AAD

Algorithms-Aided Design

From traditional drawings to the parametric diagram

"Architects do not make buildings, they make **drawings** of buildings".

Robin Evans

Architects have always drawn before building, an act that diferentiates architecture from the mere construction. Drawings have been the architects medium to organize ideas, resources, space, etc. and represent the architects' faculty to predict design outcomes. As methods of representation have evolved, new styles have emerged. Tools such as perspective in the Renaissance and projective geometry in Modernism have marked leaps forward in design. However, these tools have been dependant on a stable set of instruments for centuries: paper, drawing utensils, ruler and the compass. In this model each creative act is translated into a geometric alphabet by gestures which establish a direct link between the idea and the sign.

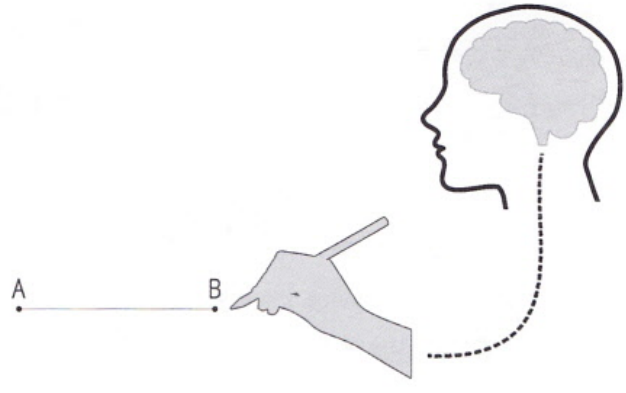


FIGURE 0.1

The act of drawing is a natural gesture when executed using traditional drawing tools which establish a direct link between ideas and signs. A natural interaction is characteristic of those tools which can be considered as a *hand mould*.

An additive process

The traditional drawing is an **additive process**, in which complexity is achieved by the addition and overlap of independent signs traced on paper. No **associative relations** can be managed. The **internal consistency** of a drawing is not guaranteed by the medium, but is entrusted to the designer. As follows, the drawing is not a **smart medium**, but rather, a **code** based on standards and conventions.

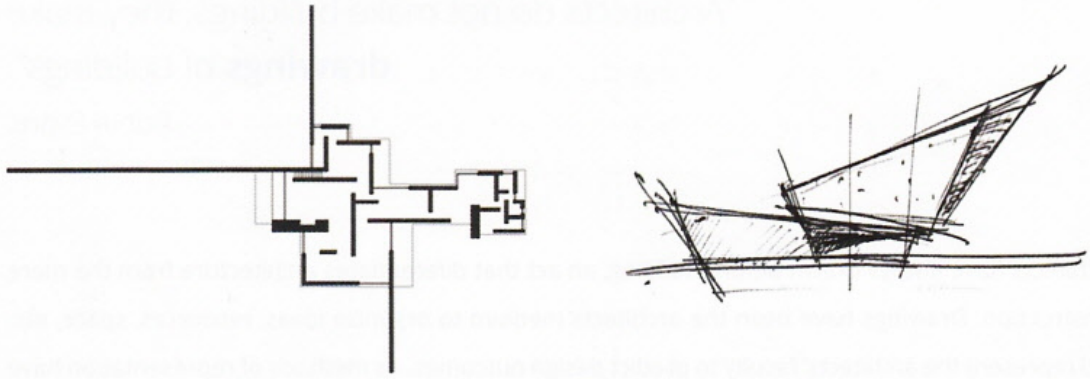


FIGURE 0.2
The traditional drawing is based on adding and overlapping independent signs on a paper. The meaning and the overall consistency of these signs is entrusted to the designers and is based on conventions. The drawing by Mies van der Rohe (on the left) is a "plan" while the sketch on the right is a "draft", nevertheless both are (ontologically) just ***signs on a paper***.

The additive logic of the traditional drawing implies two limits: first, the act of drawing differs from cognitive mechanisms underlying the creative process, which works by establishing interrelations rather than adding information. Second, the drawing process excludes physically relevant aspects that in the real world drive the generation of forms. For example, the traditional drawing cannot manage **forces** (such as gravity) and constraints which affect and restrict deformations and displacements. These limits have restricted the exploitation of the drawing and designers have been forced to reiterate definitive tectonic systems rather than innovating. Initially these limits were not overcome by the computer; CAD software simply improved the ability to perform repetitive tasks without affecting the method of design. Similar to traditional drawing, CAD entrusted the designer to determine the overall consistency by adding digital signs or geometric primitives on a digital sheet/space and controlling **CAD *layers*; this method can be seen as the translation of the additive logic within the digital realm.**

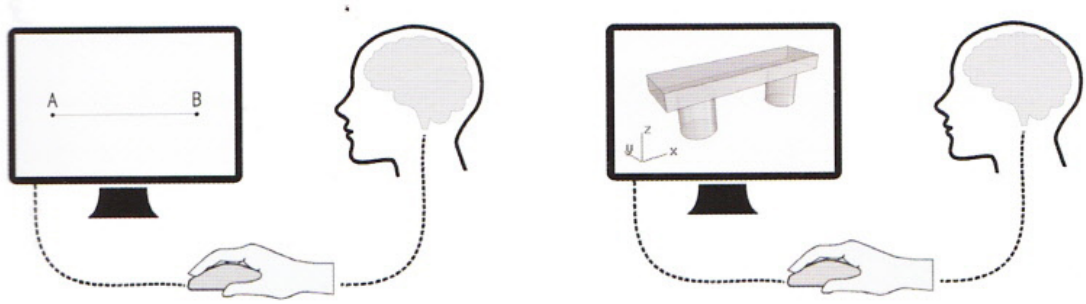


FIGURE 0.3
The mouse is still an extension of the brain. It simulates the "presence" of the hand in the digital environment.

