CADERNOS
PROARQ 18

MARIA GABRIELA CELANI E CARLOS EDUARDO VAZ

Scripts em CAD e ambientes de programação visual para modelagem paramétrica: uma comparação do ponto de vista pedagógico

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

Maria Gabriela Celani Arquiteta e Mestre pela Faculdade de Arquitetura e Urbanismo da Univesidade de São Paulo (FAU-USP), PhD pelo Massachussets Institute of Tecnology (MIT), livre-docente pela Universidade Estadual de Campinas (Unicamp) e pós-doutora pela Universidade Técnica de Lisboa. É pesquisadora e docente do curso de Arquitetura e Urbanismo na Unicamp, onde criou e coordena o Laboratório de Automação e Prototipagem para Arquitetura e Construção (LAPAC) e o grupo de pesquisas Teorias e Tecnologias Contemporâneas Aplicadas ao Projeto. gabi.celani@gmail.com

Carlos Eduardo Vaz Arquiteto pela Universidade de São Paulo (2003), Mestre e Doutor em Engenharia Civil pela Universidade Estadual de Campinas (UNICAMP). Atualmente é professor adjunto da Universidade Federal de Pernambuco, no departamento de Expressão Gráfica. cevv00@gmail.com

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*

RESUMO

Este artigo compara o uso de ambientes de programação visual para modelagem paramétrica e linguagens de script para programas de CAD (computer aided design) no ensino de conceitos computacionais aplicados ao projeto, para alunos de graduação em arquitetura. Ambos os sistemas são descritos e categorizados em termos do método de representação utilizado. É proposta uma analogia entre os ambientes de programação visual para modelagem paramétrica e as linguagens de programação visuais (VPLs), e é feita uma revisão da literatura destas. A comparação também é baseada nos resultados práticos obtidos em uma disciplina em que as duas alternativas foram utilizadas para o desenvolvimento de um exercício de projeto paramétrico. No segundo caso, a curva de aprendizado foi muito mais acentuada, os projetos desenvolvidos mais complexos e os alunos tiveram uma melhor compreensão das estratégias generativas de projeto e do uso de ferramentas computacionais específicas para a exploração de soluções de projeto. Embora devam ser consideradas as características específicas das ferramentas utilizadas na comparação, é possível concluir que o uso de ambientes para modelagem paramétrica visual pode ajudar na introdução de conceitos computacionais para estudantes de arquitetura que não possuem conhecimento de programação, preparando-os para a introdução de métodos mais abstratos. O artigo também discute a importância da habilidade dos alunos em trabalhar com diferentes métodos de representação, do mais concreto ao mais abstrato, como parte da educação em arquitetura.

Palavras-chave: Linguagem de programação. Modelagem paramétrica. Métodos de representação. Ensino.

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.*

Introdução

A inserção de novas tecnologias para a educação em arquitetura tem sido alvo de grandes discussões e publicações e tema de muitas conferências, especialmente a partir da década de 1980, quando os computadores se tornaram mais acessíveis tanto para os estudantes quanto para as universidades. A questão central dessas discussões progressivamente evoluiu da introdução de programas para simples representação para programas que podem ser realmente integrados ao processo de projeto, permitindo a geração e exploração de soluções (Mark, Martens & Oxman, 2001). O interesse nos programas generativos está se tornando mais evidente após o início do uso das máquinas de fabricação digital no campo da arquitetura, que permitem a produção de formas livres de um modo que não era possível anteriormente.

Sistema generativos de projeto foram descritos por Mitchell (1975), em seu artigo *The theoretical foundations of computer-aided architectural design*, como sendo dispositivos que são capazes de gerar soluções em potencial para um dado problema de projeto. As mais importantes estratégias do projeto generativo são as combinações, as substituições, a parametrização, as restrições de contexto, a aleatoriedade, a emergência, a otimização e a combinação de duas ou mais delas (Celani, 2008).

Os sistemas generativos, ao contrário do que se pensa, não precisam ser necessariamente implementados em um computador, mas podem ser utilizados para realizar atividades repetitivas que consumiriam muito tempo. Antes de 1980 esses sistemas dificilmente eram implementados em computador, não apenas por que era necessário ter habilidades de programação altamente especializadas, mas também pelo custo do hardware, que deveria possuir uma interface gráfica adequada e grande capacidade de memória.

Um dos primeiros guias para a implementação de sistemas generativos de projeto em computador foi elaborado por Mitchell, Ligget e Kvan (1987). Trata-se do livro *The art of computer-graphics programming*, que utiliza como linguagem de programação para o desenvolvimento das rotinas o Pascal. Outros guias para a implementação de sistemas generativos de projeto foram organizados por autores como Schmitt (1988), Coates e Thum (1995), Celani (2003) e Terzidis (2006). Cada um desses livros propõe diferentes estratégias generativas. Todos baseiam-se no uso das linguagens de script existentes em programas de CAD e tiram vantagem das funções geométricas presentes nesses aplicativos. Seu objetivo é introduzir conceitos computacionais a alunos de graduação e pós-graduação de arquitetura (Celani, 2008).

