Digital Conformity

RIMA AJLOUNI
University of Utah

In architectural design, profound changes to the nature of design instrumentations are challenging firmly-held assumptions about the relationship between the design ideation and its representation. For centuries, hand-drawn representations have been used to facilitate the cognitive dialogue; bridging the gap between the internal mental images and the external physical world. Today, architectural discourse is rapidly embracing digital mediums, through which designs are conceived and communicated via controlled digital lenses. While these digital environments offer multitude of creative spheres for exploration, the change from atoms to bits is shaping new ways of thinking and making while forcing a higher degree of imbedded submission to the tool's logic. Unfortunately, the role that digital instrumentations play in conforming our physical, virtual and perceptual realities is rarely at the center of investigation. It is therefore critical to look behind the digital interface and to unmask what is redefining design reasoning patterns within architectural discourse. To contribute to this critical discussion, this paper investigates the operating logic, graphic platforms and geometric principles behind most of the commonly used digital software in architecture today. This survey shows that the overwhelming majority of digital tools utilize the same programming and graphical platforms, which raises critical questions about the level of imbedded conformity. Most critically, the geometric concepts used in shaping the contemporary language of architecture are chiefly derived by the abilities and limitations of the software itself and rarely substantiated by architectural theory or intellectual discourse, in which the human consideration is largely missing.

INTRODUCTION

Throughout history the intimate relationship between the craftsman hand, the instrument and the medium has always been in constant negotiations. Traditionally such close-knit relationship is perceived as imbedded in a processional rhythm of improvised and itinerant dynamics; often oscillating between resistance and conformity. In this evolving system of spontaneous interconnections, creative traces and heterogeneity reside and thrive. It was not until the age of the machine that a wider distance between the hand, the instrument and the medium was created. The industrial revolution did not only challenge the notion of personal affinity in crafts, but more critically emphasized the homogeneity of output which resides at the opposite end of craftsmanship. Today, the emerging digital, fabrication and information technologies are once again transforming the way in which buildings and artifacts are conceived, communicated, crafted and experienced. While mediating the distance back to a personal space, the change from atoms to bits is challenging the direct entanglement of crafts with material culture. The shift from the physical to the virtual is shaping new ways of thinking and making, while forcing a higher degree of imbedded submission to the tool's logic. Unfortunately, the role that digital instrumentations play in conforming our physical, virtual and perceptual realities is rarely at the center of investigation.

While, the use of digital media is steadily redefining design reasoning patterns within architectural discourse, the computer as a machine is still seen as the 'black box'. It's operating logic and architecture remains a mystery to most users in the field. Unfortunately, designers are often unaware of the impact that the tool's specific architecture, logic and workflow has on design ideation and reasoning. This blind dependency is causing many challenges; provoking critical debates concerning the effect of the digitally-driven ideations on architectural products and discourse. Today a robust and a comprehensive framework underpinning the integration of digital media into architectural pedagogy and practice is urgently needed. It is therefore critical that we take a critical look at the digital environment and to unmask what is actively redefining design reasoning patterns within architectural discourse.

BACKGROUND

It is widely recognized that different representational mediums facilitate different facets of information processing during design (Cook 2008). Critical distinctions between ill-structured representations (i.e. freehand sketching, diagramming, etc.) and well-structured representations, (i.e. digital representations, computation, etc.) are clearly marked, each stimulating different aspects of the cognitive faculties. For centuries, hand-drawn representations have been used to bridge the gap between the internal mental images and the external world (Gross et al. 1988, Herbert 1988, Goel 1992, Goel 1995, Eastman 2001). Today Architectural discourse is undergoing a fundamental change regarding its design media, in which well-structured digital representations (i.e. CAD drawings, 3D digital Models, rendered views, computational models, etc.) are rapidly replacing the ill-structured hand-drawn representations (i.e. sketches, diagrams, perspective

drawings, etc.). While the cognitive properties of hand drawing have been extensively explored in the design process (Evans 1986, Van Sommers 1984), a deeper understanding of the cognitive aspects associated with using the digital environment is still unclear.

