More Translations (from Drawing to Building)

GALO CANIZARES
The Ohio State University

> "[Digital technologies] are no longer the tools for making: they are primarily tools for thinking."
>
> —Mario Carpo, The Alternative Science of Computation[1]
>
> "The theme of this article is translation...There are all those other identically prefixed nouns too: transfiguration, transformation, transition, transmigration, transfer, transmission, transmogrification, transmutation, transposition, transubstantiation, transcendence, any of which would sit happily over the blind spot between the drawing and its object, because we can never be quite certain, before the event, how things will travel what and will happen to them on the way."
>
> —Robin Evans, Translations from Drawing to Building[2]

In 2013 media theorist, Lev Manovich, wrote, "there is no such thing as 'digital media.' There is only software."[3] In other words, because all digital artifacts rely on a set of interpretations of virtual signals, software and media today are inseparable; to think of one is to think of the other. As such, designers have come to understand acts of making as fundamentally tied to a set of programs and interfaces that digitize our ideas by translating them into electric impulses. This has been the case in architectural design for the better part of the previous decade and is certainly the case today. However, while this phenomenon has led to a set of norms concerning the production of digital objects, an increase in the variety of tools available to architectural designers (particularly from video game design, visual effects industries, open-source initiatives, and app developers) has opened the door to new ways of producing and understanding architectural media. The goal of this paper is to examine architecture's evolving relationship with software, and suggest a reevaluation of the role digital mediums play in architectural education.

Beginning with Robin Evans I would argue that architects have always had an interest in examining closely the mediums on which they work. Since architectural media has now evolved into primarily virtual information, we must re-examine that relationship. Taking Lev Manovich's dictum that "there is only software" and recent pedagogical discussions on "computation as a background condition of our reality,"[4] I will suggest that, much like they cover Brunelleschi's rediscovery of perspective or Alberti's *De Pictura*, design curricula must tackle the history and theory of software in order to fully comprehend the tools and techniques involved in contemporary architectural design. This approach would allow designers to critically reflect on the impact of software on culture, and in turn the effect of digital culture on architecture.

EVERYTHING IS SOFTWARE

If Robin Evans' ceaselessly regurgitated *Translations from Drawing to Building* was to signify anything other than a debunking of the common architectural myths associated with notions of drawing buildings and building drawings, it would be the emergence of a renewed interest in the intellectual value of scrutinizing the very mediums in which we work. Writing in 1986, Evans is admittedly responding to various cultural dialogues concerning, on one hand, the fabled autonomy of the drawing, and on the other, architecture's abstract disciplinary knowledge.[5] We've heard this countless times before, and we all know that architects do not make buildings, they make drawings of buildings. Yet, it is thirty-one years later, and here we are, still ruminating on what happens when we translate between a drawing and a building. But it seems that our preoccupations today address a different kind of translation; one not necessarily concerned with whether the drawing itself exists as "the real repository of architectural art."

Before diving in, let's make two assumptions. (1) That we live in the world of ubiquitous software; and (2) that the architectural drawing (at least the kind that dreams of becoming a building) is primarily a digital artifact. Putting aside any nostalgic opprobrium this might incite, "drawing" in the case of this essay will neither refer to the intellectual act of *disegno*, nor to the drafting of lines, whether digital or analog, projective or perspectival. Instead, "drawing" should be understood as a placeholder for a variety of digital file formats that are readily used in contemporary architectural practice (eg. DWG, PDF, JPG). Therefore, given our previous assumptions, we can situate the architect as a figure whose principal task is not only to translate between drawing and building, but also to translate across a vast, ever-updating landscape of standardized file-types and graphical user interfaces. While this may appear obvious, perhaps even remedial, a critical discourse surrounding these processes has not surfaced until recently.[6]

