



**UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS DE CRATEÚS
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

DANIEL HENRIQUE DE BRITO

**GERAÇÃO PROCEDURAL DE MODELOS ARQUITETURAIS COM GEOMETRIA
ARREDONDADA UTILIZANDO *SELECTION EXPRESSIONS (SELEX)***

CRATEÚS

2021

DANIEL HENRIQUE DE BRITO

GERAÇÃO PROCEDURAL DE MODELOS ARQUITETURAIS COM GEOMETRIA
ARREDONDADA UTILIZANDO *SELECTION EXPRESSIONS (SELEX)*

Trabalho de Conclusão de Curso apresentado ao
Curso de Graduação em Ciência da Computação
do Campus de Crateús da Universidade Federal
do Ceará, como requisito parcial à obtenção do
grau de bacharel em Ciência da Computação.

Orientador: Prof. Me. Arnaldo Barreto
Vila Nova

Coorientador: Prof. Me. Ítalo Mendes
da S. Ribeiro

CRATEÚS

2021

DANIEL HENRIQUE DE BRITO

GERAÇÃO PROCEDURAL DE MODELOS ARQUITETURAIS COM GEOMETRIA
ARREDONDADA UTILIZANDO *SELECTION EXPRESSIONS (SELEX)*

Trabalho de Conclusão de Curso apresentado ao
Curso de Graduação em Ciência da Computação
do Campus de Crateús da Universidade Federal
do Ceará, como requisito parcial à obtenção do
grau de bacharel em Ciência da Computação.

Aprovada em:

BANCA EXAMINADORA

Prof. Me. Arnaldo Barreto Vila Nova (Orientador)
Universidade Federal do Ceará (UFC)
Campus de Crateús

Prof. Me. Ítalo Mendes da S. Ribeiro (Coorientador)
Universidade Federal do Ceará (UFC)
Campus de Crateús

Prof. Dr. Joaquim Bento Cavalcante Neto
Universidade Federal do Ceará (UFC)
Departamento de Computação - Campus do Pici

Prof. Dr. Markos Oliveira Freitas
Universidade Federal do Ceará (UFC)
Campus de Russas

A minha família.

AGRADECIMENTOS

Primeiramente, agradeço aos meus familiares e pessoas queridas, que sempre estiveram presentes para me oferecer ajuda nos momentos de dificuldade.

Agradeço aos professores Arnaldo Barreto e Ítalo Ribeiro, por me orientarem no desenvolvimento deste Trabalho de Conclusão de Curso.

Agradeço aos professores e professoras que me inspiram na busca pelo conhecimento, o qual nos traz uma luz para o entendimento e a resolução das questões existentes no universo. Em especial, à professora Lisieux Andrade, por seus ensinamentos na disciplina do Projeto de Pesquisa Científica e Tecnológica, e ao professor Anderson Almada, por ter me incentivado no desenvolvimento e possibilitado a publicação do HáLugar, meu primeiro aplicativo.

Por fim, agradeço à Universidade Federal do Ceará, por ter tornado este sonho realidade, e também pela concessão das diversas bolsas, as quais me proveram conhecimentos muito valiosos.

“Se eu vi mais longe, foi por estar sobre ombros
de gigantes.”

(Bernard de Chartres)

RESUMO

Modelar ambientes virtuais é uma tarefa árdua, podendo requerer grande tempo e esforço da parte dos artistas, se estes optarem por gerar cada objeto manualmente. Baseado nisto, a modelagem procedural surgiu com a proposta de trazer alguns benefícios no que se refere à geração das diversas camadas de ambientes virtuais, como vegetação, terrenos, estradas, rios, edifícios e cidades, por exemplo. Tais modelos, por sua vez, podem ser aplicados em diversos cenários, como planejamento urbano, jogos, filmes, simulações, entre outros. Contudo, também surgiram alguns desafios, como a falta de intuitividade na utilização de alguns *frameworks* existentes, a semântica em relação à disposição dos elementos nos modelos, o grau de realismo com que eles são apresentados, e a geração de formas mais complexas, como estruturas arredondadas. Diversas pesquisas tentam mitigar tais dificuldades, assim, seguindo esta premissa, o presente trabalho descreve uma abordagem pioneira para a resolução do problema da geração de modelos arquiteturais com geometria arredondada utilizando *Selection Expressions (SELEX)*, por meio da aplicação de técnicas de deformação, permitindo o arredondamento de estruturas no sentido externo e interno.

Palavras-chave: Ambientes virtuais. Modelagem procedural. Modelagem arquitetural. Gramáticas. *Selection Expressions*. Deformação.

ABSTRACT

Modeling virtual environments is an arduous task, and might require great time and effort on the part of the artists if they choose to generate each object manually. Based on this, the procedural modeling came up with the proposal to bring some benefits with regard to the generation of the various layers of virtual environments, such as vegetation, land, roads, rivers, buildings, and cities, for example. Such models, in turn, can be applied in various scenarios, such as urban planning, games, movies, simulations, among others. However, some challenges also emerged, such as the lack of intuitiveness in the use of some existing frameworks, the semantics in relation to the layout of elements in the models, the degree of realism with which they are displayed, and the generation of more complex forms, such as rounded structures. Several researches try to mitigate these challenges, thereby following this premise, the present work describes a pioneering approach to solve the problem of generating architectural models with rounded geometry using *SELEX*, through the application of deformation techniques, allowing the rounding of structures in the external and internal directions.

Keywords: Virtual environments. Procedural modeling. Architectural modeling. Grammars. Selection Expressions. Deformation.

LISTA DE FIGURAS

Figura 1 – Modelo de ambiente virtual criado com o framework <i>SketchaWorld</i> : (a) ambiente natural com estrada cruzando o rio, (b) ambiente virtual final, (c) visão em perspectiva da área urbana.	21
Figura 2 – Modelagem procedural a partir de uma imagem.	22
Figura 3 – Modelo de uma casa (à esquerda), com vista superior do primeiro andar (ao centro) e vista de dentro (à direita).	23
Figura 4 – Representação geométrica da <i>CGA Shape</i> . Esquerda: definição de uma caixa delimitadora contendo uma forma primitiva. Direita: Modelo simples de uma construção utilizando três primitivas.	24
Figura 5 – Exemplo de aplicação das regras da <i>CGA Shape</i> . À esquerda: Um <i>design</i> básico de fachada. À direita: Uma divisão simples que pode ser utilizada para os três andares superiores.	26
Figura 6 – Variações estocásticas de modelos de edifícios.	28
Figura 7 – Diferentes construções em um ambiente suburbano.	29
Figura 8 – Exemplo ilustrativo do uso da <i>CGA++</i> : (a) gramática, (b) árvore de formas e (c) resultado.	31
Figura 9 – Opções de navegação na árvore. Neste caso, a e b são valores representando as formas dos nós 1 e 2, respectivamente, enquanto X e Y são rótulos.	32
Figura 10 – Exemplo do uso de formas recuperáveis.	33
Figura 11 – Representação do manipulador de eventos.	35
Figura 12 – Representação de bloco perimetral. Acima: estratégia de modelagem desejada e os parâmetros envolvidos. Meio: exemplos de resultados para diferentes escolhas de parâmetros. Abaixo: exemplo de um resultado de planejamento urbano no mundo real.	37
Figura 13 – Modelos de arranha-céus ecológicos. Acima: principais etapas da abordagem de modelagem. Abaixo: visualizações em foco do resultado do exemplo e resultados para diferentes valores para área com gramado.	38
Figura 14 – Modelos de fachadas com elementos e alinhamentos aleatórios. Acima: abordagem de solução geral, na qual as linhas de divisão são mostradas em vermelho). Abaixo: exemplo de resultados para diferentes parâmetros.	39

Figura 15 – Exemplo de operação <i>split</i> , mostrando o resultado para diferentes valores de entrada, neste caso, 6 e 11.	39
Figura 16 – Comparação entre o <i>layout</i> gerado pela <i>CGA Shape</i> e <i>SELEX</i>	41
Figura 17 – Ilustração dos paradigmas de modelagem <i>SELEX</i> e <i>CGA Shape</i>	43
Figura 18 – As ilustrações (b), (c), (d), (e) e (f), representam diferentes perspectivas da árvore de formas em relação ao <i>design</i> da fachada (a).	44
Figura 19 – Para a forma atual (<i>leftWall</i>), é demonstrada uma forma contida (<i>win</i>), uma forma anexada (<i>balcony</i>) e uma forma conectada (<i>cornerWall</i>).	45
Figura 20 – Instruções <i>SELEX</i> para gerar a fachada da Figura 17(e).	46
Figura 21 – Grafo abstrato da seleção de nós.	48
Figura 22 – Exemplo da utilização de seletores.	49
Figura 23 – Exemplo da utilização de funções de agrupamento.	50
Figura 24 – Exemplo da utilização de <i>actions</i>	50
Figura 25 – Comparação do ajuste com e sem o uso de rótulos.	51
Figura 26 – Dois exemplos de alinhamento.	52
Figura 27 – Exemplo da árvore de formas gerada pela <i>CGA Shape</i> em (a), e pela <i>SELEX</i> em (b), com base na fachada (c).	52
Figura 28 – Estatísticas da <i>SELEX</i> e <i>CGA Shape</i> para geração de fachadas.	53
Figura 29 – Exemplo de modelagem utilizando <i>SELEX</i>	54
Figura 30 – Árvore de formas construída com base no modelo final da Figura 29(o). . .	55
Figura 31 – À esquerda, um exemplo de deformação livre dos objetos do cenário, a partir da mudança dos pontos da grade cúbica visível à direita.	55
Figura 32 – Objeto de controle de origem (a) e objetos de destino (b), (c), (d). Neste caso, O_S e O_D representam os respectivos centros dos objetos.	56
Figura 33 – Deformação de uma bola de futebol (a) com base nos objetos (b), (c) e (d), da Figura 32.	56
Figura 34 – Técnica de deformação de forma livre de Sederberg: a) antes e b) depois da transformação.	57
Figura 35 – Técnica de deformação de forma livre utilizando <i>Non-Uniform Rational B-Spline (NURBS)</i> : a) antes e b) depois da transformação.	57
Figura 36 – Operação de deformação introduzida por Thaller <i>et al.</i> (2013).	58

Figura 37 – O modelo inicial de uma casa (a) é deformado pelo usuário, enquanto preserva as propriedades típicas, como a ortogonalidade da parede e a disposição linear do piso (b).	58
Figura 38 – Aplicação de regras de modelagem da <i>Generalized Grammar</i> (G^2).	60
Figura 39 – Exemplo típico de beirais passando ao redor de uma borda.	61
Figura 40 – Aplicação de sucessivas deformações de forma livre através da G^2	61
Figura 41 – Muralha que se estende em um terreno.	62
Figura 42 – Aplicação aninhada de deformações de forma livre em uma <i>split grammar</i> . .	63
Figura 43 – Aplicação de regras de subdivisão em objetos.	64
Figura 44 – A aplicação de deformações em uma fachada reta, que é definida utilizando uma <i>split grammar</i> (a), produz um número diferente de janelas nas partes laterais (b). A imagem inferior (c) mostra as divisões que são realizadas no espaço de coordenadas locais (não deformadas) para alcançar o resultado mostrado após a deformação (b).	65
Figura 45 – Prédio comercial com estrutura arredondada.	66
Figura 46 – Uma parede dividida em nove partes, por meio de diferentes sistemas de coordenadas (cartesiana, cilíndrica e esférica).	67
Figura 47 – Variações de modelos de abóbodas.	68
Figura 48 – Exemplo arquitetural de um domo e de uma torre de castelo.	69
Figura 49 – Exemplo que está além da capacidade de modelagem da <i>SELEX</i>	70
Figura 50 – Representação do sistema de coordenadas do Blender: (a) eixos, (b) vértice, (c) aresta, (d) face.	71
Figura 51 – Representação dos índices de um objeto: (a) vértices, (b) arestas e (c) faces.	72
Figura 52 – <i>Wireframe</i> da representação do lado (a) posterior, (b) esquerdo, (c) direito e (d) frontal, em relação aos (e) eixos cartesianos.	72
Figura 53 – Fluxograma de execução.	73
Figura 54 – Diagrama de classes referente à estrutura da árvore de formas.	75
Figura 55 – Modelo de massa inicial.	77
Figura 56 – Inclusão de forma virtual.	77
Figura 57 – Resultado obtido após a operação de extrusão.	78
Figura 58 – Modelo final com estrutura frontal arredondada.	80

Figura 59 – Exemplos de variação do parâmetro (1) <i>type</i> , com (2) <i>direction</i> valorado como <i>outside</i> : (a) Forma original, (b) <i>front</i> , (c) <i>left</i> , (d) <i>right</i> , (e) <i>top</i> , (f) <i>bottom</i> .	80
Figura 60 – Exemplos de variação do parâmetro (1) <i>type</i> , com (2) <i>direction</i> valorado como <i>inside</i> : (a) Forma original, (b) <i>front</i> , (c) <i>left</i> , (d) <i>right</i> , (e) <i>top</i> , (f) <i>bottom</i> .	81
Figura 61 – Deformação frontal de um modelo de massa em forma de cubo, possuindo uma grade virtual com apenas uma linha e uma coluna, a partir de diferentes valores de (3) <i>roundingDegree</i> .	81
Figura 62 – Deformação frontal de um modelo de massa em forma de cubo, a partir de diferentes valores de (4) <i>segments</i> .	82
Figura 63 – Representação gráfica dos diferentes valores de (5) <i>sideReference</i> : <i>main_front</i> , <i>main_back</i> , <i>main_left</i> e <i>main_right</i> .	82
Figura 64 – Deformação frontal de um modelo de massa em forma de cubo, para diferentes valores de (6) <i>axis</i> : (a) <i>vertical</i> e (b) <i>horizontal</i> .	83
Figura 65 – Deformação lateral (direita) de um modelo de massa em forma de cubo, possuindo uma grade virtual com apenas uma linha e uma coluna, a partir de diferentes valores de (7) <i>insideRounding</i> .	83
Figura 66 – Modelo de massa do edifício da Figura 49.	87
Figura 67 – Variação do modelo da Figura 66(j), com modificação na área e lateral.	88
Figura 68 – Variação do modelo apresentado na Figura 29(o).	88
Figura 69 – À esquerda, variações dos resultados obtidos por Jiang <i>et al.</i> (2018), à direita.	89
Figura 70 – Variações <i>low poly</i> (a) e (b) dos modelos apresentados, respectivamente, na Figura 66(j) e na Figura 37.	90
Figura 71 – Exemplo de arredondamento superior em múltiplas faces.	90
Figura 72 – Modelo combinando arredondamento externo e interno.	91
Figura 73 – Variação de modelo obtido através da <i>CGA++</i> .	91
Figura 74 – Variação de modelo obtido através de <i>split grammar</i> .	92
Figura 75 – Modelos com múltiplas variações de arredondamento (a) externo e (b) interno.	92
Figura 76 – (a) Exemplo de faces que compartilham uma mesma aresta, e (b) solução para aplicar deformação.	94
Figura 77 – Geração do floco de neve de Von Koch utilizando <i>Lindenmayer Systems</i> (<i>L-Systems</i>).	101
Figura 78 – Geração de um modelo de alga utilizando <i>L-Systems</i> .	101

Figura 79 – Renderização tridimensional do modelo de uma <i>Mycelis</i>	102
Figura 80 – Gramática para gerar planta de igreja na forma de cruz grega.	103
Figura 81 – Diagrama que descreve a cronologia das <i>shape grammars</i>	104
Figura 82 – Exemplo de <i>split</i> com o modificador aproximado (~).	105
Figura 83 – Exemplo de <i>split</i> com repetição.	105
Figura 84 – Árvore de derivação referente à fachada representada na Figura 85.	106
Figura 85 – Fachada associada à árvore de derivação da Figura 84.	106

LISTA DE TABELAS

Tabela 1 – Dados estatísticos dos modelos gerados pela solução proposta. 93

LISTA DE ABREVIATURAS E SIGLAS

G^2	<i>Generalized Grammar</i>
<i>L-Systems</i>	<i>Lindenmayer Systems</i>
<i>NURBS</i>	<i>Non-Uniform Rational B-Spline</i>
<i>SELEX</i>	<i>Selection Expressions</i>
CGA	Computer Generated Architecture

SUMÁRIO

1	INTRODUÇÃO	18
1.1	Contextualização	18
1.2	Justificativa	19
1.3	Objetivos	19
1.3.1	<i>Objetivo geral</i>	19
1.3.2	<i>Objetivos específicos</i>	19
1.4	Estrutura do trabalho	19
2	FUNDAMENTAÇÃO TEÓRICA	21
2.1	Geração Procedural	21
2.2	CGA Shape	24
2.2.1	<i>Especificações</i>	24
2.2.2	<i>Aplicações</i>	27
2.3	CGA++	29
2.3.1	<i>Especificações</i>	30
2.3.2	<i>Linguagem</i>	34
2.3.3	<i>Aplicações</i>	36
2.4	Selection Expressions	40
2.4.1	<i>Definições de forma</i>	42
2.4.2	<i>Introdução à linguagem</i>	45
2.4.3	<i>Modelagem procedural baseada em seleção</i>	46
2.4.4	<i>Ações</i>	49
2.4.5	<i>Funções de restrição</i>	50
2.4.6	<i>Comparativo</i>	52
2.4.7	<i>Aplicações</i>	53
2.5	Deformação	55
3	TRABALHOS CORRELATOS	59
3.1	<i>Generalized Use of Non-Terminal Symbols for Procedural Modeling</i>	59
3.2	<i>Procedural architecture using deformation-aware split grammars</i>	62
3.2.1	<i>Integrando deformações de forma livre</i>	62
3.2.2	<i>Aplicações</i>	64

3.3	<i>Procedural modeling of architecture with round geometry</i>	67
4	PROPOSTA	69
4.1	Problema	69
4.2	Abordagem	70
4.2.1	<i>Ferramenta de modelagem</i>	70
4.2.2	<i>Fluxo de modelagem</i>	72
4.2.3	<i>Módulos</i>	74
4.2.3.1	<i>Imported libraries</i>	74
4.2.3.2	<i>Classes Module</i>	74
4.2.3.3	<i>Utility Functions Module</i>	75
4.2.3.4	<i>Actions Module</i>	76
4.2.3.5	<i>Outros módulos</i>	84
5	RESULTADOS	85
5.1	Modelos gerados	85
5.2	Desempenho	93
5.3	Restrições	93
6	CONCLUSÃO	95
6.1	Considerações	95
6.2	Trabalhos futuros	95
	REFERÊNCIAS	97
	APÊNDICES	100
	APÊNDICE A – Abordagens precursoras de modelagem procedural	100
A.1	L-Systems	100
A.2	Shape Grammars	102
A.3	Split Grammars	103

1 INTRODUÇÃO

Neste capítulo, serão apresentados alguns contextos nos quais a técnica de modelagem procedural é aplicada, bem como as motivações e os objetivos do presente trabalho.

1.1 Contextualização

Nos últimos anos, a modelagem procedural tem sido um eminentemente tópico de pesquisa, devido ao fato de que simulações de realidade virtual, jogos e filmes, têm se tornado cada vez mais prevalecentes. Na indústria cinematográfica, boa parte das produções utiliza recursos de Computação Gráfica. Um exemplo é o filme *Avatar*, produzido e dirigido por James Cameron, sendo a primeira obra a contar com um mundo 3D foto-realista totalmente gerado por computador: o planeta Pandora (SIMON, 2011). A indústria de jogos, por sua vez, ultrapassou a indústria do cinema, e alguns *video games* têm orçamentos maiores do que os sucessos de bilheteria de Hollywood (TEBOUL, 2011). Um dos exemplos que mais se destaca é a série *Grand Theft Auto*, que combina comportamentos quase totalmente irrestritos em ambientes virtuais inspirados em cidades dos Estados Unidos, como Nova Iorque e Miami (SIMON, 2011).

A modelagem procedural é aplicada em uma grande variedade de áreas, como na geração de texturas, plantas, terrenos, edifícios, cidades, estradas, malhas fluviais, entre outras. A geração de edifícios, em particular, é uma das mais desenvolvidas, possuindo métodos que podem ser amplamente empregados na criação de modelos detalhados e realistas (SMELIK *et al.*, 2014).

Além dos segmentos da indústria citados anteriormente, a modelagem procedural também pode ser utilizada em planejamento urbano, análises logísticas e simulações, uma vez que representações realistas de um espaço urbano podem ser aplicadas em treinamentos de socorro, análise de rotas, planos de evacuação, mapeamento de tráfego, entre outras situações (RODRIGUES *et al.*, 2015).

Seja qual for o cenário, a atividade de criar um ambiente virtual é bastante complexa, visto que existe uma crescente demanda pela modelagem de arquiteturas com características cada vez mais próximas da realidade. Portanto, técnicas de modelagem procedural estão em constante evolução, buscando atender às necessidades emergentes de artistas e animadores.

1.2 Justificativa

A geração procedural de edifícios pode reduzir de maneira significativa os custos de modelagem, uma vez que permite a produção de uma variedade de formas semelhantes a partir de um conjunto de regras generativas (SMELIK *et al.*, 2014), sendo amplamente aplicada na criação de fachadas de prédios retos. Entretanto, para geração de estruturas arquitetônicas mais complexas, com geometria arredondada, por exemplo, algumas técnicas trabalham apenas por meio da sua importação, como complemento.

Baseado nisto, o presente trabalho tem como principal contribuição apresentar uma abordagem, baseada em técnicas de deformação, cujo intuito é resolver a limitação da geração de modelos arquiteturais com geometria arredondada, por meio da utilização de *Selection Expressions*, uma linguagem de modelagem procedural relativamente recente, que representa uma evolução do conceito de *Computer Generated Architecture (CGA)*, utilizado nas clássicas *CGA Shape* e *CGA++*.

1.3 Objetivos

1.3.1 *Objetivo geral*

Gerar modelos arquiteturais com geometria arredondada utilizando *Selection Expressions*, por meio da especificação de uma nova operação de deformação.

1.3.2 *Objetivos específicos*

- Implementar e avaliar a linguagem para geração de modelos de massa com arquitetura arredondada, ou seja, com bordas suavizadas;
- Integrar e avaliar linguagem com ferramenta de modelagem 3D por meio de *scripts*;
- Avaliar a aplicação de técnicas de deformação para criação de modelos arquiteturais com geometria arredondada;
- Avaliar o resultado obtido frente a alguns exemplos do mundo real.

1.4 Estrutura do trabalho

O presente trabalho está disposto em 6 capítulos. No Capítulo 2, por meio de uma fundamentação teórica, são apresentados conceitos e técnicas do estado da arte, para o

melhor entendimento dos capítulos seguintes. No Capítulo 3, são introduzidas ideias pertinentes à resolução do problema a ser abordado, com base em trabalhos correlatos. No Capítulo 4, descreve-se o problema e uma possível abordagem para resolvê-lo. No Capítulo 5, são apresentados alguns modelos obtidos com a técnica proposta, bem como uma breve análise estatística dos resultados. Por fim, no Capítulo 6, são apresentadas as últimas considerações e possíveis tópicos para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão apresentadas as principais técnicas e conceitos relacionados à modelagem procedural, que servirão de alicerce para o entendimento das estratégias abordadas nesta pesquisa, a qual baseia-se na utilização de *Selection Expressions*.

2.1 Geração Procedural

Técnicas procedurais são segmentos de código que especificam as características de um modelo ou efeito gerado por computador. Um exemplo é a utilização de algoritmos e funções matemáticas para texturizar a superfície do mármore, ao invés da utilização de imagens escaneadas para definir o valor das cores (EBERT *et al.*, 2002).

Uma das principais vantagens da modelagem procedural é que, por meio da entrada de um conjunto regras generativas e seus respectivos parâmetros, torna-se possível criar uma grande variedade de objetos. Isto permite que a geometria real de um modelo complexo possa ser gerada apenas quando necessário (SMELIK *et al.*, 2014).

Smelik *et al.* (2014) argumentam que a modelagem procedural tem a capacidade de reduzir radicalmente a quantidade de esforço para criação de conteúdo digital. Contudo, boa parte dos métodos de modelagem procedural atuais ainda não oferece uma alternativa adequada em relação à modelagem manual, exigindo que os usuários manipulem regras e parâmetros não muito intuitivos, cujos efeitos na saída dificilmente podem ser previstos.

A modelagem procedural é bastante utilizada na geração de ambientes virtuais (Figura 1), que são importantes em muitas aplicações, sendo algo amplamente discutido por Smelik *et al.* (2011).

Figura 1 – Modelo de ambiente virtual criado com o framework *SketchaWorld*: (a) ambiente natural com estrada cruzando o rio, (b) ambiente virtual final, (c) visão em perspectiva da área urbana.



Fonte: (SMELIK *et al.*, 2011)

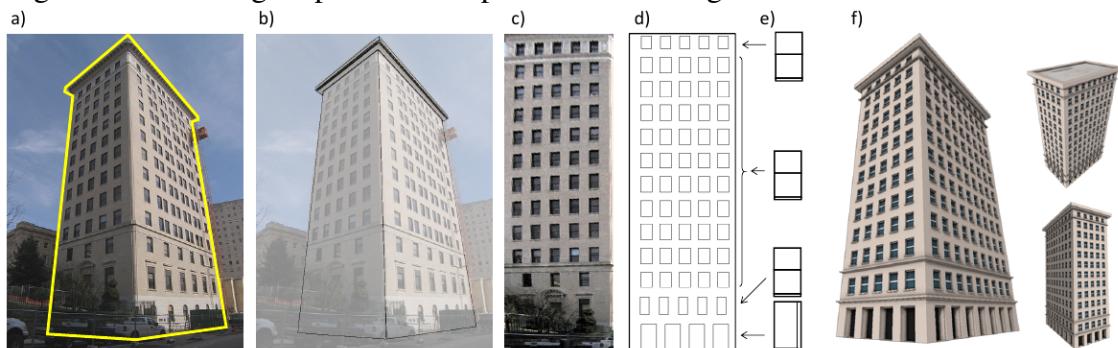
Pode-se analisar a modelagem procedural de diferentes perspectivas: do poder ex-

pressivo à qualidade da produção, do tipo de interação do usuário à complexidade computacional, do grau de popularidade à área de aplicação (SMELIK *et al.*, 2014), tais como geração de terrenos, vegetação, estradas, malhas fluviais, cidades e edifícios, os quais são o foco deste trabalho.

Smelik *et al.* (2014) afirmam que a geração procedural de edifícios é um dos campos mais desenvolvidos da modelagem procedural, havendo ampla utilização de algum sistema de reescrita formal, como *L-Systems*, *split grammars* ou *shape grammars*, a fim gerar um modelo de construção 3D, através de uma representação 2D.

Outros métodos alternativos, por sua vez, tentam reconstruir as gramáticas a partir de um conjunto de dados do mundo real, conforme apresentado por Nishida *et al.* (2018), com a utilização de fotografias. No exemplo mostrado na Figura 2, a) dada uma imagem e a marcação da silhueta de um edifício, b) na primeira etapa, sua abordagem estima os parâmetros da câmera e gera uma gramática de massa do edifício. Em seguida, c) a imagem da fachada é retificada e d) a gramática da fachada é gerada. Logo após, e) os elementos terminais representando as janelas são selecionados. f) Por fim, a gramática de saída é construída, e uma geometria 3D correspondente é gerada.

Figura 2 – Modelagem procedural a partir de uma imagem.



Fonte: (NISHIDA *et al.*, 2018)

O processo de geração de edifícios por inteiro, ou seja, envolvendo fachadas e interiores, requer a utilização de métodos distintos para cada um dos casos. Na geração de fachadas, utiliza-se, geralmente, abordagens baseadas em gramáticas. Na geração de interiores, pode-se diferenciar entre a geração a partir da planta e a solução do *layout* de móveis (SMELIK *et al.*, 2014). O trabalho de Leblanc *et al.* (2011) introduz um sistema para modelagem de edifícios completos, conforme ilustrado na Figura 3.

No contexto da geração procedural de edifícios, o conceito de semântica é algo pri-

Figura 3 – Modelo de uma casa (à esquerda), com vista superior do primeiro andar (ao centro) e vista de dentro (à direita).



Fonte: (LEBLANC *et al.*, 2011)

mordial, pois é ela que define se a estrutura dos modelos está organizada de maneira significativa, ou seja, condizente com exemplos arquiteturais do mundo real.

Conforme argumentado por Silveira *et al.* (2015), para que a geometria tridimensional de um edifício seja gerada, um conjunto mínimo de dados deve ser fornecido pelo usuário, como uma lista de arestas e vértices para definir as paredes internas de cada cômodo, ou as paredes externas do edifício, por exemplo. Na definição da parede, alguns pontos podem ser definidos para representar o centro das portas e janelas. Os quartos e suas paredes adjacentes podem receber um tipo, como cozinha, banheiro ou sala de estar. Algumas possíveis informações que podem ser definidas em uma entrada semântica são:

- **Materiais:** Categorizados por tipo, como parede externa, teto e vidro, que são utilizados para restringir os tipos de materiais utilizados para cada cômodo;
- **Estilo de janelas:** Define as informações de dimensão (largura e altura), direção da abertura, profundidade da abertura, tipos de ambiente aos quais se aplica, e materiais;
- **Estilo de portas:** Semelhantes aos estilos de janelas, de modo que diferentes características possam ser consideradas, como portas com topo arredondado, portas deslizantes ou portas com painéis de vidro;
- **Estilo de telhados:** Tipos de telhados que podem ser utilizados na construção, definindo as dimensões, materiais e formas permitidas.

Portanto, uma definição semântica não é apenas uma definição de aparências, pois atribui sentido ao que está sendo criado (SILVEIRA *et al.*, 2015).

Visando construir uma linha temporal da evolução da modelagem procedural, algumas das principais técnicas precursoras são apresentadas no Apêndice A, tais como *L-Systems* (Seção A.1), *Shape grammars* (Seção A.2) e *Split grammars* (Seção A.3). Então, seguindo a ordem cronológica, nas próximas seções, serão apresentadas técnicas mais recentes e pertinentes

ao contexto do presente trabalho.

2.2 CGA Shape

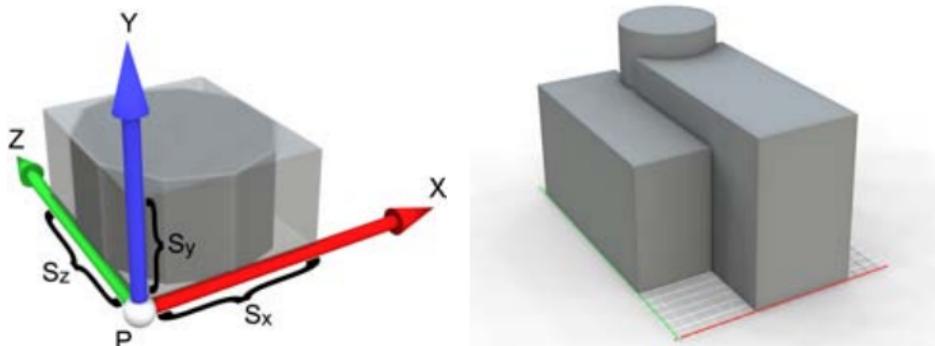
A *CGA Shape* foi proposta por Müller *et al.* (2006) para geração procedural de modelos arquiteturais, trazendo melhorias em relação às *split grammars*. Como contribuição, introduz regras de divisão com repetição, regras de redimensionamento e regras de divisão de componentes. A notação da gramática e as regras gerais para adicionar, redimensionar, transladar e rotacionar as formas são estendidas dos *L-Systems*, mas voltadas para a modelagem de arquiteturas.

2.2.1 Especificações

Visando formalizar a *CGA Shape*, Müller *et al.* (2006) apresentam as seguintes definições e recursos:

Forma: Consiste em um símbolo (*string*), atributos geométricos e atributos numéricos. As formas são identificadas por seus símbolos, sendo um terminal $\alpha \in \Sigma$ ou um não-terminal $\beta \in V$. As formas correspondentes são chamadas de formas terminais e formas não-terminais, respectivamente. Os atributos geométricos mais importantes são a posição P , três vetores ortogonais X , Y e Z , que descrevem um sistema de coordenadas, bem como um vetor de tamanho S . Em conjunto, tais atributos definem o escopo, ou seja, uma caixa delimitadora orientada no espaço, conforme mostrado na Figura 4.

Figura 4 – Representação geométrica da *CGA Shape*. Esquerda: definição de uma caixa delimitadora contendo uma forma primitiva. Direita: Modelo simples de uma construção utilizando três primitivas.



Fonte: (MÜLLER *et al.*, 2006)

Processo de produção: Um conjunto finito de formas básicas define uma configu-

ração. O processo de produção pode começar com uma configuração arbitrária de formas A , chamada de axioma, e prossegue da seguinte maneira: (1) Selecionar uma forma ativa com o símbolo B no conjunto; (2) escolher uma regra de produção com B no lado esquerdo para computar um sucessor para B , ou seja, um novo conjunto de formas $BNEW$; (3) marcar a forma B como inativa, adicionando as formas $BNEW$ à configuração, e continuar com a etapa (1). O processo de produção termina quando não existirem mais não-terminais na configuração. É importante pontuar que as formas não são excluídas, mas sim marcadas como inativas, após serem substituídas. Isto permite a consulta na hierarquia de formas como um todo, e não apenas na configuração ativa.

Notação: As regras de produção são definidas da seguinte maneira:

$$id : predecessor : cond \rightsquigarrow successor : prob,$$

onde id é um identificador único para a regra, $predecessor \in V$ é um símbolo que identifica uma forma que deve ser substituída por $successor$, e $cond$ é uma expressão lógica que deve ser avaliada como verdadeira, antes da aplicação da regra. Uma regra, por sua vez, é selecionada com probabilidade $prob$. Por exemplo, a regra:

$$1 : fac(h) : h > 9 \rightsquigarrow floor(h/3) floor(h/3) floor(h/3)$$

substitui a forma de fac por três formas $floor$, se o parâmetro h é maior que 9.

Regras do escopo: Utilizadas para modificar as formas, onde $T(t_x, t_y, t_z)$ é um vetor de translação, que é adicionado ao escopo da posição P ; $R_x(angle)$, $R_y(angle)$ e $R_z(angle)$, gira o respectivo eixo do sistema de coordenadas; e $S(s_x, s_y, s_z)$ define o tamanho do escopo. Os símbolos [e] são utilizados, respectivamente, para empilhar e desempilhar o escopo atual na pilha, similar aos *L-Systems*. O comando $I(objId)$ adiciona a instância de uma primitiva geométrica com identificador $objId$. Objetos padrões incluem um cubo e um cilindro, contudo, qualquer modelo tridimensional pode ser utilizado. A seguinte regra define o *design* do modelo de massa apresentado na Figura 4:

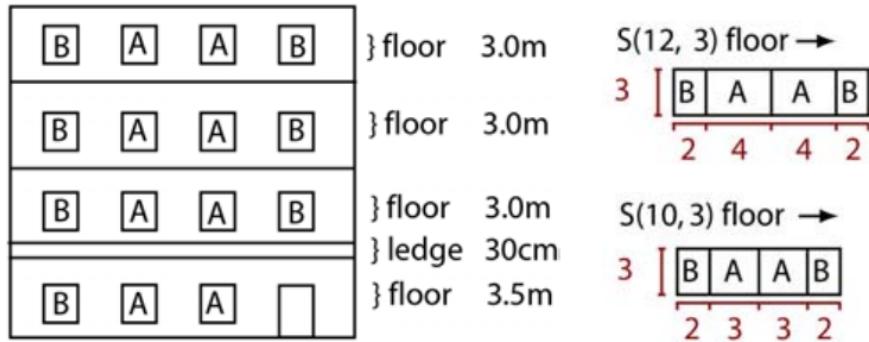
$$\begin{aligned} 1: A \rightsquigarrow & [T(0,0,6) S(8,10,18) I("cube")] \\ & T(6,0,0) S(7,13,18) I("cube") T(0,0,16) S(8,15,8) I("cylinder"). \end{aligned}$$

Regra de divisão básica: Divide o escopo atual ao longo de um eixo. Por exemplo, considere a regra para dividir a fachada da Figura 5 em quatro andares e uma saliência:

$$1: fac \rightsquigarrow Subdiv("Y", 3.5, 0.3, 3, 3, 3)\{floor|ledge|floor|floor\}$$

O primeiro parâmetro descreve o eixo dividido, sendo X , Y ou Z , e os parâmetros restantes descrevem os tamanhos da divisão. Entre os delimitadores { e } é fornecida uma lista de formas, separadas por |.

