DESIGN GENERATIVO PARA BUILDING INFORMATION MODELING

Bruno Ferreira [(1)], António Leitão [(1)]

(1) INESC-ID/IST, Lisboa

Resumo
O Design Generativo (DG) é uma abordagem algorítmica para a criação de modelos arquitetónicos, permitindo mecanizar tarefas, produzir geometrias complexas, e otimizar o modelo criado. Uma vez que as aplicações de *Computer-Aided Design* (CAD) são muito utilizadas em Arquitetura e Engenharia, foram já criadas diversas ferramentas que combinam o DG com as aplicações CAD. Um exemplo a ter em conta é o Rosetta, um ambiente de programação para DG, desenvolvido para arquitetos e engenheiros, que permite aos utilizadores escolher, não só em que linguagem querem programar, mas também em que ferramenta CAD querem produzir os seus modelos. Mais recentemente, o paradigma CAD tem vindo a ser substituído pelo paradigma BIM, o que reduz a utilidade das ferramentas de DG que apenas lidam com o paradigma CAD. Como as vantagens do DG se estendem naturalmente ao paradigma BIM, propomos uma solução que expande o Rosetta, permitindo a criação de programas de DG que tiram proveito das capacidades do paradigma BIM. A nossa solução permite não só elevada portabilidade entre ferramentas BIM, mas ainda alguma portabilidade entre os paradigmas CAD e BIM. De forma a avaliar a nossa solução, apresentamos um conjunto de casos de estudo onde mostramos o uso do Rosetta em combinação com o Revit.

1. Introdução

As aplicações de *Computer-Aided Design* (CAD) aumentaram a eficiência com a qual se realizavam diversas tarefas de design, permitindo aos arquitetos criar modelos cada vez mais precisos e complexos. No entanto, mesmo recorrendo a ferramentas CAD, a modelação e alteração de geometrias complexas continua a ser desafiante pois estas ferramentas ainda não dão ao utilizador o nível de flexibilidade que é, por vezes, necessário.

O Design Generativo (DG) é uma abordagem que resolve estes problemas [1]. DG pode ser descrito como a criação de formas através de algoritmos [2]. Estes algoritmos, implementados

sob a forma de programas, permitem a fácil geração de variações do mesmo modelo [3]. Estes programas podem também gerar geometria que seria difícil de criar por meios manuais.

Reconhecendo as vantagens da abordagem de DG, diversas ferramentas foram criadas com o intuito de permitir a criação de programas de DG. Muitas destas ferramentas foram criadas a pensar em arquitetos com poucos conhecimentos de programação e são, por isso, facilmente utilizáveis.

Uma dessas ferramentas é o Rosetta. Esta ferramenta possui um ambiente de programação para DG que permite aos utilizadores desenvolverem programas numa das diversas linguagens disponíveis. Estes programas são usados para gerar modelos em diversas ferramentas CAD [4]. Atualmente, as ferramentas CAD estão a ser substituídas por ferramentas de *Building Information Modeling* (BIM), inviabilizando as ferramentas de DG como o Rosetta, que foram desenvolvidas com as aplicações CAD em mente. Isto acontece devido ao facto das ferramentas CAD lidarem principalmente com geometria, enquanto que as ferramentas BIM lidam com objetos BIM paramétricos. Estes são definidos por um conjunto de regras paramétricas, assim como um conjunto de propriedades como material, preço, e fabricante, entre outras [5].

Estas propriedades permitem às ferramentas BIM detetar problemas nos modelos, como sejam tubagens que atravessam vigas estruturais. Para além disto, as regras paramétricas adaptam os objetos à sua utilização no projeto, permitindo, por exemplo, que uma porta inserida numa parede assuma as medidas corretas. Finalmente, toda a informação necessária durante o ciclo de vida do edifício é armazenada no modelo, podendo ser utilizada para gerar um vasto conjunto de documentos, desde orçamentos a documentos de fabricação [5].