From 60's, the architecture avant-garde tried to "force" drawing's limits using several methods to represent forces and processes that drive the generative process. For example, Eisenman's diagram for House IV impressed the entire sequence of geometric operations that led to the final object.

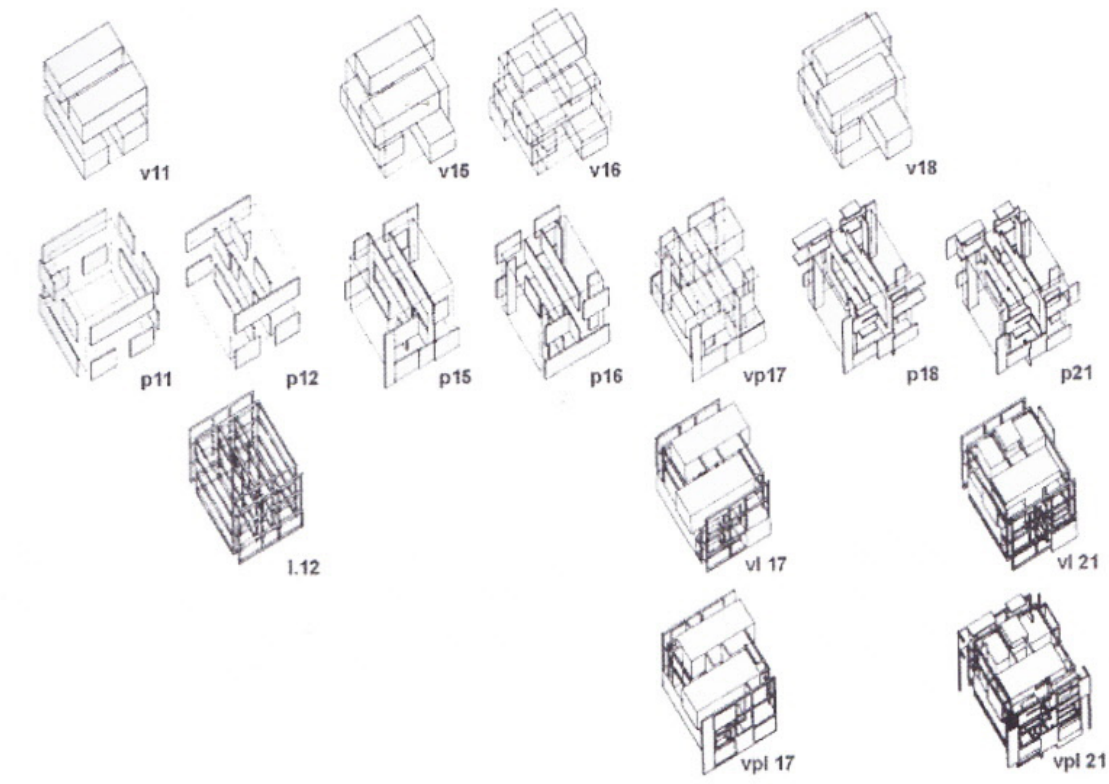


FIGURE 0.4
Peter Eisenman, House IV, Falls Village, Connecticut, 1971.

From conventional drawing to the analogue (smart) apparatus

Despite the limitations, drawings have been the stable medium of architecture over the centuries and this was possible as the architects have relied on **typology**, i.e. the use of well proven, preconceived solutions and tectonic systems. Typology made the drawing not only a communication medium but a system that enabled designers to explore and refine variations (**form-making** approach) within a specific set of formal and structural constraints.

The conventional drawing was first attacked by a new approach, the **form-finding** – emerged in architecture in late 19th century – which aimed to investigate novel and optimized structures found through complex and associative relations between materials, shape and structures.

Pioneers like Gaudi (1852-1926), Isler (1926-2009), Otto (1925-) and Musmeci (1926-1981) have rejected typology and looked to self-formation processes in nature as a way to organize buildings. Since the form could not descend from proven solutions, the traditional drawing could not be used as a tool to predict design outcomes.

FIGURE 0.5
Heinz Isler. Service Station in Deitingen, Solothurn, Switzerland (1968). Image by David P. Billington.

For this reason form-finding pioneers relied on physical models such as: soap films which found minimal surfaces, and suspended fabric which found compression-only vaults and branched structures. In other words, the drawing as a medium to investigate form was replaced with physical form finding relying on analogue devices which demonstrated how dynamic forces could mold new self-optimized architectural forms.

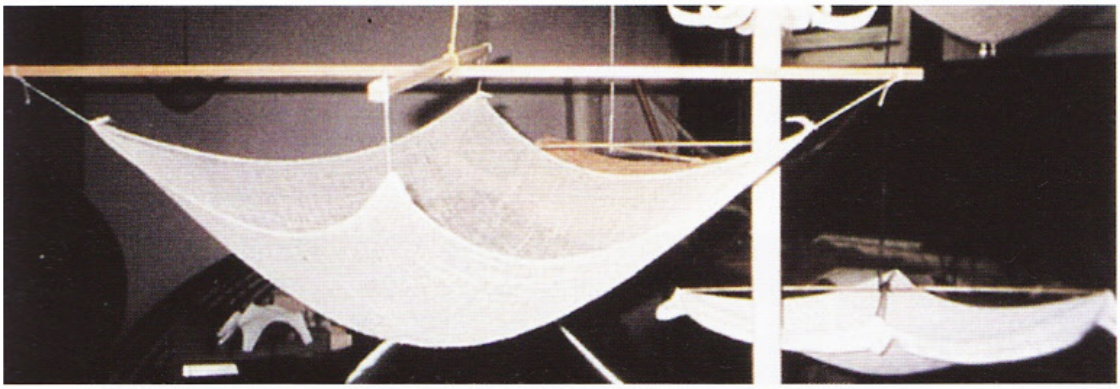


FIGURE 0.6
Forces and forms are correlated.