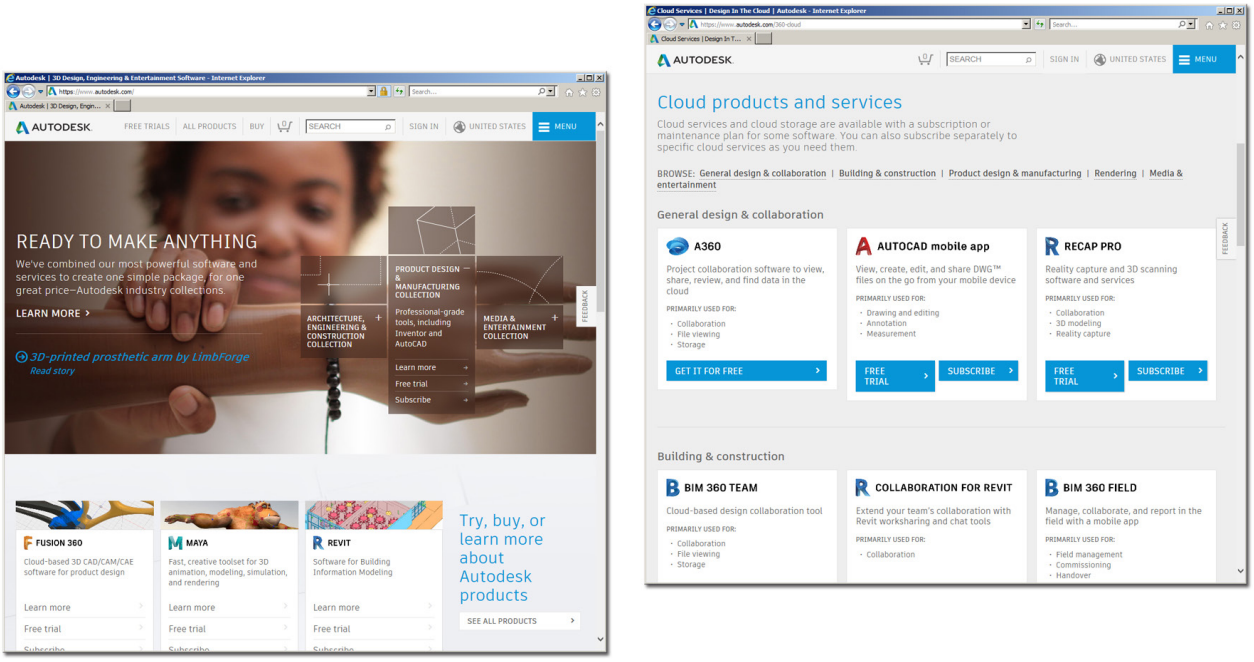

Figure 1: Autodesk advertising software which allows users to "make anything" and work across distances on the Cloud and mobile devices.

An understanding of design software has been stifled by a desire to marginalize it as a utility with the humanist paradigm that prioritizes design intellect's hegemony over mechanical tools. But this analogy of the digital as a tool for the realization of some architectural *a priori* is just the myth Evans debunks.[7] In *Translations from Drawing to Building*, he famously insists that the drawing does not reveal objective truths of architectural space, but instead obfuscates truth in favor of a series of mediated effects. The architectural drawing therefore stands apart from the technical and becomes "more abstract in appearance, more penetrating in effect…and suggestive of a perverse epistemology in which ideas are not put in things by art, but released from them."[8] And as drawing becomes ever more digitized, this dynamic interaction between thoughts and representations encompasses a vast landscape of mediums in the form of software. Thus, software is no longer a vehicle for simply communicating that which we blindly create in our heads, but rather, much like analog drawing, contributes to the formulation of that very thought from our first encounter with it. It is this reformulation of our relationship to the digital—what some are calling the second digital turn—that I wish to discuss.[9] In the context of this essay, *software* should be understood as the layer of interpretive interfaces that lie between a user and a computer's operating system. The term *digital media* will be used to refer to the outputs of this layer, the products of user's interactions with software.