Estas capacidades implicam, não só, mudar a forma de trabalho dos utilizadores, mas também a forma de trabalho das ferramentas de DG, de modo a possuírem funcionalidades que permitam tirar proveito do paradigma BIM, algo para o qual não estavam preparadas.

Desenvolver programas de DG para ferramentas BIM é um problema atual, sendo que a maioria das soluções existentes envolvem a utilização de *Application Programing Interfaces* (APIs) disponibilizadas pelas ferramentas. No entanto, estas APIs requerem conhecimentos avançados de programação e encontram-se escritas em linguagens de programação complexas como C# e C++. Estas características tornam-nas inadequadas a utilizadores com reduzida experiência de programação. Para resolver este problema, propomos uma solução que permite a estes utilizadores criar programas de DG no paradigma BIM. Na próxima secção descrevemos o trabalho relacionado analisado para o desenvolvimento da solução.

2. Trabalho Relacionado

Diversas ferramentas foram analisadas, de forma a guiar o desenvolvimento da nossa solução. O foco principal foram ferramentas que permitissem explorar DG com ferramentas BIM, assim como plug-ins que atinjam o mesmo objetivo.

2.1 Grasshopper e Lyrebird

Grasshopper é uma linguagem de programação visual desenvolvida para arquitectos e disponibilizada como um plug-in para a aplicação CAD Rhinoceros 3D. Programas escritos com esta linguagem representam um grafo de fluxo de dados que consiste num conjunto de componentes e ligações entre eles. Os componentes podem representar funções, parâmetros ou elementos geométricos [6]. Uma vez que estes componentes podem ser ligados entre si, os utilizadores podem combiná-los de forma a produzir algoritmos complexos.

Para além destes componentes, o Grasshopper disponibiliza componentes de *scripting* que permitem escrever programas em VB.NET, C# e Python [6].

Uma vez que o Grasshopper é uma linguagem visual, esta é fácil de aprender e utilizar, tornando-a popular entre arquitetos. No entanto, o seu aspeto gráfico é também uma desvantagem, pois à medida que os programas crescem, estes tornam-se difíceis de compreender e manipular. A Figura 1 ilustra um programa que torna evidente estes problemas.

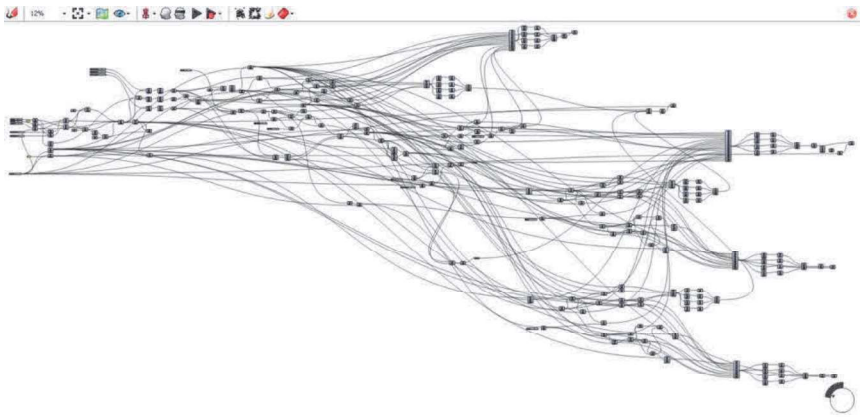

Figura 1: Exemplo de programa em Grasshopper. (source: http://workshopsfactory.wordpress.com/files/2009/08/parametric-table- grasshopper.jpg).

O Grasshopper apenas funciona com a ferramenta CAD Rhinoceros, mas a sua funcionalidade tem vindo a ser estendida com *plug-ins*, como o Lyrebird, que tornam possível a utilização da linguagem com outras aplicações, nomeadamente aplicações BIM.

