Becoming Computational: Restructuring/ Reconsidering Pedagogy Towards a (More) Computational Discipline

Chris Beorkrem, University of North Carolina at Charlotte
Nicholas Senske, University of North Carolina at Charlotte

Bootstrapping a Computational Discourse

Maya Przybylski, University of Waterloo

This paper discusses work from System Stalker Lab, a third year undergraduate design studio taught at the University of Waterloo. System Stalker Lab is an introductory exploration of design computing, aiming to instill awareness of the key structures and processes inherent in a design practice inclusive of computational strategies and techniques. The studio also seeks to seed a computationally oriented design culture within the school by clarifying and speculating on the opportunities existing within computing in relationship to architectural design. Such a practice requires that designers expand their notion of digital methodologies to include the fundamental paradigms of computer science. The focus of the paper is on the first phase of work carried out in the studio, which is committed to building a workable foundation in algorithmic thinking, representation, programming and design – core skills required for working within a computational context. The described process exposes students to the skills necessary for the conceptualization, design, and execution of a project operating within a computational discourse. Having completed the first, highly structured phase of the studio, students are enabled to continue to learn independently and to employ computational design in more open design projects.

Computation as an Ideological Practice

Nathaniel Zuelzke, École Polytechnique Fédérale de Lausanne
Trevor Patt, École Polytechnique Fédérale de Lausanne
Jeffrey Huang, École Polytechnique Fédérale de Lausanne

For computation to become an integral part of architectural design, it must be recognized as an ideological practice. Insofar as it requires explicit, precise formalizations of the factors which shape any given project, computation is a strong assertion of an author's ability to solve a problem. Becoming computational involves acknowledging this agency and the ways in which it differs from conventional paradigms of authorship, and assessing its impact in the design process.

This paper presents computation within the framework of a year-long Masters-level design studio offered at École Polytechnique Fédérale de Lausanne, Switzerland. The studio brief and assumptions are explained and the notion of a "computational engine" is introduced as an evolving document which clarifies its authors' intents. Within the studio, the creation of an engine begins with a parametric site analysis or dynamic mapping of the existing context to construct an understanding of the site which possesses a strong authorial agenda. The temporal, multi-scalar, and diagrammatic nature of the engine are discussed.

By considering solution space, computational workflows are contrasted with conventional ones. Whereas conventional solution space is largely unstructured and underexplored, the relational, combinatorial nature of computational solution space makes it both unknowable in advance yet efficient to explore. The integration of evaluation metrics and feedback systems into the computational engine further increases this efficiency without diminishing authorial will. Finally, some pitfalls are considered, the responsibilities of an author are discussed, and the mechanisms that computation has to mediate these concerns are recapitulated.

Ultimately, computation should not be viewed as an end in itself, but rather as an ideological practice which engenders criticality and promotes innovation.

Computational Design Methods

David Lee, Clemson University

The ACSA Digital Aptitudes Conference celebrates 100 years of architectural discourse. Of parallel importance to the theme of this session, the event will also mark the 50th anniversary of the internet's conception. Indeed, Licklider's concept of the 'Galactic Network' marked a revolutionary shift in thinking about how data sets could be managed and was followed by a series of influential publications that collectively laid the groundwork for the Age of Information.

While computation is not inherently about digital tools, the advent of the Information Age – spawned by the internet and fueled by technology such as, mobile computing, social networking, and GPS – is largely responsible for the current necessity for computational thought in design. Computational thinking being compulsory to the various disciplines that employ information processing, it is critical that architecture schools adopt an attitude that computational thinking be compulsory to the education of the architect. Moreover, it must be engrained in every aspect of a design education, from beginning to advanced design as well as in practice. This paper presents a series of concepts regarding the role of computation in design, specifically architectural, education. Accompanying these concepts are a series of examples of how they have been carried out in courses I have delivered at all levels of an architectural curriculum.

Integrated BIM and Parametric Modeling: Course Samples with Multiple Methods and Multiple Phases

Wei Yan, Texas A&M University

This paper presents well designed modeling samples for teaching integrated BIM and Parametric Modeling in a graduate course. The samples range from parametric curves, recursive solid models, to parametric Building Information Models. Implicit and explicit parametric modeling methods are introduced to the class. Multiple phases of one sample are also exercised. Computer programming is studied as a powerful method for modeling. The objective of integrating the two powerful modeling methods is to foster critical design thinking, which is enhanced by the understanding of the major advantages of BIM and parametric modeling: Creativity, Constructability, and Computability (3C's). The paper describes the samples and methods in detail and compares the different learning focuses and limitations of the multiple methods.