Contudo, as disciplinas que têm esse tipo de conteúdo raramente são obrigatórias na formação do arquiteto e é possível afirmar que poucos estudantes de arquitetura estão prontos para aceitar o desafio de aprender uma linguagem de programação. As razões dessa dificuldade ainda não foram comprovadas, mas

é possível especular que os estudantes acreditam que programar é complicado, que essa atividade não tem relação com sua atuação profissional e que não se trata de um conhecimento que um arquiteto deva ter.

Mesmo quando os alunos de arquitetura aprendem uma linguagem de programação, a maioria deles não irá usar essa nova habilidade para expressar suas ideias de projeto. Isso ocorre, em primeiro lugar, porque para desenvolver um programa de computador é necessário certa experiência – trata-se de uma atividade que consome muito tempo. Em segundo lugar, o desenho a mão ainda é considerado o principal – se não o único – método de produção das ideias iniciais de projeto.

É impossível questionar o papel do desenho no processo criativo, que tem sido amplamente pesquisado por muitos estudiosos (Robbins, 1997). Contudo, se considerarmos o processo de projeto como uma atividade que envolve a combinação de diferentes possibilidades, o computador pode assumir um papel importante ao permitir a geração sistemática e exaustiva de alternativas de projeto.

Recentemente, alguns pacotes CAD introduziram ferramentas com capacidades de geração de modelos paramétricos sem que haja a necessidade, por parte do usuário, da elaboração de código simbólico para sua implementação. Nesses programas é utilizada a programação visual para a elaboração de diagramas que irão representar o algoritmo que gerará o modelo paramétrico. Em geral, esse tipo de aplicativo permite uma automação limitada do processo de projeto, mas pode ser eficiente para a exploração de formas, por meio da geração automática de variações paramétricas. A maioria desses ambientes de programação permite criar algoritmos sem a utilização de código simbólico e suas capacidades podem ser estendidas por meio do uso de scripts.

Objetivos e Métodos

O objetivo desta pesquisa foi avaliar o uso de duas ferramentas baseadas em diferentes paradigmas da computação, as linguagens de programação textual e as linguagens de programação visual. Para isso foram introduzidos conceitos de programação e a partir deles os alunos deveriam realizar exercícios com duas ferramentas diferentes (o editor de algoritmos Grasshooper e a linguagem de script VBA). Nesses trabalhos os estudantes deveriam elaborar algortimos capazes de inserir componentes como linhas e faces ou criar formas complexas. Essas composições deveriam ser modeladas indiretamente pelos alunos. Ou seja, estes deveriam criar os códigos ou estruturar, como será visto, um digrama simbólico que era responsável por inserir automaticamente ou possibilitar a modificação dos componentes instancidados por parte dos alunos.

Para viabilizar a pesquisa são comparados os trabalhos de turmas de diferentes anos da disciplina "CAD no Processo Criativo". Os trabalhos, desenvolvidos por meio do uso de ambientes para modelagem paramétrica visual (Grasshopper), pertencem apenas ao grupo que participou da disciplina no ano de 2010. Os demais trabalhos apresentados pertencem a turmas de anos anteriores, que ainda utilizavam a linguagem de script VBA.

Os critérios utilizados para analisar os resultados basearam-se na capacidade de assimilação de conceitos de programação e de geração de formas por parte dos alunos. Como será visto, apesar de os alunos terem aprendido o mesmo conteúdo, com a utilização da nova ferramenta CAD, foram capazes de desenvolver trabalhos nos quais conseguiram dar resposta a um problema de projeto. O resultado foi completamente diferente em relação aos anos anteriores, em que na maior parte dos casos os estudantes apenas conseguiam instanciar linhas ou formas, mas não dar resposta a um problema específico.

Linguagens *Script* e os Ambientes para Modelagem Paramétrica Visual

As linguagens script para softwares CAD podem variar muito, não apenas em termos de sua sintaxe e estrutura, mas também em relação aos resultados que podem ser obtidos por sua aplicação. Alguns exemplos de linguagens de programação para aplicativos CAD são o Rhinoscript do Rhinoceros, o MEL do Maya, o MaxScript do 3DMax e o VBA ou Autolisp do AutoCAD. Na comparação que será realizada neste artigo foi utilizado o VBA para AutoCAD, uma linguagem orientada a objetos. Os scripts (também chamados de macros no VBA) podem ser desenvolvidos no *Visual Basic for Application Interactive Development Environment* (VBAIDE), um aplicativo incorporado aos programas do Microsoft Office (como Word ou Excell), assim como no AutoCAD.

Embora não seja uma linguagem compilada e não permita a criação de novas classes de objetos ou aplicações autônomas, a linguagem script VBA é uma ferramenta poderosa para automatizar procedimentos no AutoCAD. Seu ambiente de desenvolvimento permite a elaboração de interfaces de um modo fácil e intuitivo, o que representa uma grande vantagem sobre o AutoLisp.