Today, from Autodesk Revit coordination to the management of plug-in applications to optimizing files for printing, an ever-growing repertoire of software skills are crucial to the production, dissemination, and translation of a design project at all scales. Because humans are largely inept at decoding binary syntax, software is the mediator whenever ideas move from abstract thoughts to mechanized gestures on the screen and pixel color values transmogrify into instructions for assembly. The science-fiction reality is: that our globalized world has reached a point where most communication must be interpreted by some software. As Keller Easterling notes, "we know that as soon as communications leave our lips or fingertips they are immediately diced and rearranged into information packets better suited for streaming digital compression."[10] However, despite the ubiquity of the digital, I am not suggesting that architects should become programmers, nor that computation be conflated with design. Instead, my point is that, because it is so pervasive there needs to be a deeper understanding of the critical and cultural role that software plays in design processes from education to practice. As Lev Manovich argues in *Software Takes Command*, "[i]t is, therefore, the right moment to start thinking theoretically about how software is shaping our culture, and how it is shaped by culture in turn."[11]

Manovich has widely been regarded as the progenitor of these thoughts within the digital humanities, having been one of the earliest thinkers to synthesize and historicize the development of "cultural software:" a loose umbrella term referring to computer applications involved in the creation of cultural

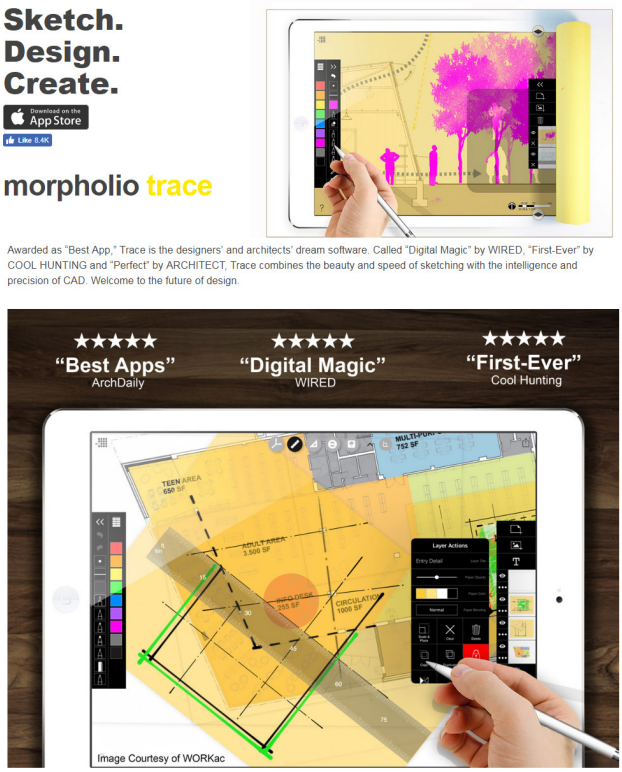

Figure 2: Morpholio Trace is a "smart" sketching app for mobile devices, which simulates pen drawing with CAD features.

artifacts, interactive services, aesthetic content, online social interactions, and virtual experiences.[12] Of course, each discipline has its own catalog of go-to software, which relies largely on a combination of specified workflows, licensing costs, industry standards, and delivery methods. In art, for example, "new media" artists often make use of video-game design programs to create interactive pieces. For us architects, we know the usual suspects: Adobe Photoshop/Illustrator/InDesign, Autodesk AutoCAD/Revit/3d Studio Max/Maya, Rhino 3d, to name a few. Yet while students and professionals use these programs on a daily basis to create artifacts, Manovich's theses posit that most discussions rarely touch on their impact on cultural conventions or their historical development.