THE DIGITAL ENVIRONMENT

In 1945 John Von Neumann introduced the basic design principles of the digital computing environment. Neumann proposed an electronic digital device that "can carry out instructions to perform calculations. The instructions which govern this operation must be given to the device in absolutely exhaustive detail. They include all numerical information which is required to solve the problem under consideration" (Neumann 1945 pp1). Neumann proposed that all numeric values are translated into machine language using a binary digits (bits) of the two values "0" and "1", in which, the "bit" constitutes the smallest measuring unit for computer memory. According to these principles, the digital environment operates by processing a numerical 'input' using arithmetical and/or logical operations to produces numerical 'output'. This process follows a linear 'fetch-execute' cycle, in which instructions are executed in a consecutive order. Accordingly, in order to perform any task using the digital environment, the problem needs to be described as algorithmic conversation between input and output (Denning 2009). Therefore, in design contexts, the question of 'computability' of the problem becomes critical to its digital execution. The rationalization of the design problem by means of digital electronics necessitates that the problem be constructed in terms of computable functions between input and output (Kotnik 2010), has an algorithmic solution and can be measured quantitatively. Therefore, it is essential to understand the nature of the design problem before adopting a digitally-driven process in search for solutions.

THE NATURE OF THE DESIGN PROBLEM

Our understanding of design as a cognitive paradigm has evolved over time; moving away from identifying design as an objective rational problem solving process (Simon and Newell 1958, Newell 1969, Newell and Simon 1972, Simon 1987), towards a new realization that identifies design as involving a complex and dynamic processes that are overwhelmingly subjective in nature (Schön 1983). In this new realization, a strong correlation exists between three cognitive aspects: the nature of the design problem under investigation (well-defined vs. ill-defined), the nature of the supporting knowledge structure (factual, informal, procedural, experiential, representational, etc.) and the nature of design representations (ill-structured representations (i.e. freehand sketching, diagramming, etc.) vs. well-structured representations (i.e. digital representations, computation, etc.). Cognitive sciences make a clear distinction between the cognitive processes associated with two types of design problems: an ill-defined design problem vs. a well-defined design problem; each activating different patterns of reasoning during design processing. A well-defined design problem mandates that all necessary information needed for defining the problem is available before initiating the search for possible solutions. This definition operates within an objective rational problem solving sphere, in which design thinking involves the process of searching for solutions that satisfy the predefined problem space. In this context, a well-defined design problem draws heavily upon objective sources of knowledge including formal, factual and procedural, while mainly benefiting from well-structured representations, such as digital and computational models. On the other hand, an ill-defined design problem operates within an undetermined problem sphere, in which the information necessary to construct the problem is lacking at the outset (Simon 1973, Goel 1992). Such design process is overwhelmingly immersed in subjective interpretation and draws upon implicit knowledge structure including experiential, social and cultural, as well as benefiting from ill-structured design representations to stimulate design dialogue.

The architectural design process encompasses a dualistic nature involving both objective and subjective interpretations (Gadamer 1986, Goel 1992, Dorst 2004). The initial phases of a design process are mostly characterized as ill-defined nature and rely on subjective interpretations to facilitate the lateral movement between multiple ideas; allowing a wider creative space for innovation (Eastman 2001, Dorst 1997, Dorst 2004, Goel 1992). Moving vertically in the depth of the design problem, design interpretations shifts to a more structured refinement and detailing toward a well-defined problem space. The later stages of the architectural design process are often immersed in objective interpretations, which responds better to well-structured representations such as digital and computational modeling.

A prerequisite for using the digital electronic system as a design tool, is understanding the "necessary limitation on computable functions as mediator between input and output" (Kotnik 2010, pp 6). Therefore, in the context of the design process, the use of digital instrumentation mandates that the problem be abstracted numerically and rationalized using deductive reasoning logic using precise arithmetic and/or logical expressions. It is therefore critical to understand that employing well-structured representations such as digital and computational modeling at early design exploration phases can obstruct the lateral search for alternative solutions, which can result in a premature crystallization of the design (Goel 1992). Moreover, similar to reading, writing, sketching and math skills, any learned external digital representation needs to be internalized in order to be effective part of the intrinsic cognitive mental reasoning skills (Eastman 2001). Therefore, in order to facilitate the creative process during design reasoning, digital skills need to be absorbed and internalized at a cognitive level (Akin & Akin 1998).

