CADERNOS
PROARQ 18

MARIA GABRIELA CELANI E CARLOS EDUARDO VAZ

CAD scripting and visual parametric modeling environments: a comparison from a pedagogical point of view

Maria Gabriela Celani Architect and MArch from the School of Architecture and Urbanism of the University of São Paulo (FAU-USP), PhD from Massachusetts Institute of Technology (MIT), lecturer at the University of Campinas (Unicamp) and Post-doctorate from the Technical University of Lisbon. She is a researcher and lecturer at the Architecture and Urbanism Faculty at Unicamp, where coordinates the Laboratory of Automation and Prototyping for Architecture and Construction (LAPAC) and the research group 'Contemporary Theories and Technologies Applied to Design'. gabi.celani @ gmail.com

Carlos Eduardo Vaz Graduated in Architecture from the University of São Paulo (2003) and finished his MA and PhD in Civil Engineering from the State University of Campinas (UNICAMP). He is currently an associate professor at the Federal University of Pernambuco, working in the Department of Graphic Expression. cevv00@gmail.com

ABSTRACT

This paper compares the use of visual parametric modeling environments and CAD scripting languages for teaching computational design concepts to novice architecture students. Both systems are described and categorized in terms of the representation method they use. An analogy between visual parametric modeling environments and visual programming languages is proposed and the literature about VPL's is reviewed. The comparison is also based on the practical results of a course in which a scripting language and a parametric modeling environment were used for a parametric design exercise. In the second case the learning curve was steeper, the designs developed were more complex, and students developed a better understanding of generative design concepts and the use of computer tools for design exploration. Although we must take into account the characteristics of the specific tools used in this comparison, it is possible to conclude that the use of visual parametric modeling environments can help introducing computational concepts to novice architecture students with no background in programming, preparing the ground for the introduction of more abstract methods. The paper also discusses the importance of the ability to shift between different representation methods, from the more concrete to the more abstract, as part of the architectural education.

Keywords: Script Languages. Parametric modeling. Representation method. Teaching.

Introduction

The insertion of new technologies in the architectural education has been a subject targeted by long discussions and publications, as well as the theme of many conferences, especially the ones starting in the 1980s, when computers became more accessible both for students and universities. The central question of those debates progressively evolved from the introduction of simple representation software to programming that can be fully integrated to the design process, allowing the generation and exploration of solutions (Mark, Martens & Oxman, 2001). The interest in generating software is becoming more evident after the beginning of digital manufacturing machines use in the architectural field, promoting the creation of free forms in such a way not possible before that.

Generative design systems were described by Mitchell (1975) in his article *The theoretical foundations of computer-aided architectural design,* as devices that were capable of generating potential solutions for a design problem. The most important strategies of the generative design are: combinations, substitutions, parameterization, context restrictions, contingency, emergence, optimization, and the combination of two or more of them (Celani, 2008).

Generative systems, conversely to the common sense, do not need to be necessarily implemented in a computer, but may be used to accomplish repetitive tasks that would consume much time otherwise. Before the 1980s, these systems were hardly implemented in computers, not only because highly specialized skills in programming were necessary, but also for the high costs of hardware that should have an appropriate graphic interface and a high memory capacity.

Mitchell, Ligget and Kvan (1987) prepared one of the first guides for the implementation of computer aided-design systems. Their book *The art of computer-graphics programming*, which suggests Pascal as the programming language for the routines development. Schmitt (1988), Coates and Thum (1995), Celani (2003) and Teridis (2006) organized other computer-aided design guides. Each of these books proposes different generative strategies. All of them are based in the existing CAD software script languages and take advantage of the geometric functions present in these software. Their main objective is to introduce computational concepts to undergraduate and graduate students in architecture (Celani, 2008).

Nevertheless, subjects with this type of content are rarely mandatory in the Architecture school, and it is possible to affirm that few architecture students are ready to accept the challenge of learning a programming language. The reasons for this difficulty are yet to be proved, but it is possible to speculate that students believe that programming is complicated, this activity is not related to the professional practice, and it is not the type of knowledge an Architect must have.

Even when Architecture students learn a programming language, most of them will not use this skill to express his design ideas. This happens, in the first place, because in order to develop a computer program a certain experience is necessary – it is a time consuming activity. Secondly, sketches are still considered the main – if not the only one – method for producing the preliminary design ideas.

It is impossible to deny the drawing role in the creative process, a theme that is largely studied by many scholars (ROBBINS, 1997). Therefore, if we consider the design process as an activity that involves the combination of distinct possibilities, the computer may assume an important role by allowing the systematic and exhaustive generation of design alternatives.