Figura 5 – Exemplo de aplicação das regras da *CGA Shape*. À esquerda: Um *design* básico de fachada. À direita: Uma divisão simples que pode ser utilizada para os três andares superiores.



Fonte: (MÜLLER *et al.*, 2006)

Regras de redimensionamento: Nem todas as partes da arquitetura são redimensionadas igualmente, por isto, é possível distinguir entre valores absolutos, que não redimensionam, e valores relativos, que redimensionam. Os valores são considerados absolutos por padrão, então, para denotar valores relativos é utilizada a letra r , conforme a seguinte regra:

I: floor ~> Subdiv("X", 2, 1r, 1r, 2) {B|A|A|B},

na qual os valores relativos r_i são substituídos por $r_i * (Scope.sx - \sum abs_i / \sum r_i)$, onde *Scope.sx* representa o tamanho de *x-length* do escopo atual.

Repetição: Permite a inclusão de um elemento específico lado a lado. Por exemplo:

I: floor ~> Repeat("X", 2) {B}

especifica que, enquanto houver espaço, *floor* será revestido com elementos do tipo *B*, ao longo do eixo x do escopo. O número de repetições é calculado a partir do valor de *repetitions* = $\lceil Scope.sx / 2 \rceil$.

Divisão de componentes: Esta nova operação permite dividir o escopo em formas de dimensões menores, conforme exemplificado no comando a seguir:

I: a ~> Comp(type, params) {A|B|...|Z},

onde *type* identifica o tipo de divisão do componente com os parâmetros (*params*) associados,

se existirem. Por exemplo, $\text{Comp}("faces") \{A\}$ cria uma forma com o símbolo A para cada face da forma tridimensional original. Da mesma maneira, $\text{Comp}("edges") \{B\}$ e $\text{Comp}("vertices") \{C\}$ são utilizados para dividir arestas e vértices, respectivamente. Para acessar apenas os componentes selecionados, são utilizados comandos como $\text{Comp}("edge", 3) \{A\}$, para criar uma forma A alinhada com a terceira aresta do modelo; ou $\text{Comp}("side faces") \{B\}$, para acessar as faces laterais de um cubo ou cilindro poligonal. Para codificar formas de menor dimensão, são utilizados escopos cujos eixos têm tamanho diferente de zero. Para voltar à dimensões superiores, pode-se utilizar o comando S , com um valor não-nulo na dimensão correspondente. Por exemplo, para realizar a extrusão de uma face ao longo de sua normal e, portanto, transformá-la em uma forma volumétrica.

2.2.2 Aplicações

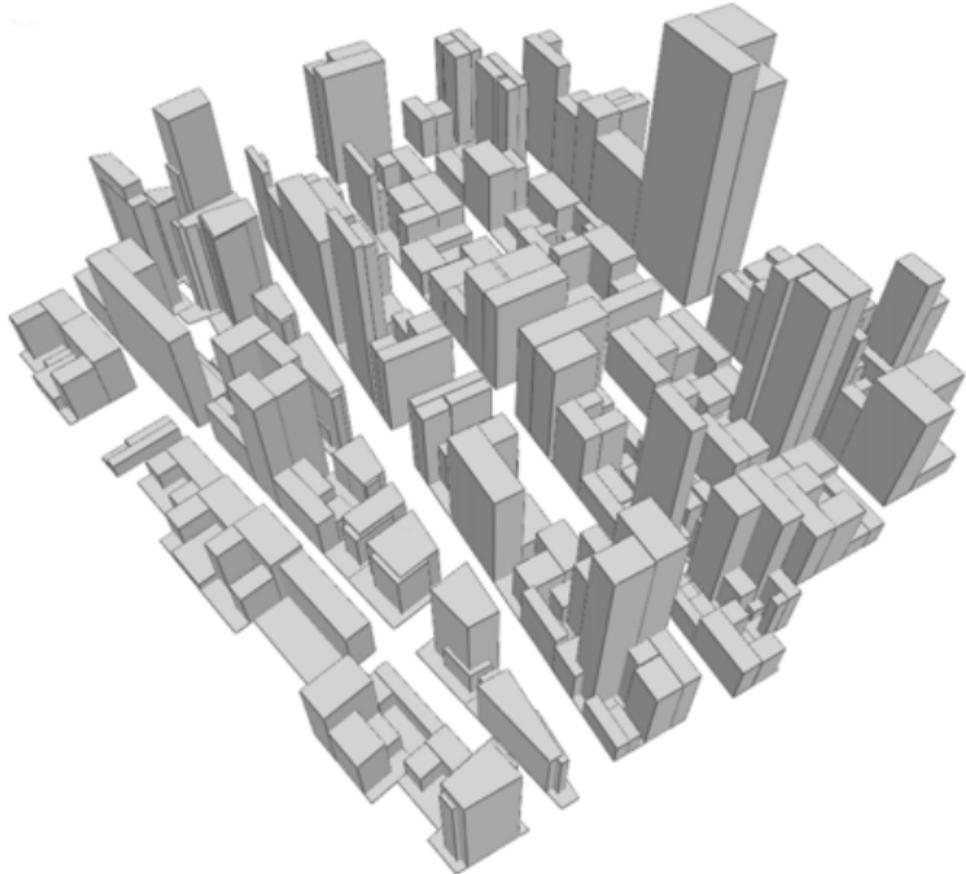
Com objetivo de demonstrar a aplicação da *CGA Shape*, Müller *et al.* (2006) apresentam exemplos nos seguintes contextos:

Prédios comerciais: Inicialmente, é definido um conjunto de regras para gerar vários modelos com base em uma gramática estocástica. O axioma da gramática é uma forma bidimensional, representando um lote de construção. As regras, por sua vez, funcionam da seguinte maneira: na regra 1, uma operação de extrusão é aplicada ao lote com altura definida pelo valor de *building_height*, utilizando um comando de tamanho para produzir uma forma tridimensional. Logo após, esta forma é dividida em duas formas menores. Uma forma volumétrica, que é o maior sólido no modelo, e uma forma que, posteriormente, será dividida em duas asas laterais. Esta divisão é realizada pela regra 2, que também gera uma lacuna entre as asas laterais. Na regra 3, é exemplificado o uso de regras estocásticas para gerar uma variedade de modelos com diferentes configurações. Além disto, são utilizadas combinações de números aleatórios e seleção estocástica de regras, a fim de criar uma variedade de formas com diferentes alturas e larguras na asa lateral. Na regra 4, ocorre a transição para a modelagem de fachada, permitindo que em estágios posteriores possam ser adicionadas portas e janelas, por exemplo. Um modelo gerado a partir destas quatro regras é ilustrado na Figura 6.

PRIORITY 1:

1. $lot \rightsquigarrow S(1r, building_height, 1r)$
 $Subdiv("Z", Scope.sz * rand(0.3, 0.5), 1r) \{ facades \mid sidewings \}$
2. $sidewings \rightsquigarrow$
 $Subdiv("X", Scope.sx * rand(0.2, 0.6), 1r) \{ sidewing \mid \epsilon \}$
 $Subdiv("X", 1r, Scope.sx * rand(0.2, 0.6)) \{ \epsilon \mid sidewing \}$
3. $sidewing$
 $\rightsquigarrow S(1r, 1r, Scope.sz * rand(0.4, 1.0)) \{ facades : 0.5 \}$
 $\rightsquigarrow S(1r, Scope.sy * rand(0.2, 0.9), Scope.sz * rand(0.4, 1.0)) \{ facades : 0.3 \}$
 $\rightsquigarrow \epsilon : 0.2$
4. $facades \rightsquigarrow Comp("sidefaces") \{ facade \}$

Figura 6 – Variações estocásticas de modelos de edifícios.



Fonte: (MüLLER *et al.*, 2006)

Residências: A gramática utilizada para gerar os ambientes da Figura 7, baseia-se na seguinte estratégia: segmentar as bordas da propriedade com uma divisão de componentes; dividir a propriedade para modelar o jardim da frente, o quintal e a construção principal; gerar

uma calçada e colocar árvores em intervalos regulares ao lado da rua; e, por fim, gerar uma calçada conectada à porta da garagem, bem como uma via conectada à porta de entrada.

Figura 7 – Diferentes construções em um ambiente suburbano.



Fonte: (MÜLLER *et al.*, 2006)

2.3 CGA++

A *CGA++* foi introduzida por Schwarz e Müller (2015) como sendo uma evolução natural da *CGA Shape*, com o objetivo de superar algumas limitações existentes na modelagem procedural de arquiteturas, tais como:

1. A coordenação do refinamento de decisões através de múltiplas formas não é diretamente suportada. Assim, qualquer decisão que afete várias formas deve ser tomada, o mais tardar, no ponto onde as linhagens destas formas divergem, para, posteriormente, ser transmitida nas regras convenientes;
2. Normalmente, não são possíveis operações envolvendo múltiplas formas, ou seja, não são permitidas operações *booleanas*, como a interseção de duas formas, nem a construção de uma forma a partir de várias outras, por exemplo;
3. Informações contextuais, geralmente, não estão disponíveis, o que impede considerar referências de outras formas, algo necessário para determinados objetivos de modelagem,

como alinhamento;

4. O suporte para explorar múltiplas formas é inexistente. Em particular, não é possível gerar uma derivação a partir de outra derivação, e consultar ou incorporar o resultado.

Assim, com base nestas dificuldades, Schwarz e Müller (2015) apresentam dois novos recursos com a *CGA++*.

Em primeiro lugar, as formas ficam diretamente expostas na gramática, permitindo que objetos individuais sejam identificados de maneira única, bem como transmitidos e armazenados como valores. Assim, as operações podem tomar formas como argumentos, permitindo operações *booleanas* em múltiplos elementos, resolvendo a limitação 2. Além disto, torna-se possível acessar, percorrer e consultar a árvore de formas gerada pelo processo de derivação, o que facilita a obtenção de informações contextuais genéricas, resolvendo a limitação 3. Árvores de formas temporárias podem ser criadas instantaneamente em tempo de execução, por exemplo, gerando uma nova derivação ou invocando uma função em uma árvore existente, algo que permite a busca por outras alternativas, resolvendo a limitação 4 (SCHWARZ; MÜLLER, 2015).

Em segundo lugar, juntamente com o dispositivo linguístico de eventos, é fornecido um mecanismo de agrupamento dinâmico e um recurso de sincronização, permitindo a coordenação entre um grupo de formas, através da troca de informações ou do estabelecimento de uma decisão coerente sobre como proceder individualmente, resolvendo a limitação 1. Além disto, os eventos podem ser utilizados para influenciar a ordem do processo de derivação, garantindo a disponibilidade de todas as formas necessárias ao realizar uma consulta contextual, resolvendo a limitação 3 (SCHWARZ; MÜLLER, 2015).

2.3.1 Especificações

De acordo com Schwarz e Müller (2015), a ideia principal para superar as limitações descritas anteriormente, na Seção 2.3, é tornar as formas e a árvore de formas disponíveis na gramática. Com o processo de derivação definindo a árvore de formas, cada elemento que é gerado durante a derivação corresponde a um nó desta árvore, o que identifica ainda mais a subárvore que tem raiz neste nó. Assim, os termos *forma*, *nó*, e *(sub)árvore* oferecem visualizações diferentes da mesma entidade. Uma vez que tal interpretação é adotada, as entidades são expostas diretamente na linguagem.

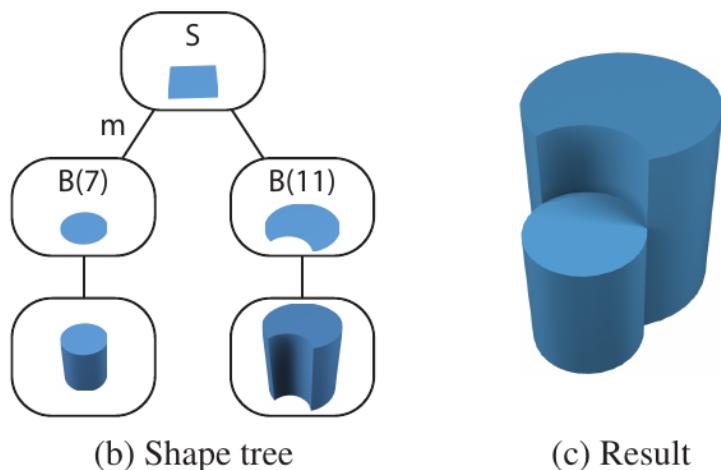
Schwarz e Müller (2015) argumentam que a *CGA++* oferece diversas opções para fazer referência a uma forma existente. Além da consulta da árvore, as formas em um corpo de regra podem ser convenientemente acessadas por nome. Uma vez que a forma é identificada, ela pode participar livremente nas expressões; em particular, pode ser utilizada como argumento para funções e operações. Como exemplo ilustrativo, na Figura 8, a gramática (a) demonstra a operação *booleana minus*, que modifica a forma atual subtraindo dela uma dada forma, podendo ser empregada para evitar a sobreposição de áreas na construção.

Figura 8 – Exemplo ilustrativo do uso da *CGA++*: (a) gramática, (b) árvore de formas e (c) resultado.

```

S      --> i("circle") m=B(7) t(3,0,0) s(10,0,10) minus(m) B(11)
B(h) --> extrude(h)
  
```

(a) Grammar



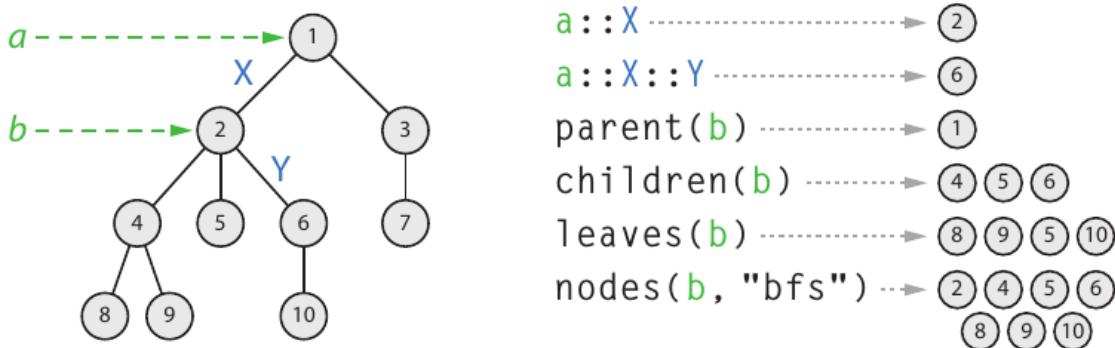
Fonte: (SCHWARZ; MÜLLER, 2015)

Com objetivo de definir operações e conceitos importantes para entendimento da *CGA++*, Schwarz e Müller (2015) apresentam as seguintes especificações:

Árvore de formas: A técnica oferece funções para navegar arbitrariamente na árvore de formas (Figura 9). Neste contexto, dado um nó, tanto seu pai quanto sua lista de filhos podem ser consultados. Além disto, também existem funções que retornam todos os nós folha de uma dada subárvore, ou uma lista de todos os nós enumerados de acordo com uma ordem de percorrido especificada. A ausência de um nó é denotada por `null` nas gramáticas. Para fazer referência a um nó filho rotulado, o operador de acesso associativo à esquerda `::` pode ser utilizado, o qual leva um nó em seu lado esquerdo e um rótulo em seu lado direito. Uma consulta na árvore de formas pode ser realizada através de atributos, permitindo abstrair

a estrutura exata da árvore, o que facilita a localização de uma determinada forma com base em algum critério. Cada forma ainda pode ter um número arbitrário de atributos, que podem ser definidos e recuperados. Um atributo é identificado por um nome, e seu valor pode ser de qualquer tipo compatível com a linguagem, incluindo uma forma.

Figura 9 – Opções de navegação na árvore. Neste caso, a e b são valores representando as formas dos nós 1 e 2, respectivamente, enquanto X e Y são rótulos.



Fonte: (SCHWARZ; MÜLLER, 2015)

Novas formas: Para gerar um novo processo de derivação dentro do atual, utiliza-se o construto `<actions>(base)`, que possibilita criar uma nova árvore de (sub)formas. Assim, empregando a forma identificada por *base* como forma inicial, as ações especificadas são executadas e a árvore resultante é retornada. Se o argumento *base* for omitido, a forma atual é utilizada.

Funções: Uma nova árvore de formas é produzida por meio de várias funções internas, podendo ser derivada de uma ou mais (sub)árvores de entrada. Um conjunto destas funções se preocupa, principalmente, com a modificação de formas. Por exemplo, a função `transformScope(source, target)` retorna uma cópia da origem, onde os escopos de todos os nós estão sujeitos à transformação, fazendo o escopo do nó raiz se tornar idêntico ao escopo de destino, ou seja, ajustando a origem no destino. Para outras operações, são fornecidas funções que tomam uma forma a ser manipulada como argumento, e retornam o resultado como uma nova forma. Por exemplo, `t(tree, Δx, Δy, Δz)` translada os escopos de todos os nós da árvore especificada pelo deslocamento dado. Além disto, existe o grupo de funções para operações de subdivisão, cada uma retornando uma lista com todas as partes resultantes, e outro que se concentra, exclusivamente, na manipulação da estrutura da árvore.

Formas recuperáveis: Servem como nós indicadores que especificam como a derivação deve ser continuada em estágios futuros. Para criação de uma forma recuperável utiliza-

se `?name(arg0, ...)`, registrando a forma atual e os argumentos fornecidos, sendo, basicamente, um nó (forma) com atributos especiais. A função `continue(tree, name0 = rule0, ...)` invoca a regra `rulei` para todas as formas recuperáveis na árvore fornecida, cujo nome corresponda a `namei`, utilizando os argumentos associados à respectiva forma recuperável, e retorna uma árvore adequadamente refinada. Na Figura 10, duas árvores de formas parciais são construídas, e a que se mostra como melhor opção é, então, completada pelo refinamento das formas recuperáveis da árvore adequada. A regra `Parcel` explora dois esquemas de desenvolvimento alternativos: primeiro, construindo parcialmente as duas árvores de formas correspondentes, depois, atribuindo às variáveis `a` e `b`, para uso subsequente. As regras `DesignA` e `DesignB`, em última análise, geram formas `Footprint` e aplicam uma operação de extrusão sobre elas, produzindo massas de construção, cujo refinamento é adiado pela emissão de uma forma recuperável `?Mass`. Empregando a função `V`, que soma os volumes das formas (folhas) de uma determinada árvore, a regra `Parcel`, então, determina a árvore com o maior volume de massa total, e continua o refinamento em suas formas recuperáveis, utilizando a regra `Mass1` ou `Mass2`, respectivamente. Por fim, a árvore de formas resultante é incorporada por meio da utilização do `include`.

Figura 10 – Exemplo do uso de formas recuperáveis.

```

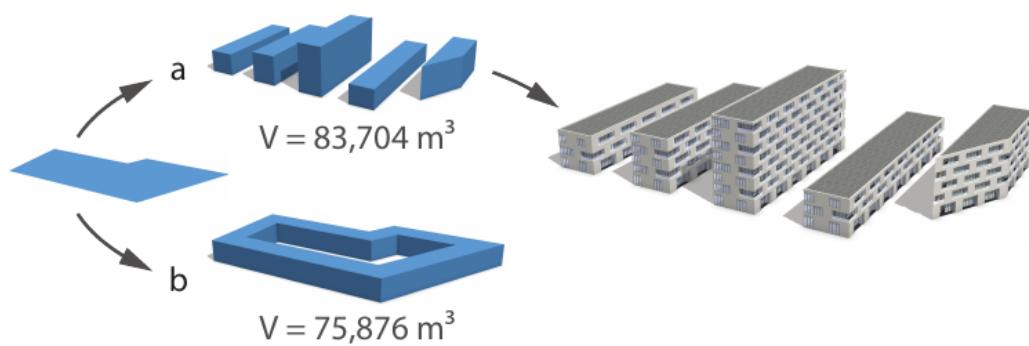
Parcel --> with(a = < DesignA >, b = < DesignB >) {
    case { V(a) > V(b): include(continue(a, Mass = %Mass1))
        | else:           include(continue(b, Mass = %Mass2)) } }

DesignA --> split("x") { 15: Footprint | ~20: NIL }*
DesignB --> setback(15) { all = Footprint }
Footprint --> extrude(rand(10, 30)) ?Mass

Mass1 --> ...
Mass2 --> ...

func V(tree) = sum(map(l : leaves(tree), volume(l)))

```



Fonte: (SCHWARZ; MÜLLER, 2015)

Eventos: Servem como pontos de sincronização, oferecendo influência na ordem em que o processo de derivação refina as formas, permitindo que vários ramos independentes da derivação troquem informações, e coordenem como proceder. São dois os principais elementos envolvidos em um evento: a operação especial `event(name)` e um manipulador de eventos. A operação gera o evento especificado dentro de uma regra, que suspende a ramificação atual da derivação e a identifica como participante do evento. Uma vez que todos os ramos ativos levantaram algum evento, o manipulador de eventos é consultado. Em seguida, cada ramificação é retomada, executando a respectiva regra determinada pelo manipulador diretamente no local, o que corresponde à substituir a operação `event` pelas ações da regra. Na Figura 11, (1) refere-se a um evento gerado com a operação `event`, seguido de (2), que sincroniza vários ramos do processo de derivação. No `event handler` (3), recebe-se as formas atuais de todos os ramos participantes como entrada, e em (4), o manipulador do eventos retorna uma regra para cada um, especificando como proceder.

Conforme descrito por Schwarz e Müller (2015), o manipulador de eventos é especificado como parte da definição de um evento, sendo representado na gramática pelo seguinte construto:

$$\text{event } name(param_0, \dots) \{ priority \} = handler.$$

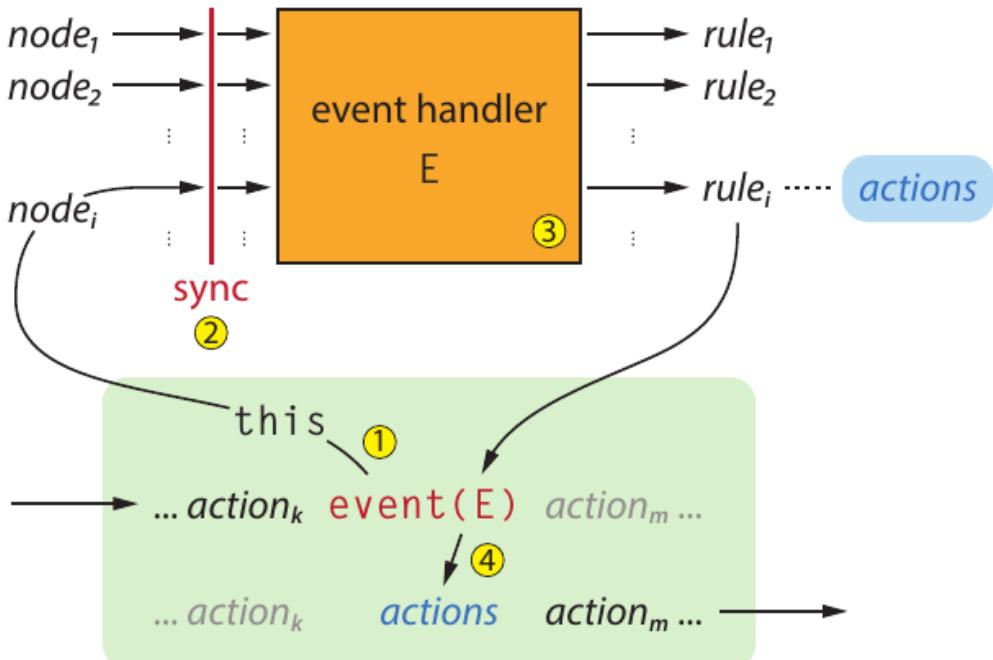
Schwarz e Müller (2015) afirmam que, ao fornecer parâmetros, uma família inteira de eventos pode ser definida, onde cada atribuição de parâmetro concreto constitui um evento separado, por exemplo, `e(0)` e `e(1)` são dois eventos distintos. Para cada evento é atribuído um número de prioridade, que pode depender dos valores dos parâmetros, sendo 0 se não for especificado, orientando, assim, a ordem em que múltiplos eventos concorrentes são tratados.

2.3.2 *Linguagem*

Os recursos da *CGA++* apresentados por Schwarz e Müller (2015) ainda são complementados por outros novos componentes de linguagem, que oferecem uma notação mais simples, aumentando a expressividade, conforme mostrados a seguir:

Objetos suportados: Além de tipos *booleanos*, números e *strings*, a *CGA++* oferece suporte à listas e tuplas. As listas são sequências com um número arbitrário de valores do mesmo tipo. As tuplas representam sequências com um tamanho fixo de valores, podendo ser de tipos distintos. A *CGA++* também suporta funções anônimas, como o construto

Figura 11 – Representação do manipulador de eventos.



Fonte: (SCHWARZ; MüLLER, 2015)

$[param_0, \dots](body)$, que produz um novo valor de função, onde qualquer variável fora da função referenciada na expressão *body* tem seu valor capturado como parte do valor da função.

Argumentos de expressão: Visam facilitar a expressão e fornecer uma sintaxe sucinta, sendo suportados por muitas funções integradas que processam vários itens, como os elementos de uma lista.

Argumentos iteráveis: Se uma função receber uma lista como um parâmetro e avaliar um argumento de expressão para cada elemento desta lista, então, um nome significativo para acessar o respectivo elemento dentro do argumento de expressão pode ser especificado como parte do argumento da lista (*name : list*). Se omitido, o elemento fica disponível por meio de uma variável implícita.

Operador de cadeia: Ao aplicar, iterativamente, uma sequência de funções a um valor, o operador de cadeia \rightarrow pode ser útil para melhorar a legibilidade, pois aplica seu lado esquerdo como primeiro argumento ao seu lado direito. Por exemplo,

`this \rightarrow t(2, 0, 2) \rightarrow s(2, 9, 6) \rightarrow r(30, 0, 0),`

é equivalente a

`r(s(t(this, 2, 0, 2), 2, 9, 6), 30, 0, 0).`

Argumentos implícitos: Em alguns casos, existe um valor bem definido $v \in \mathbb{N}$, no qual, geralmente, uma função é aplicada. Visando a concisão notacional, é possível, simplesmente, omitir este valor. Por exemplo, se uma função interna aparecer dentro de um corpo de regra e aceitar uma forma como primeiro argumento, a forma atual (`this`) será utilizada automaticamente se este argumento for omitido. Da mesma maneira, a lista de formas participantes (`$nodes`) é utilizada implicitamente como primeiro argumento em uma expressão do manipulador de eventos.

Variáveis auxiliares: Para lidar com expressões complexas ou utilizar o resultado de uma expressão diversas vezes, variáveis auxiliares podem ajudar a trazer clareza e facilidade de formulação. Para este fim, o construto a seguir é fornecido:

```
with(var1 = value1, ..., expression)
with(var1 = value1, ...){actions},
```

permitindo atribuir valores às variáveis e, posteriormente, utilizá-los em uma expressão ou dentro dos argumentos de ações.

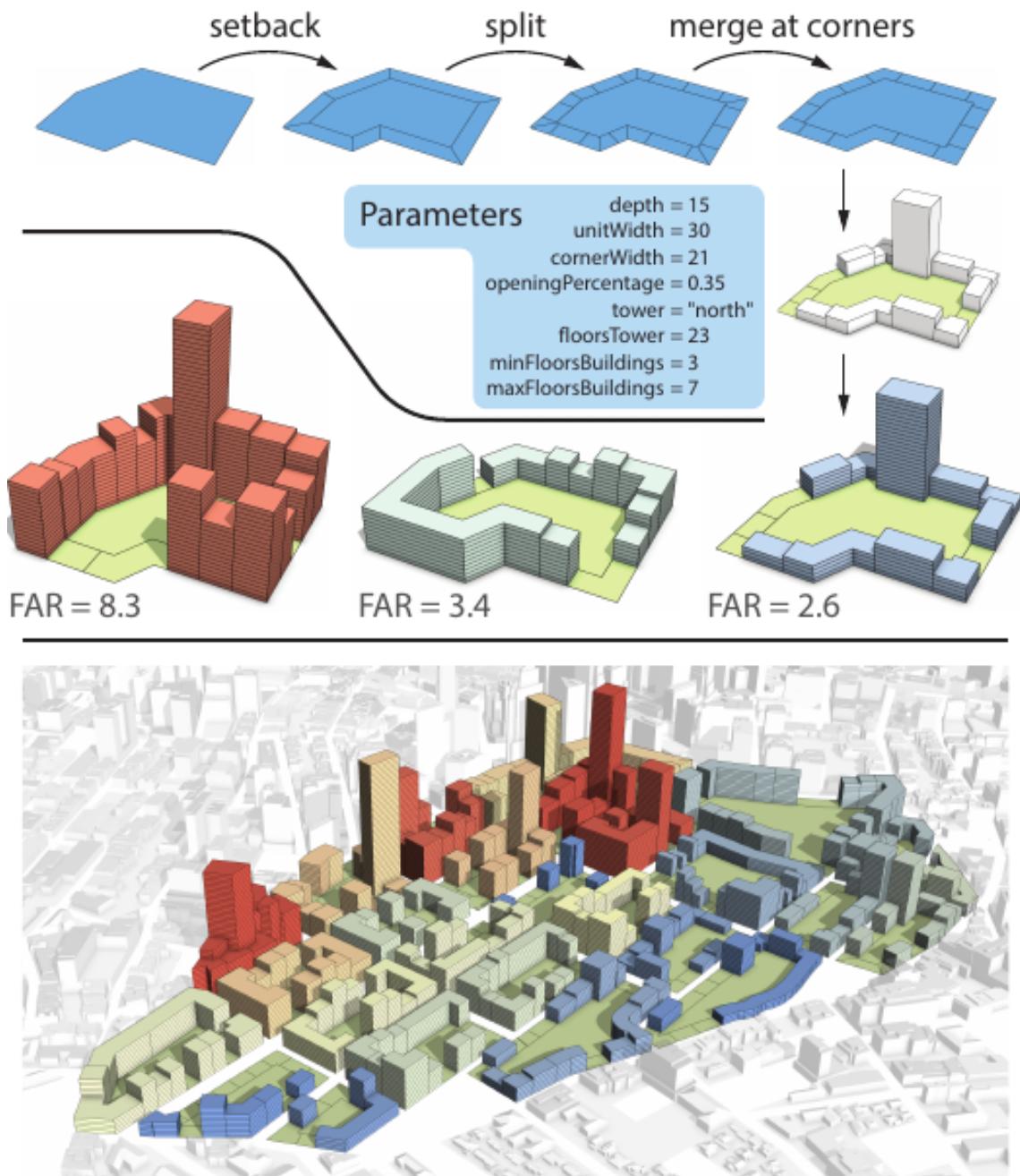
Mais informações sobre a *CGA++* foram disponibilizadas por Schwarz e Müller (2015) como material suplementar.

2.3.3 Aplicações

Com intuito de demonstrar os recursos de modelagem oferecidos pela *CGA++*, Schwarz e Müller (2015) apresentam os seguintes exemplos, cobrindo diferentes domínios de aplicação:

Planejamento urbano: Uma tarefa comum dos planejadores urbanos é projetar o chamado bloco perimetral, ou seja, um lote de blocos com edifícios em seus limites, conforme mostrado na Figura 12.

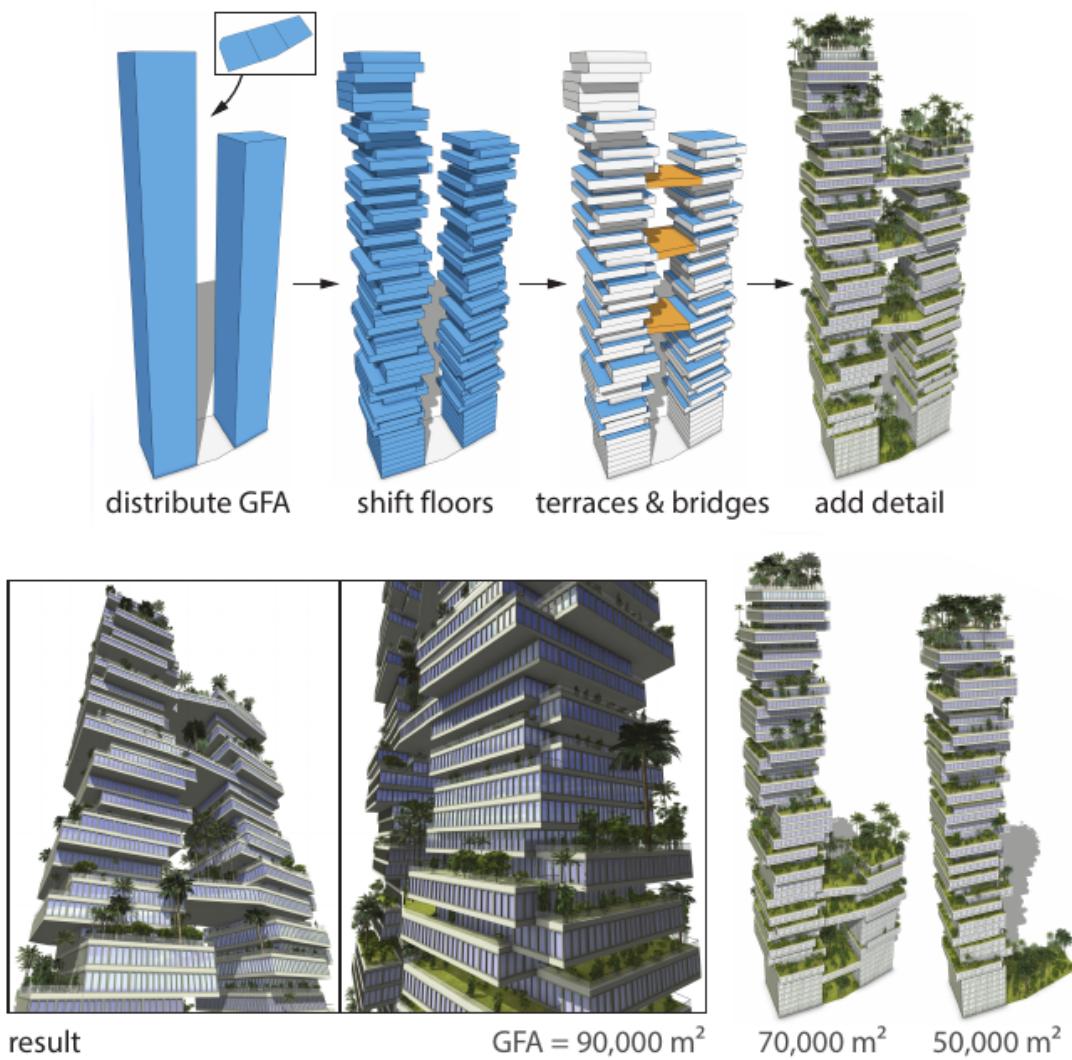
Figura 12 – Representação de bloco perimetral. Acima: estratégia de modelagem desejada e os parâmetros envolvidos. Meio: exemplos de resultados para diferentes escolhas de parâmetros. Abaixo: exemplo de um resultado de planejamento urbano no mundo real.



Fonte: (SCHWARZ; MÜLLER, 2015)

Edifícios: Na Figura 13, inspirados nos arranha-céus ecológicos de megacidades asiáticas, são apresentados modelos que consistem em, no máximo, dois blocos de torres, possuindo pisos deslocados e terraços com jardins no telhado, que são ligados por uma série de pontes suspensas.