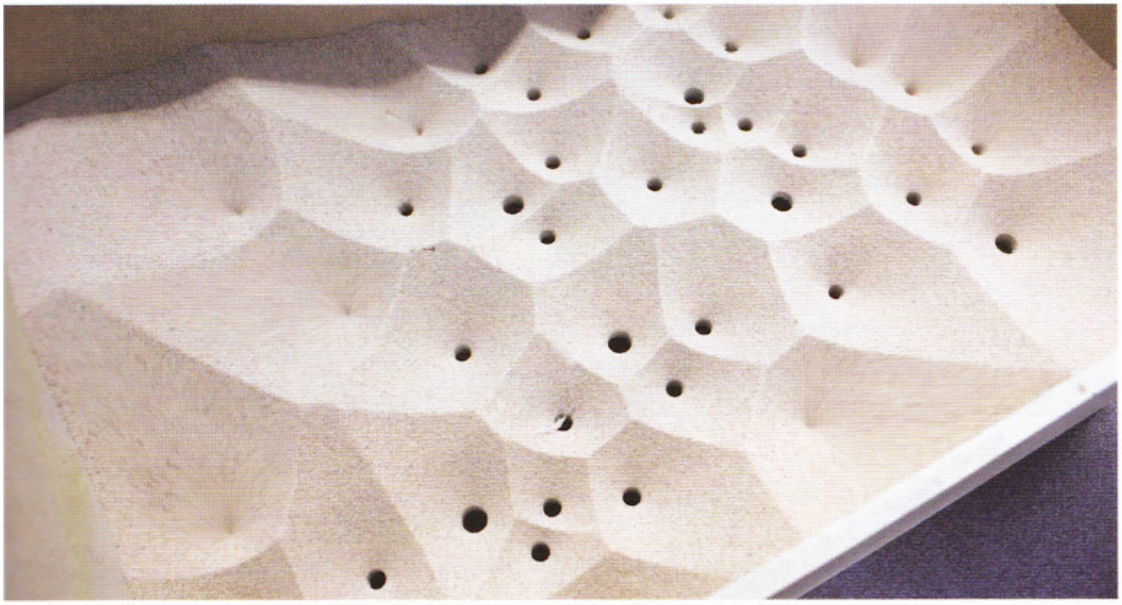


FIGURE 0.7
"Any granular material falling from a fixed point forms a cone on the surface below and a funnel within the granulate mass with the same angle of inclination, the natural angle of repose, 35 degree." Frei Otto, 1972. Sand Experiment Inspired by Frei Otto - WeWantToLearn.net, studio ran by Toby Burgess and Arthur Mamou-Mani at the University of Westminster, student: Jack Munro.

Over the last decades the increasing complexity of buildings has made ***form-finding*** an important strategy in determining the shape and form of indeterminate structures. Structural optimization through physical modeling was **mono-parametric** (gravity based) and marked a trajectory towards **multi-parametric** form-finding which aims to interact with heterogeneous data: geometry, dynamic forces, environment, social data.

Parameters: from additive to associative logic

Luigi Moretti, the Italian architect, invented the definition for "**Parametric Architecture**" in 1939. His research on "*the relations between the dimensions dependent upon various parameters*" culminated in an innovative exhibition of his models of stadiums for soccer, tennis and swimming at the 1960 Twelfth Milan Triennial. Moretti's design parameters were linked to viewing angles and economic feasibility in these projects: the final shape was generated by calculating pseudo isocurves, that attempted to optimize views from every position in the stadium.

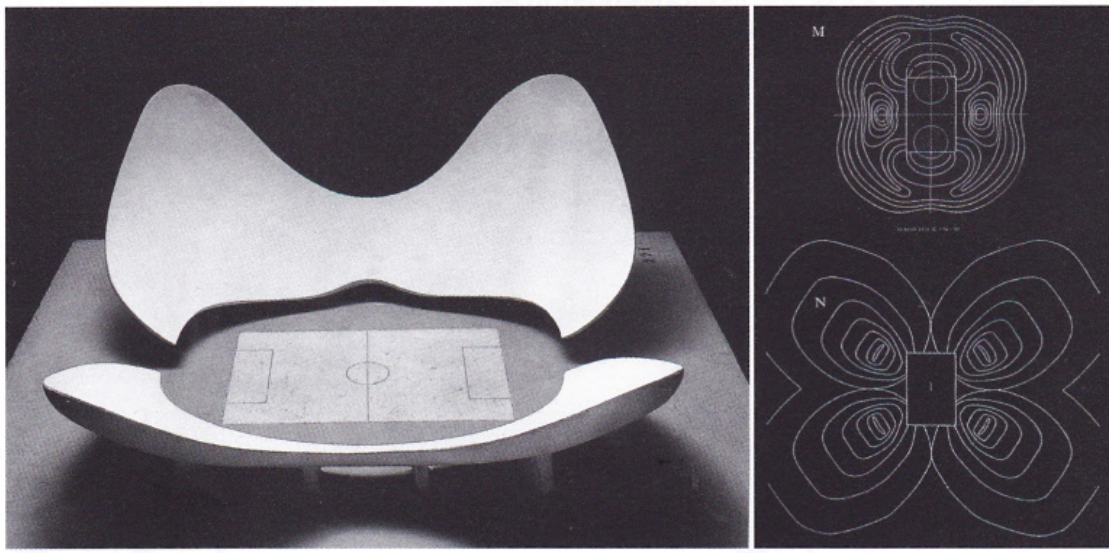


FIGURE 0.8
"Architettura Parametrica" research. Milan Triennale exhibition, 1960. Solution for a soccer stadium and diagrams drawn to generate the geometry.