But for now, let us return to the issue of translation (from drawing to building). If a major task of the contemporary architect is to manage the collection and transmission of various file-types through networks both local and international, then one would expect architectural curricula to address the fundamentals of navigating this digital landscape.[13] While design pedagogy covers disciplinary knowledge such as abstract design and communication techniques, technical skills for successfully outputting a desired object from a piece of software are not as synthesized as drawing, which is taught both theoretically and technically (think: one seldom encounters how to draw a perspective without mentioning Brunelleschi's discovery of it). In the case of software, best practices for compression, optimization, or conversion are engaged as means to particular ends; rarely as historical or theoretical concepts. Surely an immersion into the annals of software's history and socio-cultural impact is as warranted as the lineage of drawing mediums from the Renaissance. At least from a desire for a well-rounded architectural education, Ivan Sutherland's invention of interactive computer graphics must be mentioned in the same breath as Brunelleschi's perspective.

However, this need for profound computational knowledge is a relatively recent development in our discipline. It can be seen as an evolution of those digital design processes, codified during the early 2000s (a.k.a. the first digital turn), that were results of students tinkering with early modeling software.[14] Although important themes surfaced then, such as the basic discrepancies between OBJ meshes and IGES B-splines or those between raster and vector images, it should be noted that this insight was shared at the discretion of those early adopters of digital tools who saw the medium as primarily a playground for architectural form. Designers had little to no background in computation, and as a result, software was perceived as an extension of the hand. A decade later, it is evident that design pedagogy and design software have not kept up with each other. Most students enter the professional field with vast technical bravado and an ability to translate back and forth between three-dimensional and two-dimensional media, but few are able to describe exactly what takes place inside these steel, whirring boxes, or why certain operations work efficiently whereas others do not. (Pop quiz: why is an STL file better for 3D printing than an OBJ file?)[15]

I would argue that the presumption of software as a simple tool subservient to our architectural whims is an outdated pedagogical model. During the first digital turn, the assertion that a designed product is only as good as the designer would yield nods of agreement from most in the field, a sentiment derived perhaps from the skeuomorphic qualities of the digitized drafting table that is CAD's "paperspace". Yet, the number of managerial decisions to make regarding our products has grown exponentially since, warranting a need for deeper grasp of these systems. If a student sketches by hand on a tablet, she will immediately have to decide whether to save the sketch as a JPEG, PNG, TIFF, or any other image file-type as well as the resolution of the sketch. Not only has this blurred the distinction between drawing and image, but it has also added the dimension of "compatibility" to the product: the sketch can only be read by software that accepts that file-type. Let us also not forget that some file formats are proprietary and require a specific version of an application for full use. This user-level politics is