O Lyrebird é um *plug-in*, desenvolvido pelos LMN Architects, para funcionar como ferramenta de interoperabilidade entre o Grasshopper 3D e o Revit. O Lyrebird tem como foco principal transferir informação entre aplicações, em vez de traduzir geometria entre elas [7]. Este *plug-in* permite a utilização do Grasshopper para estruturar a informação necessária para produzir objetos no Revit. Esta informação é utilizada para identificar e criar instâncias das famílias corretas com os parâmetros desejados. Por exemplo, para criar uma viga, é criada uma linha no lado do Grasshopper, que é passada para o Revit de forma a, em conjunto com a família de objetos a utilizar, instanciar a viga. Um exemplo de utilização pode ser visto na figura 2.

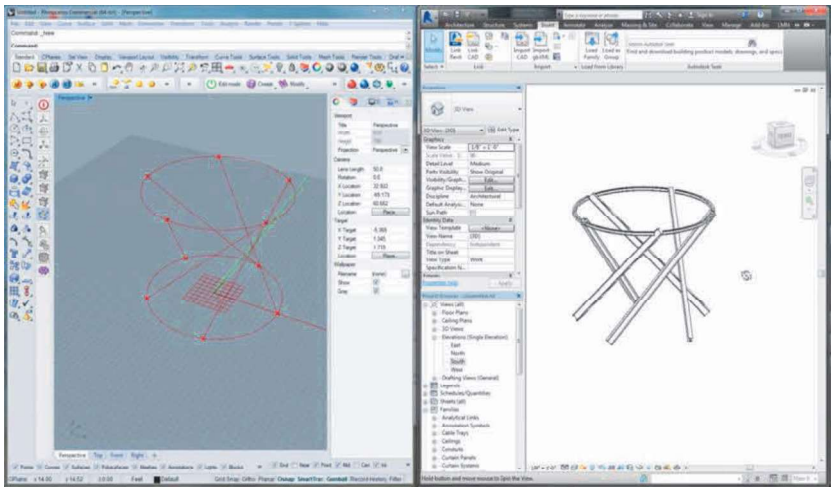

Figura 2: Criação de vigas no Lyrebird com linhas criadas em Rhinoceros (source: https://lmnarchitects.com/tech-studio/bim/superb-lyrebird/).

2.2 Dynamo

Dynamo é um *plug-in* para o Revit fortemente influenciado por linguagens de programação visuais como o Grasshopper. Tal como no Grasshopper, os utilizadores criam um grafo de fluxo de dados, utilizando nós que se encontram ligados entre si com cabos, associados a portos que cada nó contém. Os nós podem representar diversos elementos do Revit, como linhas, funções matemáticas e objetos BIM. Os utilizadores podem também definir nós personalizados, de forma a estender a funcionalidade base do Dynamo [8].

O Dynamo também suporta nós de código, que são elementos que podem conter pequenos programas escritos numa linguagem de programação como o Python.

2.3 GenerativeComponents

GenerativeComponents é um sistema paramétrico e associativo para a ferramenta BIM Microstation da Bentley.

Este sistema é baseado em propagação, ou seja, o utilizador tem de determinar as regras, ligações e parâmetros que definem a geometria desejada. Este sistema consiste num grafo dirigido acíclico gerado por dois algoritmos: um que ordena o grafo e outro que propaga os valores por este [9].

O GenerativeComponents oferece diversas formas de interação aos utilizadores, tendo em conta os seus diferentes níveis de conhecimentos. A primeira é através de uma interface gráfica que permite a manipulação direta de geometria. A segunda recorre à linguagem GCScript, permitindo criar ligações entre objetos com pequenos programas. Por fim, é também possível escrever programas na linguagem C#, permitindo a criação de algoritmos complexos.

O GenerativeComponents demonstra que uma linguagem visual pode ser mais fácil de aprender mas que à medida que o utilizador evolui e quer produzir programas mais complexos, este tende a transitar para uma linguagem mais textual.

2.4 RevitPythonShell

O RevitPythonShell é um *plug-in* desenvolvido para o Revit que permite aceder à API do mesmo na linguagem Python, simplificando assim a sua utilização [10].