Moretti's research was a collaboration with the mathematician Bruno De Finetti, wherewith he founded the Institute for Mathematical Research In Architecture (I.R.M.O.U.). Moretti said:
"The parameters and their interrelationships become [...] the code of the new architectural language, the "structure" in the original sense of the word [...]. The setting of parameters and their relation must be supported by the techniques and tools offered by the most current sciences, in particular by logics, mathematics [...] and ***computers****. Computers give the possibility to express parameters and their relations through a set of (self-correcting) routines".*[1]

NOTE 1

F. Bucci and M. Mulazzani, *Luigi Moretti opere e scritti* (Milano: Electa, 2006), 204–208.

It is evident from this quote, that Moretti immediately understood the potentials of the computer applied to design. Following Moretti, the first application for design utilizing the computer occurred in 1963. The American computer scientist Ivan Sutherland developed the ***Sketchpad***, defined as "A Machine Graphical Communication System," creating the first interactive Computer-Aided Design (CAD) program.

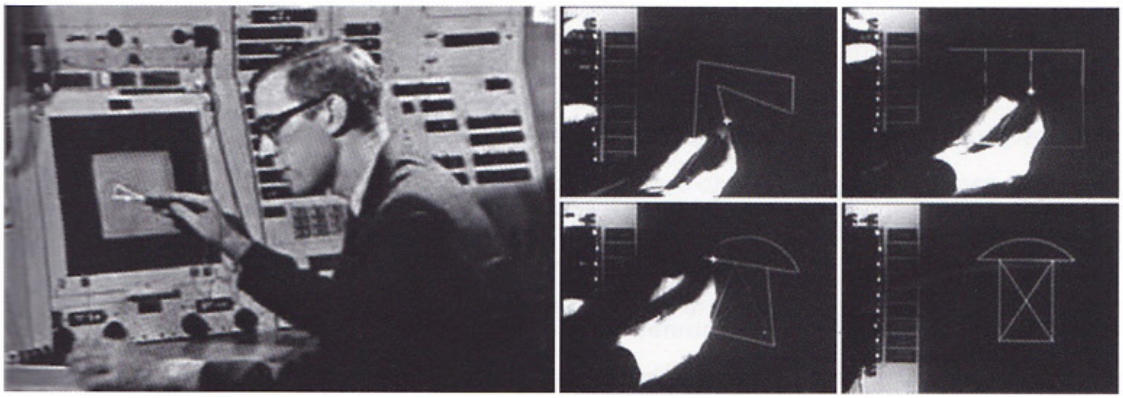


FIGURE 0.9
Ivan Sutherland on MIT Lincoln Labs' TX-2 computer (1963). The *Sketchpad* interface.

Considered as one of the most influential computer programs ever written, the sketchpad was designed to test human-computer interaction and allowed designers to draw basic primitives such as: points, lines and arcs, using a light-pen for input. The program featured many of the now typical CAD operations such as: blocks managing, zoom and snaps. Moreover, it was based on an advanced **associative logic**, the so called *atomic constraint*.

It was an innovative feature which facilitated links between objects; for example, if two lines (or more appropriately two vectors) were drawn starting from the same point A, every movement of A implied change in magnitude and direction of the lines. Constraints such as points could be combined to generate relationships between objects, overcoming the limits of the **additive logic** of traditional drawings.

The introduction of the computer to design by Moretti, and the graphical interface of Sutherland marked a revolution in architectural design techniques and moreover it upgraded the architects tools. However, the innovations brought by early CAD programs were not immediately embraced by commercial software for almost three decades.

For instance, the associative capabilities introduced by the Sketchpad were not embedded in commercially successful software, such as Autocad (1982). Autocad met the architects need to speed up repetitive tasks and manage multiple drawing layers, by in effect, digitalizing the drawing board.

The next important step forward occurred in 1987, with the introduction of Pro/ENGINEER® software,

developed by Samuel Geisberg for mechanical system design. The program allowed users to associate tridimensional parametric components which were controlled by user input constraints. For example, it was possible to create a link between a rivet and the relative hole. The user changing the rivet input size implied a propagation of modifiers which updated the tridimensional model as well as the bidimensional output.

Pro/ENGINEER reduced the cost of making design changes, and overcame the rigid constraints of tridimensional modeling.

The most profound progress has happened from the late 1980's to present day. Academic research and avant-garde practices – trying to escape simple **editing** limitations of software applications – explored new ways to manipulate software "from the inside" aiming to find unexplored solutions and forms through **programming**. Many designers soon realized that more sofisticated programs could manage **complexity beyond human capabilities** by structuring routines and procedures.

This type of modeling relies on programming languages which express instructions in a form that can be executed by the computer through a step-by-step procedure: the **algorithm**.