A linguagem contém a estrutura típica de condicionais (*if-then-else*) e *looping* (*for-each*, *for-next*, *do-while*, *select-case*, *go-to*), presentes em qualquer outra linguagem de programação. As rotinas podem ser agrupadas em funções ou sub-rotinas. A sintaxe é similar a das demais linguagens orientadas ao objeto, sendo conhecida como *dot syntax*, na qual um ponto depois do nome de um objeto é usado para acessar suas propriedades (por exemplo, *line.color = red*) e engatilha os seus métodos, seguidos por seus parâmetros (por exemplo: *line.move frompoint, topoint*).

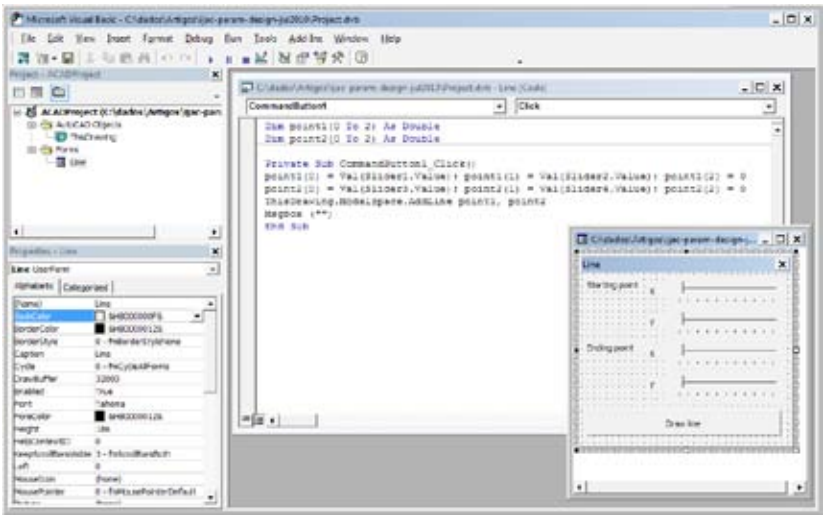
Para utilizar esse tipo de sintaxe é importante conhecer o modelo de objetos do AutoCAD, uma estrutura hierárquica que mostra como as coleções e classes de objetos se relacionam. Por exemplo, o *Model Space* é considerado uma coleção na qual certos tipos de objetos podem ser encontrados, tais como *lines*, *polylines*, *3dsolids* etc. É recomendado (mas não obrigatório) que uma rotina em VBA seja iniciada com a declaração das variáveis, incluindo a especificação de seu tipo (como números, palavras ou entidades do AutoCAD) e a definição do seu modo de acessibilidade (*public* ou *private*). O VBA utiliza *arrays* (matrizes) para armazenar dados, como as coordenadas x, y e z de um ponto. Por ser uma linguagem muito estruturada, ensiná-la requer a introdução de diversos conceitos de programação, mesmo para o desenvolvimento de programas muito simples.

Para exemplificar o uso do VBA para AutoCAD é apresentada a seguir um modelo de macro que insere linhas de maneira parametrizada, com uma interface simples que utiliza barras de rolagem. A Figura 1 mostra a interface do usuário e o código correspondente no VBAIDE. O usuário carrega a macro a partir do editor do AutoCAD e a interface automaticamente aparece na área de trabalho. Nessa interface é possível modificar os valores das barras de rolagem, alterando as coordenadas dos pontos inicial e final da linha que será gerada na área gráfica do AutoCAD. Para gerar um novo resultado, o programa deve ser recarregado. A interface também pode conter um botão para apagar a linha. O código para

FIGURA 1

Exemplo de uma interface simples criada no VBAIDE e seu código em VBA.

Fonte: adaptado de Kwok e Grondzik, 2007



implementar essa rotina é muito simples. Nele são atribuídos valores numéricos para as coordenadas *x* e *y* para cada um dos pontos (1 e 2) e depois é instanciado um objeto *line* no *Model Space* do AutoCAD.

É possível executar algo muito semelhante em certos ambientes para modelagem paramétrica com recursos de programação visual sem a necessidade de

digitar sequer uma linha de código em texto. Ao invés de apresentar uma interface para escrever linhas em um compilador esses programas contêm uma área de trabalho em que podem ser introduzidos componentes que irão compor o "código" que realizará a tarefa.

Dois exemplos desse tipo de software são o Generative Components e o Grasshopper. O primeiro é um módulo do aplicativo Microstation da empresa Bentley, enquanto o segundo é um *plug-in* para o programa Rhinoceros da empresa McNeil. O diagrama simbólico do GC é descrito como:

> ***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)***

No Grasshopper a área para a elaboração dos diagramas é chamada de *canvas* e é definida como "...the actual editor where you define and edit the history network. The Canvas hosts the objects that make up the definition." (Payne e Issa, 2009, p.5).