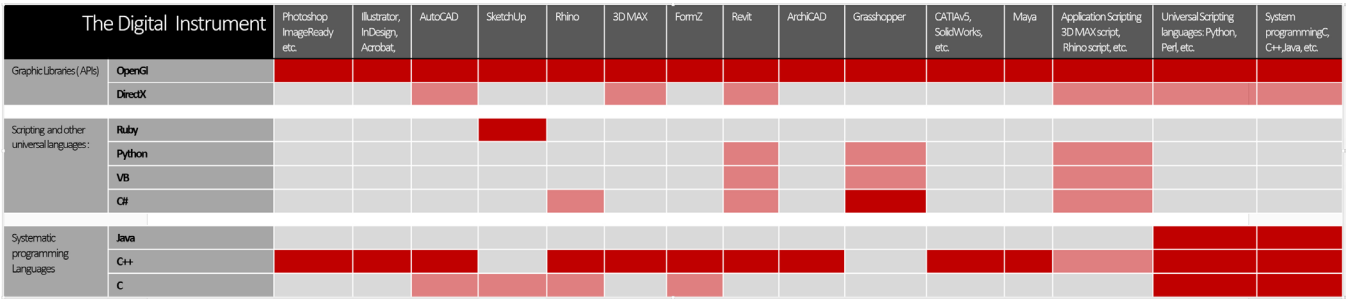

Table 1: Programming and graphical platforms used to craft digital instrumentations.

ARCHITECTURE PEDAGOGY AND DIGITAL INSTRUMENTATION

In architectural pedagogy, students are increasingly required to learn and utilize a range of different digital tools to inform their design process without a proper understanding of their internal algorithms or operating logic. Table 1 lists an array of the most commonly utilized digital instrumentations in architecture schools today. While these tools offer multiple spheres for creative exploration, they are fundamentally redefining design reasoning patterns within architectural discourse. Without an understanding of such impact, design pedagogies risk causing cognitive and technical challenges; potentially driving the educational experience out of focus. In order to understand what is deriving architectural pedagogy and practice today, this paper looks behind the digital interface and discusses three main aspects: (1) Programming and graphical platforms. (2) Graphic representations and modeling techniques and (3) digitally-driven design logic.

PROGRAMMING AND GRAPHICAL PLATFORMS

To understand the architecture behind any of the digital tools, it is beneficial to look at the underlying programming logic and graphical platforms. Table 1 lists the different programming languages and graphical libraries that are used in crafting most of the digital instrumentations in architecture today. In general, the purpose of a programing language is to provide a formal method for instructing the computer to perform certain tasks using very specific syntax (form) to convey specific semantics (meaning). However, programing languages do not provide a library for graphic representations, therefore a graphical library is needed to provide the geometric primitives for building the graphical interactive components within the software. If we survey the digital tools in table 1, we find that 100% of these instrumentations utilize the same free OpenGl graphical Library and almost 87% of the programming logic rely heavily on C++ object-oriented programming to craft their functionalities.

C++, which was developed by Bjarne Stroustrup in 1983 as an extension of the C language, is one of the most powerful and fast mid-level programming languages that is commonly used for crafting Graphic User Interface (GUI) based applications including advanced graphical software and gaming engines. C++ is an object-oriented, data centered programming which allows data and behavior to be encapsulated to create user-defined date types. For example, instead of writing a code for a general shape-drawing function, which might be very expensive in terms of storage space, the programmer can declare a class of shape object, in which every individual object is optimized in term of its drawing function. The programmer can then evoke the appropriate object's drawing function when needed. In general, class declaration in C++ provides "strong typing, data hiding and code reuse through inheritance" (Pohl 1999 pp2), which can be best used to achieve modular-programming.