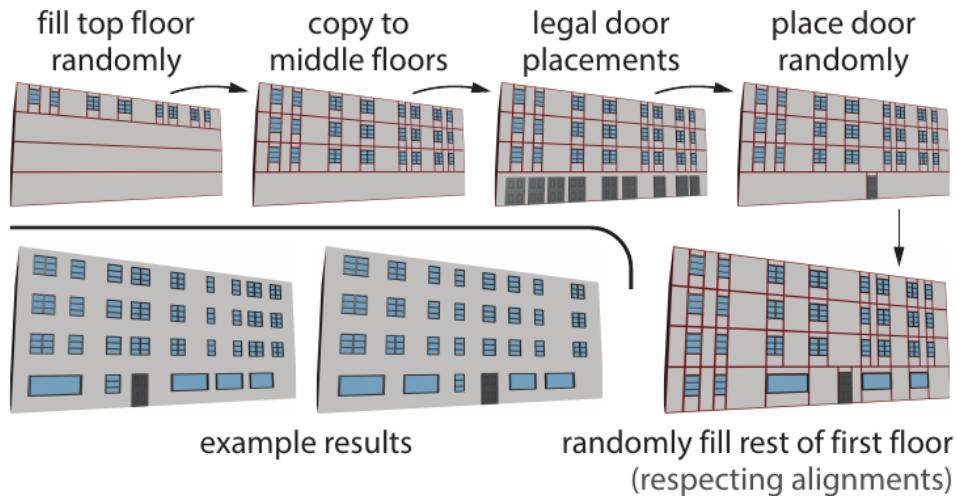
Figura 13 – Modelos de arranha-céus ecológicos. Acima: principais etapas da abordagem de modelagem. Abaixo: visualizações em foco do resultado do exemplo e resultados para diferentes valores para área com gramado.



Fonte: Adaptado de (SCHWARZ; MüLLER, 2015)

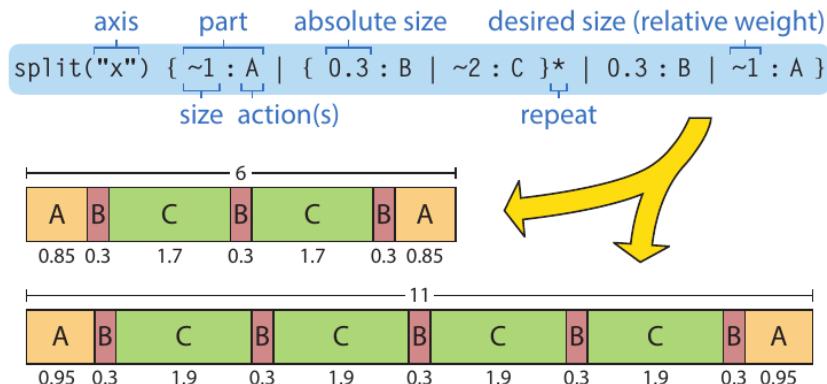
Fachadas: Na geração de fachadas aleatórias, o alinhamento desempenha um papel fundamental, assim, as Figuras 14 e 15 apresentam os recursos da *CGA++* neste contexto.

Figura 14 – Modelos de fachadas com elementos e alinhamentos aleatórios. Acima: abordagem de solução geral, na qual as linhas de divisão são mostradas em vermelho). Abaixo: exemplo de resultados para diferentes parâmetros.



Fonte: (SCHWARZ; MüLLER, 2015)

Figura 15 – Exemplo de operação split, mostrando o resultado para diferentes valores de entrada, neste caso, 6 e 11.



Fonte: (SCHWARZ; MüLLER, 2015)

Na próxima seção, serão apresentadas características e principais vantagens da linguagem de modelagem *SELEX*, em relação às suas precursoras, previamente abordadas.

2.4 Selection Expressions

A *SELEX* é uma nova abordagem para geração procedural, que foi introduzida por Jiang *et al.* (2018), e tem como ideia principal selecionar formas através de *selection-expressions*, substituindo a simples correspondência de *strings*, utilizada em técnicas como *CGA Shape* e *CGA++*.

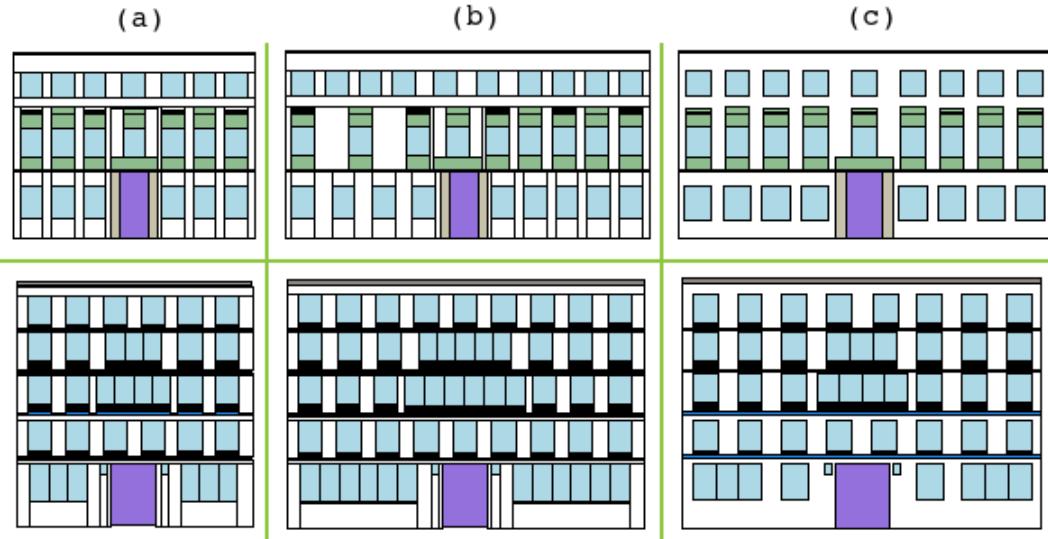
Segundo Jiang *et al.* (2018), uma *selection-expression* especifica como selecionar um subconjunto complexo de formas em uma hierarquia de formas, por exemplo, selecionar todas as janelas altas no segundo andar da fachada principal do edifício, o que permite expressar ideias de modelagem em seu contexto global. Outro recurso da *SELEX* é a aplicação de restrições de alinhamento e restrições de tamanho, permitindo que descrições procedurais possam gerar variações de fachadas e construções, sem violar as restrições de alinhamento e dimensionamento que afetam o estado da arte atual.

Para exemplificar este caso, Jiang *et al.* (2018) apresentam a Figura 16, que traz uma comparação entre a *SELEX* e a *CGA Shape*. Na coluna (a), são representados dois *layouts* de fachadas. Após uma operação de redimensionamento, na coluna (b), são mostrados os resultados da *CGA Shape*, e na coluna (c), os resultados da *SELEX*. Pode-se perceber que algumas restrições são violadas por meio da utilização da *CGA Shape*. No primeiro caso, há uma pequena diferença na largura da forma da parede no segundo andar à esquerda e à direita da fachada. Devido a esta diferença, após o redimensionamento, a regra de repetição para as janelas à esquerda e à direita produz um número diferente de repetições. No segundo caso, é mostrada uma ocorrência de desalinhamento. As janelas do segundo ao quarto andar são alinhadas no *layout* de entrada. Contudo, a *CGA Shape* quebra o alinhamento entre os andares três e quatro.

Jiang *et al.* (2018) argumentam que o principal intuito de uma gramática é derivar um *design* baseando-se em operações de divisão. Por exemplo, inicialmente, um modelo de massa é gerado. Logo após, as faces laterais do modelo de massa são extraídas como polígonos de fachada. Em seguida, os polígonos de fachada podem ser divididos em colunas ou pisos. Depois disto, os pisos podem ser divididos em ladrilhos e, por fim, em paredes, janelas ou portas. Desta maneira, a *SELEX* visa melhorar dois problemas predominantes nesta abordagem.

Em primeiro lugar, Jiang *et al.* (2018) afirmam que a *CGA Shape* fornece oportunidades limitadas para coordenar os diferentes ramos da derivação. Por exemplo, uma regra pode ser invocada para um ladrilho em algum lugar de um edifício para colocar uma janela e uma varanda. Depois, esta regra precisa decidir localmente como a janela e a varanda devem

Figura 16 – Comparação entre o *layout* gerado pela CGA Shape e SELEX.



Fonte: Adaptado de (Jiang *et al.*, 2018)

ser projetadas, de modo que as decisões de *design* sejam coordenadas com todos os outros elementos de um edifício. Uma vez que o alinhamento correto dos elementos é difícil de modelar, a solução proposta é a introdução de construções de linguagem que possibilitam uma melhor comunicação entre as diferentes partes de um *layout*, visando utilizar uma visão global para descrever as principais decisões de *design* envolvidas na modelagem. Por exemplo, ao invés das janelas decidirem localmente que tamanho, alinhamento e tipo devem ter, são escritas regras globais que descrevam onde colocar quais tipos de janelas. Assim, há uma alteração das regras no formato *label* → *actions* para regras no formato *selection-expression* → *actions*. Enquanto trabalhos anteriores concentravam-se, principalmente, em melhorias no lado direito de uma regra, a SELEX propõe extensões também no lado esquerdo de uma regra.

Em segundo lugar, Jiang *et al.* (2018) afirmam que existem algumas desvantagens na abordagem de divisão hierárquica utilizada pela CGA Shape e CGA++. Por exemplo, existem diversas maneiras de ver o mesmo edifício, ou seja, olhando para uma fachada, pode-se estar interessado em expressar uma operação de modelagem em termos de pisos, em termos de colunas, ou algum subconjunto de ladrilhos. Neste contexto, a escrita da regra força um edifício a ser dividido em uma única hierarquia, o que é um problema. Assim, se o autor da regra se comprometer com uma subdivisão com base no andar, será muito difícil expressar as operações de modelagem que precisam ser coordenadas entre várias colunas e vice-versa. Por exemplo, na Figura 17(c), a fachada é dividida em uma região à esquerda, uma coluna da porta, e uma região à direita. Em seguida, as regiões individuais são divididas em andares. Desta maneira, tal estrutura impõe uma hierarquia que tem regiões intermediárias desnecessárias que são difíceis

de nomear semanticamente, e apresentam problemas para formular seleções. Para superar estas limitações, a *SELEX* permite o gerenciamento de várias hierarquias simultaneamente, oferecendo suporte para múltiplas visualizações dos dados sem dividir as formas.

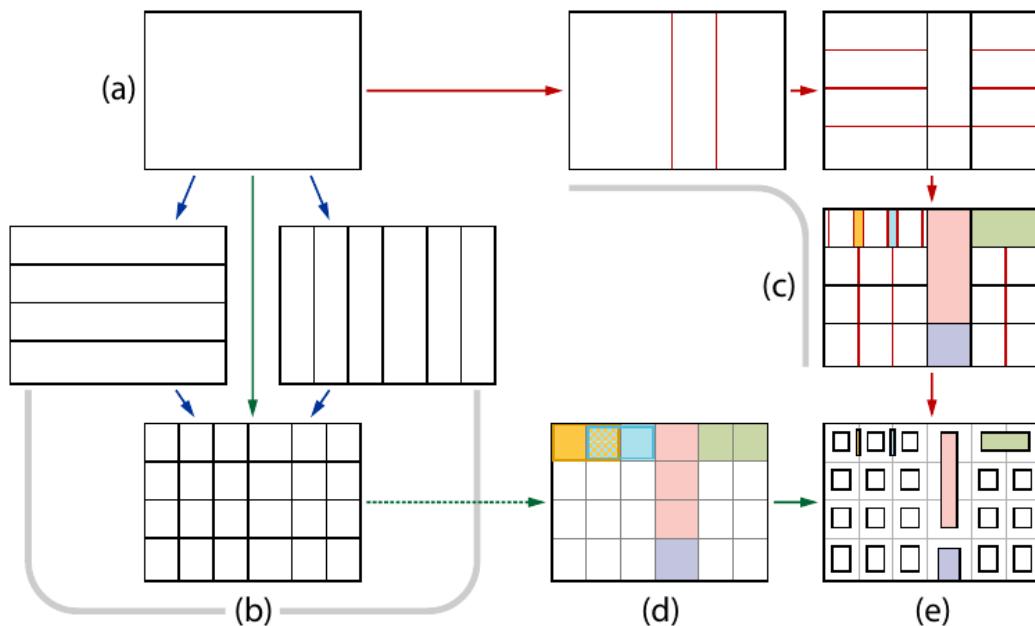
Na Figura 17, Jiang *et al.* (2018) ilustram os problemas com a abordagem anterior e as vantagens do uso da *SELEX*:

- (a) Fachada vazia;
- (b) Grades são utilizadas para permitir diferentes visualizações de *design*, como linhas, colunas, sub grades ou células individuais da grade;
- (c) Problemas de divisão das abordagens atuais, as quais não permitem a fusão de células. Desta maneira, ao se realizar divisões para estabelecer regiões de múltiplas células, sequências complexas e difíceis divisões verticais e horizontais alternadas são necessárias, o que é difícil de gerenciar;
- (d) A *SELEX* permite selecionar, arbitrariamente, sub-grades retangulares de uma grade, e posicionar elementos em relação a elas, permitindo a modelagem de uma forma natural e semanticamente significativa;
- (e) Na geração de fachadas de edifícios, para simplificar o alinhamento, os elementos são organizados de acordo com uma grade. Contudo, elementos únicos podem abranger várias células da grade ou serem colocados entre as células da grade. Em particular, pode-se incorporar elementos que abrangem duas células, cada uma delas contendo um outro elemento, como os ornamentos amarelos e azuis no último andar. Para isto, a *SELEX* fornece suporte à seleções sobrepostas, onde pode ser necessário a seleção de uma única célula para colocar cada janela, e uma seleção de célula dupla para colocar ornamentos, por exemplo.

2.4.1 Definições de forma

Jiang *et al.* (2018) argumentam que a maneira de organizar o conjuntos de formas produzidas pela linguagem é uma importante decisão de *design*. Neste contexto, uma escolha é simplesmente operar em um conjunto de formas sem qualquer hierarquia, conforme mostrado na Figura 18(e). Outra abordagem clássica é organizar as formas em uma árvore, porém, isto requer o comprometimento com uma hierarquia específica. Ou seja, para fachadas, várias hierarquias (ou árvores) podem existir a qualquer momento, e as operações de modelagem são normalmente expressas em diferentes hierarquias. Por exemplo, em alguns casos, as janelas

Figura 17 – Ilustração dos paradigmas de modelagem SELEX e CGA Shape.



Fonte: (Jiang *et al.*, 2018)

devem ser selecionadas com base nos pisos (linhas) e, outras vezes, com base nas colunas da fachada, conforme mostrado nas Figuras 18(b) e 18(c), respectivamente. Uma outra possibilidade é criar e manter explicitamente várias hierarquias em paralelo, conforme mostrado na Figura 18(d). No entanto, isto pode levar a problemas de consistência, uma vez que existe um grande número de maneiras intermediárias possíveis para agrupar outras formas.

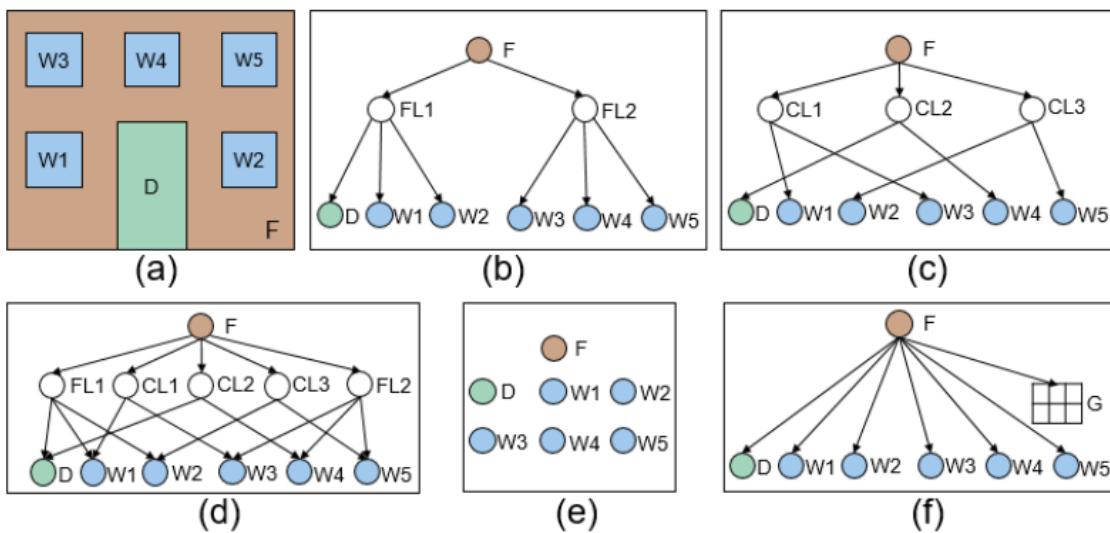
Para solucionar este desafio Jiang *et al.* (2018) utilizam grades como formas virtuais, as quais permitem o gerenciamento de qualquer sub-região como forma auxiliar, sem a necessidade de gerar a sub-região explicitamente, conforme ilustrado na Figura 18(f). Existem, assim, dois tipos de formas diferentes: formas virtuais e formas de construção. As formas virtuais possuem várias linhas e colunas, consistindo em diversas células, as quais podem ser aplicadas em formas de construção 2D, com objetivo exclusivo de orientar o posicionamento de outras formas de construção, mas não de subdividi-las. Para uma forma de construção, podem existir diversas formas virtuais, permitindo a modelagem de *layouts* complexos.

Conforme apresentado por Jiang *et al.* (2018), formas virtuais podem ser utilizadas nos seguintes casos:

- Alocar uma posição a partir da seleção de uma célula da grade, na qual determinada forma pode ser adicionada;
- Facilitar a seleção das formas de construção que estão contidas dentro delas, por exemplo, para selecionar formas na mesma linha ou coluna;

- Definir o comportamento de redimensionamento, uma vez que a especificação da grade inclui informações sobre o espaçamento de linhas e colunas, e a forma como as linhas e colunas se repetem, enquanto houver espaço disponível.

Figura 18 – As ilustrações (b), (c), (d), (e) e (f), representam diferentes perspectivas da árvore de formas em relação ao *design* da fachada (a).



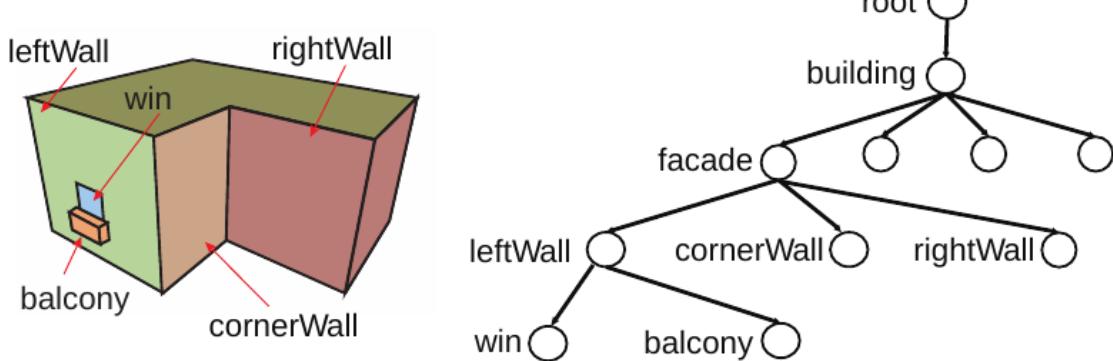
Fonte: (Jiang *et al.*, 2018)

De acordo com Jiang *et al.* (2018), a *SELEX* utiliza um conjunto de atributos integrados para cada forma. *Label* é o nome da forma, por exemplo, "window". O *Type* indica se a forma é uma forma virtual ("virtual") ou uma forma de construção ("construction"). *Dim* é uma variável binária que indica uma forma 2D ou 3D. O escopo descreve uma caixa orientada no espaço 3D, utilizando variáveis que descrevem um quadro de coordenadas (*xaxis*, *yaxis*, *zaxis*), uma posição em R^3 (denotada por *xpos*, *ypos*, *zpos*), e informações de tamanho (*xsize*, *ysize*, *zsize*).

Jiang *et al.* (2018) afirmam que a linguagem da *SELEX* desenvolve uma hierarquia de formas e as armazena em uma árvore, por meio da utilização de certas funções. Cada forma pode ter apenas um pai, e apenas formas de construção podem ter filhos. O nó raiz é uma forma com o rótulo "root". As formas 2D são utilizadas para criar subdivisões em outras formas 2D, por exemplo, fachadas ou janelas. As formas 3D são utilizadas para modelar elementos que são gerados através de operações de extrusão da fachada ou estruturas suspensas, como uma varanda ou um ornamento na janela, por exemplo. Na hierarquia de formas da *SELEX*, as formas têm filhos que são formas anexadas ou formas contidas. Uma forma anexada é uma forma 3D vinculada a uma forma 2D. Em algumas situações, a forma anexada tem uma face contida na

forma 2D, às vezes, as formas não se tocam. Por exemplo, a forma 3D de uma varanda pode ser anexada a uma forma 2D, que descreve a posição da janela dentro de uma fachada. Uma forma contida é uma forma 2D presente dentro de sua forma 2D pai, ou uma face lateral de uma forma 3D pai. Uma forma conectada é uma forma 2D que compartilha uma aresta com outra forma 2D. Topologicamente, uma forma conectada é considerada uma vizinha no contexto de *selection-expressions*. A Figura 19 ilustra o conceito de forma contida, anexada e conectada.

Figura 19 – Para a forma atual (*leftWall*), é demonstrada uma forma contida (*win*), uma forma anexada (*balcony*) e uma forma conectada (*cornerWall*).



Fonte: (Jiang *et al.*, 2018)

2.4.2 Introdução à linguagem

Jiang *et al.* (2018) explicam que a linguagem de modelagem procedural *SELEX* executa um comando por vez, podendo ser uma regra ou uma atribuição de variável. Uma regra tem o seguinte formato:

$$\text{selection-expression} \rightarrow \text{actions},$$

na qual a *selection-expression* seleciona uma lista de formas da árvore de formas, e as *actions* são comandos a serem executados. Por exemplo, na Figura 20, os comandos de C2 a C11 são regras. Uma atribuição tem o formato:

$$\text{identifier} = \text{expression},$$

na qual o *identifier* denota uma variável e a *expression* representa um valor. Os comandos rotulados como C1, na Figura 20, são atribuições de variáveis. Uma lista é uma lista de outros tipos de objetos, mas também há suporte para listas de listas. Um par consiste em dois objetos, no qual o primeiro objeto deve ser comparável, podendo ser um número, uma *string* ou um valor

booleano, e o segundo objeto pode ser de qualquer tipo.

Figura 20 – Instruções *SELEX* para gerar a fachada da Figura 17(e).

```

1  ##C1:
2  facW = 17.6; facH = 12.8;
3
4  ##C2:
5  { <> -> addShape("facade", ...); }
6
7  ##C3: split facade into cells and set it as the working node
8  { <[label=="facade"]> -> createGrid("main", ...); }
9  { <[label=="facade"]> [label=="main"] -> setHeadNode(); }
10
11 ##C4: add a door touching the ground;
12 {<cell() [colLabel=="mid"] [rowLabel=="gnd"]>
13 -> addShape("door", ...); }
14
15 ##C5: add the glass windows above the door
16 { <cell() [colLabel=="mid"] [rowIdx>1] [:groupCols()]>
17 -> addShape("glass", ...); }
18
19 ##C6: add ledges at the left side of the top floor.
20 { <cell() [colLabel=="left"] [rowLabel=="top"] [:groupPairs()]>
21 -> addShape("ledge", ...); }
22
23 ##C7: add the two-cell window at the right side of the top floor.
24 {<cell() [colLabel=="right"] [rowLabel=="top"] [:groupRows()]>
25 -> addShape("win4", ...); }
26
27 ##C8: add windows on the ground floor.
28 { <cell() [colLabel in ("left", "right")] [rowIdx==1]>
29 -> addShape("win1", ...); }
30
31 ##C9: add windows on the second and third floor.
32 { <cell() [colLabel in ("left", "right")] [rowIdx in rowRange(2,-2)]>
33 -> addShape("win2", ...); }
34
35 ##C10: add windows in the left side of the top floor.
36 { <cell() [colLabel=="left"] [rowLabel=="top"]>
37 -> addShape("win3", ...); }
38
39 ##C11: generate walls to fill the space.
40 { <root()/[label=="facade"]> -> coverShape(); }
```

Fonte: (Jiang *et al.*, 2018)

2.4.3 Modelagem procedural baseada em seleção

Segundo Jiang *et al.* (2018), o recurso mais importante da *SELEX* são as *selection-expressions*, que permitem selecionar os elementos de uma árvore de formas através de seletores

intercalados com o operador /. Cada seletor recebe uma lista de formas como entrada e retorna uma lista de formas. A entrada implícita para o primeiro seletor é uma lista contendo o nó raiz da árvore de formas. O operador / obtém uma lista de formas como entrada e executa os comandos definidos.

Jiang *et al.* (2018) argumentam que seletores são agrupados em sequências de seletores especializados, podendo ser de três tipos diferentes: seletor de topologia, como *child*, *descendant* e *parent*; seletor de atributo, como `[label=="window"]`; e seletor de grupo, como `[:: groupRows()]`. Os seletores precisam ocorrer na ordem em que foram fornecidos, ou seja, não podem ser misturados em uma sequência arbitrária.

Na definição de Jiang *et al.* (2018), uma *selection-expression* possui a seguinte configuração:

$$<[topoS][attrS \mid groupS]^*/[topoS][attrS \mid groupS]^*/\dots >,$$

onde, dentro de cada sequência de seletor, há de zero a um seletores de topologia (*topoS*), zero a muitos seletores de atributo (*attrS*), e zero a muitos seletores de grupo (*groupS*). Uma *selection-expression* vazia retorna a entrada. Quando uma forma que não existe é especificada, a seleção correspondente retornará uma forma vazia e a regra não será executada. Um exemplo é mostrado na Figura 21, por meio de um grafo abstrato, no qual os nós selecionados pela *selection-expression*

$$<[label="A"]/[label="C"]/[h \geq 2]>$$

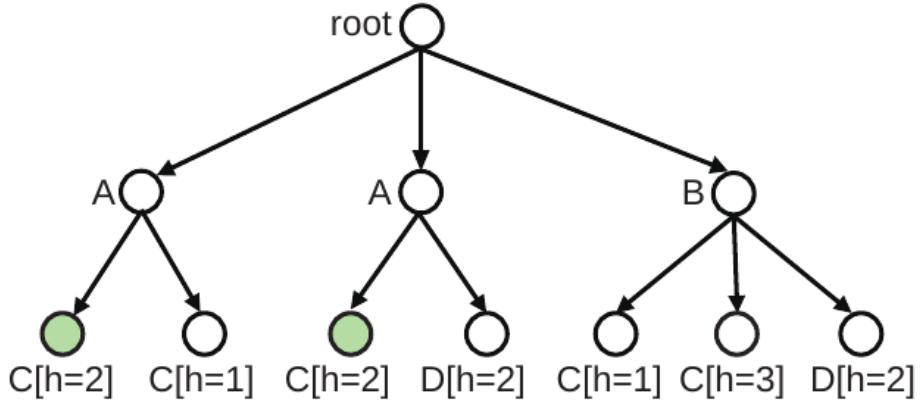
são destacados em verde. No processo, a *selection-expression* seleciona os filhos rotulados como "A" do nó raiz, então, para os dois nós rotulados como "A", todos os seus filhos rotulados como "C", com um valor de atributo $h \geq 2$, são selecionados.

Por se tratar de um importante recurso da *SELEX*, Jiang *et al.* (2018) apresentam algumas especificações mais detalhadas sobre os diferentes tipos de seletores:

Seletor de topologia: Recebe uma lista contendo uma única forma como entrada, e produz uma lista de formas com a relação de topologia especificada para a forma de entrada. Um seletor de topologia tem a forma `[topology-function()]`, e pode utilizar uma das seguintes funções: *child()*, *descendant()*, *parent()*, *root()*, *neighbor()* e *contained()*.

Seletor de atributo: Recebe uma lista de formas e retorna uma lista de formas cujos atributos satisfazem algumas condições. Em sua estrutura básica, um seletor de atributo tem a configuração `[attribute-name comparison value]`. Os operadores relacionais são especificados

Figura 21 – Grafo abstrato da seleção de nós.



Fonte: (Jiang *et al.*, 2018)

como em outras linguagens de programação, por exemplo, `==`, `<=`, `>=`, `!=` e `in`. Os exemplos são `[label=="facade"]` e `[label in ("window_arch", "window_rect")]`. Alternativamente, em sua estrutura mais geral, um seletor de atributo é uma expressão *booleana*. Alguns outros exemplos são `[isEmpty()]`, `[numCols() > 4]` e `[toShapeX(0.5) > 2]`. A função `toShapeX(0.5)` dimensiona o valor de entrada 0.5 pelo `xsize` de uma forma. Uma função importante é a `pattern(regex, pat)`, que verifica se o padrão de caracteres de `regex`, na posição do índice de uma forma, corresponde à `pat`. Por exemplo, `pattern("(AB)*", "A")` testa se uma forma de entrada está em uma posição de índice ímpar, e `pattern("A(B)*A", "A")` testa se uma forma de entrada está na primeira ou na última posição de uma lista de entrada. As funções `isEven()` e `isOdd()` são casos especiais do comando `pattern(regex, pat)`, que verificam se uma forma tem um índice par ou ímpar em uma lista de formas selecionadas. Os atributos `rowIdx` e `colIdx` são utilizados como a posição topológica de uma célula em relação à região abrangida pelas formas virtuais de entrada. Por exemplo, `rowIdx == 1 && colIdx == 1` especifica a célula superior esquerda de uma grade.

Seletor de grupo: Recebe uma lista de formas como entrada e aplica operações de agrupamento para retornar uma lista de formas combinadas. Um seletor de grupo opera apenas em formas virtuais e regrupa sub-regiões, por exemplo, combinando células de uma forma virtual para representação de pisos. Os seletores de grupo são implementados utilizando funções de agrupamento, resultando em seletores na seguinte configuração: `[:: grouping-function()]`. Algumas funções que operam juntamente com seletores de grupo são:

- `groupRows()` e `groupCols()`: mesclam formas virtuais adjacentes, ou seja, *cells*, com o mesmo índice de linha ou coluna, respectivamente;
- `groupRegions()`: mescla todas as formas virtuais adjacentes que formam uma ou várias

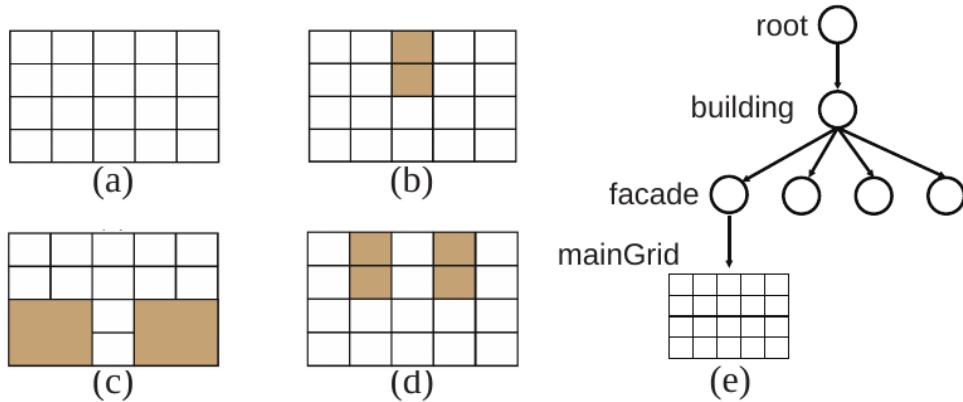
regiões retangulares;

- *groupEach(n)*: mescla todas as n formas virtuais adjacentes.
- *groupPair()*: gera todos os pares possíveis de duas formas subsequentes. Por exemplo, dado "ABCD", ele retornará "AB", "BC" e "CD";
- *cells()*: decompõe uma forma virtual como uma lista de formas virtuais com uma célula.

Nas Figuras 22 e 23, respectivamente, Jiang *et al.* (2018) ilustram a utilização de alguns seletores e funções de agrupamento:

- (a) Uma grade é utilizada para demonstrar múltiplas seleções;
- (b) `<descendant()>[label=="facade"]/[label=="mainGrid"]/[type=="cell"]`
`[rowIdx in (1,2)][colIdx==3]>;`
- (c) `<descendant()>[label=="facade"]/[label=="mainGrid"]/[type=="cell"]`
`[rowIdx in (3,4)][colIdx in (1,2,4,5)][::groupRegions()]>;`
- (d) `<descendant()>[label=="facade"]/[label=="mainGrid"]/[type=="cell"]`
`[rowIdx in (1,2)][colIdx in (2,4)][::groupCols()][::cells()]>;`
- (e) Representação da árvore de formas.

Figura 22 – Exemplo da utilização de seletores.

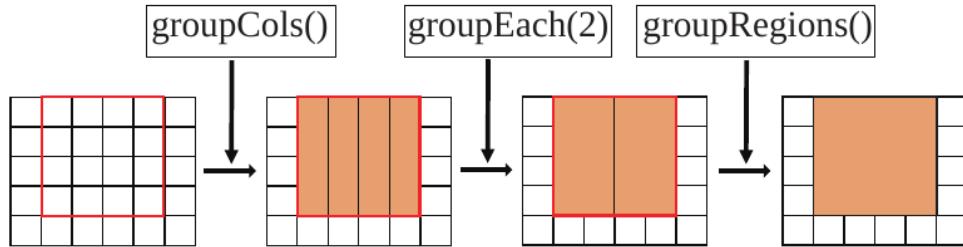


Fonte: (Jiang *et al.*, 2018)

2.4.4 Ações

As *actions* são descritas por Jiang *et al.* (2018) como funções que podem ser executadas sobre as formas retornadas por uma *selection-expression*, sendo amplamente utilizadas para criar novas formas e as adicionar na hierarquia de formas, por exemplo, "*addShape*", "*attachShape*", "*coverShape*" e "*connectShape*". O pai de uma nova forma pode ser especificado

Figura 23 – Exemplo da utilização de funções de agrupamento.

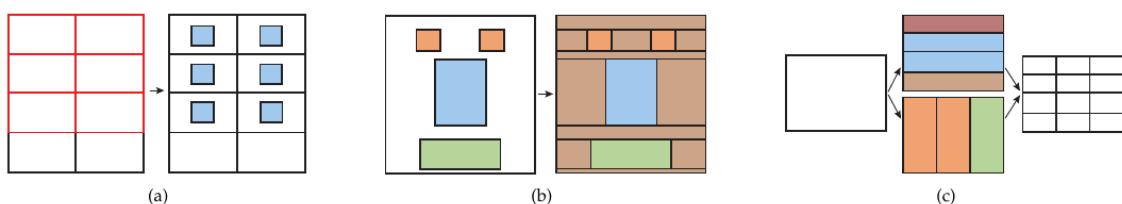


Fonte: (Jiang *et al.*, 2018)

explicitamente ou implicitamente utilizando valores padrões. Normalmente, o pai é a forma de construção de entrada ou, no caso de uma forma virtual, o primeiro ancestral que é uma forma de construção. A Figura 24(a) ilustra um exemplo da função "*addShape*". Para gerar a fachada da Figura 17, as instruções presentes na Figura 20 fazem uso intensivo da função "*addShape*", uma vez que o exemplo é 2D.

Jiang *et al.* (2018) afirmam que outro aspecto da modelagem de edifícios residenciais complexos é a exigência de um controle cuidadoso da geometria que está sendo gerada. Para isto, são utilizados comandos para criar geometria dentro de formas que não são cobertas por outras formas com a função "*coverShape*", conforme mostrado na Figura 24(b). Além disto, a *SELEX* também fornece diversas funções para criar formas virtuais (grades), como no exemplo da Figura 24(c).

Figura 24 – Exemplo da utilização de *actions*.