Em ambos os casos, a geometria é desenvolvida de um modo diagramático. Sendo assim, ela não é armazenda em um arquivo usual para modelos geométricos, mas em um tipo especial de arquivo chamado *transaction file* no GC e *definition file* no Grasshopper. Um *transaction file*:

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

A Figura 2 mostra o diagrama elaborado em Grasshopper para representar uma linha parametrizada, similar ao exemplo anterior desenvolvido em VBA. A representação é composta por barras de rolagem nas quais o usuário pode definir os valores x e y das coordenadas dos pontos inicial e final de uma linha. Cada ponto é representado, no *canvas*, por um componente que os gera na área de trabalho do Rhinoceros. Esses componentes lembram visualmente pilhas, com conectores de entrada de dados à esquerda e de saída dos resultados à direita. Os componentes que geram os pontos são conectados a um terceiro componente, que gera uma linha. Os valores que não são especificados, como no caso da coordenada z dos pontos, recebem um valor *default* (neste caso, z=0). Quando o usuário modifica a posição nas barras de rolagem os pontos mudam de lugar, e consequentemente a linha também, sendo redesenhados na área de trabalho do Rhinoceros.

No Grasshopper também é possível criar estruturas condicionais com o uso de componentes especiais. Outros componentes são capazes de subdividir for-

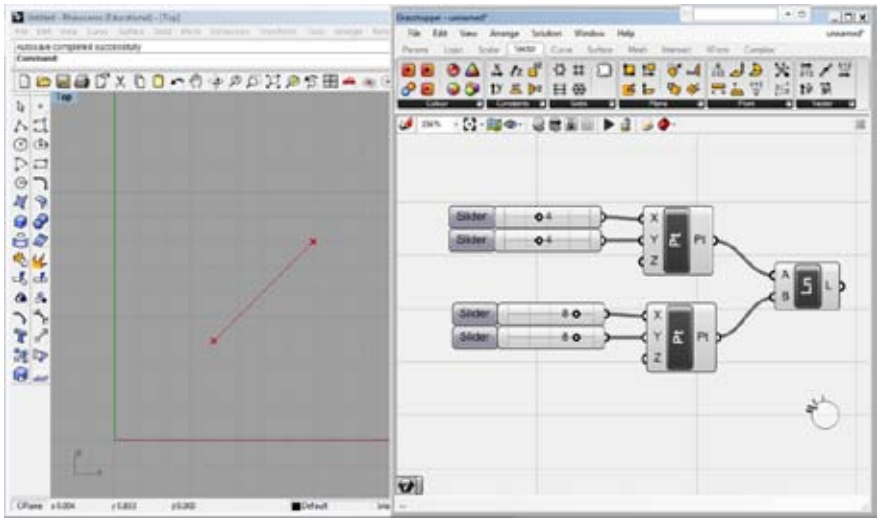
mas, tais como linhas ou superfícies sem a necessidade de escrever um *looping*. Alguns componentes são capazes de gerar números aleatórios ou sequências numéricas, tais como as séries de Fibonacci e a proporção áurea. Essas ferramentas podem ser utilizadas para implementar sistemas generativos de projeto.

Apesar da geração de uma linha parametrizada ser um caso simples, com ele já é possível mostrar como o uso de um ambiente de programação visual para modelagem paramétrica é muito mais intuitivo que um ambiente de programação por código textual, pois não requer a introdução de conhecimento teórico de programação.

FIGURA 2

Esquema para o desenho de uma linha: janela do Rhinoceros à esquerda e canvas do Grasshopper à direita.

Fonte: autores



Métodos de Modelagem em Arquitetura

Segundo Mitchell (1975), existem três métodos para a representação e modelagem em arquitetura: o icônico, o analógico e o simbólico. Os modelos icônicos são os mais literais. Exemplos típicos de seu uso em arquitetura são as plantas, elevações e maquetes. A produção desses modelos envolve transformações de escala (ampliações e reduções) e de projeção (2D e 3D). Mitchell (1975, pg. 130) enfatiza que o papel desse tipo de modelo no processo generativo é estático: "a particular state of the system actually 'looks like' the potential solution which it represents". Por meio dos modelos icônicos é possível antever como um edifício irá ficar quando pronto.

Nos modelos analógicos, segundo Mitchell (1975), um conjunto de propriedades é utilizado para representar outro conjunto de propriedades do objeto que está sendo projetado.

> ***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)***

A representação da Sagrada Família produzida por Gaudí com cabos e sacos de areia é um exemplo de um modelo analógico. Nele os cabos simbolizam os vetores de tensão, representando assim forças análogas à estrutura que seria construída. Por meio desse modelo o arquiteto foi capaz de descobrir a forma ideal para as estruturas abobadadas da catedral.

Os modelos simbólicos são representados por palavras, números, operadores matemáticos etc. Em arquitetura, os modelos simbólicos são principalmente utilizados para simulações e avaliações de estrutura, acústica, iluminação e performance térmica. Símbolos são tipicamente exibidos como fórmulas matemáticas, tabelas, matrizes e algoritmos.