Recently, some CAD packages introduced tools with parametric models generative capabilities that do not demand to its users a symbolic code elaboration. These programs use visual programming to create diagrams that will represent the algorithm to generate the parametric model. Normally, this type of software allows a limited automation of the design project, but it can be efficient to explore forms, by means of automatic generating parametric variations. The majority of these programming environments allow the creation of algorithms without the use of a symbolic code, and their capabilities can be extended through the use of scripts.

Objectives and Methods

The objective of this research is to assess the use of tools that are founded in different computing paradigms: text programming languages and visual programming languages. In this sense, programming concepts were introduced to students and, from that point, they had to develop exercises using two different tools: algorithm editor Grasshopper and script language VBA. Along their tasks, students had to prepare algorithms that were capable to insert components as *lines* and *sides or* to create complex forms. These compositions had to be modeled indirectly by the students. Therefore, they should either create codes or structure a symbolic diagram to automatically insert or enable changes in components instanced by students.

In order to make this research feasible, the student's works from classes of different years that had the subject "CAD in the Creative Process" are compared. Works that were developed through the use of visual parametric modeling environments (Grasshopper) belong only to the group that was in the 2010 class. The other works presented belong to previous years' classes that were still using the script VBA language.

The criteria used to analyze the results were based on the student's ability to assimilate programming concepts and forms generation. Despite the fact

that students learned the same contents, when utilizing the new CAD tool they developed tasks in which they were able to respond to a design problem. The result was completely different in relation to previous years, when, in most cases, students could only instance lines and forms, but not answer to a specific problem.

Script Languages and the Visual Parametric Modeling Environment

CAD software script languages can vary deeply, not only in terms of its syntax and structure, but also in relation to the possible results obtained with their application. Some examples of CAD software programming languages are Rhinoscript by Rhinoceros, MEL by Maya, MaxScript by 3DMax and VBA or Autolisp by AutoCAD. For the comparison proposed in this study, VBA by AutoCAD was used as an objects-oriented language. Scripts (also called macros in VBA) may be developed using Visual Basic for Application Interactive Development Environment (VBAIDE), a software that is part of the Microsoft Office package (such as Word or Excel), as well as AutoCAD.

Although script VBA is not a compiled language and does not allow the creation of new classes of objects or autonomous applications, it is a powerful tool to automatize operations in AutoCAD. Its development environment allows easy and intuitively interface elaboration, which represents a big advantage over AutoLisp.

The language contains the typical structure of conditionals (if-then-else) and looping (for-each, for-next, do-while, select-case, go-to) present in any other programming language. The routines may be grouped in functions or sub-routines. The syntax is similar to the other object oriented languages, also known as dot syntax, in which a dot after the object's name is used to access its properties (for example, line.color = red) and hooks its methods followed by its parameters (for example: line.move frompoint, topoint).

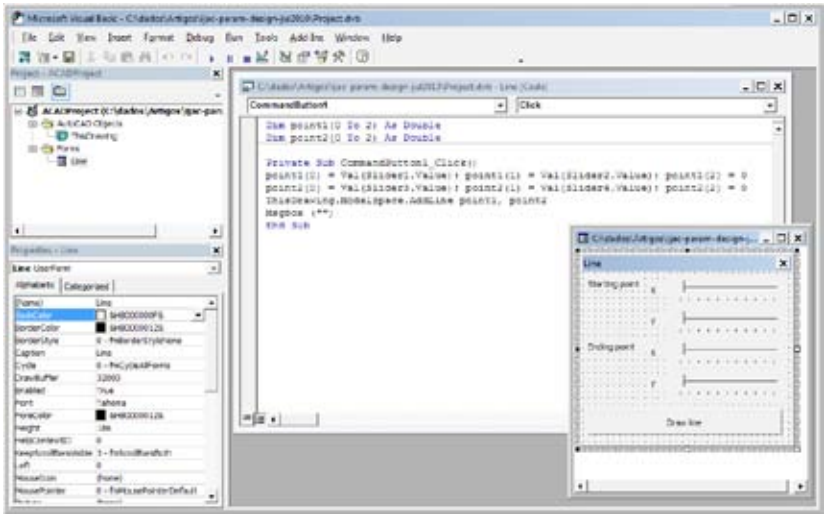
For this type of syntax's use it is important to know the AutoCAD objects model, a hierarchical structure that shows how groups and classes of objects interact. For example, *Model Space* is considered a group in which certain types of objects can be found, such as *lines*, *polylines*, *3dsolids* etc. It is recommended (but not mandatory) that a VBA routine be initiated with the variables definition, including its type specification (such as numbers, words or AutoCAD entities) and the definition of its accessibility mode (public or private). VBA uses arrays (matrixes) to store data, such as a dot's coordinates x, y and z. Because it is a very structured language, its teaching requires the introduction of various programming concepts, even for the development of very simple programs.