Fonte: (Jiang *et al.*, 2018)

2.4.5 Funções de restrição

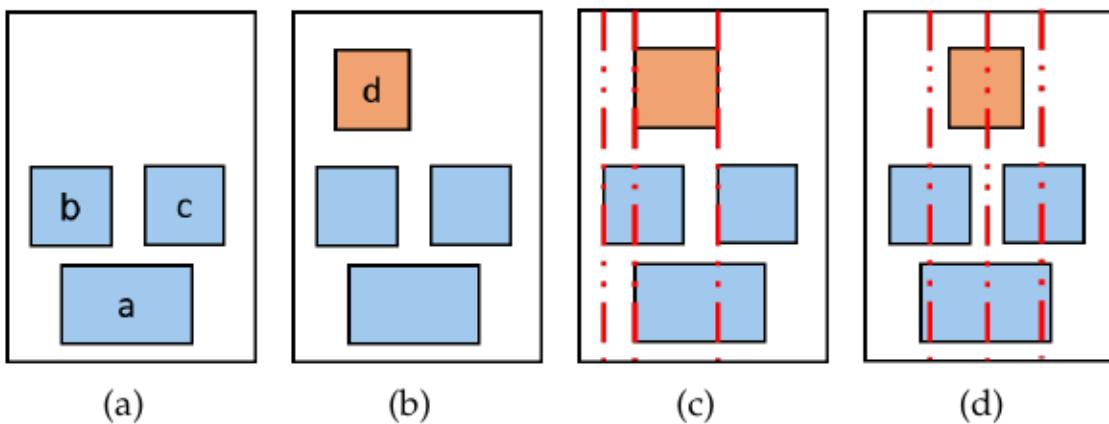
Segundo Jiang *et al.* (2018), existe uma grande dificuldade para especificar a localização de uma forma em uma gramática estocástica, pois não é possível saber exatamente quais formas foram colocadas anteriormente e onde estão. Neste contexto, a *SELEX* permite que seja definida uma sequência de restrições a partir do seguinte comando:

constrain(constraint1, constraint2, ...).

Jiang *et al.* (2018) consideram duas opções de *design* para o problema de alinhamento. A primeira escolha de *design* é decidir se as linhas de ajuste devem ser colocadas explicitamente por um comando, ou ser definidas implicitamente pelos limites das formas geradas numa etapa anterior. A segunda opção de *design* é decidir se as linhas de ajuste devem ser nomeadas com um rótulo ou não. Na Figura 25, é mostrado um exemplo comparando o ajuste com e sem rótulos, no qual:

- (a) Dado um *layout* inicial com as formas *a*, *b* e *c*;
- (b) Uma nova forma rotulada *d* é adicionada;
- (c) No ajuste sem rótulos, a forma *d* se encaixa nas linhas de ajuste mais próximas, ou seja, à esquerda da forma *a* e à esquerda da forma *c*;
- (d) Utilizar ajuste com rótulos permite um controle mais preciso, por exemplo, ajustando a coordenada x ao centro da forma *d* à coordenada x ao centro da forma *a*.

Figura 25 – Comparaçāo do ajuste com e sem o uso de rótulos.



Fonte: (Jiang *et al.*, 2018)

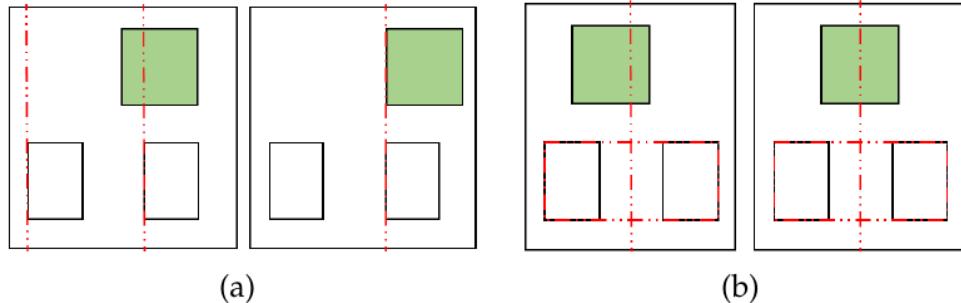
Conforme apresentado por Jiang *et al.* (2018), o alinhamento pode ser especificado de duas maneiras: uma forma de entrada ou uma forma de referência especificada por um rótulo. Os tipos de alinhamento suportados pela *SELEX* são: "left", "right", "top", "bottom", "center-x", "center-y", "one2two-x" e "one2two-y". Por exemplo, a função:

```
constrain(snap2("window", "left"), snap2("window", "center-x"))
```

especifica que uma forma de entrada deve ser alinhada à esquerda e ao centro x em relação a uma forma identificada como "window". Os exemplos de "left" e "one2two-x" são ilustrados na Figura 26, onde: (a) "left" alinha a forma de entrada em verde a uma forma de referência em branco, e (b) "one2two-x" alinha a forma de entrada em verde ao centro da caixa delimitadora

de duas formas de referência brancas. A linha tracejada vermelha indica a posição de ajuste, enquanto a caixa vermelha marca a delimitação de duas formas de referência.

Figura 26 – Dois exemplos de alinhamento.



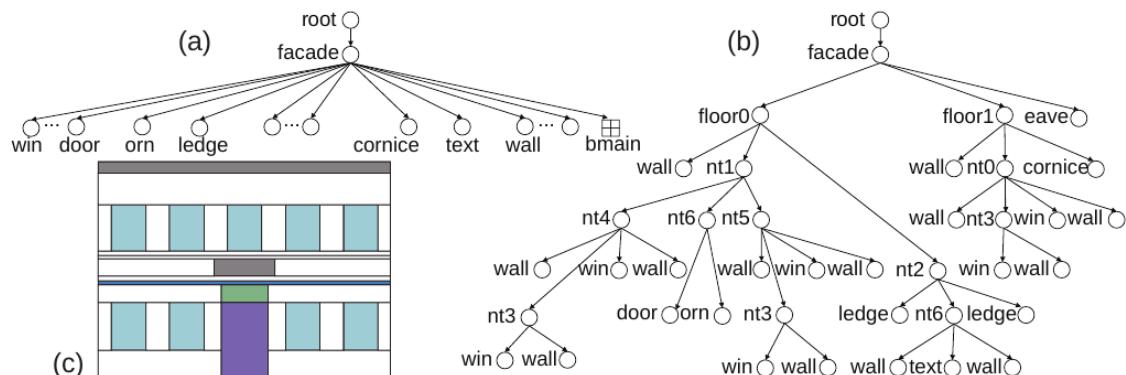
Fonte: (Jiang *et al.*, 2018)

Especificações adicionais sobre a *SELEX* foram disponibilizadas por Jiang *et al.* (2018) como material suplementar.

2.4.6 Comparativo

Para evidenciar a simplicidade com que as regiões de uma fachada são representadas na árvore de formas, Jiang *et al.* (2018) apresentam um exemplo ilustrativo, comparando o resultado da *SELEX* com o resultado da *CGA Shape*, conforme mostrado na Figura 27(a) e Figura 27(b), respectivamente. Neste caso, percebe-se que a árvore de formas produzida pela *SELEX* apresenta uma estrutura mais plana, resultando em menos nós intermediários. Isto acontece pelo fato da *SELEX* não possuir uma forma para representar os pisos, uma vez que o acesso das linhas e colunas pode ser facilmente realizado por meio de uma consulta na forma virtual.

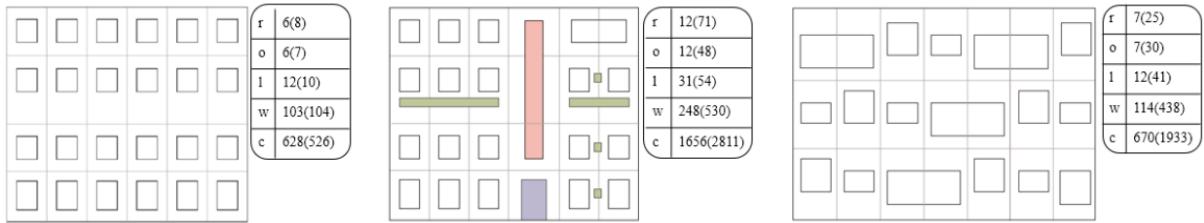
Figura 27 – Exemplo da árvore de formas gerada pela *CGA Shape* em (a), e pela *SELEX* em (b), com base na fachada (c).



Fonte: (Jiang *et al.*, 2018)

Ainda abordando estas duas técnicas, Jiang *et al.* (2018) apresentam outros três exemplos, mostrados na Figura 28, fazendo um comparativo entre a quantidade de regras r para gerar as fachadas; o número de operações o ; o número de linhas l ; o número de palavras w ; e o número de caracteres c . Neste caso, os dados referentes à *CGA Shape* estão entre parênteses.

Figura 28 – Estatísticas da *SELEX* e *CGA Shape* para geração de fachadas.



Fonte: (Jiang *et al.*, 2018)

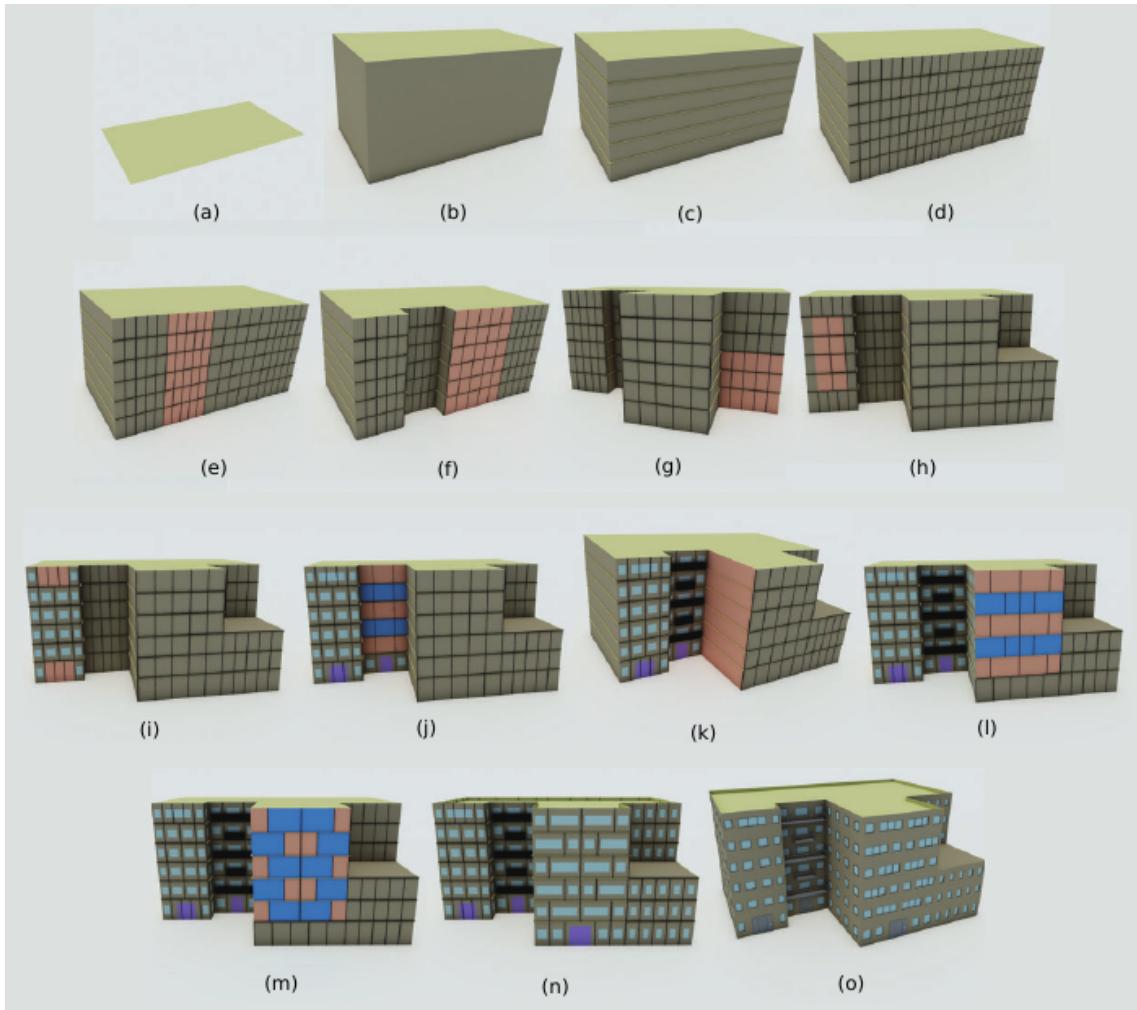
2.4.7 Aplicações

Nas Figuras 29 e 30, Jiang *et al.* (2018) ilustram a utilização de formas virtuais e *selection-expressions* no processo de modelagem de um edifício, a partir das seguintes etapas:

- (a) Polígono da planta especificada pelo usuário;
- (b) Uma operação de extrusão é aplicada ao polígono da planta, gerando um edifício;
- (c) Todas as fachadas do edifício são divididas em andares, por meio da inclusão de uma grade como forma virtual;
- (d) Cada fachada herda as informações do piso e é dividida em uma grade mais fina, especificando colunas, que são rotuladas com "colLeft", "colMidLeft", "colMidRight" e "colRight";
- (e) As colunas são selecionadas pelo rótulo "colMidLeft";
- (f) As colunas são selecionadas pelo rótulo "colMidRight" e a operação *push* é aplicada na região rotulada com "colMidLeft";
- (g) As colunas são selecionadas pelo rótulo "rowDown" e a operação de *pull* é aplicada nas regiões rotuladas com "colMidRight" e "rowDown", formando a massa do edifício;
- (h) Uma sub-região no lado esquerdo é selecionada e uma sub-grade é adicionada;
- (i) Conjuntos de células na grade principal são selecionadas para adicionar formas abrangendo várias células;
- (j) Linhas pares/ímpares em uma sub-região da segunda linha à última linha são selecionadas;
- (k) Uma forma de conexão é selecionada;

- (l) Linhas pares/ímpares são selecionadas e sub-grades diferentes são adicionadas;
- (m) Colunas largas e estreitas são selecionadas para adicionar janelas;
- (n) Janelas e portas extras são adicionadas;
- (o) Complementos são adicionados.

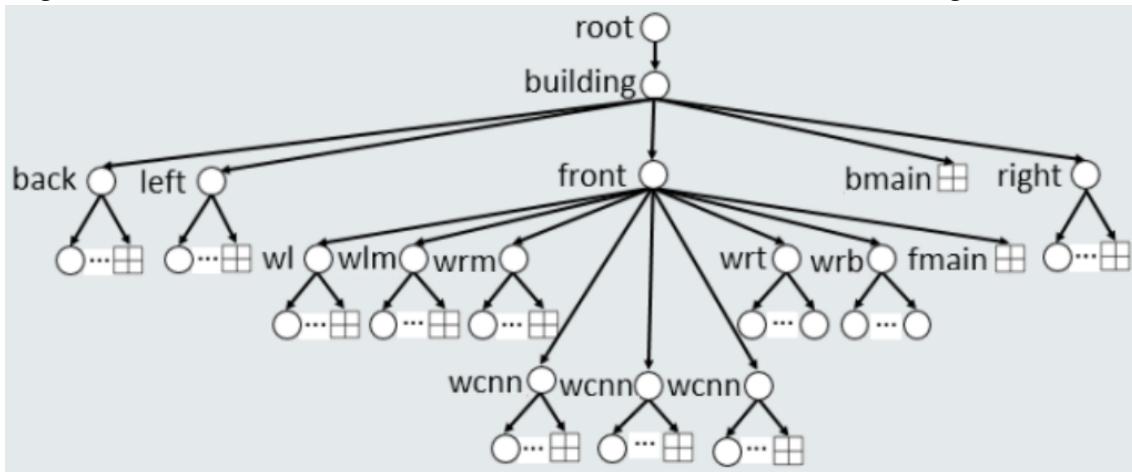
Figura 29 – Exemplo de modelagem utilizando *SELEX*.



Fonte: Adaptado de (Jiang *et al.*, 2018)

Por fim, após a apresentação das especificações, dos exemplos ilustrativos, e da análise do desempenho da *SELEX* em relação às suas precursoras, pode-se perceber que tal técnica representa um grande avanço no campo da modelagem procedural de edifícios. Assim, na próxima seção, será abordado o conceito de deformação, bem como sua relação com a modelagem procedural.

Figura 30 – Árvore de formas construída com base no modelo final da Figura 29(o).



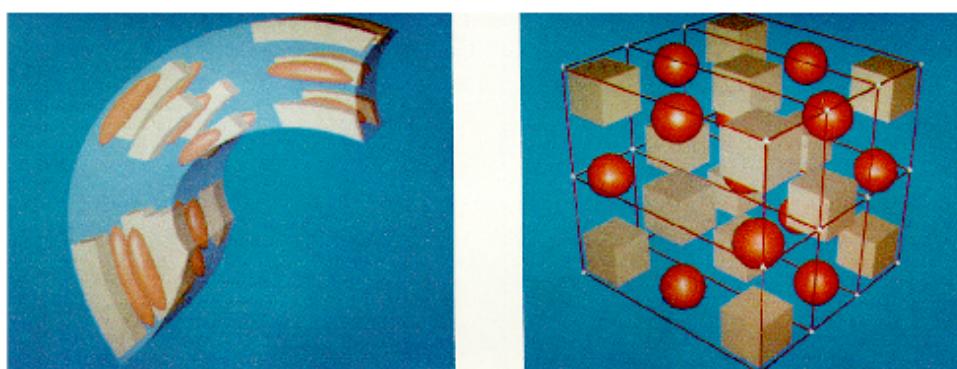
Fonte: Adaptado de (Jiang *et al.*, 2018)

2.5 Deformação

No campo da Física, a deformação de uma estrutura é qualquer mudança da configuração geométrica do corpo que leve a uma variação da sua forma ou das suas dimensões, após a aplicação de uma ação externa (TRUESELL; NOLL, 1992). Na área da Computação Gráfica, por sua vez, a modelagem de sólidos é uma das técnicas essenciais utilizadas em modeladores 3D comuns, como Maya, Rhinoceros e Blender, onde a modificação dos objetos é realizada, frequentemente, por meio da deformação de forma livre (PROCHÁZKOVÁ, 2017).

Idealizada por Sederberg e Parry (1986), a deformação de forma livre é uma transformação do espaço. Nela, é definida uma grade regular de pontos de controle, os quais podem ser descritos por uma combinação linear. Ao deslocar estes pontos de controle, uma deformação do espaço é alcançada. Geralmente, estes pontos de controle são espaçados regularmente na caixa delimitadora do elemento a ser deformado, conforme ilustrado na Figura 31.

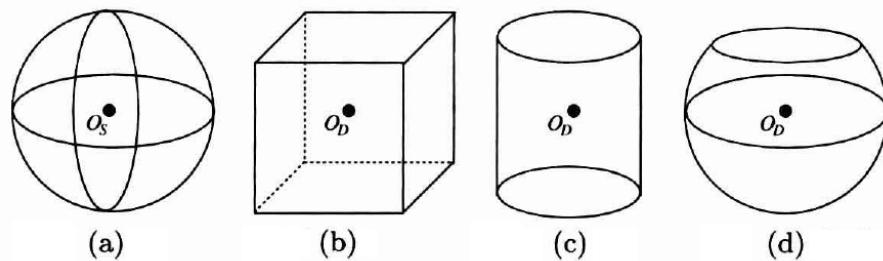
Figura 31 – À esquerda, um exemplo de deformação livre dos objetos do cenário, a partir da mudança dos pontos da grade cúbica visível à direita.



Fonte: (SEDERBERG; PARRY, 1986)

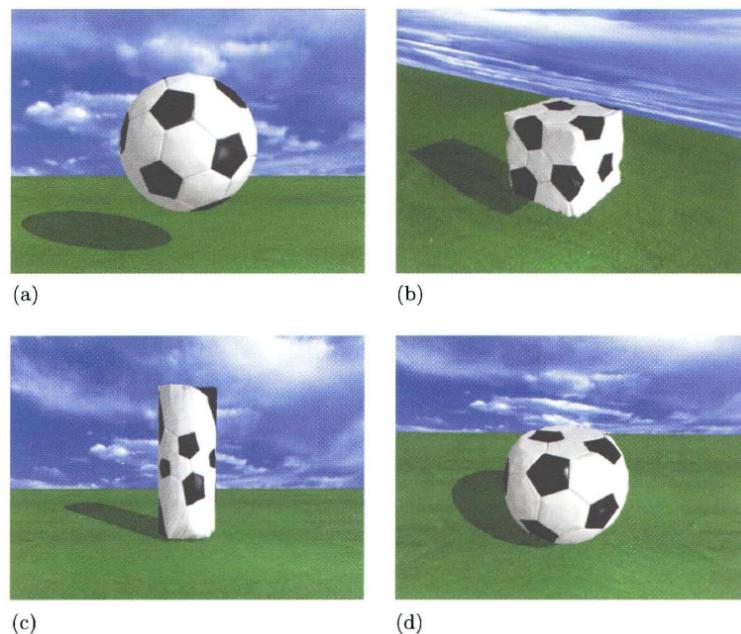
Uma evolução do trabalho de Sederberg e Parry (1986) é trazida por Jin e Li (2000), onde é apresentado um método de deformação tridimensional utilizando coordenadas polares direcionais. O usuário especifica um objeto de controle de origem e um objeto de controle de destino (Figura 32), que atuam como espaços de incorporação. Os objetos de controle de origem e de destino determinam uma transformação de volume tridimensional, que mapeia o espaço anexado no objeto de controle de origem para o espaço do objeto de controle de destino (Figura 33). Ao incorporar o objeto a ser deformado no objeto de controle de origem, a transformação ocorre sem o movimento dos pontos de controle.

Figura 32 – Objeto de controle de origem (a) e objetos de destino (b), (c), (d). Neste caso, O_S e O_D representam os respectivos centros dos objetos.



Fonte: Adaptado de (JIN; LI, 2000)

Figura 33 – Deformação de uma bola de futebol (a) com base nos objetos (b), (c) e (d), da Figura 32.

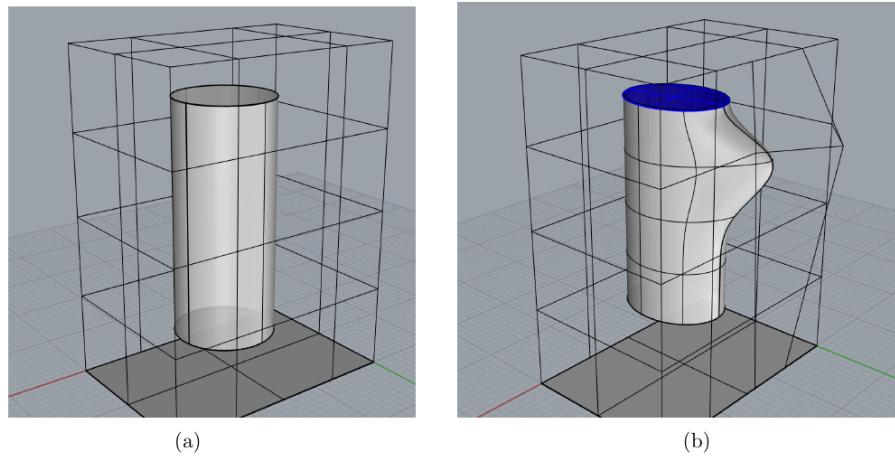


Fonte: Adaptado de (JIN; LI, 2000)

Outras duas técnicas são apresentadas por Procházková (2017), o esquema de Seder-

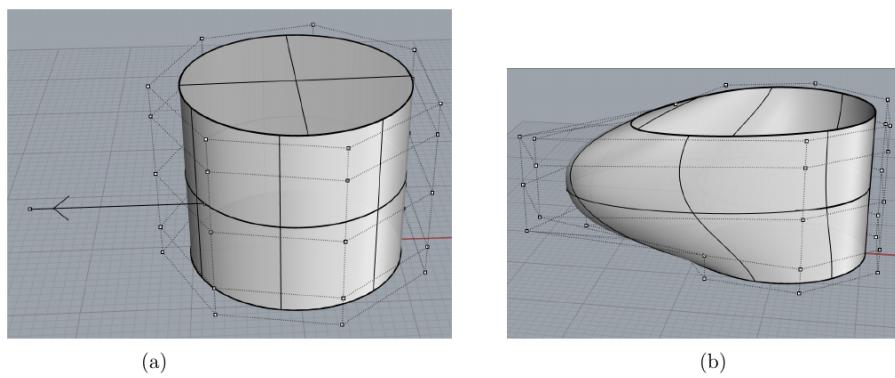
berg e Parry (1986) baseado em polinômios de Bernstein, que é a base das técnicas de deformação de forma livre (Figura 34), e o método *NURBS*, que utiliza um escopo mais elaborado e operações mais complexas (Figura 35).

Figura 34 – Técnica de deformação de forma livre de Sederberg: a) antes e b) depois da transformação.



Fonte: (PROCHÁZKOVÁ, 2017)

Figura 35 – Técnica de deformação de forma livre utilizando *NURBS*: a) antes e b) depois da transformação.

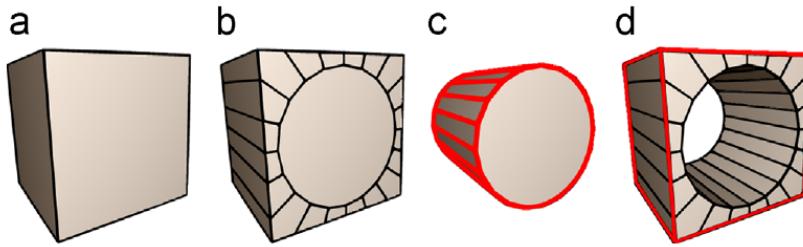


Fonte: (PROCHÁZKOVÁ, 2017)

Por achar o uso de caixas insatisfatório, Thaller *et al.* (2013) argumentam que uma maior variedade de formas pode ser obtida pelo uso de poliedros convexos como delimitador de volumes, pois as operações de divisão não ficam mais limitadas aos três eixos principais, podendo-se utilizar planos arbitrários. Um exemplo é mostrado na Figura 36, onde: (a) em um poliedro convexo, (b) um orifício circular é cortado deixando dois resultados, (c) uma parte interna convexa e (d) os arredores não convexos. A borda vermelha mostra o escopo convexo correspondente, que é igual ao escopo original.

As *deformation grammars*, por sua vez, foram introduzidas por Vimont *et al.* (2017),

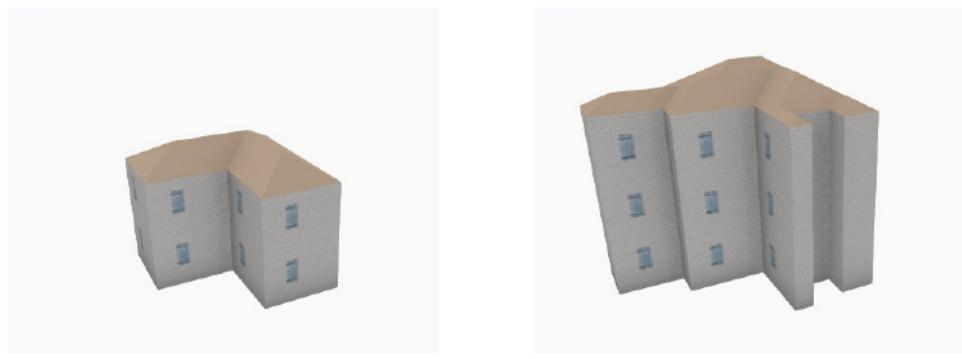
Figura 36 – Operação de deformação introduzida por Thaller *et al.* (2013).



Fonte: (THALLER *et al.*, 2013)

permitindo deformar livremente objetos complexos ou conjuntos de objetos, preservando sua consistência. Esta abordagem processa deformações de objetos como símbolos, através das regras de interpretação definidas pelo usuário. Um exemplo é mostrado na Figura 37.

Figura 37 – O modelo inicial de uma casa (a) é deformado pelo usuário, enquanto preserva as propriedades típicas, como a ortogonalidade da parede e a disposição linear do piso (b).



Fonte: (VIMONT *et al.*, 2017)

Neste capítulo, foram apresentadas algumas das principais técnicas que surgiram no decorrer da história para geração procedural de edifícios, abordando suas características, alguns exemplos de aplicação, e também suas limitações. Por meio de uma análise comparativa, percebeu-se que o emprego de *Selection Expressions*, a mais recente das abordagens, apresenta diversas vantagens em relação às suas precursoras, *CGA Shape* e *CGA++*. Contudo, mesmo com a introdução de tais melhorias pela *SELEX*, ainda existem modelos que estão além da sua capacidade de modelagem, como é o caso de edifícios com arquitetura arredondada. Assim, visando contextualizar este cenário, no próximo capítulo, serão abordadas algumas técnicas voltadas para a resolução de questões similares.

3 TRABALHOS CORRELATOS

Neste capítulo, de maneira cronológica, serão apresentadas algumas abordagens que trataram de problemáticas análogas ao do presente trabalho, descrevendo, brevemente, suas respectivas soluções.

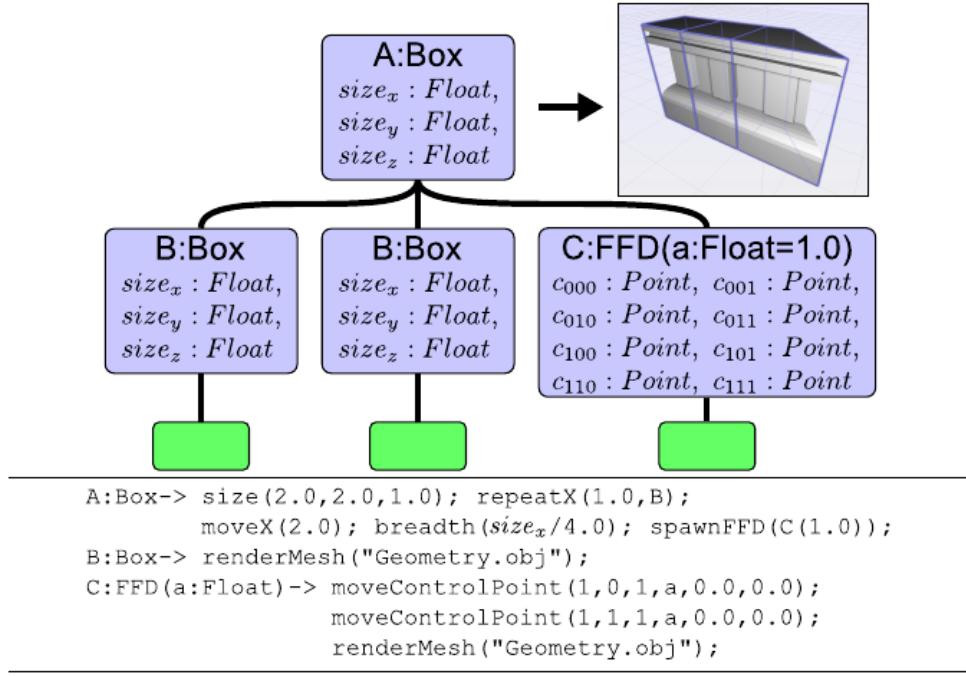
3.1 *Generalized Use of Non-Terminal Symbols for Procedural Modeling*

Krecklau *et al.* (2010) utiliza deformação de forma livre como um objeto não-terminal alternativo para superar a desvantagem da criação de objetos arredondados. Para isto, é introduzida a linguagem de modelagem procedural G^2 , que adapta vários conceitos de linguagens de programação de propósito geral, a fim de fornecer alto poder descritivo, com semântica bem definida, e uma sintaxe simples.

Segundo Krecklau *et al.* (2010), o termo "*generalized*" reflete dois tipos de generalização. Por um lado, entende-se como o escopo das linguagens de modelagem de arquitetura anteriores, permitindo vários tipos de objetos não-terminais com operadores e atributos específicos de domínio. Por outro lado, a linguagem também aceita símbolos não-terminais como parâmetros nas regras de modelagem, permitindo a definição de modelos de estrutura abstrata para reutilização dentro da gramática de maneira flexível.

Krecklau *et al.* (2010) afirmam que uma das principais características da G^2 é a introdução de classes não-terminais, as quais fornecem diferentes conceitos de modelagem. Assim, as regras da Figura 38 devem ser de um tipo específico, uma vez que os operadores de cada regra só podem ser aplicados a um determinado tipo de objeto não-terminal. Por exemplo, a classe não-terminal *Box* se comporta de maneira semelhante à *CGA Shape*, fornecendo transformações simples, bem como operadores de repetição e divisão para um objeto da cena. Além disto, as deformações de forma livre fornecem operadores para manipular os pontos de controle deste objeto. Todas as classes possuem atributos declarados implicitamente, os quais descrevem o objeto não-terminal. Uma *Box*, por exemplo, tem os três atributos $size_x$, $size_y$ e $size_z$, enquanto uma deformação de forma livre armazena as posições 3D de todos os seus pontos de controle c_{xyz} com $x, y, z \in \{0, 1\}$. Tais atributos podem ser, então, utilizados para realização de cálculos adicionais, conforme ilustrado na Figura 38, onde a regra C fornece uma declaração de parâmetro explícita, enquanto a regra A utiliza o parâmetro implícito $size_x$ do objeto não-terminal *Box*, que contém a largura atual.

Figura 38 – Aplicação de regras de modelagem da G^2 .

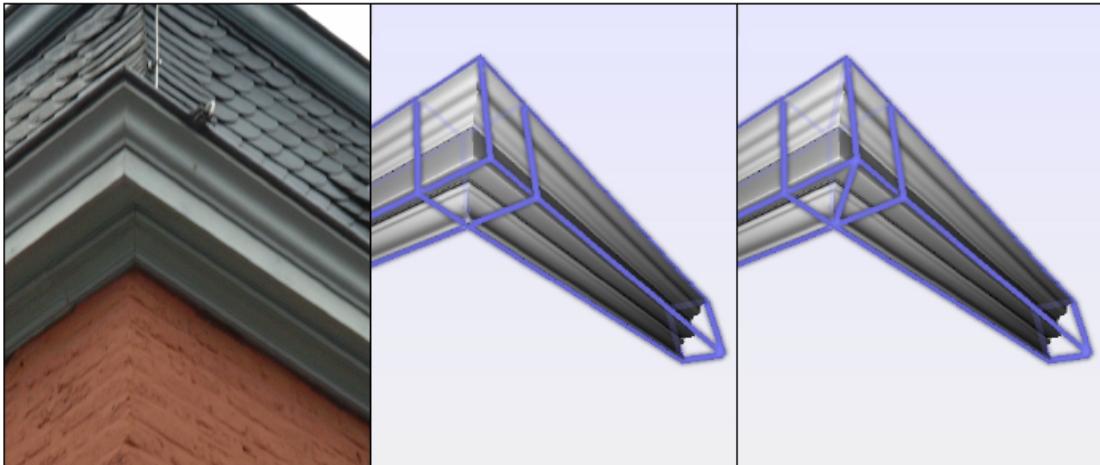


Fonte: (KRECKLAU *et al.*, 2010)

Para ilustrar a utilização da G^2 na geração de estruturas arredondadas, Krecklau *et al.* (2010) apresentam alguns exemplos. Na Figura 39 é mostrado um caso típico de beirais passando ao redor de uma borda. Na segunda imagem, uma nova geometria deve ser carregada para cobrir a região afiada. A terceira imagem mostra que deformações de forma livre resolvem o problema, e que a geometria utilizada para o beiral ao longo da parede pode ser reutilizada para o canto. A Figura 40, por sua vez, mostra a criação de bordas arredondadas após a aplicação de múltiplas deformações.

Como limitação da G^2 , Krecklau *et al.* (2010) apontam a incapacidade da realização de consultas geométricas, algo que é possível na *CGA Shape*, por exemplo. Além disto, também percebeu-se que as deformações desta abordagem são utilizadas apenas para geração de ornamentos arredondados, como os beirais da Figura 39, ou seja, não são aplicadas especificamente para geração de modelos de massa com geometria arredondada.