Os três métodos de representação descritos por Mitchell (1975) apresentam diferentes níveis de abstração: as representações icônicas se aproximam mais da realidade, enquanto as simbólicas são muito mais abstratas. Os modelos analógicos estão entre ambos. Embora os diagramas não sejam mencionados especificamente por Mitchell como um método de representação, é possível considerá-los como um tipo de representação analógica, pois permitem fácil manipulação e encontram-se entre as representações concretas e as abstratas. A tabela 1 categoriza os sistemas AutoCAD/VBA e Rhinoceros/Grasshopper em termos de métodos de representação. É possível afirmar que o primeiro sistema opera como um modelo icônico/simbólico, enquanto o segundo opera como uma representação icônica/analógica.

QUADRO 1

Categorização dos sistemas AutoCAD/VBA e Rhinoceros/Grasshopper em termos de método de representação e nível de abstração

Fonte: autores

Método de representação	Nível de abstração	Exemplo 1	Exemplo 2
Simbólico	Alto	VBAIDE	
Analógico	Médio		Grasshopper
Icônico	Baixo	AutoCAD drawing editor	Rhinoceros drawing editor

Ambientes de Programação Visual para Modelagem Para métrica e as VPLs (Visual Programming Languages)

Os ambientes de programação visual para modelagem paramética podem ser comparados às linguagens de programação visual (VPLs), também chamadas de linguagem de programação diagramáticas. As VPLs permitem que os usuários criem programas por meio da manipulação de componentes gráficos em vez do uso de linhas de código. Em outras palavras, usam uma representação analógica para os algoritmos. A Figura 3 mostra um exemplo de interface para uma VPL.

Embora os ambientes de programação visual para modelagem paramétrica não sejam exatamente uma VPL, pode-se dizer que eles possuem algumas de suas características, como o uso da interface "box-and-wire" (caixas e fios), a possibilidade de inserir códigos em alguns componentes e a organização hierárquica dos componentes, que podem ser agrupados para formar subunidades.

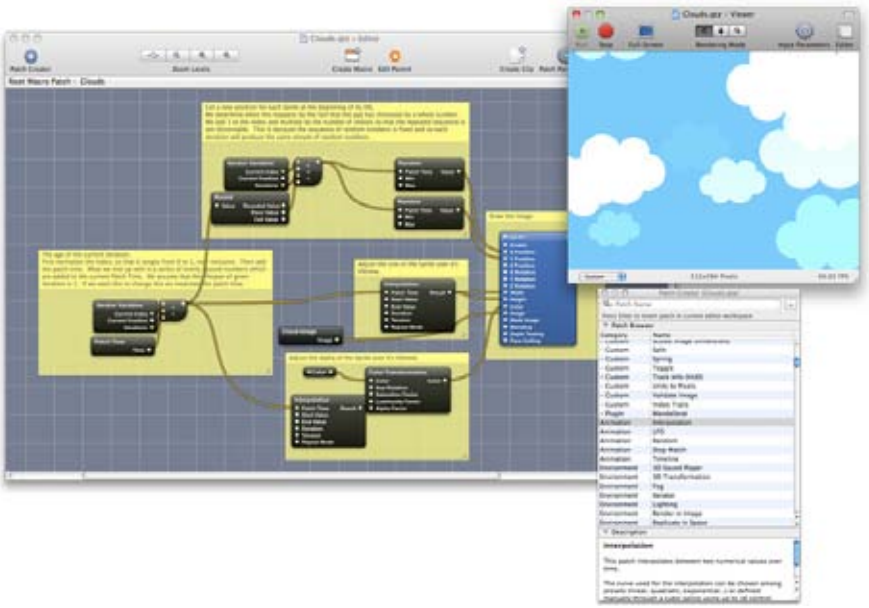


FIGURA 3
Quartz Composer, uma VPL baseada em interface "box-and-wire".

Fonte: Disponível em Wikipedia.

Green e Petre (1996) estudaram os aspectos psicológicos das VPLs por meio da comparação em termos de suas dimensões cognitivas. Segundo os autores, usar um ambiente visual para desenvolver programas é mais fácil por muitas razões:

> *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).***

De maneira semelhante, nas ferramentas de programação visual para modelagem paramétrica não é necessário declarar variáveis, mas simplesmente arrastar um parâmetro e atribuir valores a ele. Além disso, não é essencial planejar a ordem das ações e não existem regras sintáticas. Mesmo quando um código em VBScript é utilizado, este fica restrito a um componente específico, portanto os possíveis erros são facilmente localizados.

Baseados em estudos empíricos sobre o uso das VPLs, autores como Green e Navarro apud Green e Petre (1996) concluíram que o raciocínio espacial resultante da manipulação de diagramas visuais funciona como um suporte para a elaboração de ideias abstratas quando um código está sendo desenvolvido. Analogamente, os ambientes de programção visual para modelagem paramétrica podem colaborar para organizar ideias quando se planeja uma composição espacial, sem necessariamente seguir um caminho linear.

