D 2018

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

SKETCH BASED INTERACTION FOR PROCEDURAL MODELLING

DIEGO ANTÓNIO RODRIGUES DE JESUS

TESE DE DOUTORAMENTO APRESENTADA
À FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO EM
ENGENHARIA INFORMÁTICA

Sketch Based Interaction for Procedural Modelling

A DISSERTATION PRESENTED
BY
DIEGO ANTÓNIO RODRIGUES DE JESUS
TO
THE DEPARTMENT OF INFORMATICS ENGINEERING

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN THE SUBJECT OF
INFORMATICS ENGINEERING

UNDER THE SUPERVISION OF
PROFESSOR ANTÓNIO FERNANDO VASCONCELOS CUNHA CASTRO COELHO
AND
PROFESSOR ANTÓNIO AUGUSTO SOUSA

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO
PORTO, PORTUGAL
SEPTEMBER 2018

Sketch Based Interaction for Procedural Modelling

Abstract

The improvements on computer hardware have made the reproduction of larger and more detailed urban environments possible and the users are demanding the full use of such capabilities. However, these environments are complex spaces, offering a wide variety of elements and modelling them, using traditional tools, may take several man-years since every urban element must be modelled in detail. To cope with such request for greater detail, there has been a shift of focus to procedural modelling systems which, from a small set of rules and coupled with a controlled level of randomness, are capable of generating geometry and textures in a semi-automatic way, alleviating the user from the burden of manually producing the models that represent the urban environment.

The typical workflow of a procedural modelling system, however, is not intuitive, lacks predictability and controllability and requires users to have a certain level of expertise. Moreover, the traditional procedural modelling techniques for urban environments present a grammar-based approach to create and edit such rules, requiring the user to have knowledge of the underlying procedural techniques. Since the target audience for this type of systems are, typically, creative professionals, this is an obvious disadvantage.

On the other hand, these professionals tend to use paper and pencil in the early stages of content creation, drawing concept art that will guide them through the process of creating three-dimensional content. This two-stages process is often seen as a bottleneck, introducing delays in the creation process, and forces the user to focus on the modelling process, reducing the creativity. Sketch based interfaces for modelling aim at reducing this barrier while, at the same time, enhancing their focus on creativity.

The presented approach integrates Procedural Modelling techniques with sketch based interfaces to minimise the interaction problems of Procedural methods. Letting the user sketch the intended scene may allow a greater focus on the creation process while masking the intricate details of procedural techniques. The procedural rules are generated behind the scenes which can later be used to produce variations of the models easily.

Nonetheless, there are a few setbacks in current procedural modelling methods that disallow a straightforward integration with sketching interfaces. Namely, the over-reliance on split operations tends to generate rectangular or cuboid shapes which heavily contrasts with the expressiveness of sketching and the ambiguity of user selections which undermine the direct control in an interactive procedural modelling tool.

To overcome the first problem, this thesis presents an extension to shape grammars that allows the user to define layers within planar surfaces of the procedural model where vectorially defined shapes can be created. This provides the sufficient tools to create more complex shapes (such as arched windows) directly in the procedural framework. Results have demonstrated that this does increase the expressiveness of split based procedural methods.

The second problem is evident as a set of user selected shapes can, in a procedural modelling environment, have many different meanings due to its data amplification nature. Selections, in the presented thesis, are achieved not as a set of shapes but, instead, as semantic queries which upon being evaluated yield sets of shapes. These queries remain valid as long as the semantic behind the model remains relatively similar and are, thus, robust to changes. Manually creating these queries can easily become a burden to the user and, therefore, a selection inference system has been devised to infer queries from a few user selected shapes. Results have shown that these queries can express complex conditions that would be extremely complicated to achieve otherwise and that the inference subsystem can generate queries that are identical to what the user has intended.

Building on top of the previous contributions and relying on the already extensively studied field of Sketch Based Interfaces for Modelling, this thesis presents a sketch based procedural modelling system that allows the user to create full models without diverting into other elements of the user interface. Such system uses the set of available procedural operations to inform the sketch processing pipeline to infer a 3D representation of the user strokes and, afterwards, to recognise a set of 3D lines as a procedural operation candidate. To adjust the operations' parameters to closely match the user sketch, the system uses a gradient descent algorithm with a distance based function.

The presented system establishes the roots towards a full-fledged sketch based procedural modelling system oriented at buildings creation. A new set of research questions have been raised by the development of this work which, in future work, can further increase the expressiveness, intuitiveness and acceptance of urban procedural modelling methods.