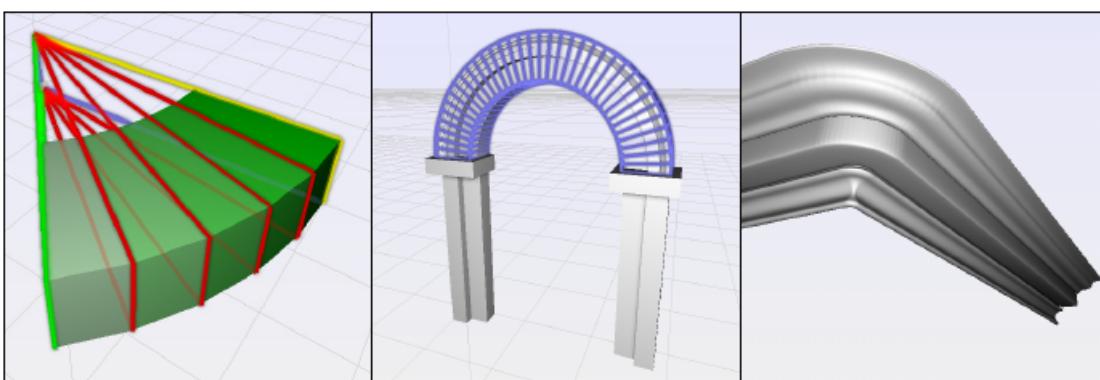
Figura 39 – Exemplo típico de beirais passando ao redor de uma borda.



```
$WallCornice:Box-> renderGeometry("Cornice.obj");
    cornerFFD(45.0, $LeftCorner);
$LeftCorner:FFD-> renderGeometry("Cornice.obj");
```

Fonte: (KRECKLAU *et al.*, 2010)

Figura 40 – Aplicação de sucessivas deformações de forma livre através da G^2 .



```
$WallCornice:Box-> renderGeometry("Cornice.obj");
    latheFFD(45.0, 2.0, $LeftCorner);
$LeftCorner:FFD-> renderGeometry("Cornice.obj");
```

Fonte: (KRECKLAU *et al.*, 2010)

3.2 Procedural architecture using deformation-aware split grammars

Uma extensão às *split grammars* é apresentada por Zmugg *et al.* (2014), permitindo a criação de arquiteturas curvadas através da integração de deformações de forma livre em qualquer nível de uma gramática.

Zmugg *et al.* (2014) afirmam que, geralmente, regras de divisão são realizadas de duas maneiras diferentes, ou podendo se adaptar às deformações, para que as repetições possam se ajustar a mais ou menos espaço, mantendo as restrições de comprimento; ou podem dividir a geometria deformada com planos, a fim de introduzir estruturas retas na geometria deformada.

De acordo com Zmugg *et al.* (2014), existem muitos edifícios e estruturas que podem ser entendidos como tendo uma forma reta, mas que, em algum momento, é distorcida para uma forma curvada. Um exemplo ilustrativo é mostrado na Figura 41, onde uma grande parede de pedra se estende por uma paisagem.

Figura 41 – Muralha que se estende em um terreno.

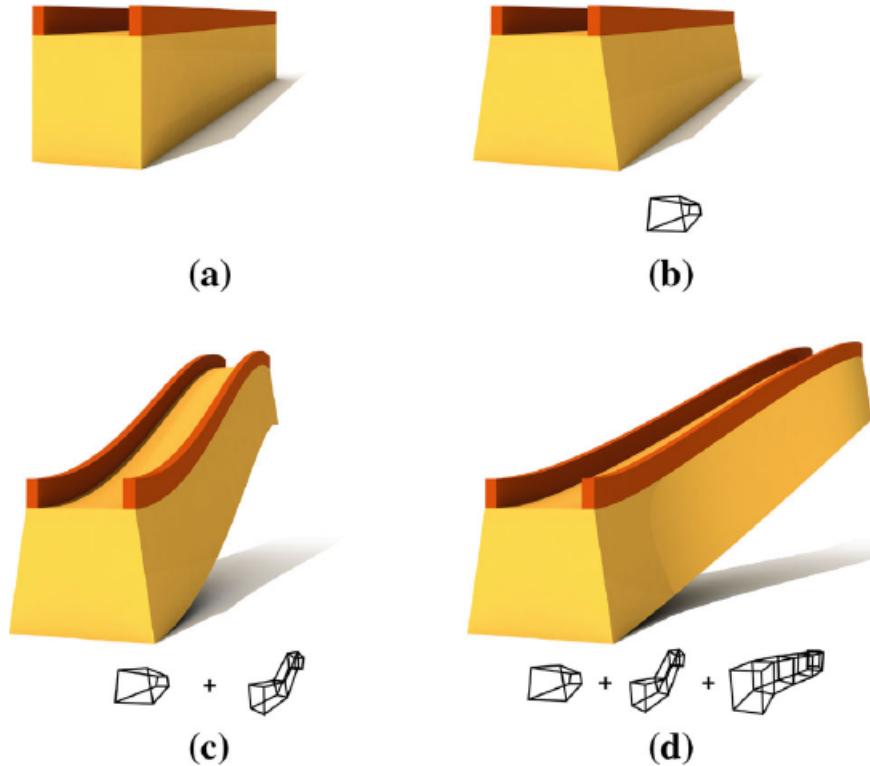


Fonte: (ZMUGG *et al.*, 2014)

3.2.1 Integrando deformações de forma livre

Segundo Zmugg *et al.* (2014), para integrar as deformações de forma livre em sua abordagem, substituiu-se a transformação rígida, utilizada em *split grammars* tradicionais, por uma lista de deformações arbitrárias, o que permite a aplicação aninhada de deformações de forma livre. Este processo é mostrado na Figura 42, onde, a partir de uma forma simples representando uma parede (a), são aplicadas três etapas de deformação: primeiro, apenas a base amarela é afetada pela deformação de alargamento (b), logo após, são aplicadas as deformações verticais (c) e horizontais (d).

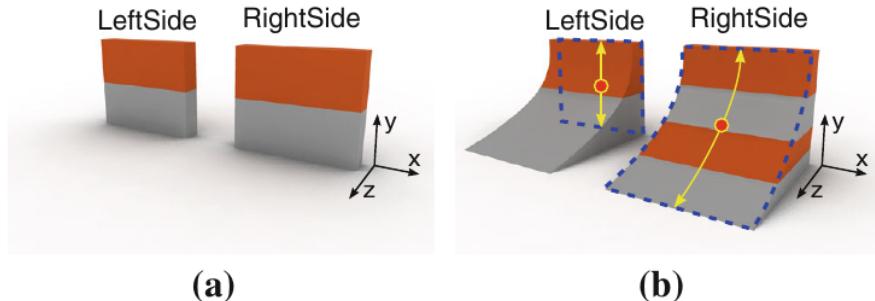
Figura 42 – Aplicação aninhada de deformações de forma livre em uma *split grammar*.



Fonte: (ZMUGG *et al.*, 2014)

Com objetivo de demonstrar a utilização das regras de deformação em um modelo geométrico, Zmugg *et al.* (2014) apresentam o exemplo da Figura 43. Nas regras mostradas na região inferior, o rótulo *Box* refere-se a uma caixa ainda não deformada, representada na Figura 43(a). A operação *deform* recebe como entrada a caixa delimitadora em coordenadas locais, o número de pontos de controle na direção dos eixos *x*, *y* e *z*, bem como uma matriz de deslocamento individual para cada um destes pontos de controle. Algumas funções utilitárias podem ser definidas para permitir uma especificação mais conveniente de deformações comuns. Assim, depois de definir a deformação, pode-se utilizar as operações de divisão padrão, como *divide*, ou novas operações de divisão, que são indicadas pelo sufixo *D*. Entretanto, para utilização de novas operações, é necessário fornecer um ponto adicional como entrada, o qual, juntamente com a direção em que a divisão deve ocorrer, é utilizado para calcular a distância entre os dois extremos no espaço deformado. Por fim, a operação de preenchimento renderiza as formas com o material definido para o atributo *mat*, resultando na Figura 43(b).

Figura 43 – Aplicação de regras de subdivisão em objetos.



Box	$\rightsquigarrow \text{Deform}(\text{localBB}, (2,4,2), [\dots])$ $\{\text{DeformedBox}\}$
DeformedBox	$\rightsquigarrow \text{Subdivide}(X, 1r, 1, 1r)$ $\{\text{LeftSide}, \text{void}, \text{RightSide}\}$
LeftSide	$\rightsquigarrow \text{RepeatD}(Y, 2, \text{rayIntersect}(Z))$ $\{\text{fill}\{\text{mat} = \text{setMaterial}(\text{index \% 2})\}\}$
RightSide	$\rightsquigarrow \text{RepeatD}(Y, 2, \text{rayIntersect}(-Z))$ $\{\text{fill}\{\text{mat} = \text{setMaterial}(\text{index \% 2})\}\}$

Fonte: Adaptado de (ZMUGG *et al.*, 2014)

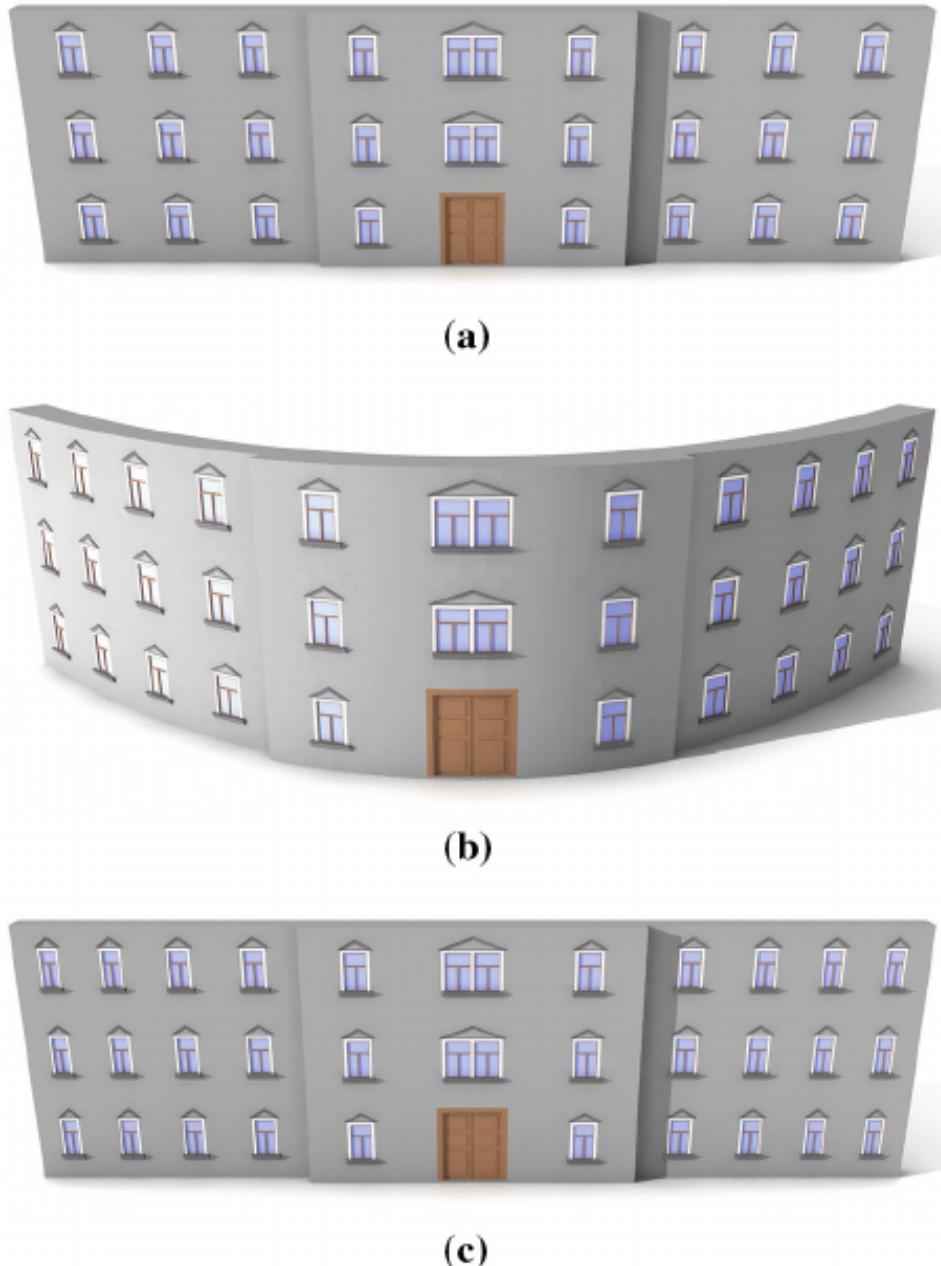
Por meio do exemplo mostrado na Figura 44, Zmugg *et al.* (2014) demonstram o efeito das operações de deformação em um modelo de fachada. As divisões ao longo da largura da fachada são feitas em relação à deformação. Para lidar com o espaço adicional fornecido pela deformação, mais divisões são introduzidas no espaço de coordenadas local, conforme ilustrado na Figura 44(c).

3.2.2 Aplicações

Um dos resultados apresentados por Zmugg *et al.* (2014) é o de um edifício oblongo que é dobrado de diferentes maneiras, através de deformações que se aproximam de formas circulares, conforme ilustrado na Figura 45. Neste exemplo, um edifício com *layout* de sala, definido utilizando uma abordagem de *split grammar* (a), se adapta de acordo com diferentes deformações que se aproximam de segmentos de círculo, ou círculos. A deformação do segmento de círculo (b) leva a uma construção como mostrado em (a). Para mudanças topológicas (c), as regras gramaticais para as paredes de contorno à esquerda e à direita de (a) foram adaptadas, e uma deformação apropriada foi aplicada para alcançar uma transição contínua.

Além da geração de modelos arquiteturais com geometria arredondada, outra vantagem relevante identificada nesta abordagem é o processo de adaptação dos elementos, como janelas e portas, após uma operação de deformação. Por exemplo, na Figura 44, após alteração da

Figura 44 – A aplicação de deformações em uma fachada reta, que é definida utilizando uma *split grammar* (a), produz um número diferente de janelas nas partes laterais (b). A imagem inferior (c) mostra as divisões que são realizadas no espaço de coordenadas locais (não deformadas) para alcançar o resultado mostrado após a deformação (b).

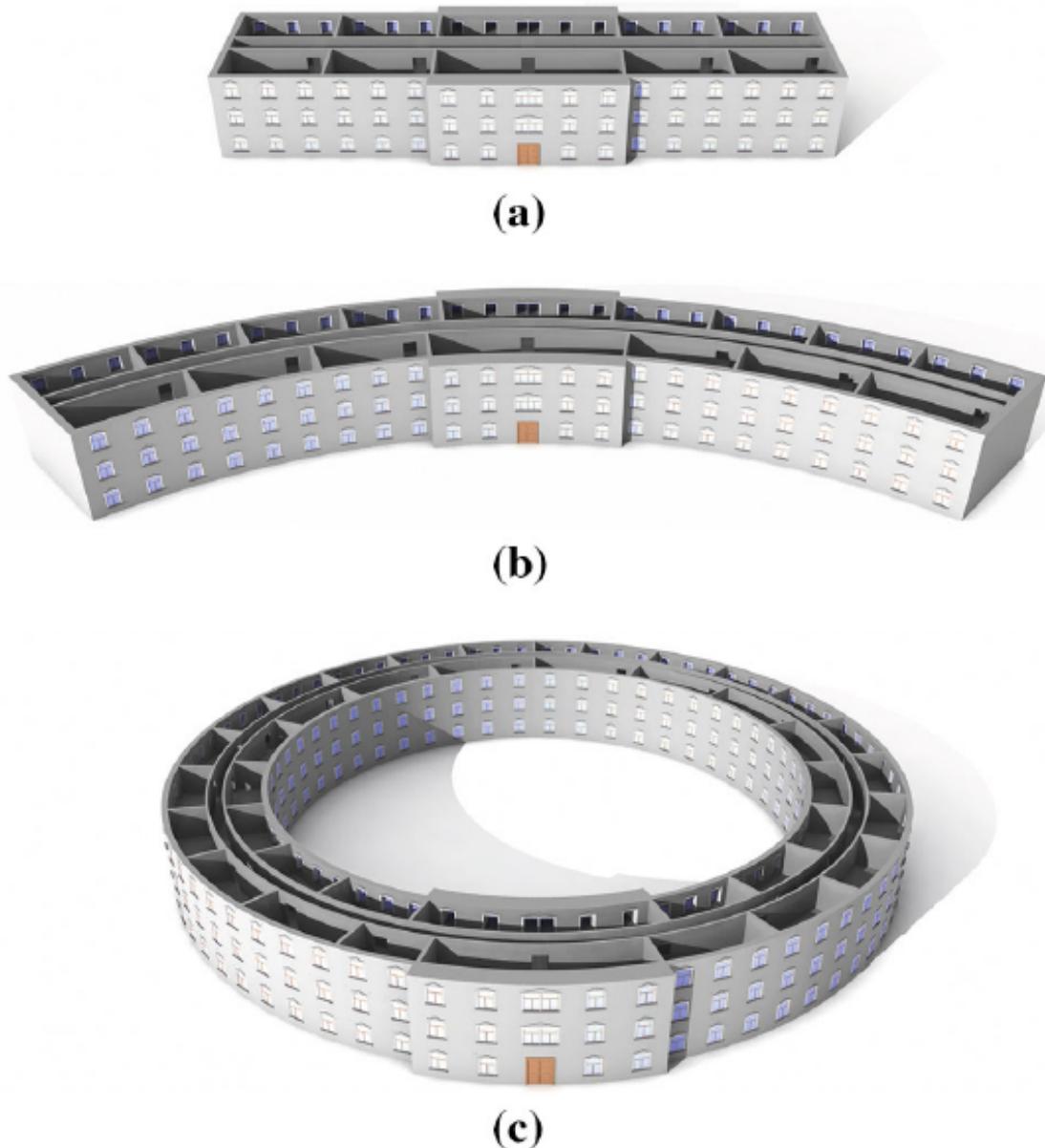


Fonte: (ZMUGG *et al.*, 2014)

curvatura da fachada, é importante notar que as janelas não são alargadas, mas sim adicionadas, a fim de se utilizar o espaço extra gerado pela deformação, todas elas possuindo a mesma largura.

Como limitação, Zmugg *et al.* (2014) mencionam que seu sistema não permite que regras adaptem o resultado de operações *booleanas* de elementos adjacentes.

Figura 45 – Prédio comercial com estrutura arredondada.



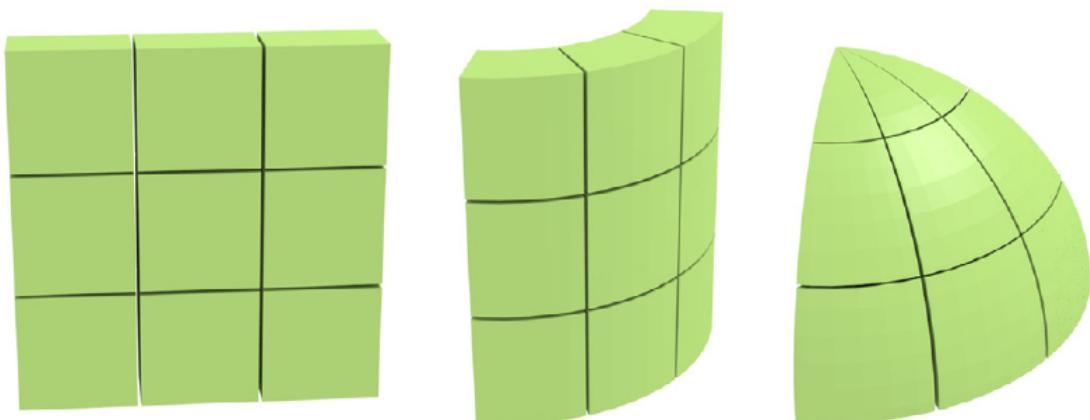
Fonte: (ZMUGG *et al.*, 2014)

3.3 Procedural modeling of architecture with round geometry

Diferentemente das abordagens apresentadas por Thaller *et al.* (2013) e Zmugg *et al.* (2014), no trabalho de Edelsbrunner *et al.* (2017) são especificados sistemas de coordenadas personalizados na *split grammar* definida pelo usuário. Os sistemas de coordenadas cilíndricas produzem geometria adequada para modelar estruturas como torres ou pilares. Sistemas de coordenadas esféricas podem ser utilizados para domos. Além disto, outros sistemas de coordenadas também são convenientes, por exemplo, para geração de estruturas em forma de cone, podendo ser aplicadas na geração de telhados.

Apesar de não apresentarem muitos exemplos práticos da utilização de regras para geração dos modelos apresentados, Edelsbrunner *et al.* (2017) afirmam que a especificação do sistema de coordenadas permite mais possibilidades na divisão de geometria. Uma divisão de parede com uma *split grammar* tradicional produz partes retangulares, assim, a partir de outros sistemas de coordenadas, também é possível dividir paredes cilíndricas ou esféricas em subpartes, conforme ilustrado na Figura 46.

Figura 46 – Uma parede dividida em nove partes, por meio de diferentes sistemas de coordenadas (cartesiana, cilíndrica e esférica).



Fonte: (EDELSBRUNNER *et al.*, 2017)

Além de permitir a utilização de diferentes tipos de sistemas de coordenadas para geração dos modelos, outro recurso interessante identificado nesta abordagem é a possibilidade do usuário especificar entradas em alto nível, a fim de organizar os elementos gerados proceduralmente. Isto permite que até mesmo usuários inexperientes modifiquem o modelo e criem diferentes variações, mas sem se aprofundar em grandes detalhamentos (EDELSBRUNNER *et al.*, 2017), conforme os exemplos da Figura 47.

Figura 47 – Variações de modelos de abóbadas.



Fonte: (EDELSBRUNNER *et al.*, 2017)

Como limitação, Edelsbrunner *et al.* (2017) argumentam que objetos produzidos por meio de deformações de forma livre, que não seguem a geometria das seções cônicas, podem ser difíceis ou impossíveis de reproduzir através da sua abordagem, podendo requerer métodos de aproximação mais complexos.

Neste capítulo, foram apresentadas três técnicas distintas, cada uma voltada para determinada área da geração procedural de modelos com estruturas arredondadas. No próximo capítulo, será discutido o problema abordado pelo presente trabalho, bem como a descrição de uma estratégia para resolvê-lo.

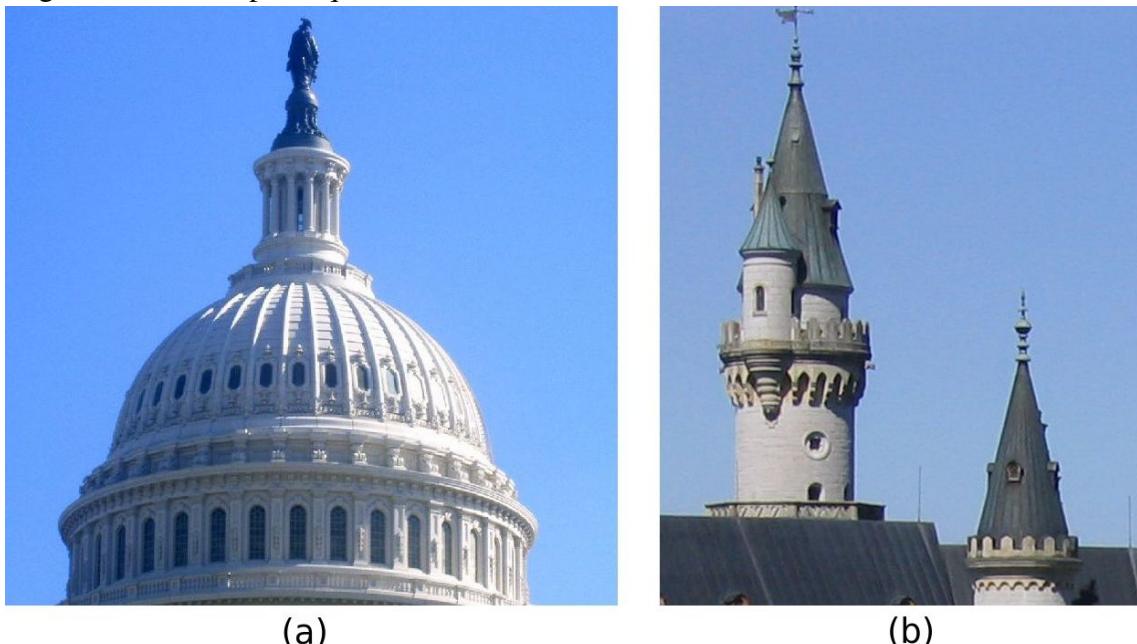
4 PROPOSTA

Neste capítulo, após o embasamento teórico sobre algumas técnicas de modelagem introduzidas no decorrer do Capítulo 2, o problema será levantado na Seção 4.1. Além disto, na Seção 4.2, por meio das ideias apresentadas na Seção 2.5 e no Capítulo 3, também será analisada uma abordagem que é pioneira na resolução do problema em questão.

4.1 Problema

Como ilustrado nas Figuras 48(a) e 48(b), muitas estruturas não retangulares comuns também possuem repetição, neste caso, o arranjo de janelas, blocos e pilares em paredes, torres e domos. Entretanto, uma *shape grammar* baseada em uma caixa padrão não permite a geração de formas nem arranjos arredondados. Apesar de já existirem métodos para modelar ou aproximar formas curvadas ou deformadas, eles ainda apresentam problemas ou falham quando o assunto é a modelagem de formas mais complexas (EDELSBRUNNER *et al.*, 2017). Portanto, se os modelos exigem *designs* curvados, sua criação é trabalhosa, pois as estruturas devem ser aproximadas por geometria plana ou ser colocadas, apropriadamente, com base em objetos pré-modelados (ZMUGG *et al.*, 2014).

Figura 48 – Exemplo arquitetural de um domo e de uma torre de castelo.



Fonte: Adaptado de (a) Veldman-Tentori (2003) e (b) Maxwell (2008)

Conforme mencionado por Jiang *et al.* (2018), uma das limitações de implementa-

ção da *SELEX* é justamente a incapacidade de modelar estruturas arredondadas diretamente, trabalhando apenas por meio de sua importação, como complementos, o que impossibilita a modelagem de fachadas curvadas, como a da Figura 49. Assim, na próxima seção, será discutida uma abordagem para resolução deste problema.

Figura 49 – Exemplo que está além da capacidade de modelagem da *SELEX*.



Fonte: (Jiang *et al.*, 2018)

4.2 Abordagem

Nesta seção, será apresentada a ferramenta de modelagem, a descrição do fluxograma de execução, bem como a estrutura do código para geração dos modelos arquiteturais com geometria arredondada, através da *SELEX*, uma linguagem de modelagem contemporânea que introduziu diversas melhorias em relação às suas precursoras.

4.2.1 Ferramenta de modelagem

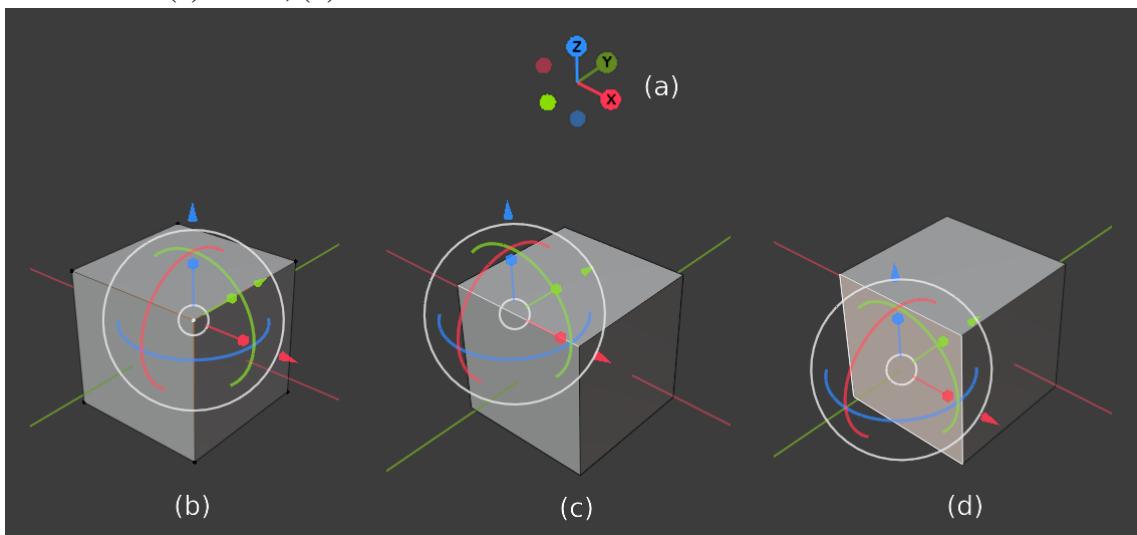
A implementação da solução a ser apresentada foi realizada no *workspace* de programação do Blender (versão 2.83.8), um *software* de modelagem *open source* mantido pela Blender Foundation (2020), utilizando *scripts* escritos em Python, linguagem de programação mantida pela Python Software Foundation (2021).

Os objetos padrões presentes na ferramenta, como plano e cubo, são utilizados

para representar, respectivamente, as formas virtuais e de construção, descritas na Seção 2.4.1. Cada um destes objetos, por sua vez, possui um conjunto de estruturas de dados que guardam informações sobre a sua representação, tais como quantidade de faces, vértices e arestas, bem como seus identificadores, posições no espaço, dentre outras. Assim, por meio destes atributos, é possível realizar a seleção de determinadas células (faces) ou grupo de células, para que operações de agrupamento, como `groupRegions`, possam ser aplicadas sobre elas.

O Blender também dispõe de múltiplos artifícios para manipulação de vértices, faces e arestas, em relação a diferentes eixos, conforme mostrado na Figura 50. Tais recursos são amplamente utilizados em operações como `addVolume` e `roundShape`, visando aplicar deformações nos modelos, a fim de gerar arquiteturas arredondadas.

Figura 50 – Representação do sistema de coordenadas do Blender: (a) eixos, (b) vértice, (c) aresta, (d) face.

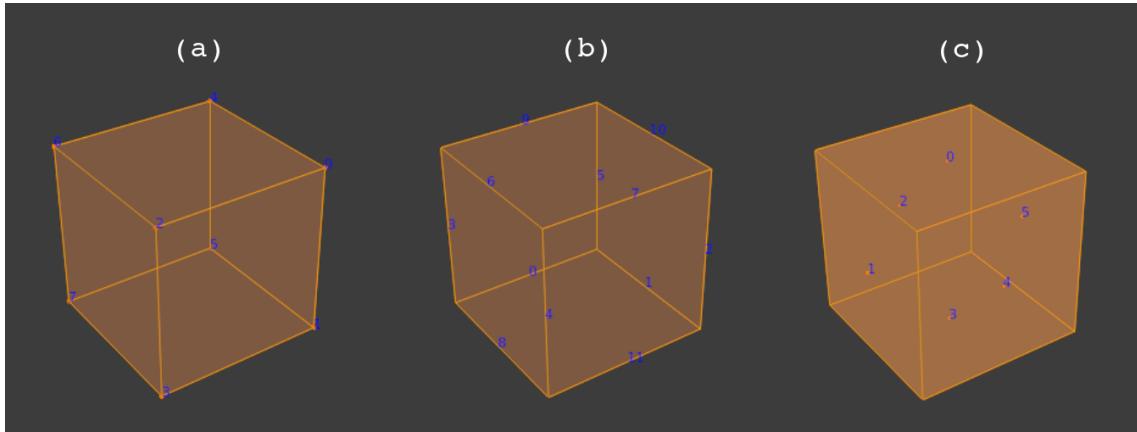


Fonte: Próprio autor

Para fins de depuração, o *software* foi configurado de maneira a prover utilidades extras para desenvolvedores, como a exibição dos índices de faces, arestas e vértices, conforme mostrado na Figura 51.

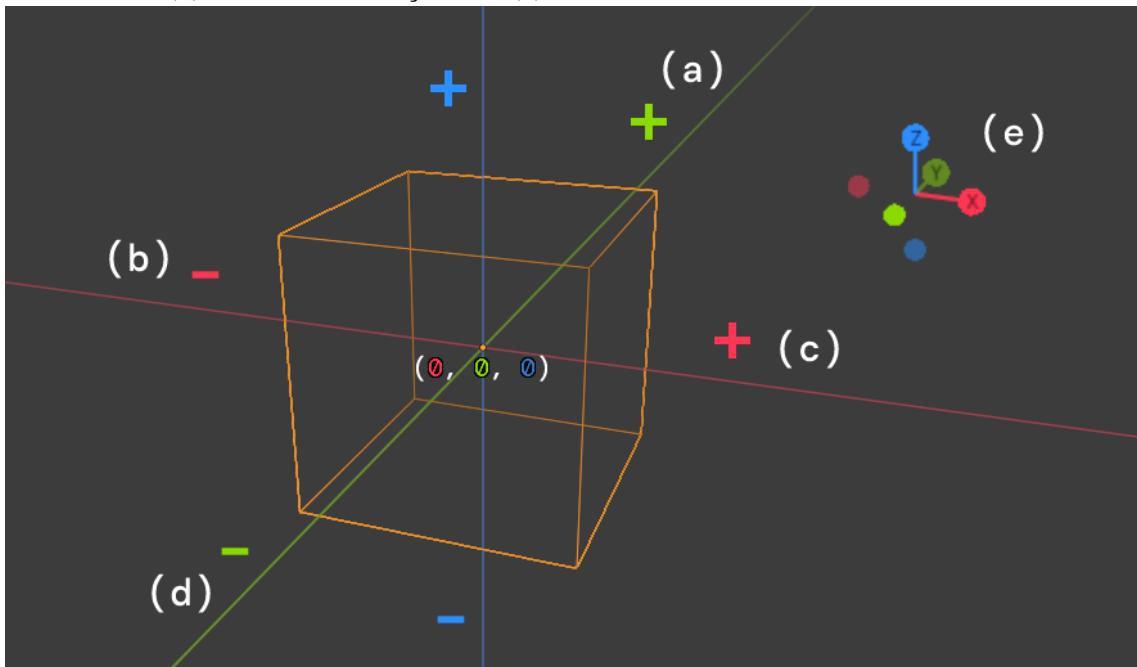
No processo de geração dos modelos, é utilizada a unidade de medida de comprimento do metro, a qual vem pré-configurada em cada projeto por padrão. Além disto, o panorama adotado para a visualização de cada modelo é mostrado na Figura 52, onde (a) representa o lado posterior, (b) representa o lado esquerdo, (c) representa o lado direito, e (d) representa a parte frontal, em relação ao eixos representados em (e).

Figura 51 – Representação dos índices de um objeto: (a) vértices, (b) arestas e (c) faces.



Fonte: Próprio autor

Figura 52 – Wireframe da representação do lado (a) posterior, (b) esquerdo, (c) direito e (d) frontal, em relação aos (e) eixos cartesianos.



Fonte: Próprio autor

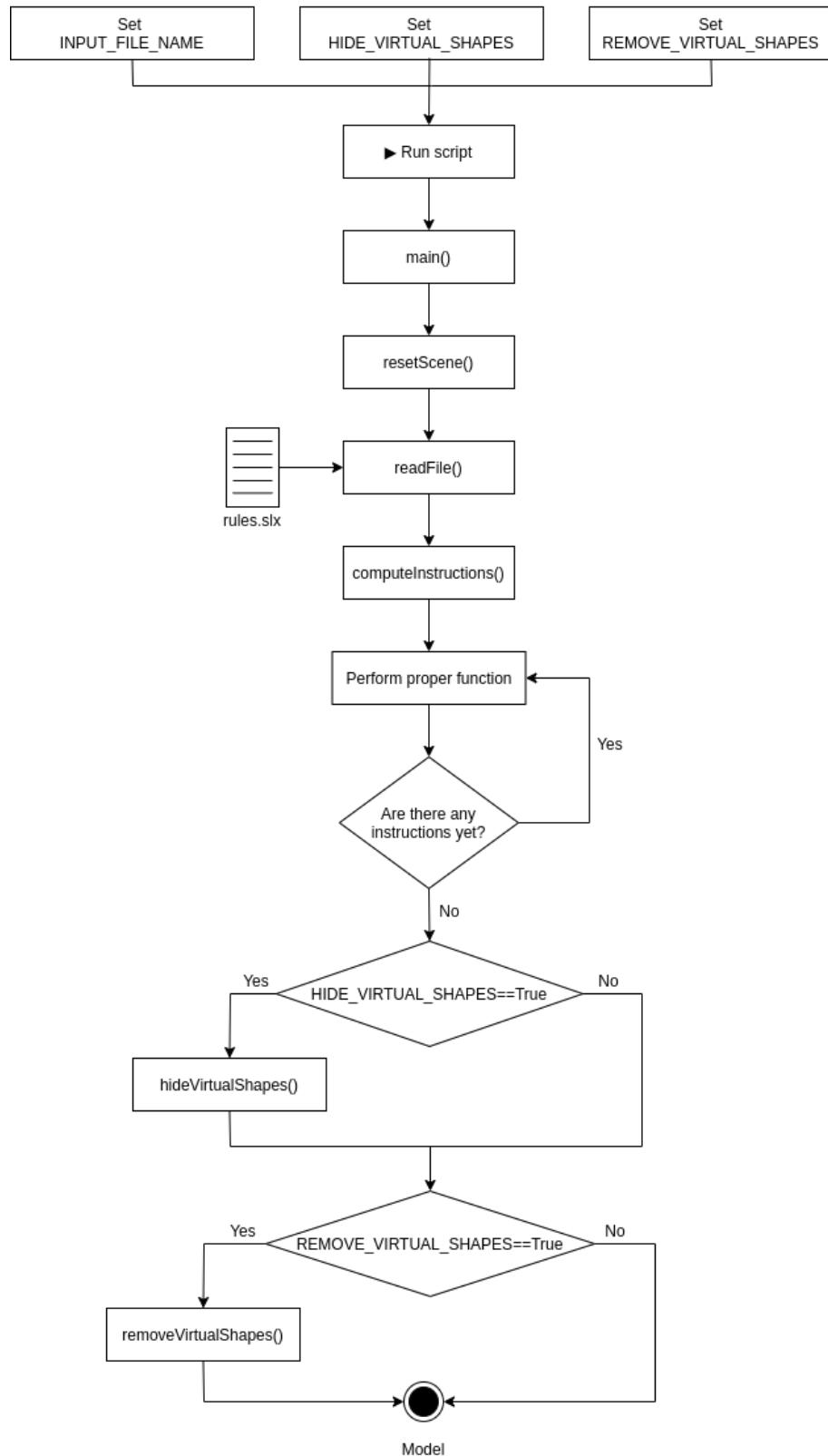
4.2.2 Fluxo de modelagem

O fluxograma da Figura 53, apresenta um *overview* das etapas que são executadas durante o processo de geração dos modelos, com base no código-fonte disponível no GitHub¹.