The OpenGl Graphic Library, on the other hand, is an interface to graphics hardware that allows the creation of 2D and 3D visual interactive programs using rendered images. Its library includes 250 distinct commands that are used to describe 2D and 3D objects (Shreiner et al. 2005). For example, the OpenGl Utility Toolkit (GLUT) includes routines for drawing many of the 3D object found in architectural drawing software, including the box, the sphere, the torus, the cone, and even the famous Utah teapot. Most critically, OpenGl allows programmers to manipulate geometry using three basic modeling transformations: glTranslatef(tx, ty, tz), glRotatef(angle, vx, vy, vz) and glScalef(x, y, z). This suggests that any geometric manipulation using this digital platform is confined within these three transformation routines. The fact that almost all digital tools in architecture are structured based on the same object oriented logic and use the same basic graphic library raises important questions regarding the level of imbedded uniformity that comes with using these common platforms.

GRAPHIC REPRESENTATIONS AND MODELING TECHNIQUES

In digital graphics there are two types of representations, raster and vector. Raster images store data as a matrix of individual pixels with a fixed resolution (height and width), which cannot be enlarged without sacrificing image quality. The raster files are used in all image manipulation software including Photoshop and ImageReady and are often used to process images for rendering 3D architectural views. With the ability to use advanced material and lighting, rendering engines allow designers to view their designs as direct visual experience that is readily interpreted as "realistic" (Turkle

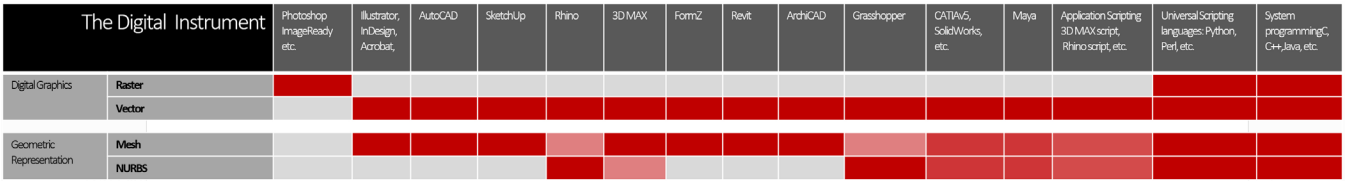

Table 2: Graphic representations and modeling techniques used in digital instrumentations.

2009), which engage very different cognitive processes than orthographic abstract projections (Scheer 2014). While these 'realistic" views are often accepted as accurate reflections of reality (Sontag 1977), the implications of the absence of the creative cognitive distance that facilitates abstractions during design processing are still unclear.

In vector data, geometric shapes are defined with coordinates (2D (x,y) or 3D (x,y,z)). Every point is saved in a matrix of 2D or 3D array and manipulated mathematically through arithmetic operations (addition, subtraction, multiplication and division). Complicated scenes are built from basic geometrical primitives (i.e. points, lines, curves, polygons). While almost 94% of architectural software utilize vector graphics to describe and manipulate geometry (table 2), it is rarely that designers are exposed to the mathematical calculations of vector geometry hidden behind the digital interface. While understandings these principles can be instrumental in enabling a better control over design representations and geometric manipulation, unfortunately Math and Calculus requirements are gradually taking the back seat in architectural education.

MODELING TECHNIQUES

In the 3D modeling environment, there are two main approaches for modelling geometry (i.e. subdivision surfaces (Polygonal Mesh, and NURBS (Non-Uniform Rational B-Spline), each enabling a different geometric language for crafting architecture. A polygonal mesh model is a collection of joined polygons that are explicitly represented by a list of vertex coordinates. In this method a curve is defined by a collection of points connecting line segments along its length. Any complicated surface can be smoothed by recursively subdividing it to smaller segments. The use of recursive functions for smoothing complicated geometry has proven to be most valuable for crafting and animating characters and primarily used in the gaming and animation industries.