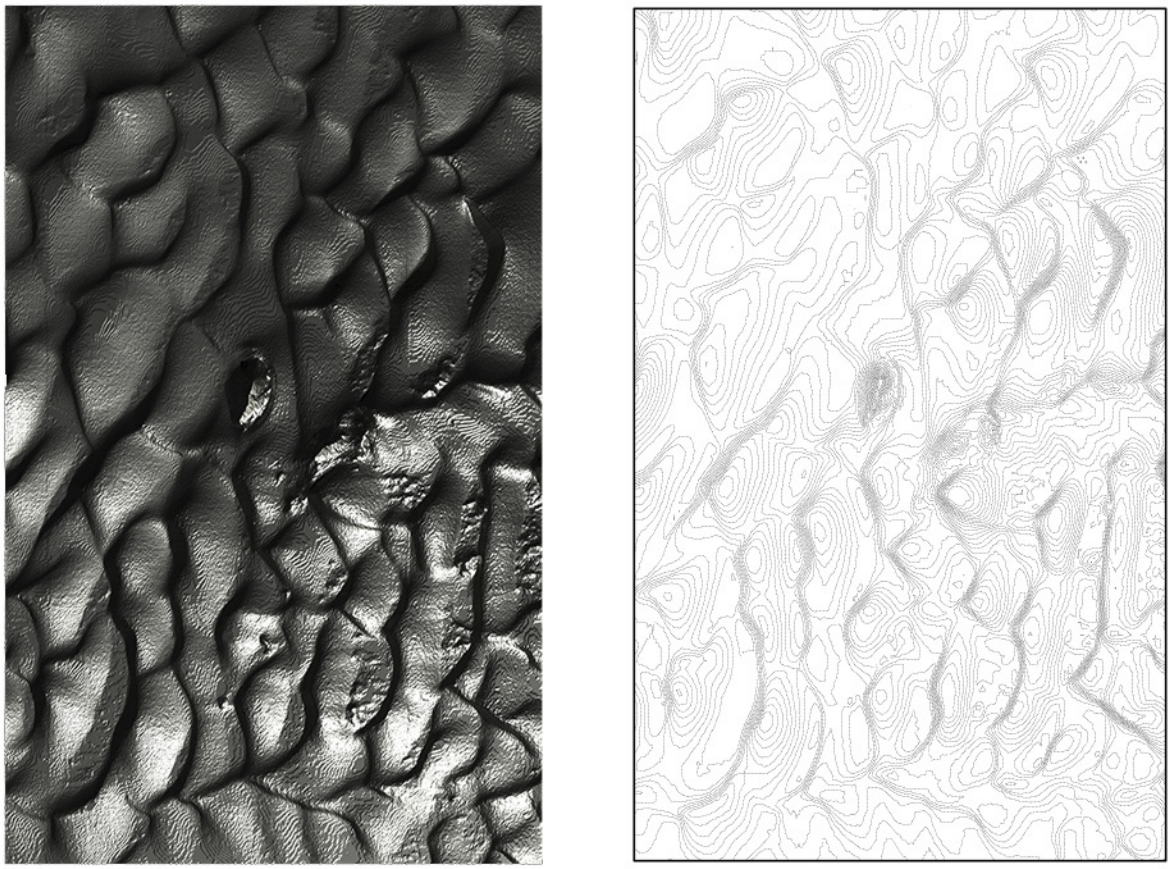

Figure 3: Difference between a displacement mapped landscape and a contoured drawing of the same landscape.

further problematized as designers are forced to contemplate backwards compatibility, which excludes users of outdated software, and legal predicaments of falsely licensed applications, which may have worse repercussions. Knowing that the drawing contains more data than relationships between pixels such as security settings, permissions, and compression methods sheds light on the politics of the drawing at a different level.

As Building Information Modeling software spreads its reach and an increasing number of design "apps" enter the discipline, fewer analogies accurately depict our relationship to these virtual interfaces. For example, if drafting in CAD is akin to drafting by hand, then BIM is like building the building, before you build the building: a simulated act. Not only is it a digital simulation, but it is also dependent on data management. Thus, the process of translating from drawing to building today, depends less on one's ability to form an apt analogy of what a "drawing" is or what it represents, and more on one's expertise in navigating information management systems, and coordinating between file-types from Navisworks, Revit, AutoCAD, Revit MEP, RISA 3D, Autodesk360, etc. But what if instead of learning the actions to perform in each program, design software was regarded as a theoretical concept; a discourse through which software interfaces, workflows, and file-types are dissected to relate to our shared experiences with other cultural media as a whole.[16] This new addition to architectural discussions, dedicated to a broader view of design software, might tease out new methods for translating from drawing to building.

MORE THAN ONE WAY TO SKIN A CAT

Apropos of the above, let us look at a hypothetical scenario. Take a simple topographical survey; a set of information common to both architects and landscape architects, not to mention a key component to a comprehensive building project. A traditional approach to modeling such a survey is to extract contour lines at specific intervals, which results in an abstract, stepped representation of the specific topography. To represent the surface as a smoother continuous surface, one would have to interpolate between these lines using common B-spline modifiers, either "lofting" or "draping" complex doubly-curved surfaces over the contour splines as formwork. The result would be an approximation of a more realistic terrain. However, let's say that the designer had taken a course in the history of computer graphics. She would have most likely