To exemplify the use of VBA for AutoCAD, a macro model that inserts parametric lines with a simple interface using scrollbars it is presented below. Figure 1 shows the user interface and the corresponding code in VBAIDE. The user uploads the macro using the AutoCAD editor and the interface automatically appears in the desktop. It is possible to change scrollbar's values, altering the initial points coordinates and the final line that will be generated in AutoCAD's editing window. In order to create a new result, the program must be reloaded. The interface may also have a button to erase the line. The code for implementing this routine is very simple. Numerical values for coordinates x and y for each of the points (1 and 2) are given and then a *line* object is instanced in AutoCAD's Model Space.

FIGURE 1

Example of a simple interface created in VBAIDE and its VBA code.

(Source: adapted from Kwok e Grondzik, 2007)



It is possible to execute something very similar in some parametric modeling environments with visual programming resources with no need of typing even one code line in text. Instead of presenting an interface to write lines in a compiler, these programs contain a desktop where components that will compose the "code" to accomplish the task may be introduced.

Two examples of this type of software are: Generative Components (GC) and Grasshopper. The former is a Microstation software module by Bentley, while the latter is a plug-in for Rhinoceros, a software designed by McNeil corporation. The symbolic diagram of GC is described as:

> *A view of the geometric and non-geometric features you are placing, in graph form. The features are capsules with the feature type noted underneath the feature name. (...) The lines connecting the features show any dependencies between features. The arrows show the direction of the dependency. The Symbolic Diagram visually expresses dependencies that may not be as apparent in the Geometric view, but which influence other dependent features and so the behaviors of the whole model. (Bentley Institute, 2008, p.13)*

In Grasshopper, the area for the preparation of diagrams is called *canvas* and it is defined as "...the actual editor where you define and edit the history network. The Canvas hosts the objects that make up the definition" (Payne & Issa, 2009, p.5).

In both cases, the geometry is developed diagrammatically. Thus, it is not stored in a usual geometric models file, but in a special type of file called *transaction file* in GC and *definition file* in Grasshopper. A *transaction file*:

> ***Contains the instructions that will generate geometry. When you open one in Generative Components, you see the working environment. It is comprised of the Generative Components dialog, the Symbolic Diagram and a Geometric view. (Bentley Institute, 2008, p.13).***

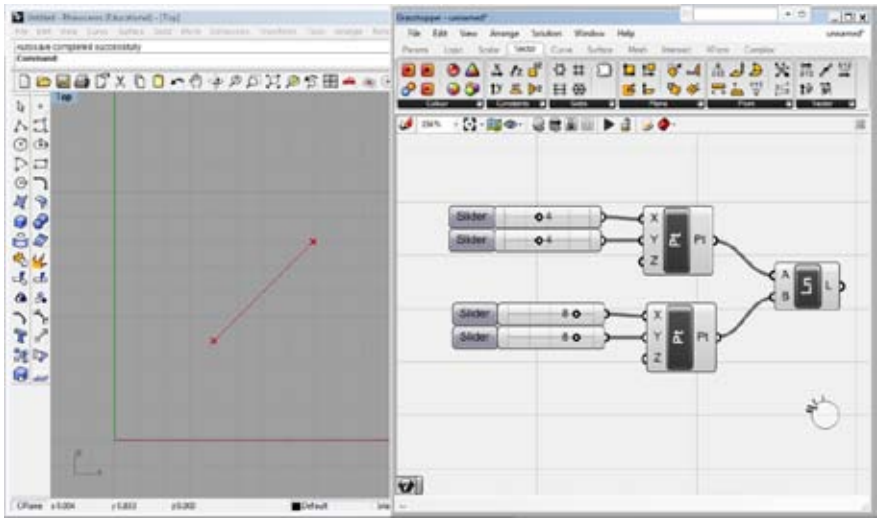
Figure 2 show the diagram prepared in Grasshopper to represent a parameterized line, similar to the previous example developed in VBA. The representation is composed by scrollbars where the user may define coordinate values x and y of the initial and final points of a line. Each is represented in the canvas by a component that generates them in the Rhinoceros desktop. These components visually resemble batteries, with input connectors to the left and output results to the right. The components that generate points are connected to a third component, which generates a line. The values that are not specified, as in the case of coordinate z point, receive a default value (in this case, z=0). When the user changes the scrollbars position the points change place and, consequently, the line follows, being represented in Rhinoceros desktop.