O *plug-in* utiliza o IronPython como linguagem e um editor de código fornecido pela Python Tools. Com este editor, os utilizadores têm acesso a um *Read-Eval-Print-Loop* (REPL) que lhes permite experimentar mais facilmente as funcionalidades da API. Com a REPL, basta apenas escrever uma expressão, avaliá-la, ver os resultados e passar à próxima expressão [10].

2.5 Comparação

A tabela 1 apresenta uma comparação entre as ferramentas analisadas.

Tabela 1: Comparação entre as ferramentas analisadas. O símbolo ✓- indica suporte limitado, ✓ indica suporte e × indica o não suporte. Na última coluna o número de ✓ traduz a quantidade de funcionalidade suportada.

	Visual	**Textual**	**Suporte BIM**	**Operações Geométricas**	**Operações BIM**
Grasshopper	✓	✓-	×	✓	×
Lyrebird	✓	×	✓	×	✓✓
Dynamo	✓	✓-	✓	✓	✓✓✓
GenerativeComponents	✓	✓	✓	✓	✓✓✓
RevitPythonShell	×	✓	✓	✓	✓✓✓

Como é possível verificar, muitas destas ferramentas: (1) tiram proveito de linguagens de programação visuais, dada a sua simplicidade e facilidade de aprendizagem; (2) oferecem algum suporte a linguagens de programação textual, apesar de se restringirem a pequenos programas; e (3) oferecem suporte a operações BIM, demonstrando a necessidade de explorar as ferramentas BIM com DG.

2.6 Problemas a Resolver

Uma das desvantagens das ferramentas analisadas está relacionada com a utilização de linguagens visuais. Apesar de mais fáceis de aprender, estas não escalam bem com a complexidade dos programas, ou seja, os programas tornam-se mais difíceis de perceber e editar [11].

Em relação às ferramentas que oferecem suporte a linguagens textuais, muitas destas tiram proveito de linguagens profissionais, como C#, que são difíceis de serem utilizadas por principiantes. Para além disto, algumas das ferramentas requerem conceitos avançados de programação, o que dificulta a aprendizagem das mesmas.

Por fim, todas estas ferramentas estão ligadas a apenas uma ferramenta BIM/CAD e utilizam linguagens de programação muito específicas. Caso o utilizador queira mudar de linguagem ou ferramenta de modelação, terá de reescrever o seu programa.

Estes são os problemas que pretendemos resolver com a nossa solução.

3. Design Generativo para BIM

De forma a permitir que os utilizadores desenvolvam programas que escalam bem com a complexidade, desenvolvemos uma solução baseada em linguagens de programação textuais. No entanto, como estas linguagens são mais difíceis de aprender, decidimos utilizar linguagens que têm qualidades pedagógicas, como Python e Racket. Além disto, queremos disponibilizar operações BIM fáceis de compreender e utilizar, assim como oferecer portabilidade entre aplicações BIM.

Por estas razões, a nossa solução é constituída por três elementos: (1) um *Integrated Development Environment* (IDE) onde o utilizador escreve os seus programas; (2) uma Camada de Abstração que fornece as operações necessárias; e (3) um *plug-in* que permite comunicar e produzir os resultados nas ferramentas BIM [12]. A arquitetura da solução pode ser vista na figura 3.

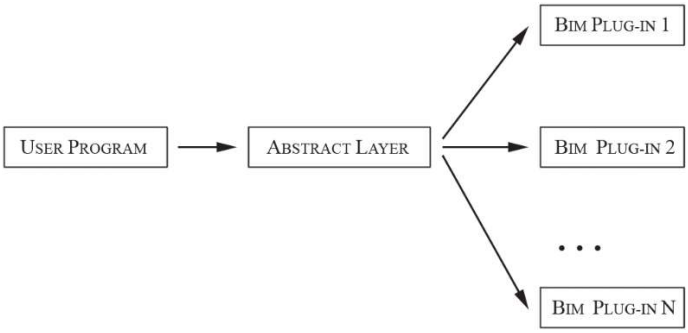

Figura 3: Arquitetura da solução.

Nas próximas secções exploramos cada um destes componentes.

