defined instructions for accomplishing some task. A computer program is essentially an algorithm that determines and organises the computer's sequence of operations. For any computational process, the algorithm must be rigorously defined through a series of precise steps that determine the order of computation.
10 The Los Alamos National Laboratory, officially known only as Project Y, was launched as a secret centralised facility in New Mexico in 1943 solely to design and build an atomic bomb. Scientists from all over the world and from different fields of study came together to make Los Alamos one of the US's premier scientific research facilities.
11 Source: http://en.wikipedia.org.
12 Christopher G Langton, 'Cellular automata', in *Proceedings of an Interdisciplinary Workshop*, Los Alamos, New Mexico, US, 7–11 March 1983. See also John von Neumann, *Theory of Self-Reproducing Automata*, edited and completed by Arthur W Burks, University of Illinois Press (Urbana, IL), 1966.
13 See also Karl Chu, op cit.
14 Martin Gardner, 'Mathematical Games: The fantastic combinations of John Conway's new solitaire game "life"', *Scientific American* 223, October 1970, pp 120–3.
15 Ibid. 'Conway chose his rules … after a long period of experimentation, to meet three desiderata: 1. There should be no initial pattern for which there is a simple proof that the population can grow without limit; 2. There should be initial patterns that apparently do grow without limit; 3. There should be simple initial patterns that grow and change for a considerable period of time before coming to end in three possible ways: fading away completely (from overcrowding or becoming too sparse), settling into a stable configuration that remains unchanged thereafter, or entering an oscillating phase in which they repeat an endless cycle of two or more periods.'
16 Ibid.
17 Aristid Lindenmayer, 'Mathematical models for cellular interaction in development: Parts I and II', *Journal of Theoretical Biology*, 18 (1968), pp 280–315.
18 The working of L-systems is here exemplified through a simple one-dimensional system consisting of variables (AB), constants (none), a start point (A) and rules (A -> B and B -> AB). With each pass through the system the rules are applied. The rule B -> AB replaces a B with AB. Consequently, stage after stage the system's pattern alternates.
Stage 0: A
Stage 1: B
Stage 2: AB
Stage 3: BAB
Stage 4: ABBAB
Stage 5: BABABBAB
Stage 6: ABBABBABABBAB
Stage 7: BABABBABABBABBABABBAB
Any letter combination can also be used to introduce another set of rules.
19 Kostas Terzidis, *Expressive Form: A Conceptual Approach to Computational Design*, Spon Press (London and New York), 2003, p 71.
20 Ibid, p 72.
21 Ibid, p 73.
22 Strategies for coding and re-coding were explored in my design studio 'Re-coded' at the University of Pennsylvania School of Design, Graduate Program for Architecture, in spring 2004. The work was shown at the 'Re-coded: Studio Rocker' exhibition at the Aedes Gallery East in Berlin during July and September 2005. See also Ingeborg M Rocker, *Re-coded: Studio Rocker*, Aedes East (Berlin), 2005.