Algorithmic modeling

What is an algorithm? An algorithm[2] is a procedure used to return a solution to a question – or to perform a particular task – through a finite list of basic and well-defined instructions. Algorithms follow the human aptitude to split a problem into a set of simple steps that can be easily computed, and although they are strongly associated with the computer, algorithms could be defined independently from programming languages. For example, a recipe can be considered as something similar to an algorithm. We can set a procedure for cooking a chocolate cake, based on a simple list of instructions:

0. Mix ingredients;
1. Spread in Pan;
2. Bake the cake in the oven;
3. Remove the cake from oven;
4. Cool.

Nevertheless, such a procedure cannot be properly considered an algorithm since the instructions

NOTE 2

The term "Algorithm" is named after the 9th century Persian mathematician Al-Khwarizmi.

are far from being well-defined and contain ambiguities: "mix ingredients" but which ingredients? How long should the cake cook? This basic example points out some important properties of algorithms:

- *An algorithm is an **unambiguous** set of properly defined instructions.*
  Algorithms depend on entered instructions. The result will be incorrect if the algorithm is not properly defined. Put another way, if steps in the cake are inverted or skipped, the chances of a successful cake diminish.

- *An algorithm expects a defined set of input.*
  Input can be different for type and quantity. The step {0} requires ingredients, the step {2} requires quantitative information such as baking temperature and time. Moreover, each input has a precondition, e.g. a requirement which must be met, such as a range of baking temperatures, for example: 160°C – 200°C.

- *An algorithm generates a well defined output.*

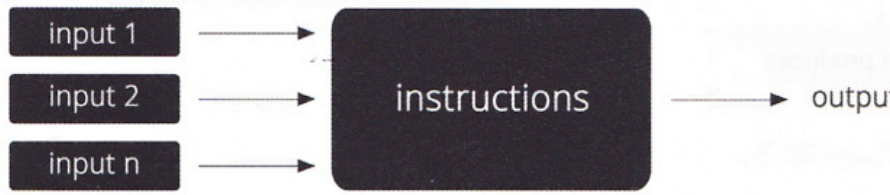


FIGURE 0.10
Schematic representation of an algorithm.

If an ambiguos recipe leads to an inedible cake, in the digital realm:

- *An algorithm can produce error messages and warnings within the specific editor.*
  Input is specific. If preconditions are not met, e.g. numbers are imputted instead of text, the algorithm will return an error.

Although algorithms are often studied abstractly they harness the potential of the computer which has the capacity to perform tasks according to a set of instructions. When algorithmic calculations are executed by a computer, a specific **editor** is used to type instructions. Editors can be **standalone** applications or embedded in a software application. For example, standalone editors include C#, Python etc. and embedded editors are script editors provided by programs such as Rhinoceros and Autocad that allow users to write instructions to automate tasks.
Algorithms consist of different classes, an algorithm class which leads to a number is called a

computation procedure, while an algorithm that generates a *yes* or *no* is called a *decision procedure*.

Algorithms can also lead to geometries. For instance, if an integrated editor is used within CAD or another modeling software, a 3D geometry is created by manipulating the standard set of primitives provided by the software or procedurally defined by a sequence of instructions. For instance, a line can be defined by two points, a start and an end; points in turn can be defined by their coordinates {x,y,z}. For example, a vase model can be defined as a revolution of a profile curve around an axis, and more complex objects can be obtained by establishing a set of rules.

Objects are no longer manipulated with a mouse, instead they are defined by procedures expressed in a specific program language: AutoLisp® in Autocad®, RhinoScript® in Rhinoceros®, MEL® in Maya® or other cross platform languages such as Python®.

Such an approach – usually referred to as **scripting** – is completely new for designers and transforms the link between the idea and the final output.

Scripting consists of two working environments:

- the editor (A);
- the 3D modeling environment (B).

Moreover it produces **two outputs**:

- the algorithm;
- the output of the algorithm, constituted by **associative** 3D or 2D geometry.

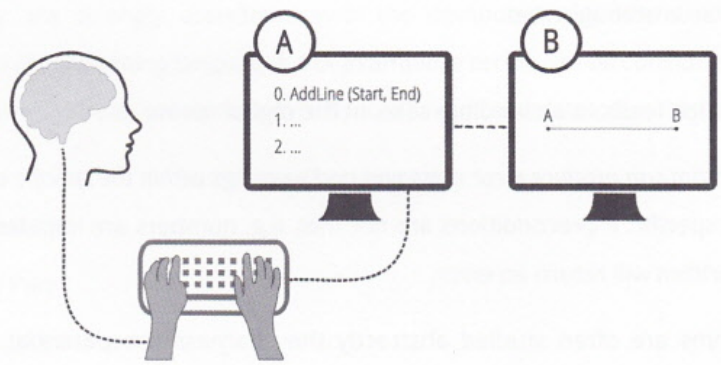


FIGURE 0.11
The algorithmic modeling based on scripting consists of two main "windows": the editor and the 3D modeling environment.

The final output is not just a "digital sign" but it can be considered as an interactive digital model responding to variations in the input by manipulating the entire system. For example, if the points

coordinates are changed from {x,y,z} to {x^1,y^1,z^1} of the mentioned line, the algorithm maintains the established relationship that the line is defined by the two points not their location. Algorithms establish associative relations between different entities such as numbers, geometric primitives and data. For example, complex geometries can be defined by an unambiguous sequence of instructions which drive interrelations. **Algorithmic design enables users to design a process rather than just a single object.**