3.1 Integrated Development Environment

A solução utiliza linguagens de programação textuais, dado a sua flexibilidade e escalabilidade, embora estas sejam mais complexas para principiantes. Para além da linguagem, é necessário um IDE com funcionalidades adequadas para principiantes, para escrever e depurar os programas. Estas funcionalidades são de extrema importância, pois ajudam a ultrapassar as barreiras de entrada das linguagens textuais. Por estas razões, decidimos usar o Rosetta e tirar proveito do IDE que este utiliza, o DrRacket, um ambiente de programação pedagógico [13] pensado para principiantes.

Ao utilizar o Rosetta, os utilizadores podem escolher em que linguagem de programação querem escrever os seus programas. Racket, Python, Processing e Javascript são alguns exemplos de linguagens disponíveis. Esta possibilidade ajuda utilizadores que saibam algumas destas linguagens, pois elimina a necessidade de terem de aprender a programar numa nova linguagem. No entanto, mesmo que tenham de aprender, todas as linguagens disponibilizadas são pedagógicas e usadas no ensino, o que facilita o processo.

3.2 Camada de Abstração

Para escreverem programas de DG que produzam os resultados desejados, os utilizadores precisam de funções que lhes permitam ter acesso a funcionalidades da ferramenta BIM. Estas funções devem permitir aos utilizadores criar objetos BIM e definir toda a informação necessária para produzir um projeto completo.

Por exemplo, se um utilizador quiser produzir uma parede com uma porta, este necessita de funções capazes de produzir ambos os elementos. A função capaz de criar paredes deverá receber informação acerca da altura, comprimento, posição e tipo de parede, de forma a criar o objeto BIM correto. Após criar a parede, a função que cria a porta será utilizada, recebendo não só a posição e tipo da porta, mas também a parede onde a porta será colocada, criando uma dependência entre os objetos.

Todas estas funções têm de ser disponibilizadas numa Camada de Abstração, uma biblioteca de funções que simplifica e traduz as necessidades do utilizador para código, de forma a que possam ser utilizadas nas diversas aplicações BIM. De forma a implementar esta camada, decidimos expandir o Rosetta, tirando proveito das funções e abstrações já criadas para as ferramentas CAD, como sistemas de coordenadas e entidades geométricas e criámos novas funcionalidades, agora adaptadas ao paradigma BIM.

3.3 *Plug-In*

Finalmente, de forma a permitir que os utilizadores executem os seus programas e produzam os modelos na aplicação BIM desejada, foi criado um componente que permite ao Rosetta comunicar com uma aplicação BIM.

Uma vez que as ferramentas BIM disponibilizam uma API que expõe as suas funcionalidades, este componente pode ser implementado como um *plug-in*, escrito com a API mencionada.

Todas as funções implementadas na Camada de Abstração terão uma função correspondente no *plug-in*. Quando uma função é executada no programa, a informação necessária é enviada para o *plug-in* e a função correspondente é executada na ferramenta BIM, produzindo os resultados na mesma.
Cada ferramenta BIM necessita de um plug-in para comunicar com o Rosetta, e para testarmos a nossa solução começamos por introduzir um *plug-in* para o Revit. Este foi desenvolvido utilizando a API do Revit, tendo sido escrito na linguagem C#. No entanto, toda a complexidade deste é escondida pela Camada de Abstração mencionada anteriormente, sendo esse um dos seus principais objetivos. Desta forma, os utilizadores podem utilizar funções simples e intuitivas, estando toda a complexidade necessária para trabalhar com as ferramentas escondida no *plug-in*. É neste componente que todos os cálculos para criar os objetos são efetuados, assim como toda a gestão de transações aplicacionais necessárias para visualizar os resultados.

4. Avaliação

De forma a avaliar a nossa solução, desenvolvemos um conjunto de casos de estudo que nos permitiram demonstrar as diversas capacidades que pretendíamos alcançar. Nas próximas

secções iremos descrever alguns modelos desenvolvidos que ilustram as funcionalidades BIM da solução, a portabilidade desta e a sua aplicação.