No que se refere à avaliação do código em desenvolvimento, Green e Petre (1996) afirmam que:

> ***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)***

Em outras palavras, programadores não experientes necessitam testar o código a cada pequeno passo dado no desenvolvimento do código. Nas ferramentas de programação visual para modelagem paramétrica, o usuário pode ver, em tempo real, o resultado das modificações no código na janela que mostra o modelo geométrico. Além disso, no *Grashhoper*, quando o usuário realiza uma conexão incorrenta entre dois componentes, estes automaticamente se tornam vermelhos. Nas linguagens simbólicas os usuários só notam os erros quando tentam executar o código. Embora o VBAIDE identifique a maioria dos erros e tenha ferramentas de *debug*, tais como *flags* e *variable watches*, identificar certos erros de sintaxe pode demorar algum tempo. A maioria dos estudantes sente dificuldades em utilizar essas ferramentas, pois sua inexperiência torna a tarefa de identificar e corrigir os erros muito difícil.

Estudo de Caso

Tanto o sistema AutoCAD/VBA como o Rhinocerus/Grassshopper foram utilizados em uma disciplina do curso de graduação em Arquitetura e Urbanismo da Faculdade de Engenharia Civil, Arquitetura e Urbanismo da Universidade Estadual de Campinas (FEC-Unicamp). A disciplina "CAD no Processo Criativo" tem duração de um semestre e aulas de duas horas de duração, uma vez por semana. As turmas são tipicamente formadas por trinta alunos. Os tópicos abordados são:

- *Computer-aided architectural design* – histórico e definições;
- Estratégias de geração de formas em arquitetura: simetrias, parametrização, aleatoriedade, recursão e substituição (fractais), projeto baseado em regras (gramáticas da forma), scripts e algoritmos e projetos baseados em desempenho. Cada um dos temas é seguido de um exercício que utiliza um código pronto;
- Um exercício de projeto generativo que deve ser realizado com o uso de uma ferramenta computacional. Normalmente apenas o VBA para AutoCAD é utilizado, mas neste caso foram usados pelos alunos tanto o VBA como o Grasshopper.

Em ambos os casos foram introduzidos apenas controles, procedimentos e operadores matemáticos simples aos estudantes, para permitir que estes desenvolvessem experiências com formas paramétricas. Não foram ensinados no curso estruturas condicionais ou *loopings*. A primeira etapa envolveu a utilização do VBA para o desenvolvimento de uma simples composição abstrata. A segunda tinha como objetivo a elaboração de uma cobertura para o portão de acesso do campus utilizando como ferramenta computacional o Grasshopper.

A dinâmica da disciplina é baseada na introdução de cada conceito computacional aplicado ao projeto seguido por um exercício curto ou uma pesquisa de referências projetuais relacionadas a esse conceito. Por exemplo, na aula de recursão e substituição os estudantes elaboraram composições a partir de uma rotina elaborada em VBA. Em seguida, os alunos buscaram exemplos da aplicação desses conceitos em arquitetura.

A introdução ao VBA foi feita em uma aula apenas e a de Rhinoceros e Grasshopper em duas. Em seguida, foi dado um prazo de duas semanas aos alunos para desenvolverem um pequeno exercício utilizando cada uma das ferramentas. A análise dos resultados foi feita a partir do resultado obtido pelos alunos e por meio de um curto texto sobre o uso das ferramentas CAD no processo criativo. As opiniões dos alunos foram comparadas às dos estudantes dos anos anteriores, quando apenas a linguagem de programação textual era utilizada. As Figuras 4 e 5 mostram alguns dos resultados obtidos pelos alunos nos exercícios com VBA e com Grasshopper.

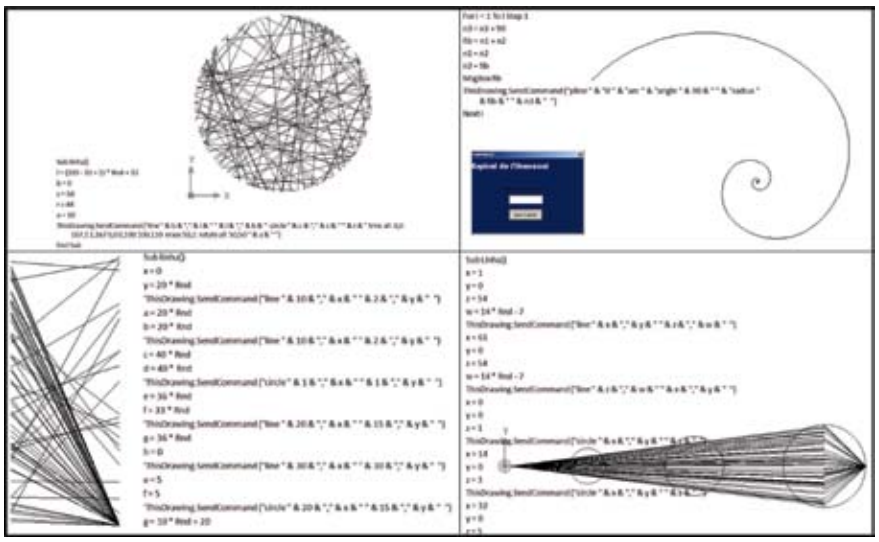
No que se refere à curva de aprendizado, a quantidade de atendimentos realizados durante o segundo exercício foi muito menor do que no primeiro, o que significa que os alunos tiveram menos dúvidas e estavam capacitados a resolver os problemas sozinhos. Esses alunos já haviam aprendido a usar o programa AutoCAD em uma disciplina no semestre anterior, mas não tinham qualquer experiência em Rhinoceros. Mesmo assim, eles ficaram tão entusiasmados com o uso do ambiente de programação visual para modelagem paramétrica que rapidamente aprenderam a utilizar o novo software CAD.