In Grasshopper it is also possible to create conditional structures with the use of special components. Other components are capable of subdividing forms, such as lines of surfaces, without the necessity of typing a *looping*. Some com-

FIGURE 2

Diagram for a line drawing: Rhinoceros window to the left and Grasshopper canvas to the right.

(Source: authors)



ponents are able to generate random numbers or numeric sequences, such as the Fibonacci series and the golden section. These tools may be used to implement design generative systems.

Despite the fact that a parameterized line generation be a simple case, it is already possible to show how its use in a parametric modeling visual programming environment is much more intuitive than a text code programming environment, since it does not require the introduction of a theoretical programming knowledge.

Architectural Modeling Methods

According to Mitchell (1975), there are three methods to architectural representation and modeling: iconic, analogic and symbolic. The iconic models are the most literal. Typical examples of their use in architecture are plans, elevations and physical models. The production of these models involves scale (amplifications and reductions) and projection (2D and 3D) transformations. Mitchell (1975, p.130) emphasizes that the role of this type of model in the generative process is static: "a particular state of the system actually 'looks like' the potential solution which it represents". Through iconic models it is possible to foresee how the building will look when it is ready.

In the analogic models, according to Mitchell (1975), a set of properties is used to represent other set of properties of the object being designed.

> ***Analogue generative systems often represent potential designs by settings of wheels, dials, sliding columns, etc. The operations performed to change the state of the system (that is, to describe a new potential design) are thus mechanical, for example, the spinning of wheels, setting dials, sliding columns alongside each other." (Mitchell, 1975, p. 131)***

The representation of 'Sagrada Familia' produced by Gaudí with cables and sand bags is an example of an analogic model. The cable symbolizes the its tension vectors that represent, thus, analog forces of the structure to be built. Through this model the architect was able to find the ideal form for the cathedral's arched structures.

Words, numbers, mathematic operators etc. represent the symbolic models. In Architecture, the symbolic models are mainly used for the simulation and evaluation of structures, acoustics, lighting and thermal performance. Symbols are typically displayed as mathematic formulas, tables, matrixes and algorithms.

The three representation methods described by Mitchell (1975) present different abstraction levels: the iconic representations better approach reality, while the symbolic ones are much more abstract. The analogic models are between both. Although Mitchell did not specifically mention diagrams as a representation method, it is possible to consider them as a type of analogic representation,

since they allow easy manipulation and are among the concrete and abstract representations. Table 1 categorizes AutoCAD/VBA and Rhinoceros/Grasshopper systems in terms of representation methods. It is possible to affirm that the former operates as an iconic/symbolic model, while the latter operates as an iconic/analogic representation.

TABLE 1 – Categorization of AutoCAD/VBA e Rhinoceros/Grasshopper systems in terms of representation methods and abstraction levels.

Representation Method	Abstraction Level	Example 1	Example 2
Symbolic	High	VBAIDE	
Analogic	Medium		Grasshopper
Iconic	Low	AutoCAD drawing editor	Rhinoceros drawing editor

Source: the author

Visual Programming Environments for Parametric Modeling and VPLs (Visual Programming Languages

Visual programming environments for parametric modeling can be compared to visual programming languages (VPLs), also called diagrammatic programming languages. VPLs allow the user to create programs by the manipulation of graphic components, instead of using code lines. In other words, they use an analogic representation for the algorithms. Figure 3 shows an example of interface for VPL.

Although parametric modeling visual programming environments are not exactly a VPL, we may say that they have some of its characteristics, such as the use of interface "*box-and-wire*", the possibility of inserting codes in some of its components, and the components hierarchical organization, which can be grouped to form subunits.

Green and Petre (1996) studied the psychological aspects of VPLs by means of comparison in terms of their cognitive dimensions. According to these authors, the use of a visual environment to develop programs is easier for many reasons:

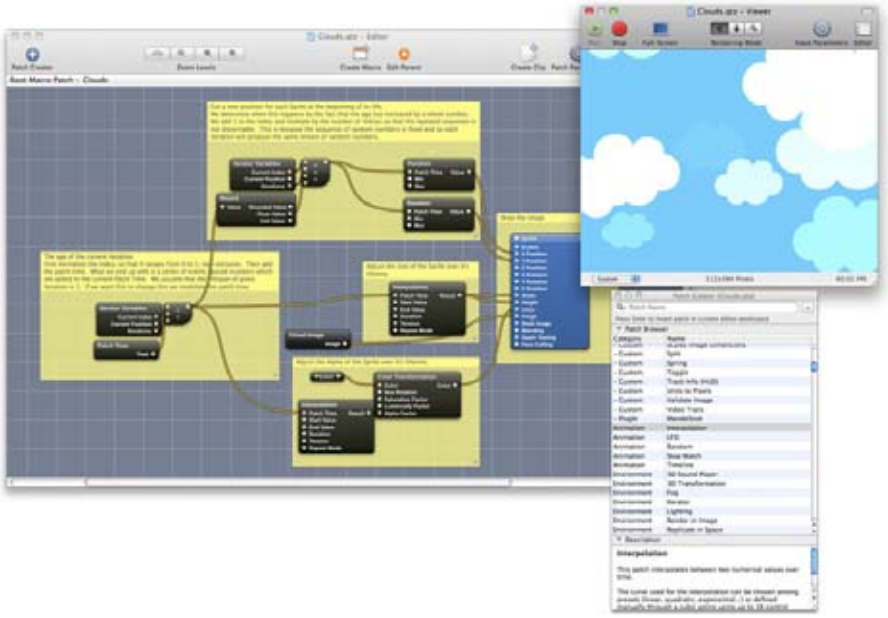


FIGURE 3
Quartz Composer, a VPL based on interface "box-and-wire".

Source: Available in Wikipedia.

> *There are fewer syntactic planning goals to be met, such as paired delimiters, discontinuous constructs, separators, or initializations of variables; higher-level operators reduce the need for awkward combinations of primitives; and the order of activity is freer, so that programmers can proceed as seems best in putting the pieces of a program together (Green e Petre, 1996, p.40).*

Similarly, it is not necessary to define variables in the parametric modeling visual programming tools; instead simply dragging a parameter and attributing values to it is enough. Besides, it is not essential to plan the order of actions e there are no syntactic rules. Even when a code in VBScript is used, it is restrict to a specific component, thus the possible errors are easily located.

Based in empirical studies on the VPLs use, authors such as Green and Navarro (*apud* Green and Petre, 1996) concluded that the resulting spatial reasoning of the visual diagrams manipulation works as a support to the elaboration of abstract ideas when a code is being developed. Analogically, visual programming environments for parametric modeling can be helpful with the organization of ideas when we plan a spatial composition, without the need of following a linear path.

Regarding the developing code assessment, Green and Petre (1996) affirm that:

> *The less experienced the programmer, the smaller the amount that is produced before it must be evaluated. Novices need 'progressive evaluation', an environment where it is easy to check a program fragment before adding to it. (Green e Petre, 1996, p.8)*

In other words, non-experienced programmers need to test the code along each step of the code development. In the visual programming tools for parametric modeling, the user can visualize, in real time, the result of the code changes

in the window that shows the geometric model. Besides, in Grasshopper, when the user makes an incorrect connection between two components, these become automatically red. In the symbolic languages users only notice the errors when trying to execute the code. Although VBAIDE identifies the majority of the errors and has debug resources, such as flags, and variable watches, identifying certain syntactic errors may take some time. Most students have difficulties in the use of these tools, because their inexperience makes the task of identifying and fixing errors very difficult.

Case Study

Both AutoCAD/VBA and Rhinoceros/Grasshopper systems were utilized in a subject of the undergraduate course in Architecture and Urbanism at the Civil Engineering School of State University of Campinas (FEC-Unicamp). "CAD in the Creative Process" is a semester long subject and classes meet once a week for two hours. The classes normally have thirty students. The topics addressed are:

- Computer-aided architectural design – history and definitions;
- Architectural forms generation strategies: symmetry, parameterization, casualty, recursion and substitution (fractals), design based in rules (form grammaticism), performance-based scripts, algorithms and projects. Each of the themes are followed by exercise that uses a pre-defined code;
- A generative design exercise that must be done using the computational tool. Normally only VBA to AutoCAD is used, but in this case, both VBA and Grasshopper were used by students;

In both cases, only controls, procedures and simples math operators were introduced to students, to allow the development of experiences with parametric forms. Conditional structures or looping were not taught in the course. The first phase involved the use of VBA for developing a simple abstract composition. The second's objective was the preparation of a cover for the campus access gate, using Grasshopper as a computational tool.

The subject's dynamic is based in the introduction of each computational concept applied to the project, followed by a short exercise and design references research, related to this concept. For example, in the class that addressed recursion and substitution, students prepared compositions using a VBA routine. Afterward, students sought for examples of these concepts application in Architecture.