4.1 Funcionalidades BIM

Dado que o que pretendíamos atingir era a capacidade de utilizar o paradigma BIM com DG, começámos por desenvolver um modelo que tirasse proveito das capacidades das aplicações BIM, nomeadamente os objetos e as ligações e dependências entre estes. Para tal, desenvolvemos para BIM um modelo que tinha sido previamente desenvolvido com DG para CAD: as *Dubai Towers*, criadas por Sama Dubai. Este modelo tinha sido produzido por um programa que gerava apenas geometria e era nosso objetivo produzir o mesmo modelo, mas com conceitos BIM. Utilizando as novas funções que introduzimos na Camada de Abstração do Rosetta, foi possível produzir o modelo em Revit, visível na figura 4.

Ao examinarmos o modelo, pudemos verificar que todos os elementos eram objetos BIM criados corretamente. Verificámos também que os elementos estavam devidamente associados, estando cada laje associada a um nível e cada parede associada a um nível de base e topo, o que condicionava a altura das mesmas. Para além disto, foi também possível criar as paredes com as propriedades esperadas de um elemento deste tipo, apesar da sua forma irregular.

4.2 Portabilidade

Uma das principais funcionalidades oferecidas pelo Rosetta é a portabilidade dos programas de DG entre ferramentas CAD. Desta forma, quisemos manter esta funcionalidade para BIM.

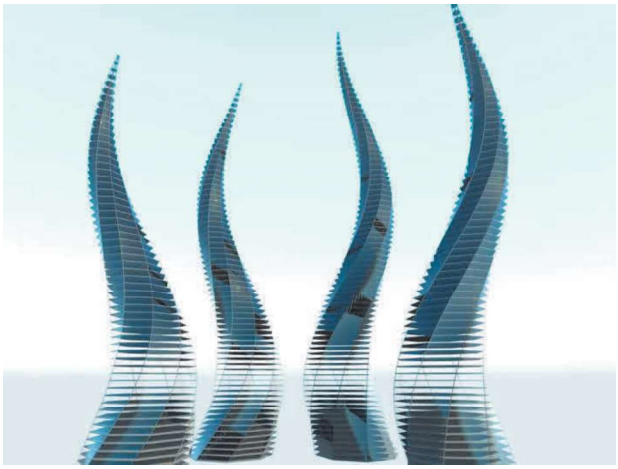

Figura 4: *Dubai Towers*, produzidas através de DG no Revit.

Para demonstrar que seria possível obter resultados semelhantes com o mesmo programa em duas ferramentas BIM, desenvolvemos um programa que produzia um modelo das *Absolute Towers*, criadas pelos MAD Architects. Dado que introduzimos funcionalidades a pensar em conceitos gerais BIM, utilizamos o programa para produzir o modelo em Revit e em ArchiCAD, a segunda ferramenta BIM que está atualmente a ser introduzida no Rosetta. Com algum trabalho de uniformização, facilitado pela Camada de Abstração, foi possível disponibilizar as mesmas funções para ambas as ferramentas, o que permitiu utilizar o mesmo programa para produzir os modelos visíveis na figura 5.