No primeiro exercício os alunos desenvolveram rotinas simples a partir de exemplos apresentados, copiando partes deles e modificando as variáveis que produzem as variações paramétricas. No segundo exercício foram obtidos resultados muito acima do esperado. Alguns estudantes pesquisaram tutoriais na Internet e desenvolveram composições bastante complexas. Os resultados mostram que os alunos se entusiasmaram e geraram formas muito mais sofisticadas com o uso do ambiente de programação visual para modelagem do que com a linguagem script.

FIGURA 4

Resultados do primeiro exercício.

Fonte: autores



No exercício 1 a maioria dos alunos utilizou linhas retas, exceto um deles, que já possuía conhecimento em VBA e desenvolveu uma espiral baseada nas séries de Fibonacci. No exercício 2 a maioria dos alunos utilizou formas orgânicas, que provalmente não seriam capazes de modelar no AutoCAD.

Uma comparação entre os textos produzidos pelos alunos sobre os conceitos computacionais aplicados à arquitetura da turma deste ano com as demais mostra que os estudantes desenvolveram uma melhor compreensão sobre as estratégias de projeto generativo e o uso das ferramentas computacionais para a exploração de soluções de projeto. Nos anos anteriores os estudantes mencionavam frequentemente que era interessante implementar sistemas generativos de projeto, mas que não seria algo que eles fariam profissionalmente. Após a utilização do Grasshopper os alunos reconheceram na modelgagem paramétrica uma forma de pensar arquitetura e algo que eles gostariam de utilizar futuramente.

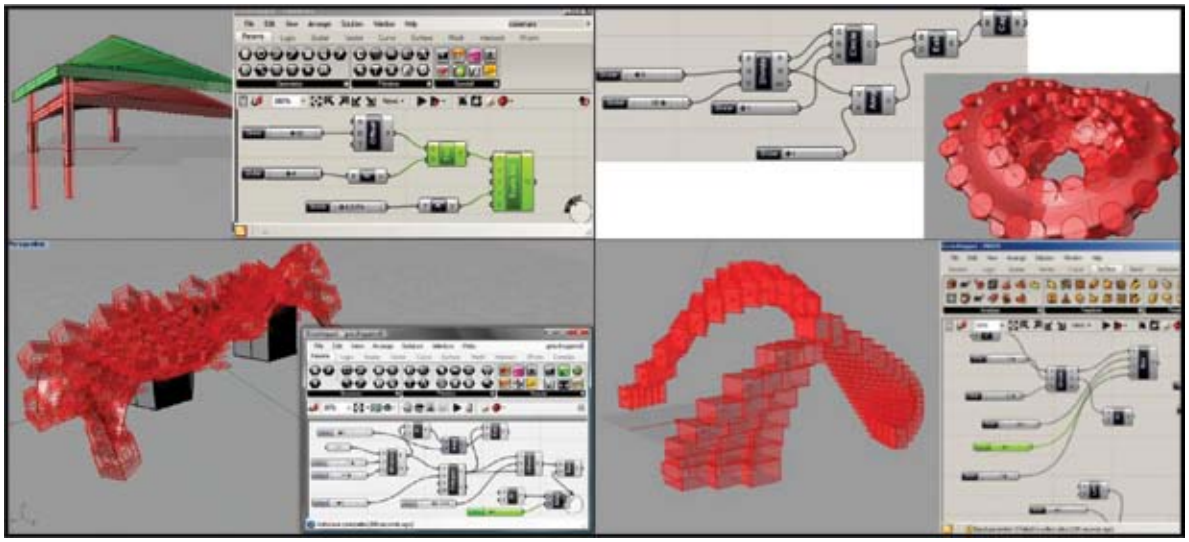


FIGURA 5
Resultados obtidos com a utilização do Grasshopper.