covered early attempts at representing complex textures using displacement mapping algorithms, such as ones used by Pixar's early RenderMan engine. In this process, surfaces are subdivided into triangular meshes whose vertices' height coordinate correspond to a specific distance from an origin managed by a greyscale "heightmap." In other words, instead of interpolating offset contour lines, a displacement map on a mesh surface recreates a topography based on a series of points dictated by pixel grayscale value. On a typical 8-bit RGB image, this allows for up to 256 different height values, resulting in a high-fidelity, realistic terrain [figure 3].

Now, the curious part about this hypothetical scenario is that the concepts remain software-independent. Most modeling software today will facilitate both methods (contouring and displacement) to some extent. However, disciplinary bias separates these ways of working. Architectural design favors the former, contour-based model, and video game/VFX design favors the latter. Tracing the historical lineage of these biases, our student will find that displacement methods were much more computationally demanding, and thus were reserved for industries with large budgets.[17] Architects, using relatively low-cost software in the early days of CAD such as Form*Z and Rhino 3D, would naturally gravitate towards a simpler, faster, and more abstract method for representing topographies. Indeed, many of our disciplinary proclivities for certain methods trickle down from an era where computation was a precious resource to be conserved. As Mario Carpo has recently noted, however, the second digital turn could be understood as a shift from the formal vocabulary of calculus to that of infinite datasets. He suggests that if calculus is a compression tool used to express a complex geometrical order in a simple way, this compression is no longer necessary when processors can add up a large dataset to represent the same figure.[18] Our modeling scenario presents another analogy for this shift: given the level of computing power today, why represent a landscape through approximate curves, when you can recreate topography with a pixel-to-polygon level of resolution?

There are obviously more complex translations at play in the execution of displacing bitmaps into three-dimensional landscapes. For one, heightmaps are only produced in a few ways: (1) by compositing satellite imagery at different points in time to calculate elevation data, or (2) by randomly generating grayscale fields with various bitmap algorithms, such as Perlin noise.[19] Both call for interactions with software outside the canon of Adobe and Autodesk; a daunting task for most students I've taught. Such an endeavor would require one to be fluent in the kinds of file-formats available and be able to translate from one to another with minimal loss of information. But more importantly, our hypothetical designer would have to choose which software combination would yield faster (or cheaper, or more detailed, or lighter) results, instead of forcing a method into a program better suited for another technique.

When I introduce students to Autodesk 3d Studio Max, I usually begin by retelling a short history of modeling software. Typically, this involves an explanation of the OBJ, FBX, DXF/DWG file standards, a short anecdote about early visual effects technologies, a brief mention of Form*Z's influence (as well as Peter Eisenman's influence on Form*Z), and Autodesk's monopolization of the field. After understanding the program's background, we start to familiarize ourselves with the user interface and tools. What quickly emerges through this process are a set of cross-platform terms and common techniques. For instance, if one knows the workflow for producing a texture map in Rhino, then doing so in other modeling programs should be straightforward. But this understanding would not come easily by teaching each program discretely. In order to facilitate this general theoretical knowledge of tools, software should be regarded as a species: each with unique traits, but nevertheless related.

While there are discrepancies throughout this vast landscape of software species, the sheer abundance of them now allows designers more flexibility, particularly when it comes to smaller experimental tasks. Additionally, "app culture" has infiltrated the design field with lighter versions of bigger software as well as open-source programs, in effect democratizing access to design tools. Not only does this allow us to quicken the pace of working, but naturally extends our repertoire of techniques, such as drawing with our fingers on a touchscreen or navigating 3D models on mobile devices. It is therefore inevitable that educators teach not only software as a tool, but software as an extension of our everyday interactions with interfaces. As these interactions continue to evolve, I have an increasing suspicion that themes from gaming, interface, software, and internet studies will keep finding their way into design curricula at large. The need for tutorial-based, step-by-step sequencing of interactions is already dwindling in favor of a more expansive approach where students experiment across a variety of differing media, testing the limits of file-formats, discovering new workflows, and translating their concepts across platforms seamlessly.[20] Educators should engage this shifting mode of operating if we are to come to terms with the truth of the matter: which is that software is the background condition of our reality.