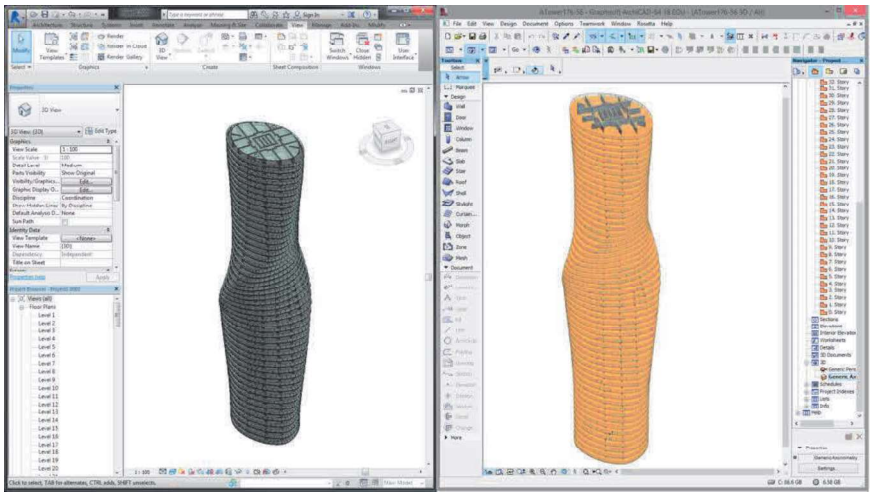

Figura 5: *Absolute Towers*, produzidas no Revit (à esquerda) e no ArchiCAD (à direita).

Como é visível na figura, o modelo obtido em ambas as ferramentas BIM era idêntico, possuindo o mesmo tipo de relações entre os objetos, assim como os mesmos tipos de objetos. Conseguimos assim obter portabilidade entre ferramentas BIM, tal como acontecia com as ferramentas CAD. E uma vez que a portabilidade é obtida através do programa, sendo o modelo gerado nativamente na ferramenta alvo, não existem perdas de informação, um problema que identificámos quando passávamos o modelo de uma ferramenta para a outra usando IFC.

Para além da portabilidade entre ferramentas BIM, em modelos simples conseguimos também alcançar portabilidade entre ferramentas CAD e BIM. Ao introduzir funções simples que representam conceitos BIM para as ferramentas CAD, é possível ter algumas funcionalidades BIM disponíveis em todas as ferramentas. Utilizando estas funções foi possível produzir um modelo simples com lajes e uma fachada sinusoidal composta por vigas. Este modelo foi produzido por um único programa de DG, gerando o mesmo modelo em todas as ferramentas de visualização suportadas pelo Rosetta. O resultado é visível na figura 6.

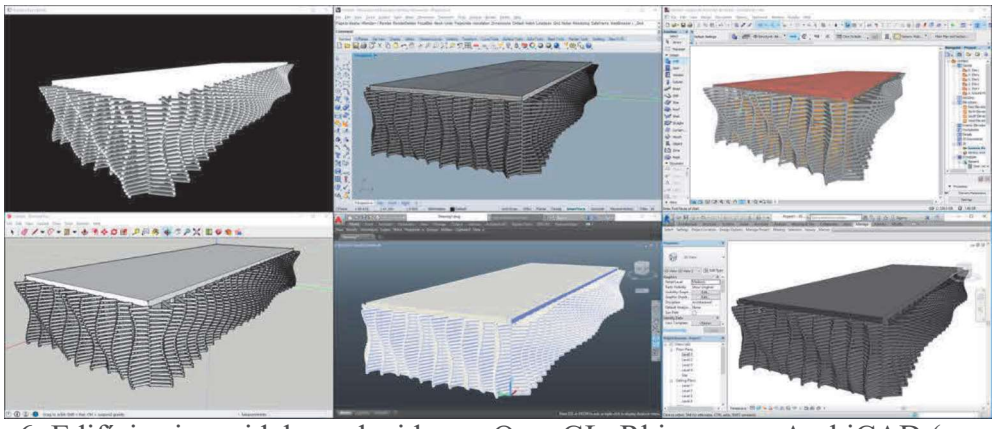

Figura 6: Edifício sinusoidal, produzido em OpenGL, Rhinoceros, ArchiCAD (em cima), Sketchup, AutoCAD e Revit (em baixo).

A portabilidade é uma funcionalidade de extrema utilidade em DG, pois permite tirar proveito dos pontos fortes de cada ferramenta, como seja a performance de ferramentas como o OpenGL, para visualizar variações simples e iniciais com rapidez, passando numa fase mais avançada para ferramentas BIM com mais detalhe e informação.