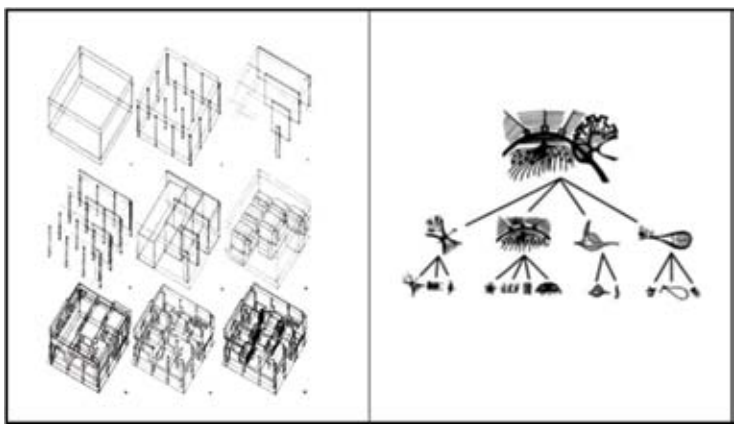
Fonte: autores

Discussões

Alexander (1971), baseado na teoria dos grafos e dos conjuntos, e Eisenman (1963), inspirado pela desconstrução de Deleuze e Derrida, usaram diagramas para representar projetos. De acordo com Somol (1999), na segunda metade do século XX o diagrama se tornou uma técnica e um procedimento fundamental no conhecimento de projeto, além de uma ferramenta para a produção e representação do discurso em arquitetura (Figura 6). Os modelos analógicos desenvolvidos sobre o *canvas* do Grasshopper podem ser entendidos como diagramas de projeto, que exigem certo nível de abstração, porém não se distanciam tanto da realidade como os códigos de programção em texto. A partir de exercícios como esse é possível encorajar os alunos a rever suas estratégias de projeto e começar a utilizar o computador como uma ferramenta que colabora ativamente no processo criativo. Um ambiente de programção visual para modelagem paramétrica pode ser um bom início para se atingir esse objetivo.

FIGURA 6
Diagramas de transformação para a Casa II, de Peter Eisenman, e diagram de um vilarejo, de Christopher Alexander.

Fonte: COSTA, 1998.



Conclusões e Trabalhos Futuros

É possível concluir que os ambientes de programação visual para modelagem paramétrica permitem a implementação de estratégias de projeto generativo de um modo relativamente simples. Esses ambientes podem ser vistos como uma oportunidade para introduzir conceitos computacionais aplicados à arquitetura para estudantes que não têm conhecimento em programação. Contudo, os resultados não excluem a importância de introduzir linguagens de script num momento adequado do curso.

Nesta pesquisa foi considerada apenas a modelagem paramétrica como estratégia generativa. Trabalhos futuros deverão comparar a implementação, nos dois ambientes analisados, de outras estratégias generativas, como os sistemas baseados em regras. A Figura 7 mostra a implementação de uma gramática da forma para os encostos das cadeiras em estilo Hepplewhite, baseada no trabalho de Knight (1980), em Grasshopper.

O diagrama utiliza componentes com condicionais para aplicar diferentes regras na composição. As barras de rolagem possibilitam a exploração interativa dos parâmetros. Contudo, o diagrama parece um pouco confuso, pois, nos modelos analógicos, quanto maior sua complexidade, maior a possiblidade de se perder a inteligibilidade – o que é muito importante nas representações abstratas.

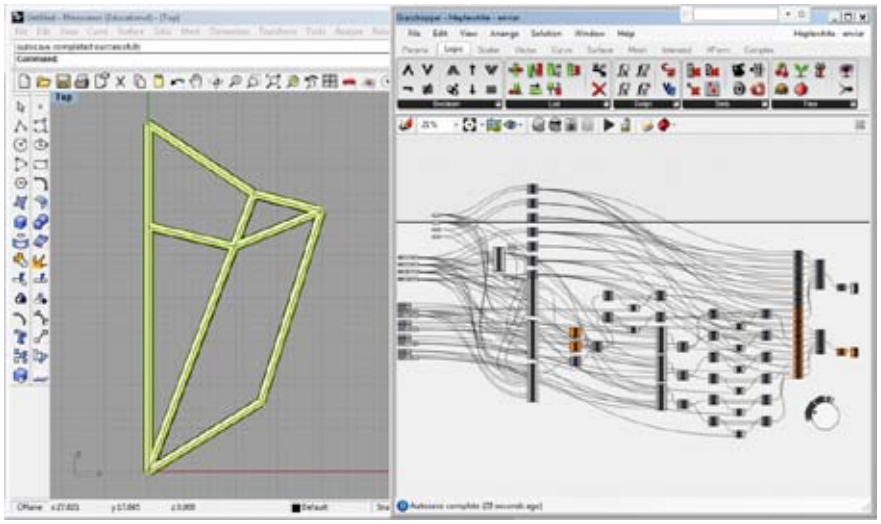


FIGURA 7
Uma implementação da gramática dos encostos das cadeiras Hepplewhite em Grasshopper

Fonte: Carlos VAZ.

Por meio de discussões como essas é possível encorajar os alunos a repensar suas estratégias generativas e começar a utilizar os computadores como uma ferramenta para ampliar as possibilidades de busca de soluções de projeto.

Agradecimentos

Os autores agradecem às turmas que cursaram a disciplina AU303 desde 2004 por seu entusiasmo e interesse nas teorias computacionais aplicadas ao projeto.

Referências

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.