The introduction to VBA was given in only one class and to Rhinoceros and Grasshopper in two. Then, a two weeks time spam was given to students to develop a small exercise using each of the tools. The analysis of the results was

done with the result obtained by students and through a short text on the use of CAD tools in the creative process. The students' opinions were compared to the previous years students', when only the textual programming language had been used. Figures 4 and 5 show some of the results obtained by students in exercises with VBA and Grasshopper.

Regarding the learning curve, the total consultations for students during the second exercise was much lower than in the first one, meaning that students had less doubts and were more prepared to solve the problems on their own. These students had already learned to use the AutoCAD software in a previous semester subject, but had no experience in Rhinoceros. Even though, they were so excited with the parametric modeling visual programming environment that rapidly learned to use the new CAD software.

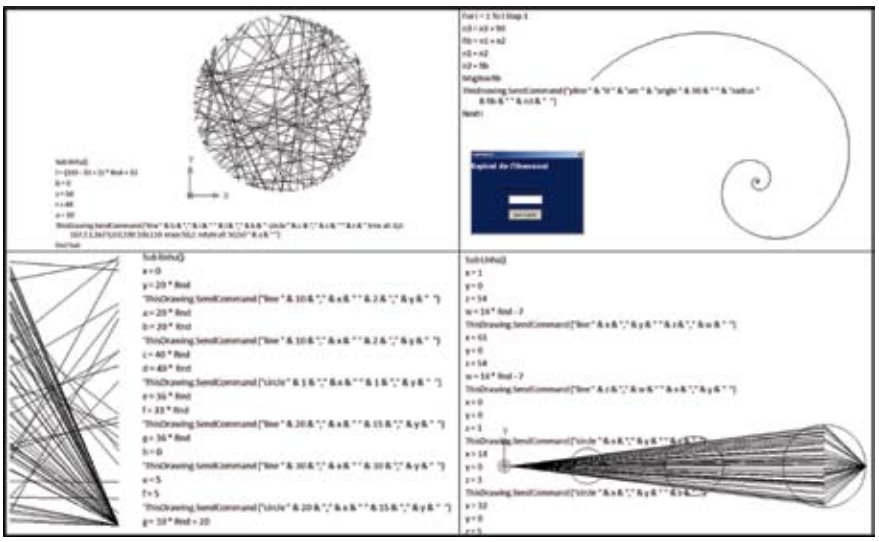
In the first exercise the students developed simple routines from the examples presented, copying part of them and changing variables that produced the parametric variations. In the second exercise the results obtained were much above expectation. Some students studied tutorials in the Internet and developed very complex compositions. The results show that students were excited and generated forms much more sophisticated with the use of the visual programming environment for modeling than with the script language.

In exercise 1, most of the students used straight lines but one that already had knowledge in VBA and developed a spiral based in Fibonacci series. In exercise

FIGURE 4

Results of the firt exercise.

Source: the authors.



2, the majority of students used organic forms, which probably, they would not be able to create in AutoCAD.

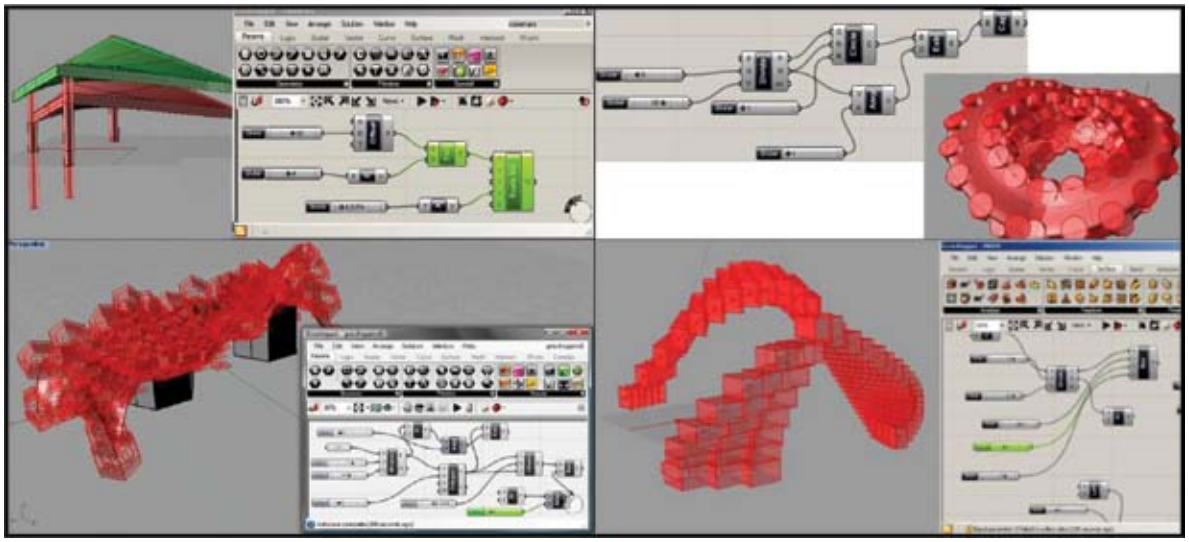
A comparison between texts produced by 2010 class and the other year's students on the computational concepts applied to Architecture show that stu-