5. Conclusão

DG foi introduzido em Arquitetura como uma forma de utilizar algoritmos para explorar formas complexas, difíceis de obter por meios tradicionais. DG foi inicialmente combinado com ferramentas CAD mas, com os avanços na área da Arquitetura, passou a existir a necessidade de combinar DG com ferramentas BIM. Assim, diversas ferramentas têm sido desenvolvidas com o intuito de permitir ao utilizador produzir modelos BIM com programação. No entanto, a maioria destas ferramentas condiciona os utilizadores a linguagens de programação inadequadas ou limitam-no a uma só ferramenta BIM ou CAD.

Para resolver estes problemas propomos uma solução que expande o Rosetta, um ambiente de programação para DG adequado para principiantes. Através da nossa solução, o utilizador pode criar programas de DG para ferramentas BIM, tirando proveito das diversas funcionalidades que o paradigma oferece. Para além disto, o utilizador tem a liberdade de escolher em que linguagem de programação quer programar, produzindo programas que podem ser facilmente portados entre ferramentas BIM e, no caso de programas simples, também ferramentas CAD. Através dos casos de estudo, mostramos a capacidade de produzir modelos BIM corretos com programas DG desenvolvidos com a nossa solução. É também possível verificar a utilidade da portabilidade, permitindo alternar entre as ferramentas mais adequadas a cada fase do projeto, utilizando ferramentas com melhor performance para variações, e outras capazes de mais detalhe em fases finais.

Como trabalho futuro pretendemos expandir as funcionalidades disponibilizadas pela ferramenta, assim como adicionar suporte para mais ferramentas BIM. Outro objetivo será explorar formas de otimização de modelos, em combinação com as ferramentas BIM, permitindo gerar e avaliar variações do modelo automaticamente.

Referências

[1] J. McCormack, A. Dorian, and T. Innocent, "Generative design: a paradigm for design research," in Proceedings of Futureground (2004), Design Research Society, Melbourne, Australia, 2004.

[2] R. Garber, BIM Design, Realising the Creative Potential of Building Information Modelling. Hoboken, N.J.: Wiley 2014.

[3] R. Fernandes, "Generative Design: a new stage in the design process," Tese de Mestrado, Instituto Superior Técnico, Lisboa, Portugal, 2013.

[4] J. Lopes, and A.Leitão, "Portable generative design for CAD applications," in Proceedings of the 31st annual conference of the Association for Computer Aided Design in Architecture (2011), Calgary/Banff, Canada, 2011, pp. 196-203.

[5] C. M. Eastman, P. Teicholz, R. Sacks, and K. Liston, BIM handbook : a guide to building information modeling for owners, managers, designers, engineers, and contractors. Hoboken, N.J.: Wiley, 2008.
[6] A.Payne, and R. Issa, Grasshopper Primer, Zen Edition. Robert McNeel & Associates, 2009.
[7] T. Logan, "Superb Lyrebird", February, 2014. Acedido em Abril de 2015. Disponível em: http://lmnts.lmnarchitects.com/bim/superb-lyrebird/
[8] Dynamo. Acedido em Dezembro de 2014. Disponível em: http://dynamobim.com/learn/
[9] R. Aish, and R. Woodbury, "Multi-level interaction in parametric design," in Smart Graphics (2005), Springer, Berlin, Germany, 2005.
[10] D. Thomas, "Introducing RevitPythonShell", December, 2009. Acedido em Abril de 2015. Disponível em: http://darenatwork.blogspot.pt/2009/12/introducing-revitpythonshell.html
[11] A. Leitão, and L. Santos, "Programming languages for generative design," in Respecting fragile places - Proceedings of the 29th Conference on Education in Computed Aided Architectural Design in Europe (2011), Ljubljana, Slovenia, 2011, pp. 549-557.
[12] B. Ferreira, and A. Leitão, "Generative design for building information modeling," in Real Time - Proceedings of the 33rd International Conference on Education and Research in Computed Aided Architectural Design in Europe (2015), Vienna, Austria, 2015, pp. 635- 644.
[13] R. B. Findler, C. Flanagan, M. Flatt, S. Krishnamurthi, and M.Felleisen, "DrScheme: A pedagogic environment for Scheme," in Programming Languages: Implementations, Logics, and Programs, Springer, 1997, pp. 369-388.