ENDNOTES

1 Mario Carpo, "The Alternative Science of Computation," e-flux Architecture (June 2017) accessed June 30, 2017. http://www.e-flux.com/architecture/artificial-labor/142274/the-alternative-science-of-computation/

2 Robin Evans, "Translations from Drawing to Building," *AA Files*, no. 12. (London: Architecture Association Press, 1986).

3 Lev Manovich, *Software Takes Command* (London: Bloomsbury Academic, 2013), 152.

4 Ellie Abrons, “Becoming Digital” (syllabus, University of Michigan, Ann Arbor, MI, 2017).

5 Ibid.

6 Abrons’ project, “Becoming Digital” asks students to critically engage computation as a “background condition of our reality.” See also John May’s didactic look at digital images in “Everything is Already an Image” in *Log* 40, ed. Cynthia Davidson (New York: Anyone Corporation, 2017).

7 Curtis Roth has described how Evans’ thesis suggests a paradox of the origins of architectural thought. See “On Our Dark Products” in *Some Dark Products: A Travelogue of Nine Instruments for Architecture* (Stuttgart: Edition Solitude Press, 2017).

8 Evans, “Translations…” 14

9 Mario Carpo, “ACADIA 2016 Keynote,” Vimeo video, 55:29. Posted [March 2017], https://vimeo.com/210622365

10 Keller Easterling, “Another Part of Speech,” in *Dispute Plan to Prevent Future Luxury Constitution*, ed. Benjamin H. Bratton (Berlin: Sternberg Press, 2015).

11 Manovich, “Introduction,” in *Software Takes Command*. 20.

12 Manovich provides a rudimentary list of this category, but claims it will continue to evolve. Ibid. 23

13 Roth has developed a syllabus for introducing some of these themes in architecture school. See “A Syllabus for Yung Distance,” in S*ome Dark Products*.

14 An excellent example would be the development of Form*Z at The Ohio State University, where architects and educators worked together to develop a comprehensive 3d modeling program. Peter Eisenman also famously contributed to the design of the program. See Pierluigi Serraino, *History of Form*Z* (Basel: Birkhauser, 2002), 23.

15 Thought both file types describe geometry, STL is optimized for stereolithography meaning it organizes the geometry in layers which can be read by a stereolithography machine, while OBJ organizes the geometry primarily for viewing in 3D graphics engines, and is thus more complex.

16 This is already happening within the Digital Humanities. Manovich is part of a group of editors at the MIT Press who are publishing a collection of books under the subject of “Software Studies.” See https://mitpress.mit.edu/books/series/software-studies.

17 In fact, The Ohio State University used to have a course within the Advanced Computing Center for the Arts and Design, which covered such topics. See Wayne Carlson, “A Critical History of Computer Graphics and Animation,” (syllabus, The Ohio State University, Columbus, OH, 2007) https://excelsior.asc.ohio-state.edu/~carlson/history/.

18 Carpo, “ACADIA 2016 Keynote.”

19 19. Michelle Chang has recently covered a quick history of the Perlin noise algorithm, suggesting that some scholars are starting to historicize these moments. See “Turning a Banana Inside Out,” in *PLAT* 6.0 Absence, ed. Melis Uğurlu (Houston: Rice School of Architecture, 2017).

20 Luke Pearson encourages his students to analyze video game environments and their critical potential. See “Designing by Decoding: Exposing Environments Mediated by ‘Cultural Software’.” *Journal of Architectural Education* 71:2 (2017), 197-210. https://doi.org/10.1080/10464883.2017.1340773