Em linhas gerais, inicialmente, deve-se adicionar o arquivo `main.py` como *script* no *workspace* de programação do Blender. Logo após, deve-se fornecer o nome do arquivo que contém as regras para geração do modelo, podendo ainda serem definidas algumas *flags* para tratamento das formas virtuais ao final do processo, permitindo sua ocultação ou exclusão.

¹ <https://github.com/DanielBrito/monografia/tree/main/Codigo-Fonte>

Figura 53 – Fluxograma de execução.



Fonte: Próprio autor

Uma vez que o arquivo com as regras é definido, ao executar o *script*, a função `main` é acionada, removendo os objetos presentes na cena, e invocando a função de carregamento

do arquivo. Durante a leitura das linhas, cada uma das regras é computada e a devida *action* é executada.

Por fim, após a realização de todas as instruções, o modelo final é obtido, podendo ainda ser modificado por meio das ferramentas disponíveis no próprio *software*. Contudo, tais alterações não refletem na árvore de formas, devido ao fato dela ser construída, exclusivamente, através das regras.

Entre as principais operações suportadas pelo presente trabalho estão `createShape`, `createGrid`, `groupRegions`, `addVolume` e `roundShape`, as quais serão mais bem detalhadas na próxima seção.

4.2.3 Módulos

Nesta seção, além da descrição de alguns detalhes de implementação dos módulos presentes no código-fonte, serão apresentados exemplos básicos do resultado de cada operação de modelagem.

4.2.3.1 Imported libraries

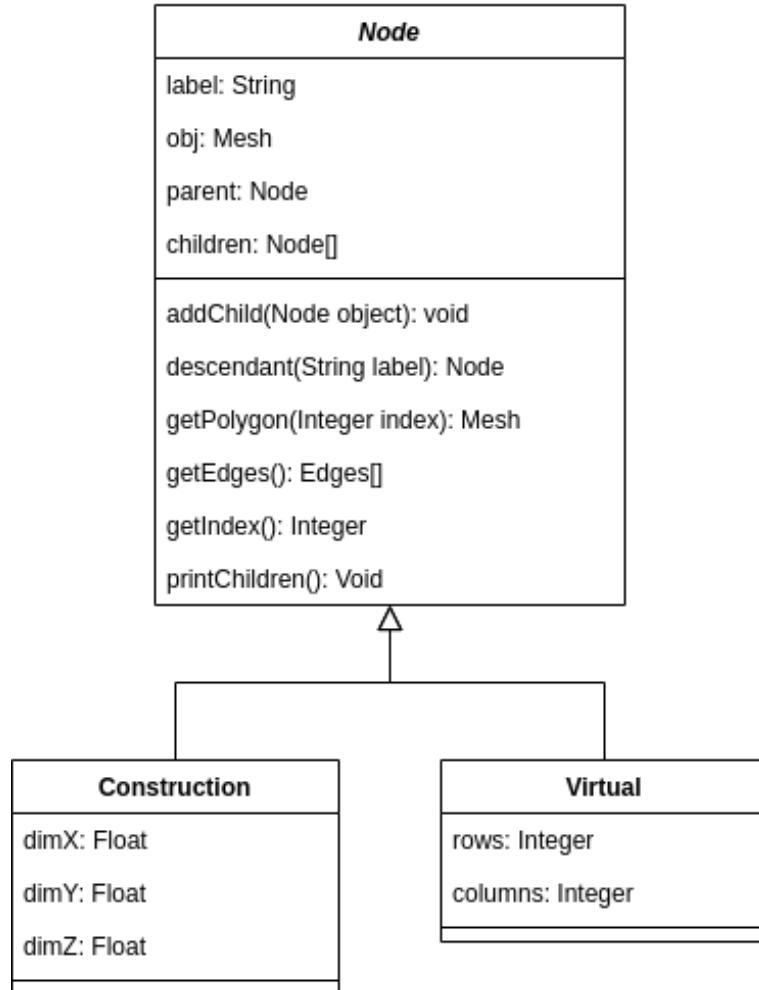
Informações disponibilizadas pela Python Software Foundation (2021) e Blender Foundation (2020) acerca das bibliotecas utilizadas no presente trabalho:

- **bpy**: Fornece acesso aos dados, classes e funções do Blender, amplamente utilizadas no processo de criação e modificação dos objetos;
- **math**: Fornece acesso às funções matemáticas, utilizadas para manipulação de valores numéricos;
- **os**: Fornece métodos para utilização de funcionalidades do sistema operacional, como o acesso ao diretório onde se encontra o arquivo com as regras que definem os modelos;
- **re**: Fornece operações para correspondência de expressões regulares, utilizadas para extração das informações contidas nas regras, obtidas a partir do arquivo de entrada.

4.2.3.2 Classes Module

Com base no paradigma de Programação Orientada a Objetos, a estrutura é organizada por meio classes, conforme o diagrama da Figura 54, as quais são utilizadas no gerenciamento da árvore de formas.

Figura 54 – Diagrama de classes referente à estrutura da árvore de formas.



Fonte: Próprio autor

Neste caso, **Node** é a representação genérica de um nó da árvore, já as subclasses **Construction** e **Virtual**, são entidades mais específicas, que representam, respectivamente, as formas de construção e as formas virtuais.

4.2.3.3 Utility Functions Module

Definição das principais funções utilitárias que auxiliam no processo de geração dos modelos, por meio da integração com as *actions*:

- **create3DMass**: Invocada durante o processo de execução da *action* `createShape`, adicionando o modelo de massa padrão, e redimensionando-o de acordo com as configurações definidas nas regras iniciais, por meio dos valores de `width`, `depth` e `height`;
- **vert** e **face**: Utilizadas em conjunto durante o processo de execução da *action* `createGrid`,

atuando no posicionamento dos vértices e na criação das faces;

- **selectNode**: Invocada para recuperar um nó da árvore de formas, com base nos *labels* presentes nas regras;
- **placeVirtualShape**: Trabalha em conjunto com a *action* `createGrid`, por meio do posicionamento da forma virtual sobre uma determinada face do modelo de massa;
- **selectToBeVolume** e **duplicateShape**: Utilizadas em conjunto para selecionar uma sub-região de uma face do modelo, através de uma forma virtual, a fim de realizar uma extrusão por meio da *action* `addVolume`;
- **indexRange**: Utilizada para selecionar um grande intervalo de células, tornando a regra mais sucinta. Conforme especificado na *SELEX*, recebe um índice inicial, um índice final, e retorna como resultado uma lista com os valores no intervalo. Por exemplo, `indexRange(1, 5)`, retorna a lista `[1, 2, 3, 4, 5]`.

Algumas outras funções também presentes neste módulo são relacionadas ao gerenciamento da cena, como `resetScene`, que remove os objetos antes da execução das regras; `hideVirtualShapes`, que oculta as formas virtuais ao final da geração; e `removeVirtualShapes`, que remove as formas virtuais, com o propósito de gerar um objeto final com menos polígonos, reduzindo o tamanho do arquivo do modelo.

4.2.3.4 Actions Module

Definição das principais funções que executam as *actions* responsáveis pela geração e manipulação das formas virtuais e de construção, com base nas especificações da *SELEX*. Para exemplificar cada um dos casos, serão utilizadas algumas regras que produzem como resultado final um modelo de massa simples.

Primeiramente, define-se as dimensões do modelo:

#C1: Initial settings

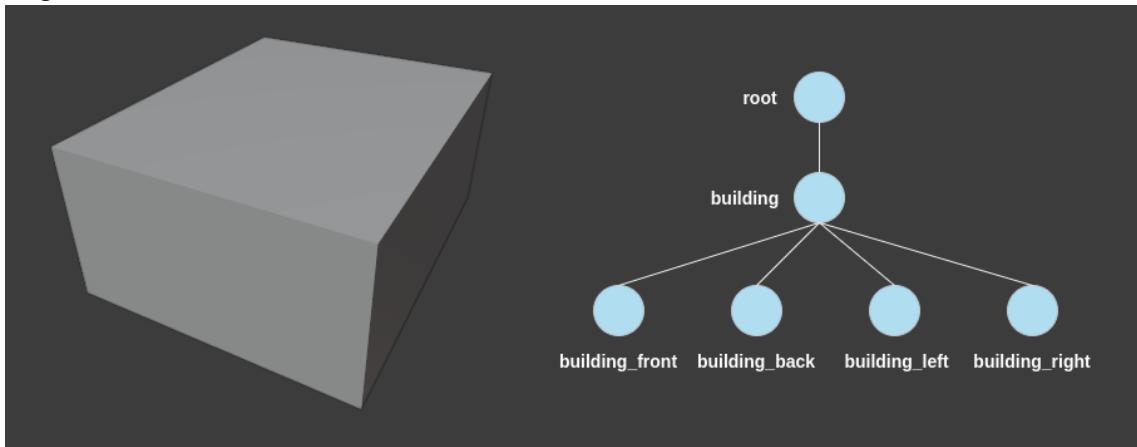
```
label = "building"; width = 9; depth = 11; height = 5;
```

Então, por meio da função `createShape`, que recebe os valores anteriores, o modelo de massa é gerado e a árvore de formas é atualizada, conforme mostrado na Figura 55.

#C2: Generating mass model

```
{<> -> createShape(label, width, depth, height)};
```

Figura 55 – Modelo de massa inicial.



Fonte: Próprio autor

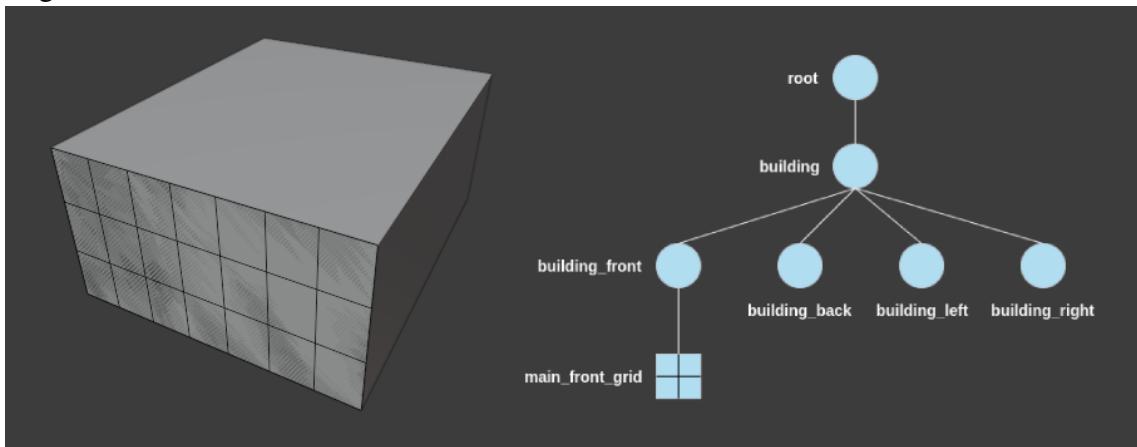
A fim de selecionar uma sub-região da parte frontal do modelo, é necessário adicionar uma forma virtual, o que pode ser feito a partir da *action* `createGrid`, que recebe como parâmetros o *label*, o número de linhas e o número de colunas, produzindo o resultado mostrado na Figura 56.

#C3: Adding virtual shape

```

{<descendant()[label=="building"]/[label=="building_front"]>
-> createGrid("main_front_grid", 3, 7);}
  
```

Figura 56 – Inclusão de forma virtual.



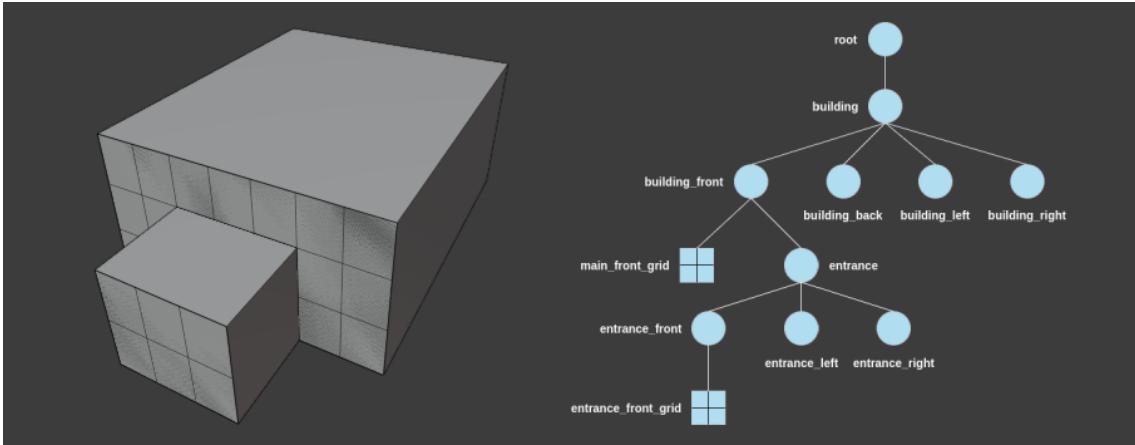
Fonte: Próprio autor

Uma vez que a grade foi criada e devidamente posicionada, torna-se possível selecionar as células desejadas, através dos índices de linha e coluna, com objetivo de realizar operações de agrupamento e extrusão, por meio do `groupRegions` e da *action* `addVolume`, respectivamente. Conforme as especificações da *SELEX*, `addVolume` recebe como parâmetros o *label* do volume a

ser produzido, o polígono que sofrerá a transformação, o tamanho da extrusão, e os *labels* das faces laterais do volume gerado. Tal resultado é mostrado na Figura 57.

```
#C4: Selecting regions and performing extrusion
{<descendant()>[label=="building"]/[label=="building_front"]
 [label=="main_front_grid"]/[type=="cell"]
 [rowIdx in (2, 3)] [colIdx in (3, 4, 5)][::groupRegions()]
 -> addVolume("entrance", "building_front", 3,
 ["entrance_front", "entrance_left", "entrance_right"]);}
```

Figura 57 – Resultado obtido após a operação de extrusão.



Fonte: Próprio autor

Como proposta, o presente trabalho introduz uma nova *action*, com objetivo de gerar estruturas arredondadas nos modelos. Assim, a operação roundShape pode ser definida da seguinte maneira:

roundShape(type, direction, roundingDegree, segments, sideReference, axis, insideDegree),

onde cada um dos parâmetros recebidos desempenha um papel específico:

1. ***type***: Define o tipo de arredondamento, podendo ser classificado como *front*, *left*, *right*, *top* ou *bottom* (Figura 59);
2. ***direction***: Define a direção do arredondamento, podendo ser classificada como *outside* ou *inside* (Figura 60);
3. ***roundingDegree***: Representa o grau de arredondamento, podendo ser obtido através da divisão do número de colunas selecionadas pela quantidade total de colunas da grade. O mesmo cálculo vale para operações em termos de linhas, visando um arredondamento na

horizontal. Por exemplo, na Figura 57, o modelo possui uma forma virtual com 3 linhas e 7 colunas na região frontal. Portanto, para realizar uma operação de arredondamento vertical na região das colunas 3, 4 e 5, o valor ideal do parâmetro *roundingDegree* é dado por $3/7 = 0.42$, podendo ainda ser reduzido, a fim de produzir um arredondamento menos acentuado (Figura 61);

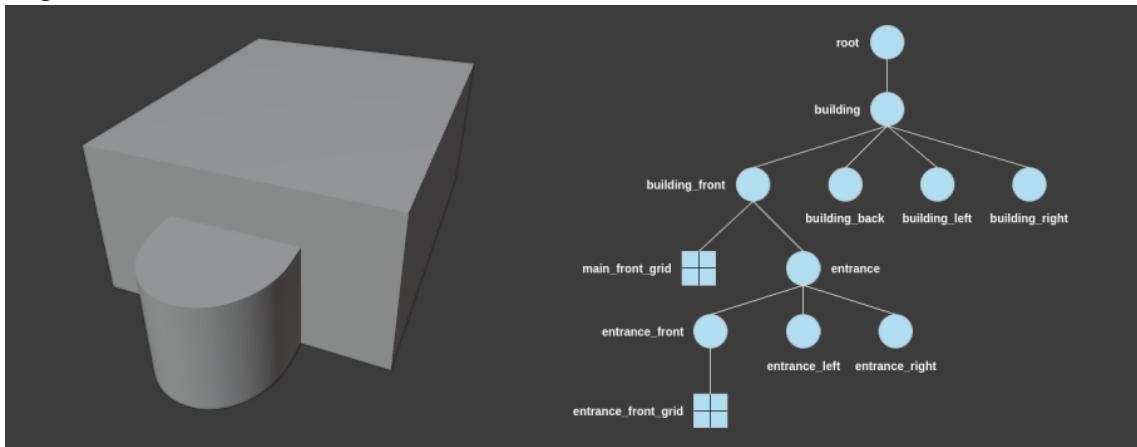
4. ***segments***: Representa a quantidade de novos segmentos (faces) a serem criados no processo de deformação da região. Portanto, quanto mais segmentos, maior o grau de realismo. Contudo, também aumenta-se o custo computacional para geração do modelo. É importante ressaltar que deformações do tipo *front* geram o dobro de segmentos, uma vez que atuam nas regiões esquerda e direita, ou inferior e superior, simultaneamente (Figura 62);
5. ***sideReference***: Referência da direção para qual o vetor normal da região está voltado, podendo assumir valores *main_front*, *main_back*, *main_left* ou *main_right*, uma vez que a consulta de vértices para operações na região frontal e posterior são efetuadas em relação ao eixo x, mas em operações laterais, utiliza-se como referência o eixo y (Figura 63);
6. ***axis***: Definido para operações de arredondamento cujo *type* possui valor *front*, podendo ser classificado como *vertical* ou *horizontal* (Figura 64);
7. ***insideRounding***: Define o grau de arredondamento interno, sendo 0.05 por padrão. A diminuição gradativa deste valor, faz a região interna se aproximar cada vez mais de um ângulo reto (Figura 65).

Então, por meio da introdução deste novo recurso, é possível aplicar uma deformação na sub-região frontal do modelo (Figura 57), a fim de se obter o resultado da Figura 58, através da seguinte regra:

```
#C5: Applying roundShape deformation
{<descendant(){[label=="building"]/[label=="building_front"]/
[label=="entrance"]/[label=="entrance_front"]>
-> roundShape("front", "outside", 0.42, 30, "main_front", "vertical");}
```

Vale ressaltar que a *action* *roundShape* atua como uma interface para a operação *bevel*, um artifício interno do Blender que suaviza bordas e cantos dos objetos.

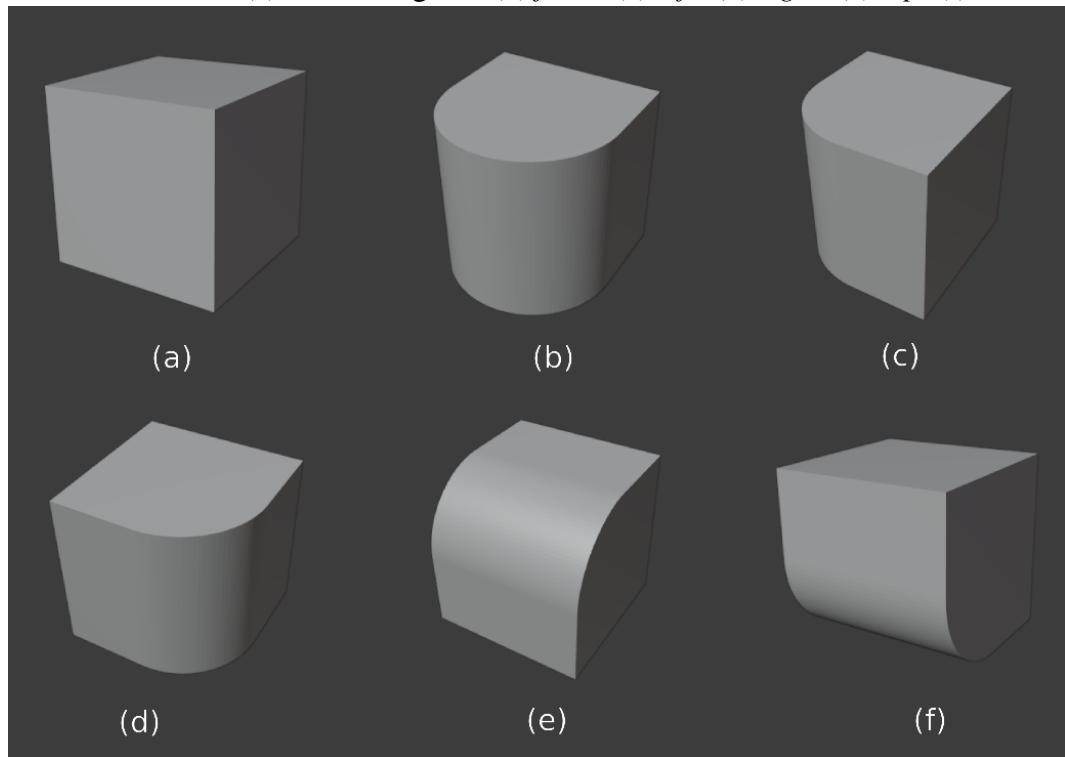
Figura 58 – Modelo final com estrutura frontal arredondada.



Fonte: Próprio autor

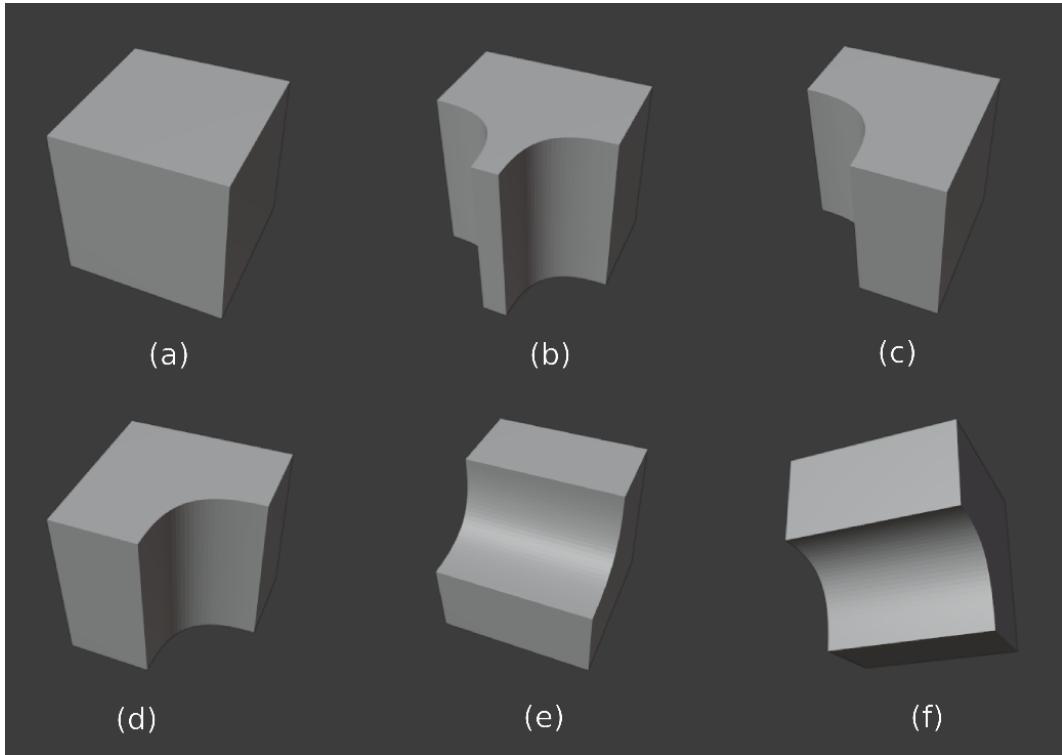
A seguir, serão apresentados outros exemplos ilustrativos com variações de valores para cada um dos parâmetros da *action roundShape*, os quais, em conjunto, são capazes de gerar modelos de massa mais complexos:

Figura 59 – Exemplos de variação do parâmetro (1) *type*, com (2) *direction* valorado como *outside*: (a) Forma original, (b) *front*, (c) *left*, (d) *right*, (e) *top*, (f) *bottom*.



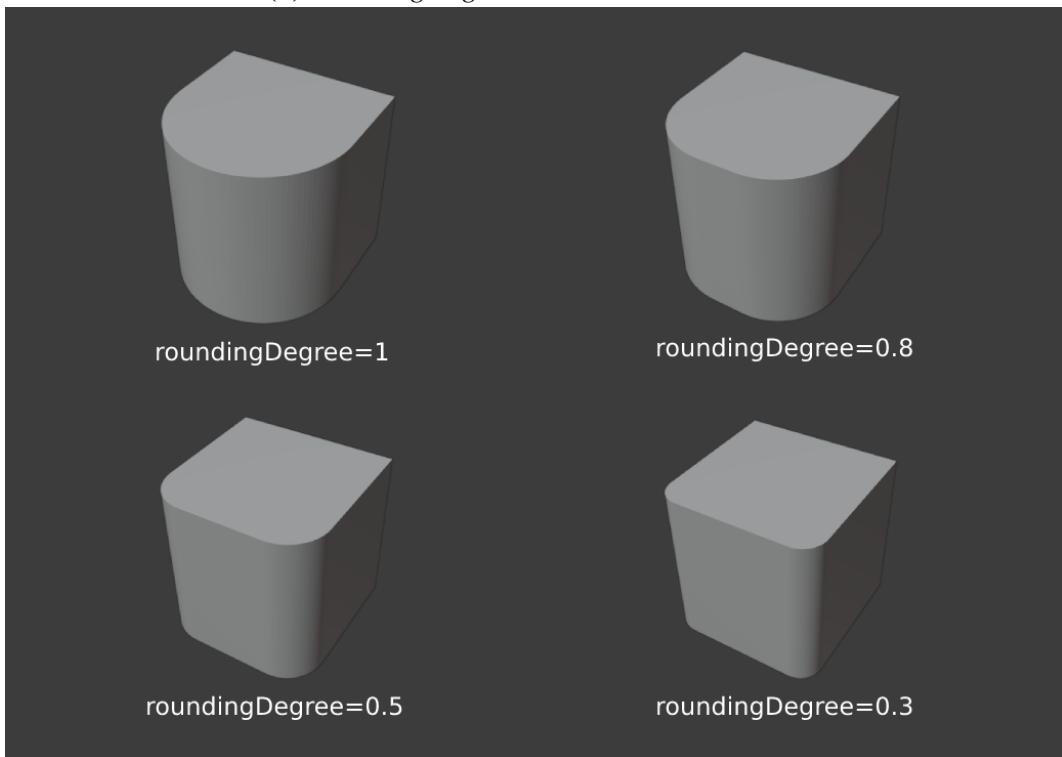
Fonte: Próprio autor

Figura 60 – Exemplos de variação do parâmetro (1) *type*, com (2) *direction* valorado como *inside*: (a) Forma original, (b) *front*, (c) *left*, (d) *right*, (e) *top*, (f) *bottom*.



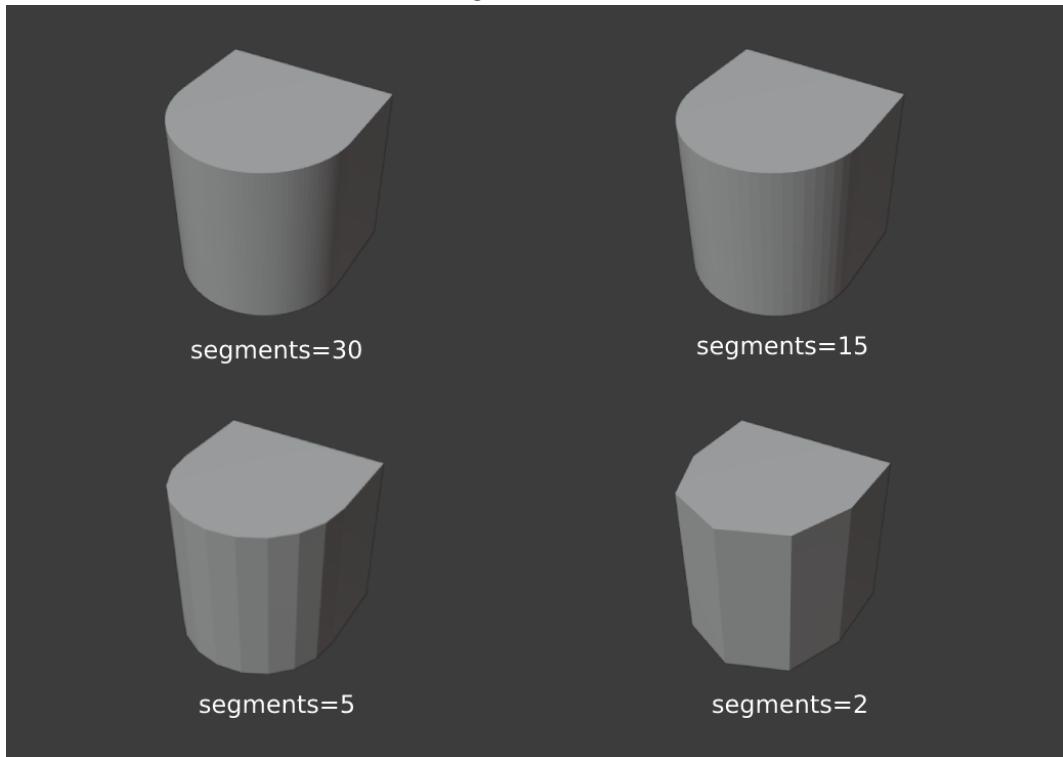
Fonte: Próprio autor

Figura 61 – Deformação frontal de um modelo de massa em forma de cubo, possuindo uma grade virtual com apenas uma linha e uma coluna, a partir de diferentes valores de (3) *roundingDegree*.



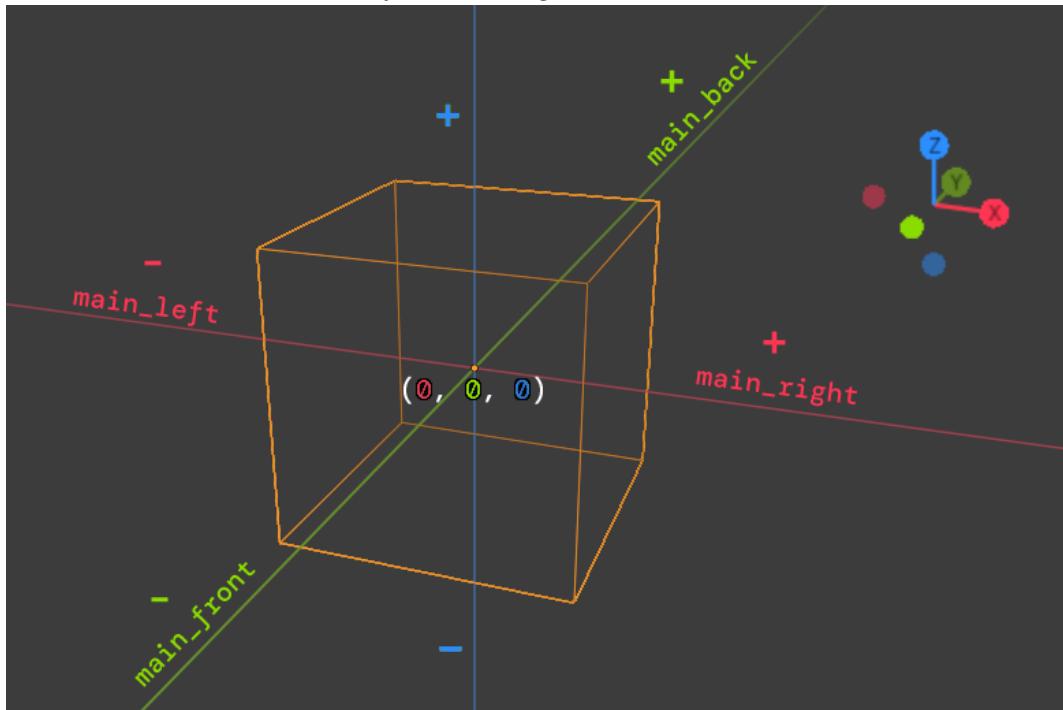
Fonte: Próprio autor

Figura 62 – Deformação frontal de um modelo de massa em forma de cubo, a partir de diferentes valores de (4) segments.



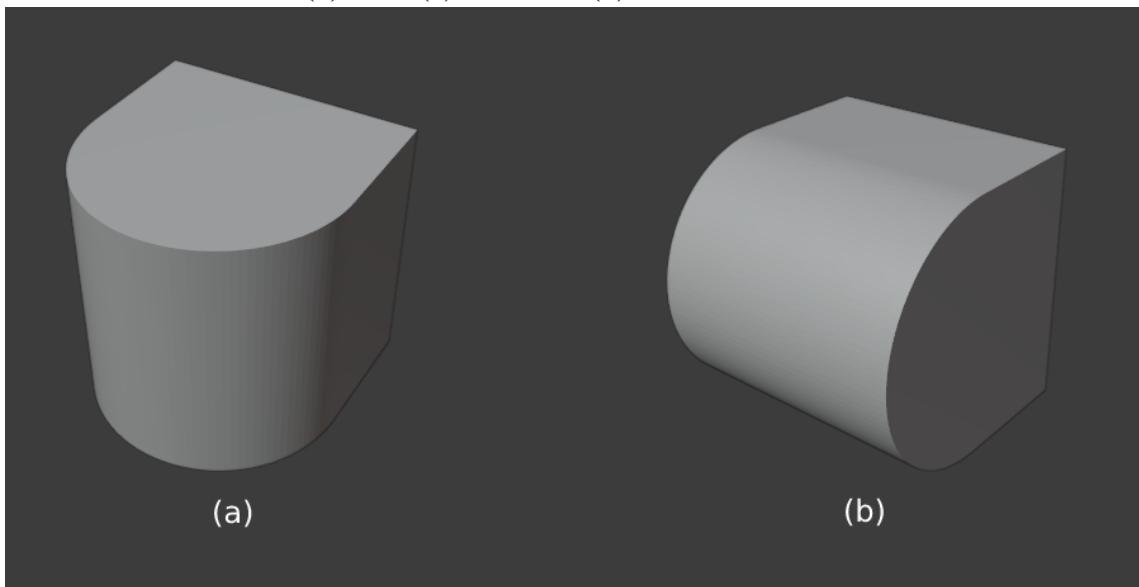
Fonte: Próprio autor

Figura 63 – Representação gráfica dos diferentes valores de (5) sideReference: main_front, main_back, main_left e main_right.