The NURBS modeling method, on the other hand defines a curve or a surface by a parametric function utilizing only few control points around its vicinity. The curvilinear geometry changes based on manipulating the location of the control vertices around it. The development of the computational curve and the basis behind the NURBS geometry was chiefly driven by the automotive and aerospace industries as a response to their need for a common mathematical language to model smooth curvilinear surfaces used in computer numerically controlled (CNC) manufacturing (Townsend, 2014). In 1959, working for the French automotive company Citroen, the mathematician Paul de Faget de Casteljau developed the first algorithm for computing a curve using few control points (Casteljau, 1999). In the 1966 Pierre Bezier, an engineer working for the French automotive company Renault, invented the Bezier curve, which is used in many graphic software like Adobe illustrator (Bézier 1998). The B-Splines, a more accurate controllable algorithm was discovered by Carl de Boor, a researcher at GM automotive company (Boor 1978). Later the aerospace industry, namely Boeing, was instrumental in devising a general geometrical model that could be used to describe all types of complex free-form curves and surfaces and was called the Non-Uniform Rational Basis Splines (Blomgren and Kasik 2002). These advances were later adopted by the mainstream computing communities and integrated into software applications for many of the design fields.

In the past few years the popularity of the NURBS modeling has increased in architecture, mostly due to the adoption of NURBS-based software (i.e. Maya, Rhinoceros, CATIA, etc.). This has led to the emergence of a new language of "digital architecture" that is based on complex geometry and organic freeform typology. As a result, complicated forms and arbitrary exotic complexities are steadily streaming out of architectural schools. While the NURBS geometry has proven to be very beneficial for testing and modeling fluid surfaces based on aerodynamic principles for the automotive and aerospace industries, in architectural design however, the rationalization of the NURBS freeform geometry for production and construction has been very challenging and largely unsubstantiated. Few architects were able to overcome these limitations (i.e. Zaha Hadid, Frank Gehry, etc.), however these one-off projects were proven to be very complicated and expensive to conceive and fabricate (Townsend, 2014). In 1999 the term "Blobitecture" was coined; referring to the adoption of amorphous, blob-like organic topology used in shaping the new language of contemporary architecture. The advances in the NURB modeling coupled with availability of the computer-aided manufacturing (CAM) allowed designers to move away from the Cartesian space and embrace the new topological vocabulary; forming an experimental paradigm of "form finding" largely driven by the manipulation of topological geometries available within the digital tool palette (i.e. stretching, folding, bending, etc.).

The Digital Instrument		Photoshop ImageReady etc.	Illustrator, InDesign, Acrobat,	AutoCAD	SketchUp	Rhino	3D MAX	FormZ	Revit	ArchiCAD	Grasshopper	CATIAv5, SolidWorks, etc.	Maya	Application Scripting 3D MAX script, Rhino script, etc.	Universal Scripting languages: Python, Perl, etc.	System programming C, C++, Java, etc.
Output-driven Models	Direct modeling															
	Representations															
Relationship-driven Models	BIM															
	Parametric															
Process-driven Models	Generative															
	Genetic algorithms, flocking, cellular automaton															

Table 3: Digitally-driven design logic vs. digital instrumentations.

Today many proponents of the "digital architecture" are promoting the "Blob" vocabulary into the main stream design domain (Lynn 1999), however the consideration for the human condition is largely missing from this proposition. With the democratization of complex free-form and fluid-like blob architecture, corners, edges, boundaries, orthogonal and Euclidean geometry are gradually dissolving into the fluidity of this new language of architecture. It is therefore critical that we ask "how does this new environmental language respond to the human condition?"

In 2014, three neuroscientists won the Nobel Prize for their discoveries of brain cells (place cells, grid cells, head direction cells and boundary cells) that constitute a positioning system in the brain (Richard 2014). Amazingly, our brain maps the environment by firing signals based on a perfect hexagonal symmetry. It has been established that the brain's GPS system is largely influenced by the shape of the environment (Richard 2014, Krupic, et al. 2015), in which the brain's positioning functions are sharpened with orthogonal and symmetrical environments. The lack of edges, corners and finite boundaries can have an irreversible negative impact on the development of essential navigating functions of the brain (Krupic, et al. 2015) and ultimately affect our ability to function in a space. The blind adoption of digitally-driven architectural languages without a proper understanding of their impact on our physical and perceptual realities raises many ethical questions about the responsibility of the architect to uphold the minimum ethical standards for responding to the human condition.