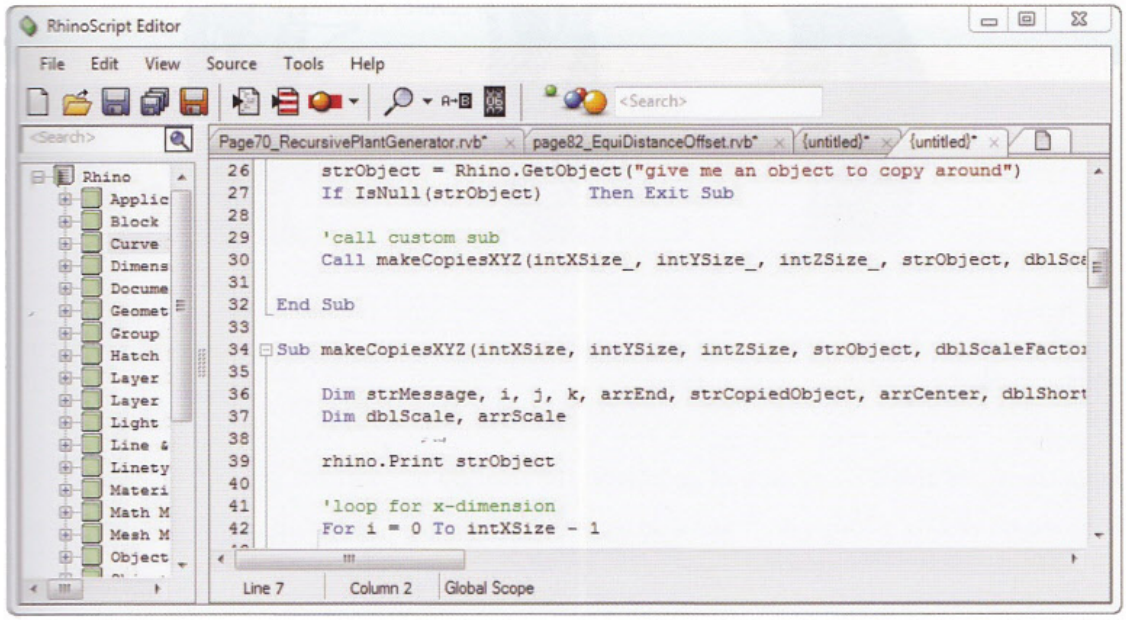


FIGURE 0.12
The RhinoScript editor.

Bruce Mau in his 1998 *Incomplete Manifesto for Growth*, states that a "*process is more important than outcome. When the outcome drives the process we will only ever go to where we've already been. If process drives outcome we may not know where we're going, but we will know we want to be there*". Mau's quote summarizes the core concept of algorithmic design; the potential to generate and control design-complexity beyond human capabilities. A set of well defined associative rules and constraints can lead to unprecedented shapes or unpredictable results that are coherent with the parameters established. Algorithmic design allows designers to find new solutions and step beyond the limitations of traditional CAD software and 3D modelers.

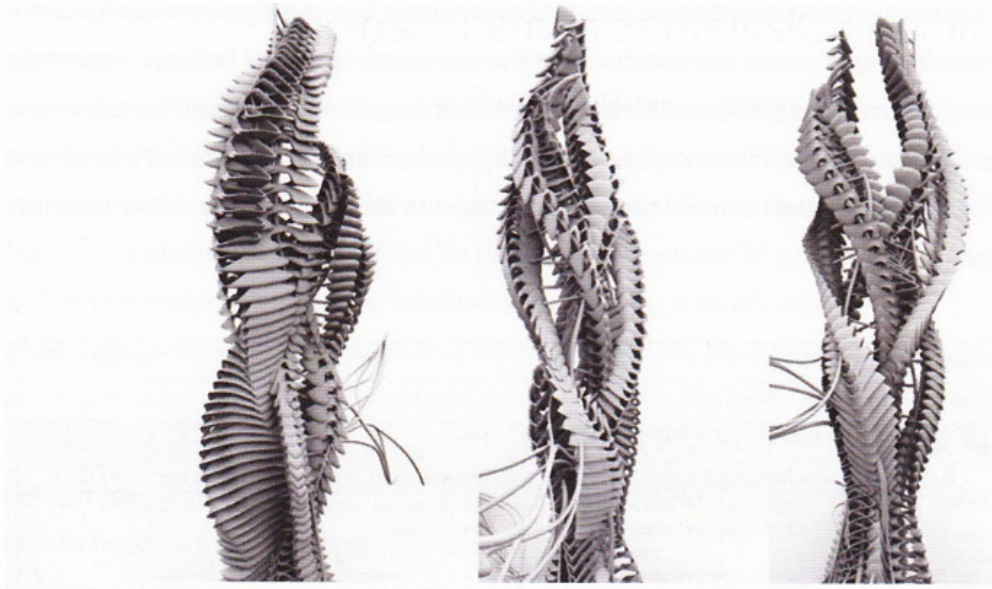
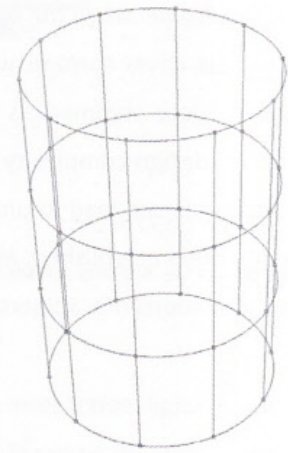


FIGURE 0.13
Parametric Urbanism - Dominiki Dadatsi, Fountoulaki Elrini, Pavlidou Eleni. The image shows three different configurations of an algorithmic-designed tower.

Algorithms can define every type of geometry. The method to construct geometries procedurally is based on writing a rough draft and translate it into a programming language. For example, the image below can be sketched by writing the following list of instructions:

0. Draw four circles;
1. Subdivide the four circles into N parts; we get N points for each circle;
2. Connect the corresponding points.