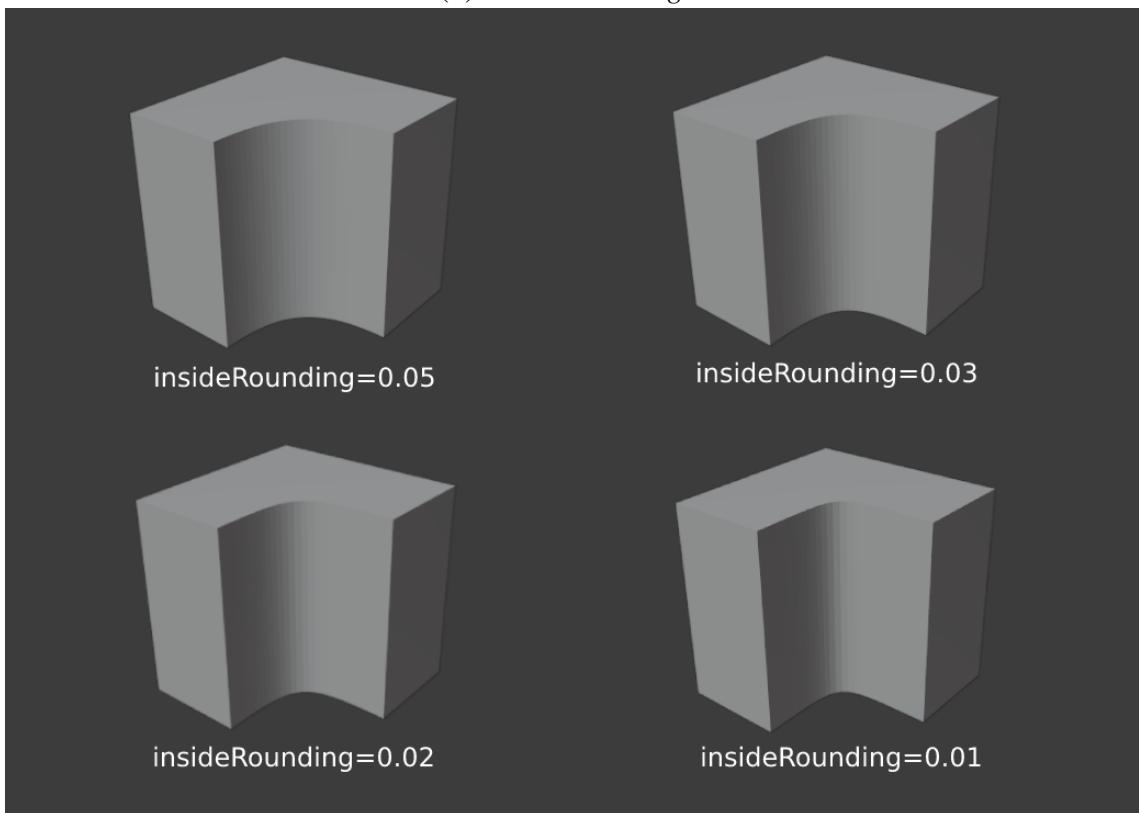
Fonte: Próprio autor

Figura 64 – Deformação frontal de um modelo de massa em forma de cubo, para diferentes valores de (6) *axis*: (a) *vertical* e (b) *horizontal*.



Fonte: Próprio autor

Figura 65 – Deformação lateral (direita) de um modelo de massa em forma de cubo, possuindo uma grade virtual com apenas uma linha e uma coluna, a partir de diferentes valores de (7) *insideRounding*.



Fonte: Próprio autor

4.2.3.5 Outros módulos

No *User Configuration Module*, o usuário informa a *string* referente ao nome do arquivo que contém as regras, sendo atribuída à variável INPUT_FILE_NAME, e também define o estado das *flags booleanas* para tratamento das formas virtuais, neste caso, HIDE_VIRTUAL_SHAPES e REMOVE_VIRTUAL_SHAPES.

No *Input Stream Module*, são definidas as funções responsáveis pela divisão dos *tokens*, com o propósito de extrair os parâmetros a serem passados para as respectivas *actions*:

- **loadSettings**: Utilizada para o carregamento das configurações do modelo de massa, as quais são enviadas para *createShape*;
- **loadCreateGrid**: Utilizada para o carregamento das informações referentes às grades virtuais, as quais são enviadas para *createGrid*;
- **loadAddVolume**: Utilizada para o carregamento das informações referentes à extrusão de uma determinada região do modelo, as quais são enviadas para *addVolume*;
- **loadRoundShape**: Utilizada para o carregamento das informações referentes ao arredondamento de uma determinada região, as quais são enviadas para *roundShape*.

No *File Module*, são especificadas as instruções para leitura do arquivo com base no valor da variável INPUT_FILE_NAME. Todas as linhas iniciadas pelo símbolo # são tratadas como comentários, e o restante são armazenadas em uma lista, a fim de serem enviadas para a função *computeInstructions*.

No *Execution Module*, é definida a função de inicialização main, utilizada para invocar algumas funções auxiliares, como *resetScene* e *readFile*, ou ainda *hideVirtualShapes* e *removeVirtualShapes*, se as variáveis HIDE_VIRTUAL_SHAPES e REMOVE_VIRTUAL_SHAPES possuírem valor True, respectivamente.

Este capítulo descreveu a problemática e algumas das principais características de implementação do presente trabalho. No próximo capítulo, serão apresentados modelos mais elaborados e uma breve análise estatística da solução proposta.

5 RESULTADOS

Neste capítulo, com base na solução proposta, serão apresentados alguns resultados obtidos a partir de diferentes conjuntos de regras. Logo após, serão avaliados dados estatísticos de cada um dos modelos gerados. Tais resultados foram obtidos por meio da utilização de um *notebook* com processador Intel Core i5, 2.20GHz, 8GB de memória RAM, sistema operacional Linux Mint 20 Cinnamon, arquitetura de 64 bits e placa de vídeo Intel Corporation HD Graphics 5500. Por fim, também serão apresentadas algumas restrições encontradas.

5.1 Modelos gerados

O primeiro resultado a ser apresentado baseia-se no edifício da Figura 49, o qual, segundo Jiang *et al.* (2018), ainda representa um desafio de modelagem para a *SELEX*. As regras a seguir representam cada uma das etapas de geração. Posteriormente, na Figura 66, são mostrados os modelos resultantes de cada operação, bem como a árvore de formas final.

Inicialmente, são definidas as características do modelo por meio da regra:

#C1: Initial settings

```
label = "building"; width = 9; depth = 8; height = 5;
```

Logo após, o processo continua a partir das seguintes regras:

(a) *#C2: Generating mass model*

```
{<> -> createShape(label, width, depth, height)};
```

(b) *#C3: Adding virtual shape*

```
{<descendant()>[label=="building"]/[label=="building_front"]>
-> createGrid("main_front_grid", 3, 6)};
```

(c) *#C4: Selecting left region and performing extrusion*

```
{<descendant()>[label=="building"]/[label=="building_front"]/
[label=="main_front_grid"]/[type=="cell"]
[rowIdx in (1, 2, 3)] [colIdx in (1, 2)] [:groupRegions()]>
-> addVolume("fac_1", "building_front", 2.5,
["fac_1_front", "fac_1_left", "fac_1_right"]);}
```

(d) *#C5: Applying roundShape deformation*

```
{<descendant()>[label=="building"]/[label=="building_front"]/
```

```

[label=="fac_1"]/[label=="fac_1_front"]>
-> roundShape("front", "outside", 0.33, 30, "main_front", "vertical"));

(e) #C6: Selecting middle region and performing extrusion
{<descendant() [label=="building"]/[label=="building_front"]/
[label=="main_front_grid"]/[type=="cell"]
[rowIdx in (1, 2, 3)] [colIdx in (3, 4)] [:groupRegions()]>
-> addVolume("fac_2", "building_front", 3,
["fac_2_front", "fac_2_left", "fac_2_right"]);}

(f) #C7: Applying roundShape deformation
{<descendant() [label=="building"]/[label=="building_front"]/
[label=="fac_2"]/[label=="fac_2_front"]>
-> roundShape("front", "outside", 0.33, 30, "main_front", "vertical"));

(g) #C8: Selecting bottom-right region and performing extrusion
{<descendant() [label=="building"]/[label=="building_front"]/
[label=="main_front_grid"]/[type=="cell"]
[rowIdx in (2, 3)] [colIdx in (5, 6)] [:groupRegions()]>
-> addVolume("fac_3", "building_front", 3.5,
["fac_3_front", "fac_3_left", "fac_3_right"]);}

(h) #C9: Applying roundShape deformation
{<descendant() [label=="building"]/[label=="building_front"]/
[label=="fac_3"]/[label=="fac_3_front"]>
-> roundShape("front", "outside", 0.33, 30, "main_front", "vertical"));

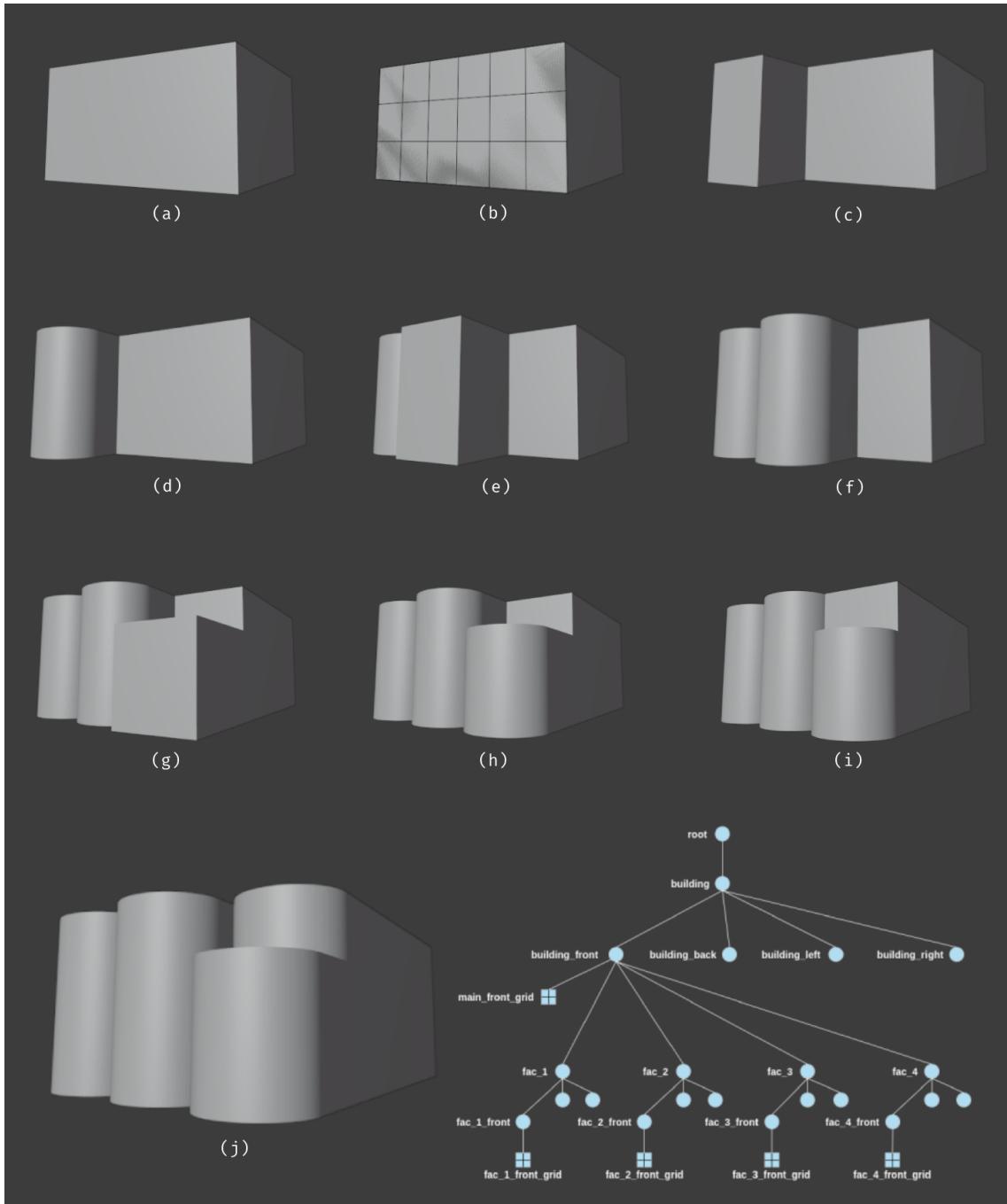
(i) #C10: Selecting top-right region and performing extrusion
{<descendant() [label=="building"]/[label=="building_front"]/
[label=="main_front_grid"]/[type=="cell"]
[rowIdx in (1)] [colIdx in (5, 6)] [:groupRegions()]>
-> addVolume("fac_4", "building_front", 2,
["fac_4_front", "fac_4_left", "fac_4_right"]);}

(j) #C11: Applying roundShape deformation
{<descendant() [label=="building"]/[label=="building_front"]/
{<descendant() [label=="building"]/[label=="building_front"]/
[label=="fac_4"]/[label=="fac_4_front"]>

```

```
-> roundShape("front", "outside", 0.33, 30, "main_front", "vertical");
```

Figura 66 – Modelo de massa do edifício da Figura 49.



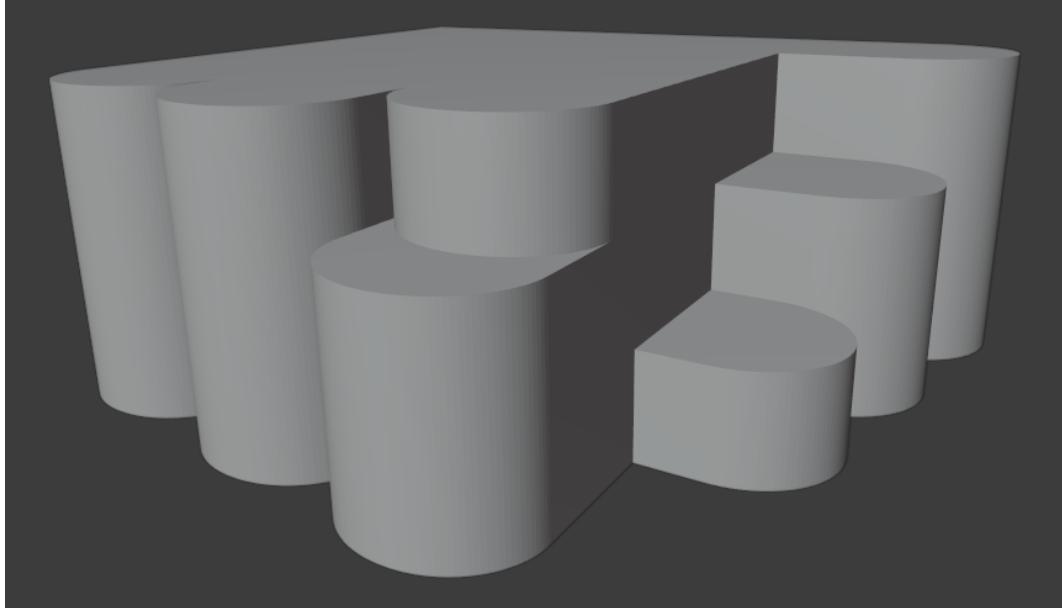
Fonte: Próprio autor

Na sequência, serão apresentados outros exemplos de modelos que exploram diferentes combinações de parâmetros. Pelo fato do conjunto de regras utilizadas para gerá-los ser extenso, decidiu-se pela sua disponibilização no GitHub¹, para uma eventual consulta.

¹ <https://github.com/DanielBrito/monografia/tree/main/Resultados>

A Figura 67 representa uma variação do modelo final obtido na Figura 66(j), por meio da inclusão de estruturas arredondadas na região lateral direita.

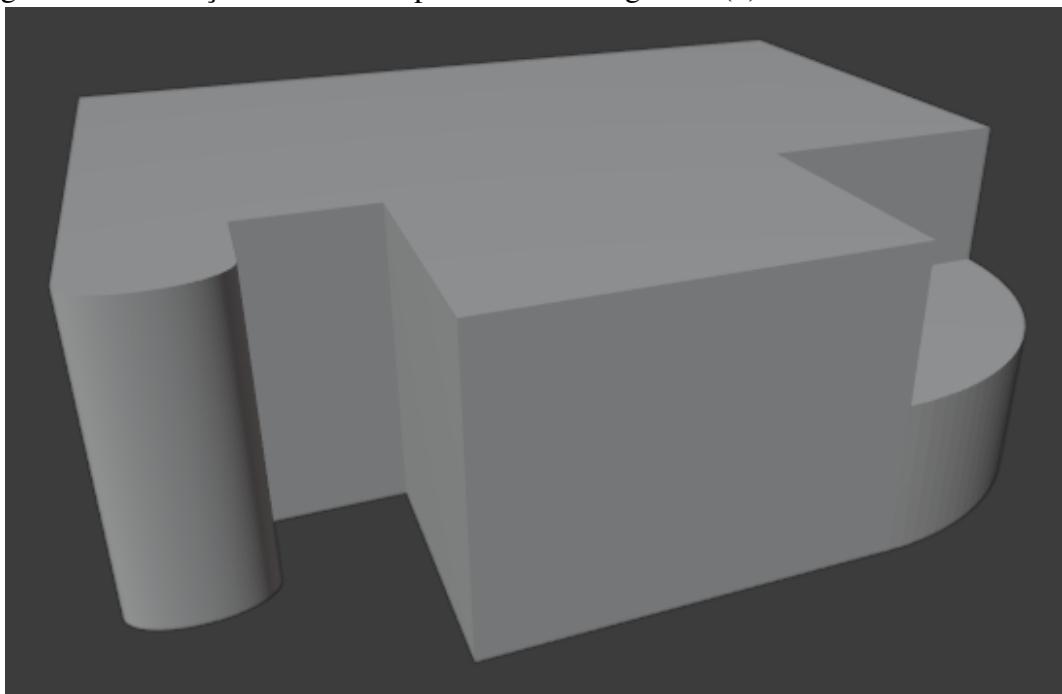
Figura 67 – Variação do modelo da Figura 66(j), com modificação na área e lateral.



Fonte: Próprio autor

A Figura 68(a), por sua vez, representa uma variação do modelo apresentado por Jiang *et al.* (2018), conforme ilustrado na Figura 29(o).

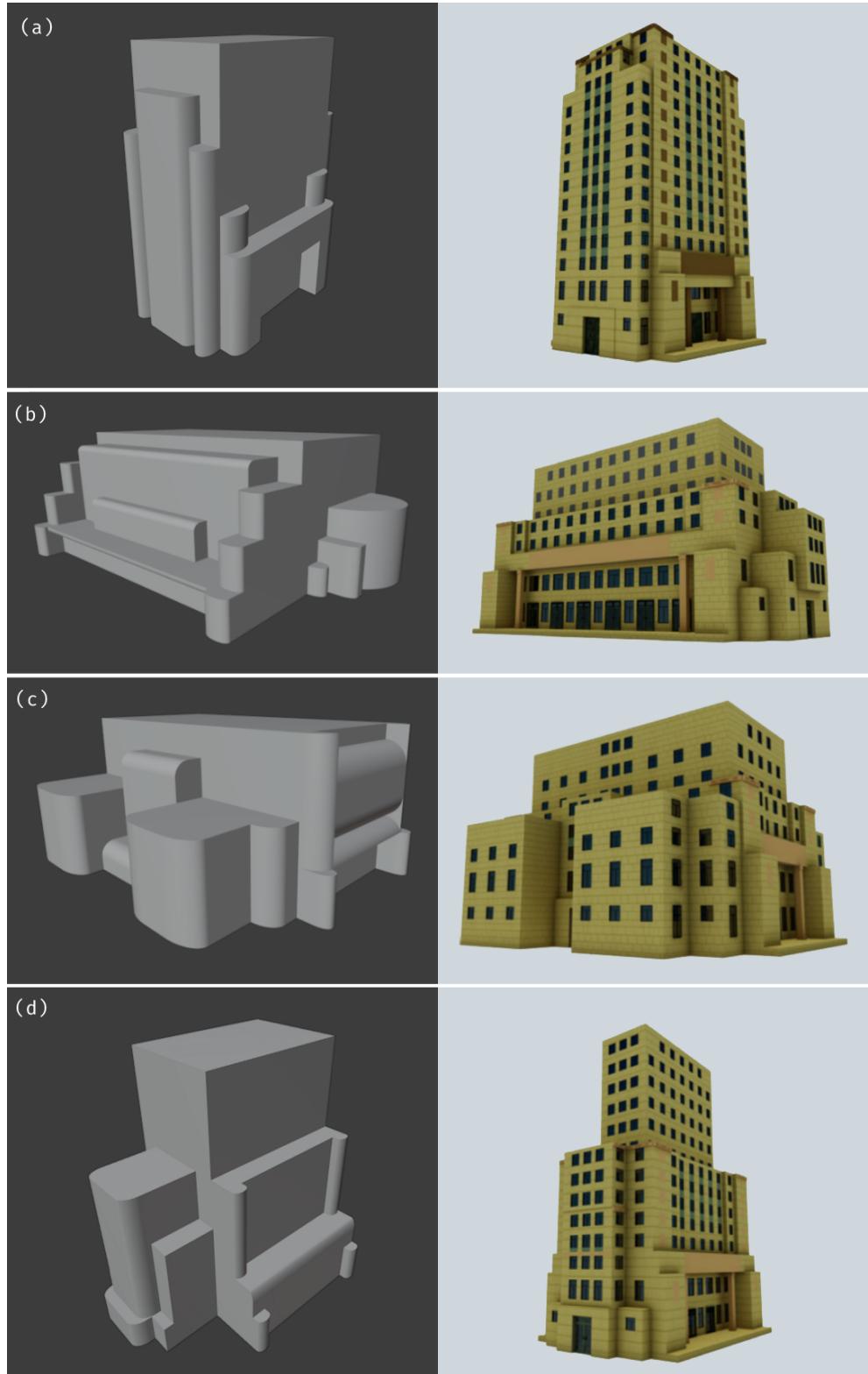
Figura 68 – Variação do modelo apresentado na Figura 29(o).



Fonte: Próprio autor

Os modelos apresentados nas Figuras 69(a), 69(b), 69(c) e 69(d), representam variações dos resultados obtidos por Jiang *et al.* (2018).

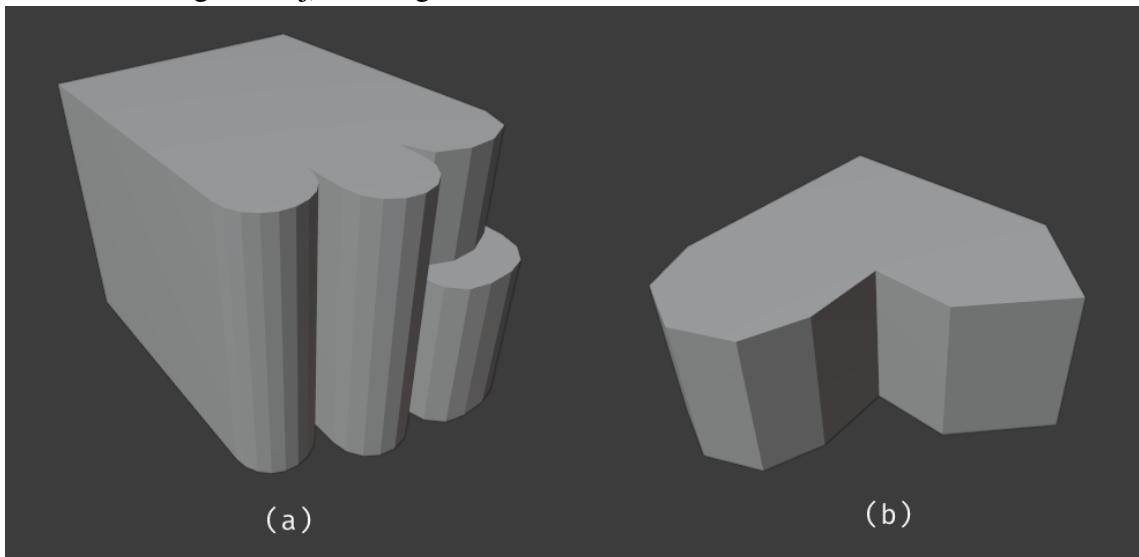
Figura 69 – À esquerda, variações dos resultados obtidos por Jiang *et al.* (2018), à direita.



Fonte: Próprio autor e Jiang *et al.* (2018)

A Figura 70 ilustra um dos diferenciais da solução proposta, que é a geração de modelos no formato *low poly*, ou seja, com poucos polígonos, os quais podem ser utilizados em situações nas quais um alto grau de realismo não é requerido.

Figura 70 – Variações *low poly* (a) e (b) dos modelos apresentados, respectivamente, na Figura 66(j) e na Figura 37.



Fonte: Próprio autor

O modelo da Figura 71 ilustra o resultado da operação `roundShape` na região superior, em diferentes faces.

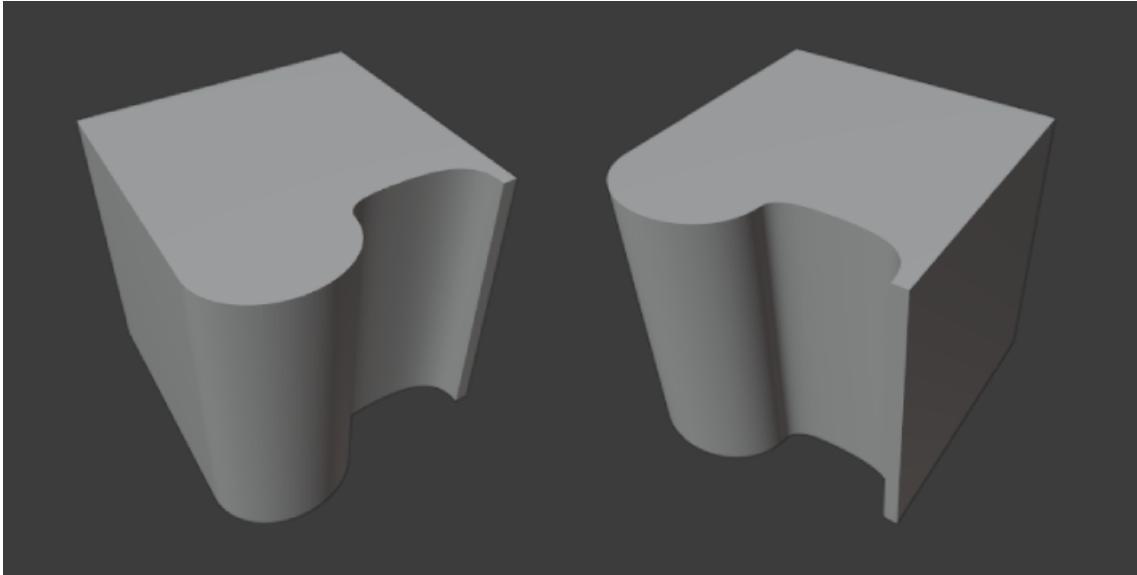
Figura 71 – Exemplo de arredondamento superior em múltiplas faces.



Fonte: Próprio autor

A Figura 72 demonstra o resultado da geração de um modelo obtido através da combinação de arredondamento externo e interno, produzindo uma arquitetura ondulada.

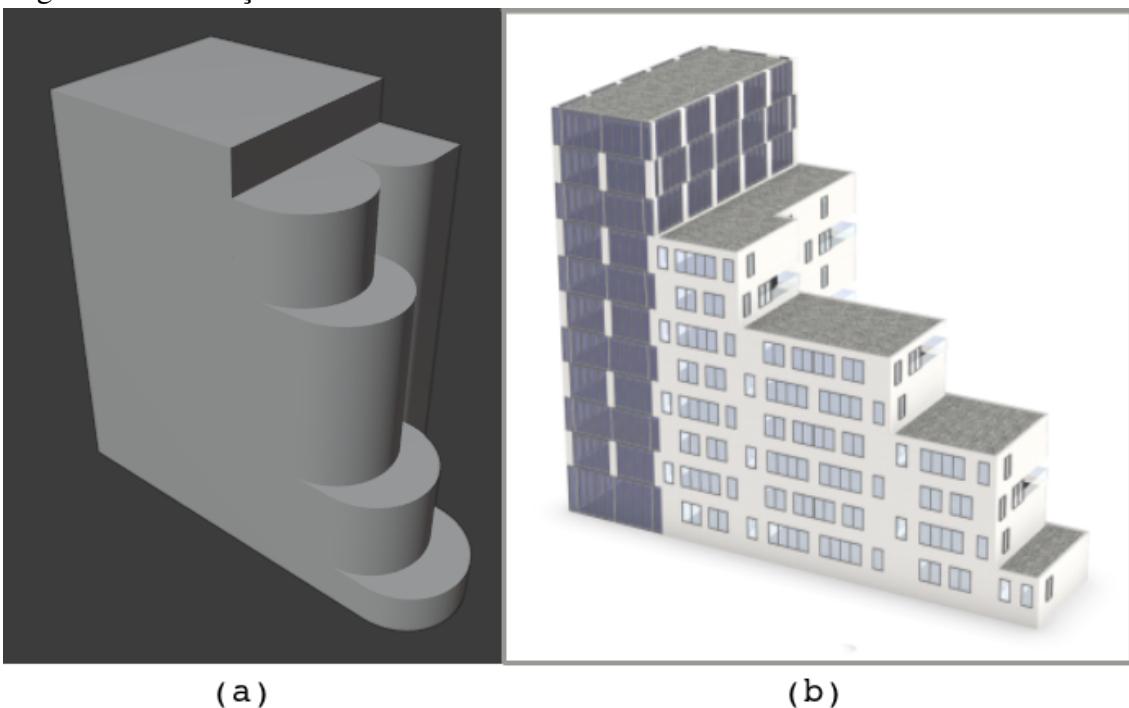
Figura 72 – Modelo combinando arredondamento externo e interno.



Fonte: Próprio autor

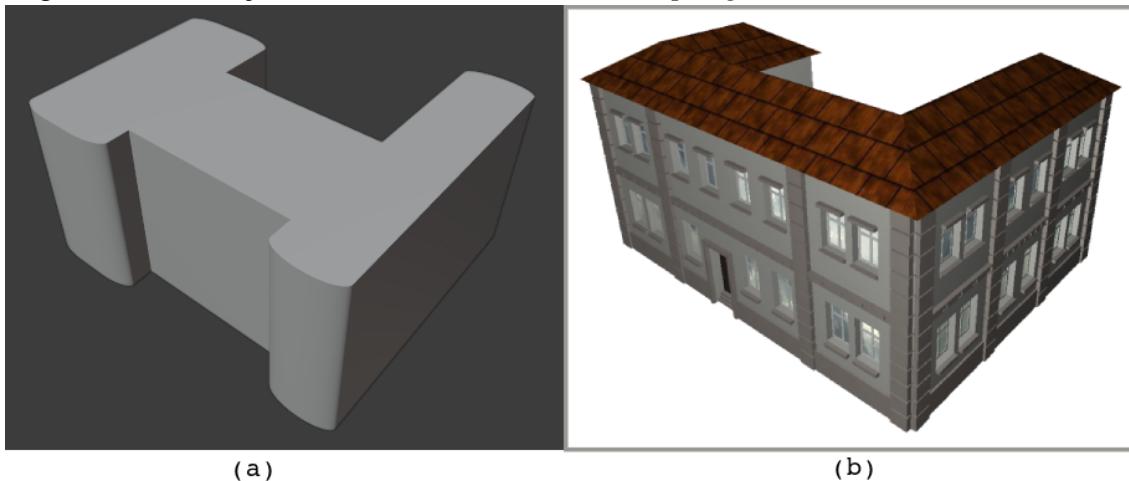
Nas Figuras 73 e 74, são ilustradas variações dos modelos apresentados por Schwarz e Müller (2015) e Wonka *et al.* (2003), respectivamente.

Figura 73 – Variação de modelo obtido através da *CGA++*.



Fonte: (a) Próprio autor e (b) adaptado de Schwarz e Müller (2015)

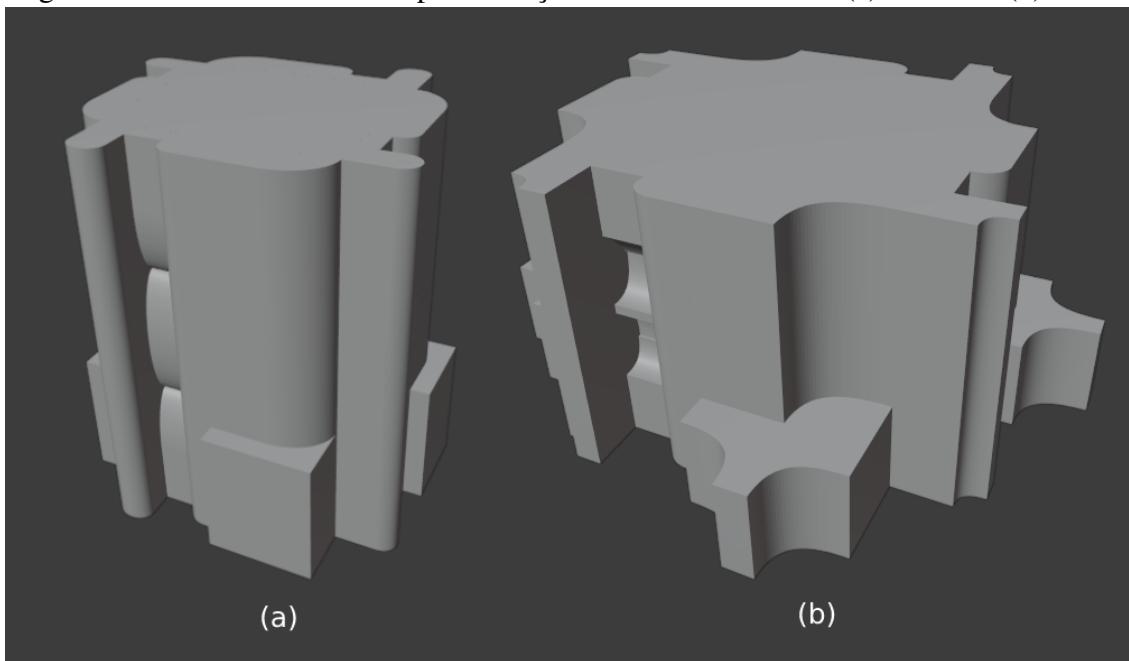
Figura 74 – Variação de modelo obtido através de *split grammar*.



Fonte: (a) Próprio autor e (b) Wonka *et al.* (2003)

Por fim, a Figura 75 ilustra modelos de massa mais complexos, que podem ser obtidos através de uma grande variedade de regras e parâmetros.

Figura 75 – Modelos com múltiplas variações de arredondamento (a) externo e (b) interno.



Fonte: Próprio autor

Uma vez que a capacidade de modelagem da solução proposta foi demonstrada, através de exemplos ilustrativos originais e de variações baseadas nos trabalhos descritos no Capítulo 2, nas próximas seções, serão apresentados dados estatísticos acerca dos modelos gerados, bem como algumas limitações identificadas.

5.2 Desempenho

As técnicas apresentadas nos Capítulos 2 e 3 são aplicadas em diversos cenários, possuindo métricas de desempenho distintas entre si. Portanto, como o presente trabalho é pioneiro na geração procedural de modelos arquiteturais com geometria arredondada utilizando *Selection Expressions*, ainda não existem parâmetros de comparação que estimem a sua qualidade. Logo, a Tabela 1 apresenta apenas informações estatísticas obtidas a partir dos modelos gerados pela solução proposta, mas que poderão servir como base em trabalhos futuros. Neste caso, não houve uma diferença perceptível no tempo necessário para gerar cada modelo, assim, ele foi descartado.

Tabela 1 – Dados estatísticos dos modelos gerados pela solução proposta.