DIGITALLY-DRIVEN DESIGN LOGIC

The rationalization of design thinking in terms of digital computation requires designers to logically abstract the design problem in terms of input, algorithmic conversation and output. Accordingly, the degree of 'computability' of the design intent is closely linked to the computational reasoning logic that is used to derive design alternatives. In this context, three levels of digitally-driven modeling logic are defined: descriptive (output-driven), parametric (relationship-driven), and generative (process-driven).

The **descriptive** design logic is an output-driven process, in which the designer is utilizing the digital environment for modeling a preconceived geometry; focusing primarily on the desired output rather than design manipulation at the algorithmic level (Oxman 2006, Kotnik 2010). The range of the digital environments that are used for descriptive modeling is wide and includes all available CAD modeling software (table 3). However, using available software mandates that design representations be confined within a limited set of predefined geometric functions, which limits the geometric manipulations to a specific tool pallet and have no control over internal function definitions. In this context, the use of programming languages provides a method to break free from the limitations associated with using existing tools. For example, figure 1 shows a predefine Circle command from the Rhino tool pallet (left) and an independently written Circle function using C++ and OpenGl library (middle). The Rhino Circle command and the Circle function draw a circle using two parameters; a center location (x,y) and a radius (rad). However, while the rhino command limits the designer's control to only manipulating the center position of the circle and the radius, the independently written Circle function allows control over the function definition, which enables access to the algorithmic level including the number of circle segments, variation of the radius, etc. (right). The ability to manipulate and craft independent functions allows the designers a wider space for exploration and control over geometric manipulation.

The **parametric** design logic is a relationship-driven design logic, in which design intent is encoded in a predefined set of relationships and parameters as a conversation between input and output following a deductive reasoning logic. In this process a geometric model is expressed in a set of parametric functions (equations), in which variations of the dependent 'output' are generated based on testing ranges of variations across the independent 'input' parameter(s). Parametric design reasoning is informed by mathematical formulas for testing performative or formative output(s), which frame the design problem and solution within a quantitative perspective and can potentially steer the creative energy towards optimization (Kotnik 2010). Accordingly, parametric logic can only operate within a well-defined design problem sphere; which risks confining the 'design space' to a fixed field of explicit relationships as well as narrowing the 'solution space' to a limited range of possibilities. Moreover, available CAD graphic software (table 3), are not designed to facilitate parametric explorations at the algorithmic level. Therefore, knowledge of programming languages is needed to extend

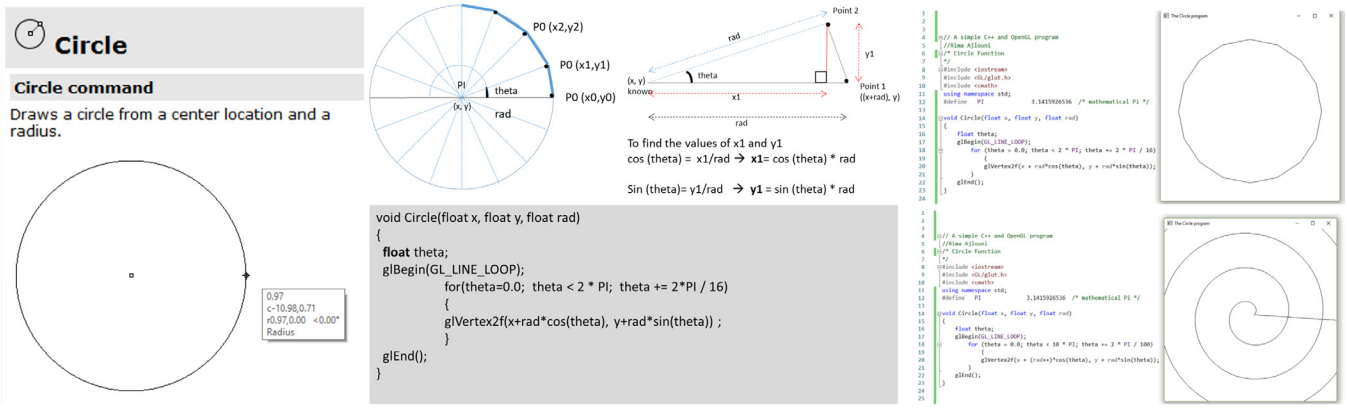

Figure 1: Left: the Rhino Circle command. Middle: Circle function written in C++ and OpenGl library. Right: Manipulations of the C++ Circle function.

the functionality of existing tools and to enable parametric manipulations.