The same element could be defined by different parameters, but it is natural to write an algorithm in a way that establishes relations between the variable parts of an object. In the example the number of lines is affected by the number of subdivisions (N), which is the main parameter. Nevertheless, the algorithm is still ambiguous since it does not specify unique origins with respect to the z coordinate: $\{x,y,z^1\}$, $\{x,y,z^2\}$ and $\{x,y,z^3\}$, a radius for each circle (r^1), (r^2) and (r^3), and the method of connecting the lines. These refinements are made when transitioning from the rough draft to the final algorithm.



The Parametric Diagram as a smart medium

In recent years many software houses have developed **visual** tools in order to make scripting more accessible to users with little to no programming skills. In effect, associative rules and dependencies can be expressed using a graphical method based on **node diagrams**.

Sutherland's *Sketchpad* represented all the constraints defined during the drawing process. Through a special diagram – a *flow chart* – the user could not only visualize the tree of dependencies but could manipulate the graph with instant effects on the drawing.

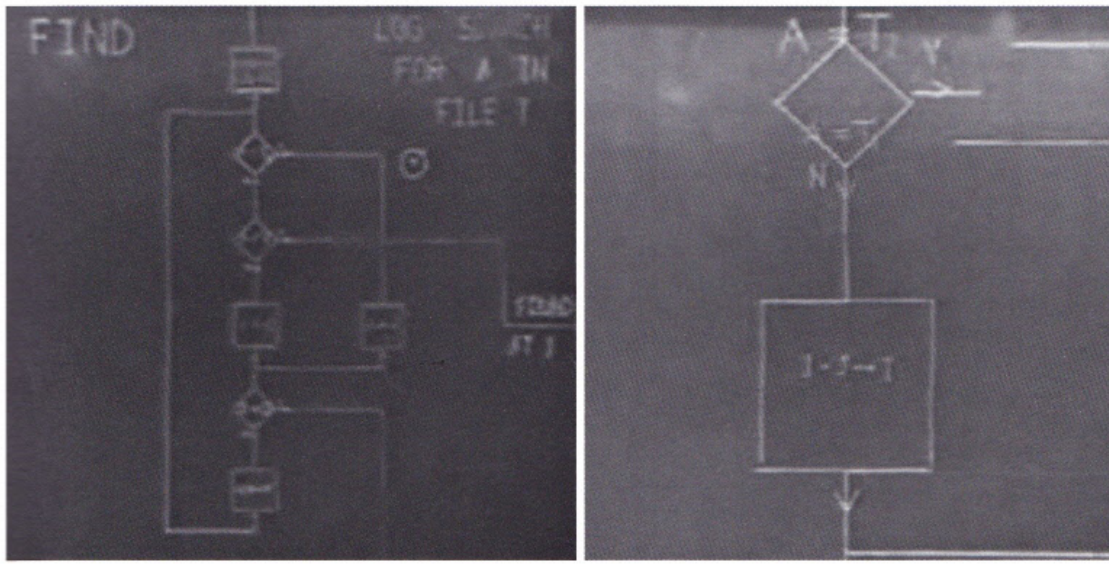
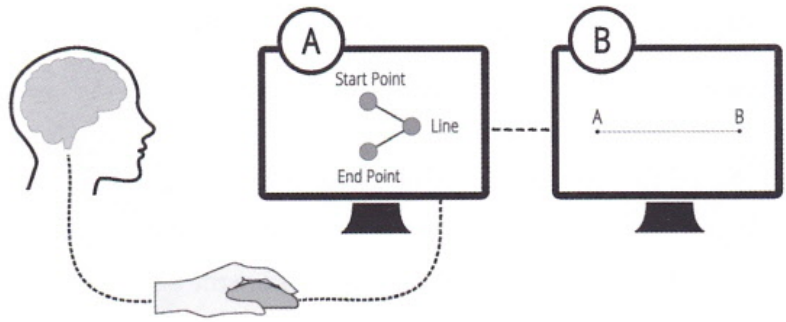


FIGURE 0.14
The Sketchpad (1963) provided a graphic visualization of design-constraints through a diagram called *flow chart*.

Many software have enabled users to interact with digital objects either by a direct manipulation and via node-based diagrams. Node based software systems such as Generative Components® by Bentley Systems and Grasshopper® by Robert McNeel & Associates, are two softwares that enable users to build up complex geometries by associating parametric primitives. Visual scripting makes possible a process were a line can be built by connecting two point objects, a square by connecting four line objects etc.

"In principle any conceivable network of relations between a given set of element attributes can be constructed".[3]

NOTE 3
P. Schumacher, *The Autopoiesis of Architecture, A New Framework for Architecture*, (John Wiley & Sons, 2010), vol. I, p. 353.



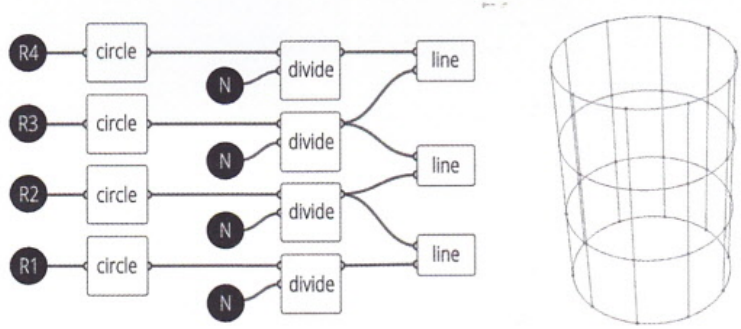
Similar to scripting, visual scripting is based on two main working environments:

- the Visual editor (A);
- the 3D modeling environment (B).