dents developed a better understanding on use of generative design strategies and on the computational tools to explore design solutions. In previous years, students frequently mentioned that it was interesting to implement generative systems to the project, but that they would not do it professionally. After using Grasshopper, students recognized in the parametric modeling an architectural reasoning and something they would enjoy using in the future.

FIGURE 5

Results obtained with Grasshopper.

Source: the authors.



Discussions

Alexander (1971), based on the graph and groups theory, and Eisenman (1963), inspired by Deleuze and Derrida's deconstruction, used diagrams to represent design projects. According to Somol (1999), in the second half of the 20th century, the diagram became a fundamental technique and procedure for the design knowledge, as well as a tool for the production and representation of the architectural discourse (Figure 6). The analogic models developed on Grasshopper *canvas* can be understood as design diagrams, which require certain level of abstraction, but are not too distant from reality, such as the text programming codes. Through exercises such as these ones, it becomes possible stimulating students to rethink their design strategies and to start using computers as tools that are actively helpful in the creative process. A parametric modeling visual programming environment can be a good start to achieve this objective.

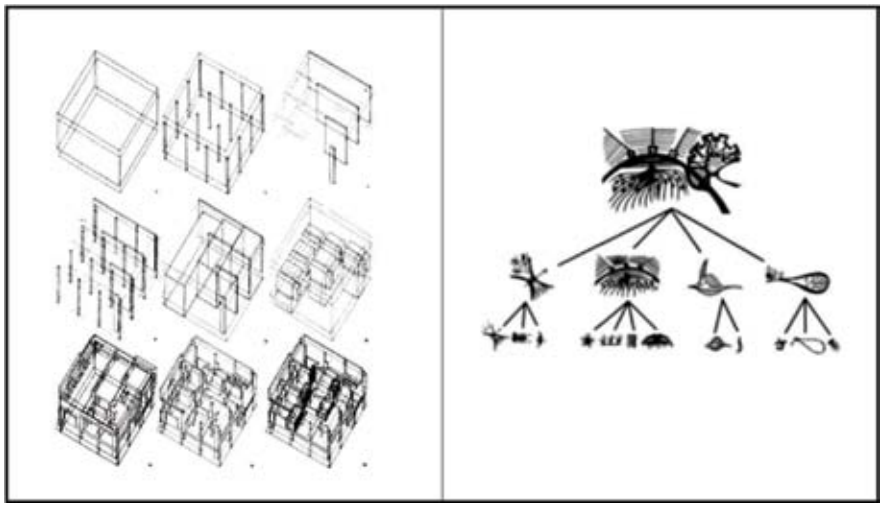


FIGURE 6
Transformation diagrams for House II, by Peter Eisenman, and diagram of a town, by Christopher Alexander.

Source: COSTA, 1998.

Conclusions and Future Unfolding

It is possible to conclude that parametric modeling visual programming environments allow the implementation of generative design strategies in a relatively simple way. These environments can be seen as an opportunity to introduce computational concepts applied to Architecture to students that have no knowledge in programming. However, the results do not exclude the importance of introducing script languages in a more appropriate moment of the course.

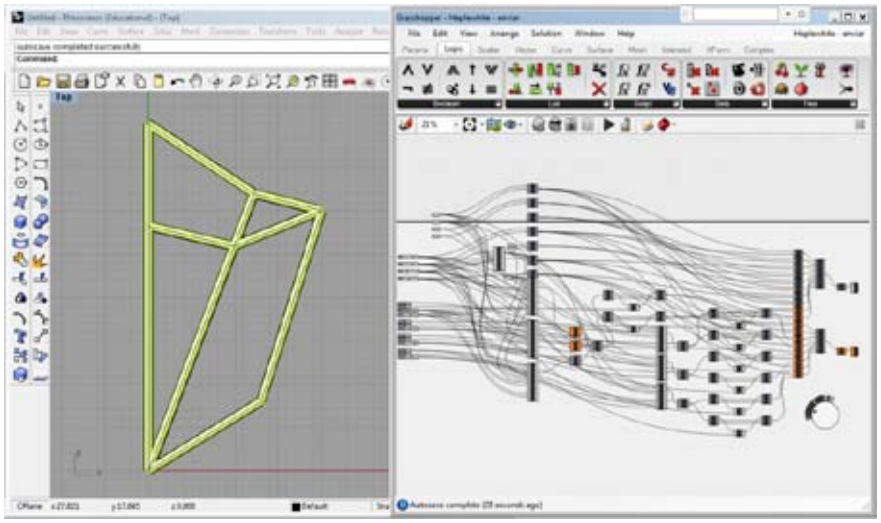
This research considered only the parametric modeling as a generative strategy. Future researches may compare the implementation of other generative strategies, in the two environments analyzed, such as systems based on rules. Figure 7 shows the implementation of the grammaticism of form in Hepplewhite style's chairs back supports, based on Knight's (1980) work, in Grasshopper.