The **generative** design logic, requires that design intent to be encoded in sequences of processes that are structured to evolve independently beyond the designer's initial input. In this evolutionary perspective, design output(s) emerge as a global response to the intensive processes at the local level. The interaction among local entities give rise to the collective global behavior. In architectural design, generative thinking logic is seen as an experimental domain for exploring behavioral-driven form-intelligence at an abstract level, which is more about phenomena recognition than a formal design method. Such exploration is currently focused on abstracting complex systems from nature (i.e. genetic algorithms, flocking, cellular automaton, etc.) and adopting them as design generators. However, it is not clear how the geometric output responds to the initial design intent, in which an objective evaluating criteria can be challenging. Moreover, knowledge of programming languages and formal scientific knowledge are essential for deploying the generative computational design reasoning logic.

CONCLUSION

Based on this review, it is evident that beyond the variations in the digital interface, most of the digital instrumentations used in architecture today share the same operating logic, graphical platform and geometric principles. This fact raises important questions about the level of conformity that comes with using these common platforms. Moreover, the integration of the digital environment into the design process mandates that the design problem be quantitatively defined in an explicit arithmetic form before initiating the search for numeric solutions. While such design environments provide many creative spheres for exploration and optimization, this quantitative perspective on design is not structured to explore ideas, to make connections between society and the built environment or most critically to bring the consideration of human condition to bear during design. Instead, the inbuilt emphasis on geometric manipulation and quantitative optimization can unintentionally steer the designer's attention away from the qualitative aspects of design.

Unfortunately, architecture profession is rarely involved in developing the digital instrumentations or graphical concepts that it uses. Architectural curriculum rarely offers opportunities to look behind the interface and understand its abstract logic or geometric principles. The lack of the essential knowledge that enables the designer to have cognitive and technical control over the tool's limitations and possibilities risks confining the design exploration to the tool's limited pallet as well as be bounded by the designer's level of technical skills. It is critical to recognize that the use of the digital instrumentations in architecture discourse affect the human condition on multiple levels, including cognitive, technical, perceptual and physical. What is urgently needed today is a comprehensive framework underpinning the integration of digital environment into architectural discourse that enables the digital architects to regain control over the tool and the medium, to contemplate diversity of considerations in their designs, and most critically to bring the human considerations back into balance with those of efficiency and optimization.

REFERENCES

1 Akin, Ö. and Akin, C. 1998. "On the process of creativity in puzzles, inventions, and designs." *Automation in Construction* 7 (2–3): 123-138.

2 Bézier, Pierre. 1998. "A View of the CAD/CAM Development Period." *IEEE Annals of the History of Computing* 20(2): 37-40.

3 Blomgren, Robert M. and Kasik, David J. 2002. "Early investigation, formulation and use of NURBS at Boeing." *ACM SIGGRAPH Computer Graphics* 36 (3): 27-32.

4 Boor, de Carl. 1978. *A Practical Guide to Splines*. Carl de Boor's Homepage. Retrieved from: http://pages.cs.wisc.edu/~deboor/

5 Casteljau, Paul de Faget. 1999. "De Casteljau's autobiography: My time at Citroën." *Computer Aided Geometric Design*, 16(7): 583–586.

6 Cook, Peter. 2008. *Drawing: The Motive Force of Architecture.* (Chichester: John Wiley & Sons Ltd.)

7 Denning, Peter J. 2009. "The Profession of IT: Beyond Computational thinking," *Communications of the ACM*. 52(6): 28-30.

8 Dorst, C.H. 1997. *Describing Design - a Comparison of Paradigms*, thesis TUDelft.

9 Dorst, K. 2004. "The Problem of Design Problems - Problem Solving and Design Expertise." *Journal of Design Research* 4(2).

10 Eastman C. 1969. "Cognitive processes and ill-defined problems: a case

study from design." In D.E. Walker & L. M. Norton (eds), *Proceedings Joint International Conference on Artificial Intelligence:* 669–690. Bedford, MA: The Mitre Corporation.