Such a process generates **two outputs**:

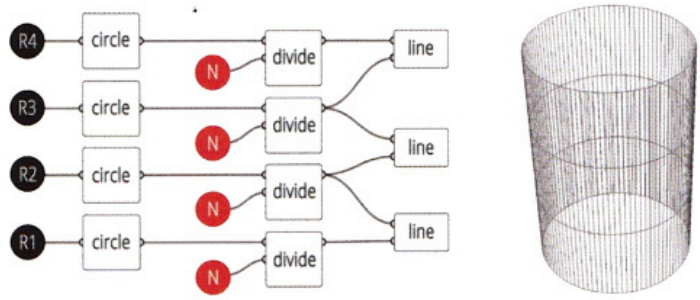
- the node diagram, also called **parametric diagram** or visual algorithm;
- the output of the parametric diagram constituted by parametric 3D or 2D geometry.

Node diagrams can be used to create geometries. For example, the following figure is the visual transposition of the algorithm "drafted" at page 26.

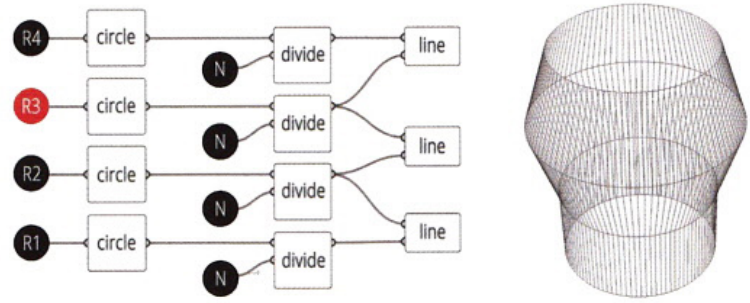


The diagram consists of nodes and connections. Square nodes are the main functions: draw a circle, divide a circle, create a line. The circular nodes are the parameters: the radius of each circle, and the number of subdivisions. The diagram's output is the same geometry generated through the step-by-step procedure shown previously.

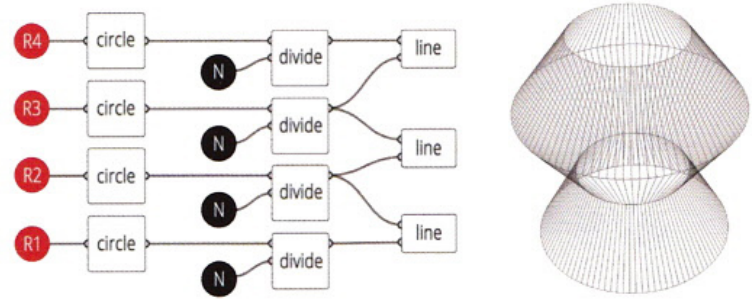
The advantage of a node diagram resides in the intuitive logic which allows you to quickly interact with parameters. For example, if the N parameter is modified, more lines are generated.



The geometry can be further modified by manipulating one of the (R) parameters, the radius of (R3) is increased in figure.



The geometry can be once again be modified by manipulating (R1, R2, R3, R4) as shown in figure.



The parametric diagram has the potential to create associative models that explore multiple configurations through control of the input parameters.

Patrick Schumacher is quoted:

"While the attributes of the graphic/digital primitives [...] are fully determined and fixed at any time, within the parametric diagram they remain variable. This variability might be constrained within a defined range

on the basis of associative functions that imbue the diagrammatic process with an in-built intelligence"[4]. The parametric diagram can be considered a smart medium for architecture and design, since it provides an internal self-consistency transposed in a graphic language which can be easily manipulated, enabling designers to explore form-finding and form-making strategies.

Jerry Laiserin is quoted:

"Form-making, loosely defined, is a process of inspiration and refinement (form precedes analysis of programmatic influences and design constraints) versus form-finding as (loosely) a process of discovery and editing (form emerges from analysis). Extreme form-making is not architecture but sculpture [...]. Extreme form-finding also is not architecture but applied engineering, where form exclusively determined by function".[5]

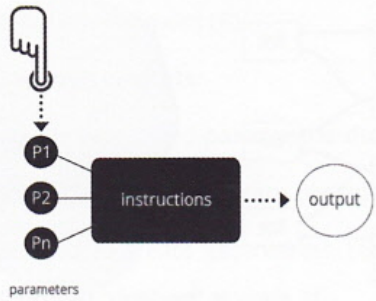


FIGURE 0.15
Conceptual representation of the form-making approach.

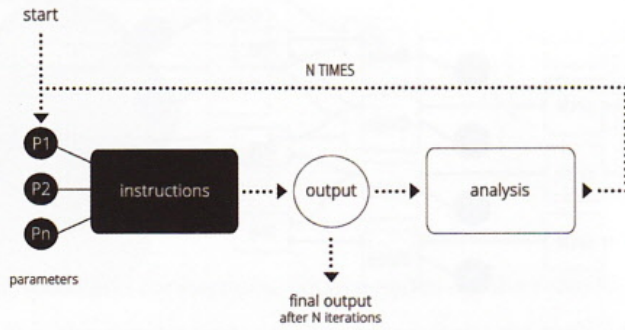


FIGURE 0.16
Conceptual representation of the form-finding approach.

NOTE 4

P. Schumacher, *The Autopoiesis of Architecture, A New Framework for Architecture* (John Wiley & Sons, 2010), vol. I, p. 352.

NOTE 5

J. Laiserin, 2008, *Digital Environments for Early Design: Form-Making versus Form-Finding*. First International Conference on Critical Digital: What Matters(s)? - 18-19 April 2008, Harvard University Graduate School of Design, Cambridge (USA), pp. 235-242.