Figura	Regras	Vértices	Arestas	Faces	Tamanho (MB)
66(f)	11	520	780	270	1,0
67	18	904	1356	468	1,1
68	8	212	319	114	1,0
69(a)	36	1576	2364	822	1,3
69(b)	31	1380	2070	720	1,3
69(c)	26	1056	1584	552	1,3
69(d)	24	892	1338	474	1,3
70(a)	11	96	144	58	1,1
70(b)	8	36	54	24	1,0
71	30	824	1236	438	1,2
72	9	272	408	144	1,1
73(a)	13	588	882	306	1,1
74(a)	12	520	780	270	1,1
75(a)	58	2392	3588	1254	1,5
75(b)	62	2152	3228	1134	1,5

5.3 Restrições

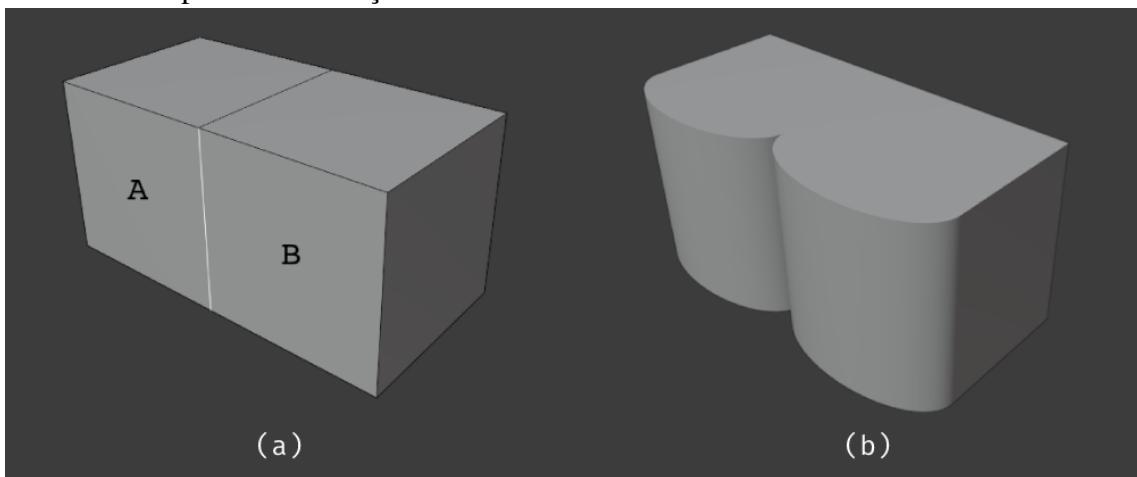
O presente trabalho representa um avanço no que se refere à geração procedural de modelos arquiteturais com geometria arredondada. Contudo, no decorrer do seu desenvolvimento, também foram identificadas algumas restrições.

A regra correspondente à *action roundShape* deve ser definida imediatamente após uma operação de extrusão, representada pela *action addVolume*. Isto significa que não é possível a realização de todas as operações de extrusão desejadas para que, posteriormente, sejam

realizadas as operações de deformação.

Além disto, não é possível realizar uma operação de arredondamento em faces paralelas que possuem aresta em comum. Por exemplo, no modelo mostrado na Figura 76(a), as faces A e B compartilham a aresta destacada em branco. Logo, não é possível realizar uma operação de arredondamento nesta região. Entretanto, este problema pode ser contornado por meio da extrusão individual de ambas as faces. Uma vez que isto acontece, pode-se aplicar a operação de deformação normalmente, conforme ilustrado na Figura 76(b).

Figura 76 – (a) Exemplo de faces que compartilham uma mesma aresta, e (b) solução para aplicar deformação.



Fonte: Próprio autor

Na implementação atual, também não é realizada uma representação da região arredondada na árvore de formas. Isto ocorre devido ao fato dos índices dos diversos segmentos criados após a operação `roundShape` não seguirem um padrão específico de numeração, o que dificulta a recuperação da referência dos respectivos polígonos para armazenamento.

Por fim, este capítulo descreveu as principais características da solução proposta, desde detalhes de implementação até algumas limitações encontradas. Por meio dos modelos gerados, percebe-se que a operação de deformação introduzida é capaz de produzir resultados interessantes, com estruturas arquitetônicas arredondadas que remetem a edifícios do mundo real. Assim, no próximo capítulo, serão apresentadas as conclusões finais.

6 CONCLUSÃO

Neste capítulo, serão apresentadas considerações acerca do que foi desenvolvido, bem como algumas questões em aberto que podem ser abordadas em trabalhos futuros.

6.1 Considerações

No decorrer do Capítulo 1, contextualizou-se o cenário em que a modelagem procedural é aplicada, apresentando também suas vantagens e desvantagens. Então, nos Capítulos 2 e 3, buscou-se descrever alguns conceitos e técnicas fundamentais para o entendimento da proposta apresentada no Capítulo 4. Por fim, no Capítulo 5, foram ilustrados alguns resultados obtidos por meio da estratégia desenvolvida, a qual é pioneira no que se refere à geração procedural de modelos arquiteturais com geometria arredondada utilizando *Selection Expressions*, o que representa um avanço na área, ainda mais porque integra-se com o Blender, uma poderosa ferramenta *open source* de modelagem.

Entre os diferenciais da solução proposta está a possibilidade da geração de modelos no formato *low poly*, priorizando o grau de desempenho ao invés do realismo, e também a possibilidade de edição do resultado final, através das ferramentas disponíveis no próprio Blender, após a geração do modelo por meio das regras definidas.

Além disto, outra característica a se destacar é o fato deste trabalho ter como resultado um projeto *open source*. Assim, pessoas interessadas podem contribuir com melhorias a partir do código-fonte e documentação, disponíveis no GitHub¹.

6.2 Trabalhos futuros

O presente trabalho teve como foco a utilização de *Selection Expressions* para geração procedural de modelos de massa. Contudo, baseado nas especificações da *SELEX*, ainda existem diversas funcionalidades a serem exploradas. Portanto, destacam-se como tópicos para trabalhos futuros:

- Manipulação da forma virtual em relação à região arredondada, visando uma representação na árvore de formas;
- Disposição de elementos como janelas e portas, através das formas virtuais, bem como a inclusão de telhados;

¹ <https://github.com/DanielBrito/monografia>

- Geração estocástica de múltiplos modelos, a fim de produzir um ambiente urbano;
- Definição de atributos de *design*, como cor e textura;
- Criação de *add-on* para simplificar a leitura do arquivo que contém as regras.

REFERÊNCIAS

- BLENDER FOUNDATION. **Blender**. 2020. Free and Open 3D Creation Software. Disponível em: <<https://www.blender.org>>. Acesso em: 24 out. 2020.
- EBERT, D. S.; MUSGRAVE, F. K.; PEACHEY, D.; PERLIN, K.; WORLEY, S. **Texturing and Modeling: A Procedural Approach**. 3rd. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002. ISBN 1558608486. Disponível em: <<https://dl.acm.org/doi/book/10.5555/572337>>.
- EDELSBRUNNER, J.; HAVEMANN, S.; SOURIN, A.; FELLNER, D. W. Procedural modeling of architecture with round geometry. **Computers & Graphics**, v. 64, p. 14 – 25, 2017. ISSN 0097-8493. Cyberworlds 2016. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0097849317300134>>.
- GU, N.; BEHBAHANI, P. A. Shape grammars: A key generative design algorithm. In: _____. **Handbook of the Mathematics of the Arts and Sciences**. Cham: Springer International Publishing, 2018. p. 1–21. ISBN 978-3-319-70658-0. Disponível em: <https://doi.org/10.1007/978-3-319-70658-0_7-1>.
- HAUBENWALLNER, K. Procedural generation using grammar based modeling and genetic algorithms. Graz University of Technology - Institute of Computer Graphics, Austria, 2016. Disponível em: <https://old.cescg.org/CESCG-2016/papers/Haubenwallner-Procedural_Generation_using_Grammar_based_Modelling_and_Genetic_Algorithms.pdf>.
- Jiang, H.; Yan, D.; Zhang, X.; Wonka, P. Selection expressions for procedural modeling. **IEEE Transactions on Visualization and Computer Graphics**, v. 26, n. 4, p. 1775–1788, 2018. Disponível em: <<https://ieeexplore.ieee.org/document/8502874>>.
- JIN, X.; LI, Y. F. Three-dimensional deformation using directional polar coordinates. **J. Graph. Tools**, A. K. Peters, Ltd., USA, v. 5, n. 2, p. 15–24, fev. 2000. ISSN 1086-7651. Disponível em: <<https://doi.org/10.1080/10867651.2000.10487521>>.
- KNIGHT, T. W. **Transformations in Design: A Formal Approach to Stylistic Change and Innovation in the Visual Arts**. USA: Cambridge University Press, 1995. ISBN 0521384605. Disponível em: <<https://dl.acm.org/doi/book/10.5555/572943>>.
- KRECKLAU, L.; PAVIC, D.; KOBBELT, L. Generalized use of non-terminal symbols for procedural modeling. **Computer Graphics Forum**, v. 29, n. 8, p. 2291–2303, 2010. Disponível em: <<https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2010.01714.x>>.
- LEBLANC, L.; HOULE, J.; POULIN, P. Component-based modeling of complete buildings. In: **Proceedings of Graphics Interface 2011**. Waterloo, CAN: Canadian Human-Computer Communications Society, 2011. (GI '11), p. 87–94. ISBN 9781450306935. Disponível em: <<https://dl.acm.org/doi/10.5555/1992917.1992932>>.
- LINDENMAYER, A. Mathematical models for cellular interactions in development. **Journal of Theoretical Biology**, v. 18, n. 3, p. 300 – 315, 1968. ISSN 0022-5193. Disponível em: <<http://www.sciencedirect.com/science/article/pii/0022519368900805>>.
- MAXWELL, S. **Capitol Dome**. 2008. Disponível em: <<https://www.freeimages.com/photo/capitol-dome-1207905>>. Acesso em: 19 out. 2020.

- MÜLLER, P.; WONKA, P.; HAEGLER, S.; ULMER, A.; GOOL, L. V. Procedural modeling of buildings. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 25, n. 3, p. 614–623, jul. 2006. ISSN 0730-0301. Disponível em: <<https://doi.org/10.1145/1141911.1141931>>.
- NISHIDA, G.; BOUSSEAU, A.; ALIAGA, D. G. Procedural Modeling of a Building from a Single Image. **Computer Graphics Forum**, Wiley, v. 37, n. 2, 2018. Disponível em: <<https://hal.inria.fr/hal-01810207>>.
- PROCHÁZKOVÁ, J. Free form deformation methods – the theory and practice. 02 2017. Disponível em: <https://www.researchgate.net/publication/314261522_free_form_deformation_methods_-_the_theory_and_practice>.
- PRUSINKIEWICZ, P.; LINDENMAYER, A. **The Algorithmic Beauty of Plants**. Berlin, Heidelberg: Springer-Verlag, 1996. ISBN 0387946764. Disponível em: <<https://dl.acm.org/doi/book/10.5555/235579>>.
- PYTHON SOFTWARE FOUNDATION. **Python**. 2021. Our Documentation - Python.org. Disponível em: <<https://www.python.org/doc/>>. Acesso em: 13 mar. 2021.
- RODRIGUES, F.; CAVALCANTE-NETO, J.; VIDAL, C. Split grammar evolution for procedural modeling of virtual buildings. In: . [S.l.: s.n.], 2015.
- SCHWARZ, M.; MÜLLER, P. Advanced procedural modeling of architecture. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 34, n. 4, jul. 2015. ISSN 0730-0301. Disponível em: <<https://doi.org/10.1145/2766956>>.
- SEDERBERG, T. W.; PARRY, S. R. Free-form deformation of solid geometric models. In: **Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques**. New York, NY, USA: Association for Computing Machinery, 1986. (SIGGRAPH '86), p. 151–160. ISBN 0897911962. Disponível em: <<https://doi.org/10.1145/15922.15903>>.
- SILVEIRA, I.; CAMOZZATO, D.; MARSON, F.; DIHL, L.; MUSSE, S. Real-time procedural generation of personalized facade and interior appearances based on semantics. In: . [s.n.], 2015. p. 89–98. Disponível em: <<https://ieeexplore.ieee.org/document/7785845>>.
- SIMON, L. Procedural reconstruction of buildings : towards large scale automatic 3d modeling of urban environments. 07 2011. Disponível em: <<https://tel.archives-ouvertes.fr/tel-00637638>>.
- SMELIK, R.; TUTENEL, T.; de Kraker, K.; BIDARRA, R. A declarative approach to procedural modeling of virtual worlds. **Computers & Graphics**, v. 35, n. 2, p. 352 – 363, 2011. ISSN 0097-8493. Virtual Reality in Brazil Visual Computing in Biology and Medicine Semantic 3D media and content Cultural Heritage. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0097849310001809>>.
- SMELIK, R. M.; TUTENEL, T.; BIDARRA, R.; BENES, B. A survey on procedural modeling for virtual worlds. **Comput. Graph. Forum**, The Eurographs Association & John Wiley & Sons, Ltd., Chichester, GBR, v. 33, n. 6, p. 31–50, set. 2014. ISSN 0167-7055. Disponível em: <<https://doi.org/10.1111/cgf.12276>>.
- STINY, G.; GIPS, J. ‘shape grammars and the generative specification of painting and sculpture’. In: . [s.n.], 1971. v. 71, p. 1460–1465. Disponível em: <<http://www.cs.bc.edu/~gips/ShapeGrammarsIFIPS71.pdf>>.

TEBOUL, O. **Shape grammar parsing : application to image-based modeling.** Tese (Theses) — Ecole Centrale Paris, jun. 2011. Disponível em: <<https://tel.archives-ouvertes.fr/tel-00628906>>.

THALLER, W.; KRISPEL, U.; ZMUGG, R.; HAVEMANN, S.; FELLNER, D. W. Shape grammars on convex polyhedra. **Computers & Graphics**, v. 37, n. 6, p. 707 – 717, 2013. ISSN 0097-8493. Shape Modeling International (SMI) Conference 2013. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0097849313000861>>.

TRUESELL, C.; NOLL, W. The non-linear field theories of mechanics. In: _____. **The Non-Linear Field Theories of Mechanics**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992. p. 1–579. ISBN 978-3-662-13183-1. Disponível em: <https://doi.org/10.1007/978-3-662-13183-1_1>.

VELDMAN-TENTORI, R. **Neuschwanstein Castle.** 2003. Disponível em: <<https://www.freeimages.com/photo/neuschwanstein-castle-1565577>>. Acesso em: 19 out. 2020.

VIMONT, U.; ROHMER, D.; BEGAULT, A.; CANI, M.-P. Deformation grammars: Hierarchical constraint preservation under deformation. **Computer Graphics Forum**, v. 36, n. 8, p. 429–443, 2017. Disponível em: <<https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13090>>.

WONKA, P.; WIMMER, M.; SILLION, F.; RIBARSKY, W. Instant architecture. In: **ACM SIGGRAPH 2003 Papers**. New York, NY, USA: Association for Computing Machinery, 2003. (SIGGRAPH '03), p. 669–677. ISBN 1581137095. Disponível em: <<https://doi.org/10.1145/1201775.882324>>.

ZMUGG, R.; THALLER, W.; KRISPEL, U.; EDELSBRUNNER, J.; HAVEMANN, S.; FELLNER, D. Procedural architecture using deformation-aware split grammars. **Vis. Comput.**, v. 30, p. 1009–1019, 09 2014. Disponível em: <<https://link.springer.com/article/10.1007/s00371-013-0912-3>>.

APÊNDICE A – ABORDAGENS PRECURSORAS DE MODELAGEM PROCEDURAL

Neste apêndice, de maneira cronológica, serão apresentadas algumas das principais técnicas precursoras de modelagem procedural, tais como *L-Systems* (Seção A.1), *Shape grammars* (Seção A.2) e *Split grammars* (Seção A.3).

A.1 L-Systems

Os *L-Systems* foram introduzidos por Lindenmayer (1968) para referenciar o estudo matemático da estrutura e desenvolvimento de organismos filamentosos. Sua ideia principal consiste em alterar partes de um objeto inicial, aplicando sucessivas regras de substituição, com objetivo de gerar objetos mais elaborados.

Conforme descrito por Simon (2011), uma forma é processada em duas etapas separadas. Na primeira, é realizado um processo de derivação a partir de um elemento inicial, o axioma. Na segunda, a *string* pode ser interpretada geometricamente utilizando gráficos tartaruga. Nos gráficos tartaruga, a interpretação geométrica é obtida tomando cada símbolo como um desenho ou comando de movimento. Sintaticamente, um estado da tartaruga é dado pelo terceiro (x, y, α) , onde (x, y) representa uma posição cartesiana, e α representa um ângulo que define a direção para qual a tartaruga está voltada. Dados um tamanho de passo d e um ângulo β , a tartaruga pode responder aos seguintes comandos:

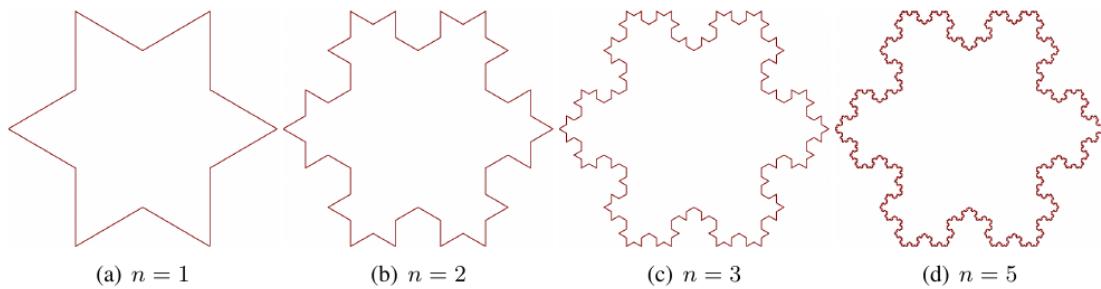
- F Mover à frente uma distância d . Desta maneira, o estado da tartaruga muda para (x', y', α) , onde $x' = x + d\cos(\alpha)$ e $y' = y + d\sin(\alpha)$, ou seja, uma linha entre (x, y) e (x', y') é desenhada;
- $+$ Girar à esquerda por um ângulo β , fazendo o estado mudar para $(x, y, \alpha + \beta)$;
- $-$ Girar à direita por um ângulo β , fazendo o estado mudar para $(x, y, \alpha - \beta)$;
- $[$ Empilhar a posição atual na pilha de localização;
- $]$ Ir para a posição gravada no topo da pilha e a desempilhar.

Historicamente, F , $+$, $-$, foram os primeiros símbolos introduzidos visando a modelagem de curvas fractais. Para modelar estruturas ramificadas, como plantas e árvores, os dois símbolos de colchetes foram adicionados, um para registrar a posição atual da tartaruga em uma pilha, e outro para recuperá-la (SIMON, 2011).

O floco de neve de Von Koch, representado na Figura 77, é um exemplo clássico de

curva fractal gerado por *L-Systems*. O axioma é $\omega = F + +F + +F$, e a gramática é composta por uma única regra $F \rightarrow F - F + +F - F$. Cada rotação corresponde a um ângulo $\alpha = 60$. A representação geométrica é mostrada para diferentes valores de n (SIMON, 2011), parâmetro que representa a quantidade de iterações da derivação.

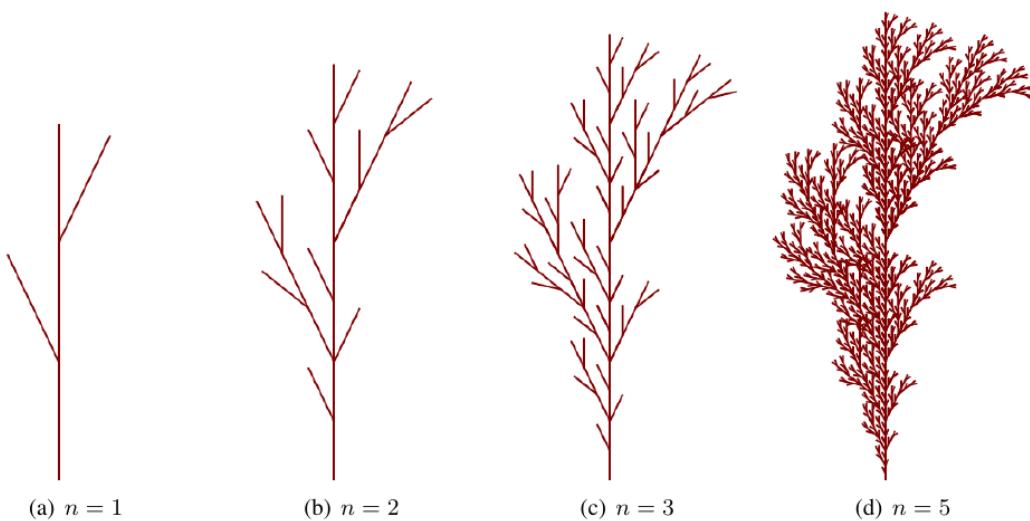
Figura 77 – Geração do floco de neve de Von Koch utilizando *L-Systems*.



Fonte: (SIMON, 2011)

A Figura 78 representa o sistema de uma alga, com intuito de ilustrar o uso dos colchetes. O axioma é $\omega = F$, e uma única regra $F \rightarrow F[+F]F[-F][F]$ cria duas ramificações. São mostradas diferentes escalas de representação para um ângulo $\alpha = 25.7$ (SIMON, 2011). Da mesma maneira como no exemplo anterior, a representação geométrica é mostrada para diferentes valores de n . Além disto, por meio da extensão da gramática e das regras é possível obter resultados similares ao da Figura 79.

Figura 78 – Geração de um modelo de alga utilizando *L-Systems*.



Fonte: (SIMON, 2011)

Figura 79 – Renderização tridimensional do modelo de uma *Mycelis*.



Fonte: (PRUSINKIEWICZ; LINDENMAYER, 1996)

A.2 Shape Grammars

A ideia central das *shape grammars* para criação de modelos 3D é iniciar com uma forma básica, como um cubo, e modificá-lo até que se pareça com o modelo desejado. Este procedimento é realizado aplicando-se várias operações à forma básica, como translação e rotação, até outras mais complexas, como extrusão e divisão, resultando em um modelo mais elaborado. O formalismo das *shape grammars* é dado pela declaração de um estado inicial (axioma), bem como pela atribuição de símbolos às formas, e também pela definição de regras de produção, que ditam como gerar símbolos e como aplicar as diferentes operações. Uma vez que as operações podem criar novas formas e o modelo resultante depende do encadeamento de múltiplas operações, geralmente, é utilizada uma árvore de derivação para armazenar tal sequência (HAUBENWALLNER, 2016).

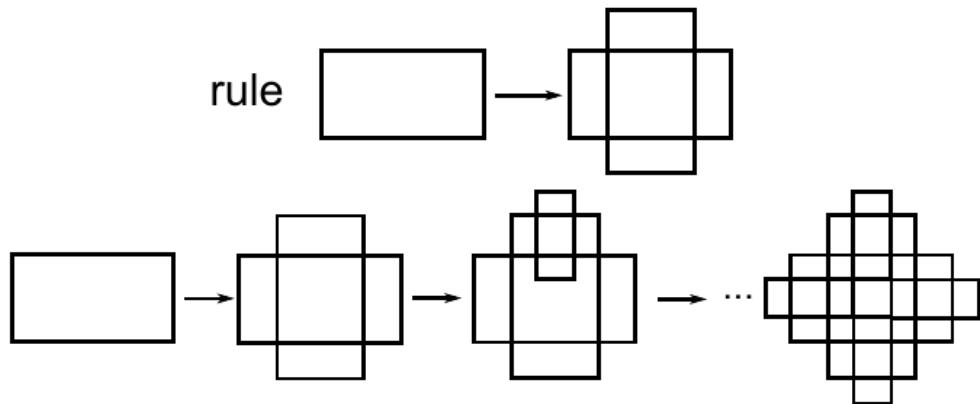
Stiny e Gips (1971) introduziram as *shape grammars* para representar a geração procedural de geometria. Por definição, uma *shape grammar* é uma quádrupla (V, L, R, ω) , onde:

- V : é um conjunto finito de formas (vocabulário);
- L : é um conjunto finito de símbolos (rótulos);
- R : é um conjunto finito de regras no formato $\alpha \rightarrow \beta$, onde α e β são formas rotuladas;
- ω : é uma forma atômica chamada de axioma.

Uma regra $\alpha \rightarrow \beta$ se aplica a uma dada forma γ se existe uma transformação δ , tal que $\delta(\alpha)$ está contido em γ . Após a aplicação de uma regra, a forma de saída é obtida substituindo-se na forma γ , a subforma $\delta(\alpha)$ por $\delta(\beta)$.

A Figura 80 ilustra um exemplo clássico de *shape grammar*, criado por Knight (1995), para descrever a planta de igrejas no formato de cruz grega. Contém uma única regra, transformando um retângulo em outros dois perpendiculares. Na primeira linha, define-se a única regra da gramática. Na segunda linha, apresenta-se uma possível derivação em n etapas, a partir de um retângulo como axioma.

Figura 80 – Gramática para gerar planta de igreja na forma de cruz grega.



Fonte: (KNIGHT, 1995)

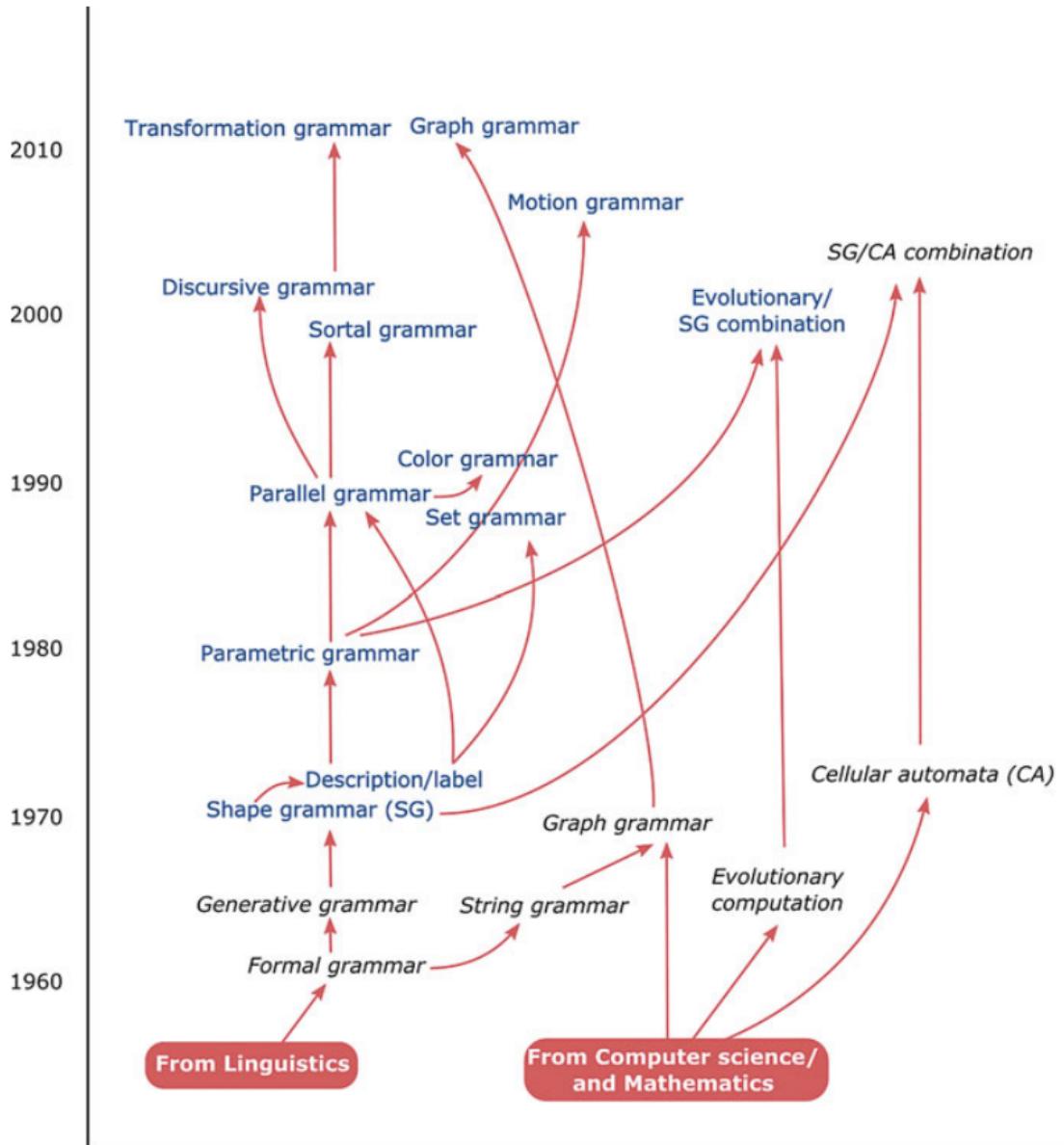
Na Figura 81, através de uma perspectiva cronológica, Gu e Behbahani (2018) representam os diferentes desenvolvimentos que ampliaram as *shape grammars* básicas, visando atender diferentes necessidades de *design*.

A.3 Split Grammars

As *split grammars* foram introduzidas por Wonka *et al.* (2003), sendo utilizadas, geralmente, para modelagem procedural de fachadas de edifícios. Uma descrição do formalismo é apresentada por Rodrigues *et al.* (2015), na qual uma *split grammar* é denotada por uma tupla $G = (N, T, \sigma, R)$, onde N representa o conjunto de símbolos não-terminais, T representa o conjunto dos símbolos terminais, $\sigma \in R$ representa a regra inicial, e R representa o conjunto de regras. As regras $r \in R$, por sua vez, têm a forma $LHS \rightarrow RHS$, na qual $LHS \in N$ é um único símbolo não-terminal, e RHS é uma operação *split* que aplicará as regras sucessoras de LHS .

Rodrigues *et al.* (2015) explicam que uma operação de *split* divide uma região

Figura 81 – Diagrama que descreve a cronologia das *shape grammars*.



Fonte: (GU; BEHBAHANI, 2018)

geométrica horizontalmente ou verticalmente, e tem a seguinte configuração:

$$\text{split}(\text{axis}) \rightarrow (\text{repeat})\{\mu_0 s_0 : r_0, \dots, \mu_n s_n : r_n\},$$

onde axis representa o eixo em que a região do espaço será dividida; o símbolo repeat , exemplificado na Figura 83, é um modificador de regra opcional, que indica se as divisões devem ser aplicadas novamente, caso ainda haja espaço; s_i define o tamanho da divisão a ser aplicada na geometria, que resultará em uma nova região retangular, na qual será aplicada a produção $r_i \in (N \cup T)$; e μ_i é o modificador da divisão, podendo ser de três tipos:

Absoluto: Utilizado para criar uma nova região de tamanho igual a s_i ao longo do

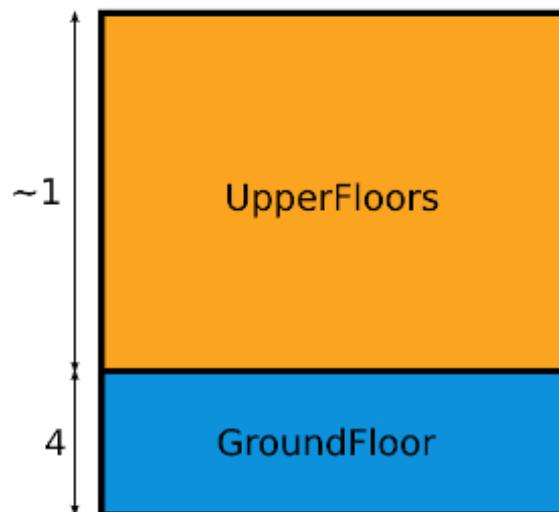
eixo selecionado, se houver espaço suficiente.

Aproximado: Força a nova região a ter tamanho, pelo menos, s_i , sendo representado na gramática pelo símbolo \sim . Ao final de todas as divisões da operação *split*, se ainda existir espaço sobrando, as regiões marcadas com este modificador aumentarão de tamanho por um número proporcional a s_i , fazendo com que toda a região passada como entrada para a regra seja ocupada.

Relativo: Dado um valor t como o tamanho da forma inicial, este modificador, representado na gramática pelo símbolo $'$, atua de maneira a criar uma nova região de tamanho $s_i * t$.

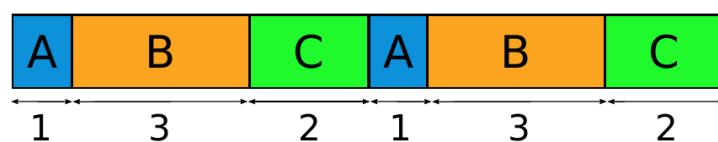
Segundo Rodrigues *et al.* (2015), numa única operação de *split* é permitido que cada divisão possua um modificador diferente. Na Figura 82, por exemplo, é mostrado o uso dos modificadores absoluto e aproximado em uma mesma operação, através da regra *Facade* \rightarrow *split(Y)4 : GroundFloor;~1 : UpperFloors*. A Figura 83, por sua vez, ilustra o uso do *repeat*, por meio da regra *S* \rightarrow *repeatsplit(X)1 : A, 3 : B, 2 : C*.

Figura 82 – Exemplo de *split* com o modificador aproximado (\sim).



Fonte: (RODRIGUES *et al.*, 2015)

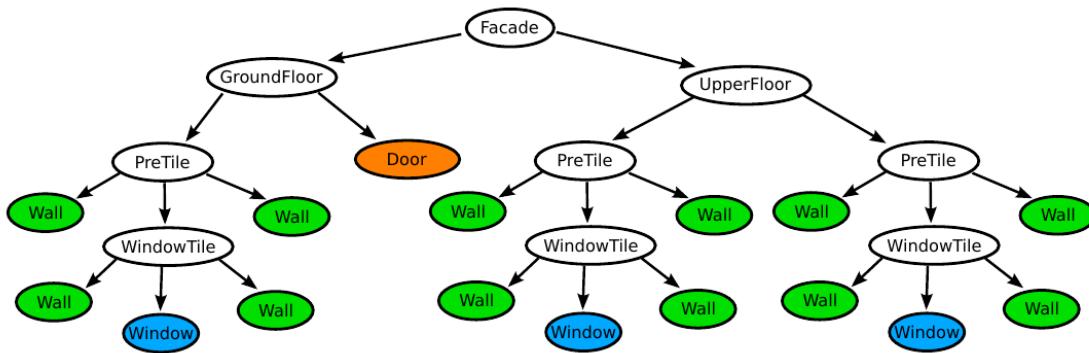
Figura 83 – Exemplo de *split* com repetição.



Fonte: (RODRIGUES *et al.*, 2015)

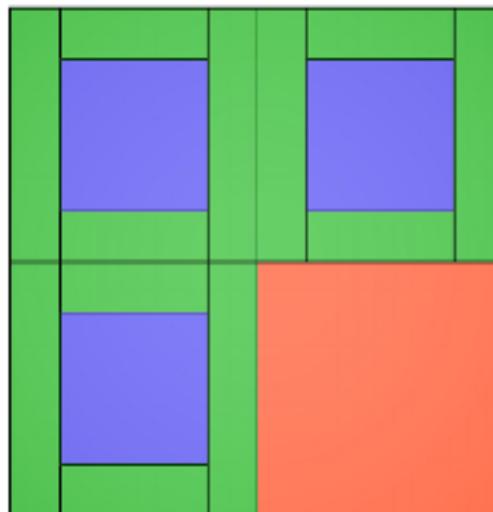
Conforme argumentado por Rodrigues *et al.* (2015), a partir de uma palavra de uma dada gramática, é possível representar uma fachada por meio de uma árvore de derivação, onde cada nó desta árvore associa uma região retangular do espaço a um símbolo da gramática $\varphi \in (T \cup N)$. Os nós intermediários particionam esta região através da aplicação de uma regra $\eta \in R$, e os nós folhas, por sua vez, representam os elementos atômicos que compõem a fachada, como janelas, portas e paredes, os quais são identificados pelos símbolos terminais $\tau \in T$ associados a eles. Para ilustrar esta ideia, a Figura 84 representa a árvore de derivação correspondente à fachada da Figura 85.

Figura 84 – Árvore de derivação referente à fachada representada na Figura 85.



Fonte: (RODRIGUES *et al.*, 2015)

Figura 85 – Fachada associada à árvore de derivação da Figura 84.



Fonte: (RODRIGUES *et al.*, 2015)