11 Eastman, C. 2001. *New Directions in Design Cognition: Studies of Representation and Recall*. In C. Eastman, M. McCracken & W. Newstetter (eds), Design Knowing and Learning: Cognition in Design Education; Amsterdam: Elsevier Science: 147-198.

12 Evans, Robin. 1986. *Translations from Drawing to Building. AA Files* (Architectural Association School of Architecture) 12: 3-18.

13 Gadamer, H. G. 1986. *The Relevance of the Beautiful and Other Essays*. R. Bernasconi & N. Walker (eds). Cambridge: Cambridge University Press.

14 Goel, V. 1992. "Ill-structured Representations for Ill-structured Problems." The Fourteenth Annual Conference of the Cognitive Science Society. Hillsdale, NJ. Lawrence Erlbaum.

15 Goel, V. 1995. *Sketches of Thought*. Cambridge, MA: MIT Press.

16 Gross, M.D., Ervin, S.M., Anderson, J.A. and Fleischer, A. 1988. "Constraints: knowledge representation in design." *Design Studies* 9(3): 133-143.

17 Herbert, D.M. 1988. "Study Drawings in Architectural Design: Their Properties as a Graphic Medium." *Journal of Architectural Education* 41(2): 26-38.

18 Kotnik, T. 2010. "Digital Architectural Design as Exploration of Computable Functions." in *International Journal of Architectural Computing* 08(1): 1-16

19 Krupic, J., Burton, S., Barry C. and O'Keefe, J. 2015. "Grid cell symmetry is shaped by environmental geometry." *Nature* 518: 232–235.

20 Lynn, G. 1999. *Animate Form.* Princeton: Princeton University Press.

21 Neumann, V. 1945. First Draft of a Report on the EDVAC. University of Pennsylvania. https://web.archive.org. Accessed on Jan 05 2016.

22 Newell A. 1969. Heuristic Programming: Ill-structured Problems. In Julius Aronofsky (ed.), *Progress in Operations Research* 3: 360-414. John Wiley & Sons.

23 Newell, A. & Simon, H. A. 1972. *Human Problem Solving*. (Englewood Cliffs. NJ: Prentice Hall).

24 Oxman, R. 2006. "Theory and Design in the First Digital Age." *Design Studies* 27(3): 229-265.

25 Pohl, I. 1999. C++ for C Programmers. Third Edition. Reading, Massachusetts: Addison-Wesley.

26 Richard, Van N. 2014. "Nobel for microscopy that reveals inner world of cells. Three scientists used fluorescent molecules to defy the limits of conventional optical microscopes." *Nature* 514: 286.

27 Scheer, David Ross. 2014. *The Death of Drawing: Architecture in the Age of Simulation.* (New York and London: Routledge Taylor & Francis Group).

28 Schön, D. 1983. *The Reflective Practitioner: How Professionals Think in Action*. New York: Basic Books.

29 Shreiner, D. Woo, M. Neider, J. and Davis, T. 2005. OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2, Fifth Edition. Addison-Wesley.

30 Simon, H. & Newell, A. 1958. "Heuristic Problem Solving: The Next Advance in Operations Research." *Operations Research* 6(1):1-10.

31 Simon, H. 1973. "The Structure of Ill-structured problems," *Artificial Intelligence* 4:181-203.P

32 Simon, H. 1987. Problem forming, problem finding, and problem solving in design. In A. Collen & W. W. Gasparski (eds.), *Design and Systems: General Applications of methodology* 3: 245-257. New Brunswick, NJ: Transaction Publishers.

33 Sontag, Susan. *On Photography.* New York: Picador, 1977.

34 Townsend, A. 2014. "One the Spline: A Brief History of the Computational Curve." *Applied Geometries* 3.

35 Turkle, Sherry. 2009. *Simulation and its Discontents*. (Cambridge: The MIT Press).

36 Van Sommers, Peter. 1984. *Drawing and Cognition: Descriptive and Experimental Studies of Graphic Production Processes.* Cambridge: Cambridge University Press.