The diagram uses components with conditionals in order to apply different rules to the composition. The scrollbars allow the interactive exploring of parameters. However, the diagram seems a little confusing, since in analogic models, the higher its complexity, the larger the possibility of losing its intelligibility – what is very important in abstract representations.

Through these kinds of discussions, it becomes possible encouraging students to rethink their generative strategies and start utilizing computers as a tool to amplify the search of design solutions possibilities.

FIGURE 7
An implementation of the Hepplewhite's chair seat gramaticism in Grasshopper

Source: Carlos VAZ.



Acknowledgements

The authors are thankful to students that took the subjects AU303 since 2004 for their enthusiasm and interest in the computational theories applied to the design.

References

ALEXANDER, C. **Notes on the synthesis of Form**. Cambridge, Massachusetts: Harvard University Press, 1964.

Bentley Institute. **GenerativeComponents V8i Essentials: Course Guide**. Bentley Institute, 2008. Disponível em:<http://ftp2.bentley.com/dist/collateral/docs/microstation_generativecomponents/microstation_GC_v8i_essentials_book.pdf>. Acesso em: 20 abr. 2011.

CELANI, G. **CAD Criativo**. Rio de Janeiro: Campus-Elsevier, 2003.

CELANI, M. G. C. **Beyond analysis and representation in Cad: a new computational approach to design education**, 2002, p., Tese de doutorado, Massachusettz Institute of Technology

CELANI, M. G. C. Generative design in architecture: history and applications. In: **New Strategies, Contemporary Techniques**, 2008, Barcelona. New Strategies, Contemporary Techniques. Disponível em: < http://www.simae.net/en/index.php >. Acesso em: 20 abr. 2011.

CELANI, M. G. C. **Teaching CAD programming to architecture students**. Revista Gestão & Tecnologia de Projetos: v. 3, n. 2, p. 1-23, 2008b.

COATES, P.; THUM, R. **Generative modeling: student workbook**. Londres: University of East London, 1995.

EISENMAN, P. **The formal basis of modern architecture**. Tese de doutorado, Universidade de Cambridge, 1963, 378 p.

GREEN, T. R. G.; PETRE, M. **Usability analysis of visual programming environments: A cognitive dimensions framework**. Journal of Visual Languages and Computing: v. 7, p. 131-174, 1996.

KNIGHT, T. W. **The generation of Hepplewhite-style chair back designs. Environment and Planning B: Planning and Design**, Londres: n. 7, p. 227-238, 1980.

Mark, E.; Martens, B.; Oxman, R. The Ideal Computer Curriculum. In: H.Penttila (Ed.), **Architectural Information Management**, 19th eCAADe Conference Proceedings. Helsinki (Finlandia): Helsinki University of Technology, pp. 168-175, 2001.

MITCHELL, W. J.; LIGGET, R. S.; KVAN, T. **The art of computer graphics programming**. Nova York: Van Nostrand Reinhold, 1987.

PAYNE, A.; ISSA, R. **The Grasshopper Primer: for version 0.6.0007**. 2009. Disponível em:< http://www.liftarchitects.com/journal/2009/3/25/the-grasshopper-primer-second-edition.html>. Acesso em: 20 abr. 2011.

ROBBINS, E. **Why Architects Draw**. Cambridge: The MIT Press, 1997.

SCHMITT, G. **Microcomputer Aided Design for Architects and Designers**. Nova York: John Wiley & Sons, 1988.

SOMOL, R. E. **Dummy Text, or The Diagrammatic Basis of Contemporary Architecture**. Risco: v. 5, n.1, p. 168-178, 2007.

TERZIDIS, K. **Algorithmic architecture**. Cambridge: Architectural Press, 2006.

WIKIPEDIA. **Quartz Composer**. Disponível em:< http://en.wikipedia.org/wiki/Quartz_Composer>. Acesso em: 20 abr. 2011.