Sketch Based Interaction for Procedural Modelling

Resumo

Os avanços no *hardware* de computador tornaram possíveis a reprodução de ambientes urbanos virtuais maiores e com mais detalhe. Por sua vez, os utilizadores finais destes ambientes requerem a utilização de todas as capacidades disponibilizadas por estes avanços. No entanto, estes ambientes são espaços complexos, oferecendo uma grande variedade de elementos cuja modelação usando ferramentas tradicionais pode demorar vários *pessoa-ano* pois todos os elementos urbanos têm de ser modelados individualmente e em detalhe. Para satisfazer tal necessidade, foram já propostas várias técnicas de modelação procedimental que, a partir de um pequeno conjunto de regras e com um nivel de aleatoriedade controlado, são capazes de gerar geometrias e texturas de uma forma semi-automática, aliviando o utilizador de produzir manualmente os modelos que representam o ambiente urbano.

O fluxo de modelação tipico num sistema de modelação procedimental é, no entanto, pouco intuitivo, imprevisivel, disponibilizando um baixo nível de controlo sobre os modelos e requerendo que os utilizadores tenham um conhecimento extenso das ferramentas. Adicionalmente, as técnicas mais comuns de modelação procedimental para ambientes urbanos apresentam uma abordagem baseada em gramáticas formais para criar e editar as re-

gras, exigindo do utilizador conhecimento avançado destas técnicas. Visto que a audiência desejada deste tipo de sistemas é, tipicamente, utilizadores com antecendentes creativos (e.g., *designers*), tal é uma desvantagem óbvia.

Por outro lado, estes profissionais usam, nas primeiras etapas da creação de conteúdo, papel e lápis com o intuito de esboçar conceitos que os irá ajudar durante o processo de criação de conteúdo tridimensional. A divisão deste processo em duas etapas é visto como obstáculo introduzindo atrasos no processo criativo e forçando o foco do utilizador no processo de modelação, reduzindo a criatividade. As interfaces caligráficas (*Sketch Based Interfaces*) para a modelação têm como objectivo diminuir esta barreira e, ao mesmo tempo, aumentar o foco na criatividade.

A proposta apresentada nesta tese integra técnicas de modelação procedimental com interfaces caligráficas para a modeleção procedimental com o objectivo de diminuir os problemas de interação inerentes. Permitir ao utilizador esboçar a cena desejada pode aumetar a criatividade do utilizador mascarando, também, os detalhes complexos das técnicas procedimentais. As regras são, portanto, geradas transparentemente e podem ser utilizades posteriormente para, facilmente, criar varições dos modelos.

Contudo, há um conjunto de problemas em aberto no domínio da modelação procedimental que impedem uma fácil integração com as técnicas de modelação procedimental urbana. Nomeadamente, a dependência em operações de corte têm tendência a criar geome-

trias rectangulares ou cubóides, contrastando com a expressividade do esboço. Por sua vez, a ambiguidade da seleção de formas por parte do utilizador debilitam o controlo directo no contexto de uma ferramenta interactiva de modelação procedimental.

Para superar o primeiro problema, esta tese apresenta uma extensção às *shape grammars* que permite ao utilizador definir camadas em superficies planas do modelo procedimental onde formas vectoriais podem ser definidas. Isto permite a criação de geometrias mais complexas (tais como janelas em arco) diretamente na ferramenta procedimental. Os resultados alcançados demonstram que tal contribuição aumenta a expressividade em comparação com métodos baseados em cortes.

O segundo problema torna-se evidente uma vez que um conjunto de geometrias selecionadas pelo utilizador num ambiente interativo de modelação procedimental podem ter vários significados diferentes devido à natureza amplificadora de dados destes métodos. Nesta tese, as seleções são definidas, não como um conjunto de geometrias, mas sim como consultas semânticas que, após serem avaliadas, resultam num conjunto de geometrias. Estas consultas têm a capacidade de se manter válidas desde que a semântica do modelo procedimental se mantenha relativamente semelhante e apresentam, portanto, robustez à mudança. Contudo, a criação manual destas consultas pode facilmente tornar-se demasiado complexa para o utilizador e, desta forma, um sistema de inferência foi também desenvolvido. Este tem como objectivo inferir consultas semânticas a partir de algumas geometrias selecionadas pelo utilizador. Os resultados demonstram que estas consultas conseguem expressar condições que seriam extremamente complexas de alcançar de outra forma e que o sistema de inferencia pode gerar consultas que são identias àquelas que o utilizador pretendia.

Assentando nestas contribuições e tendo em conta o amplamente estudado domínio das interfaces caligráficas para a modelação, esta tese apresenta um sistema de modelação procedimental caligráfico que permite ao utilizador criar modelos completos sem que seja necessário recorrer a outros elementos da interface. Este sistema usa o conjunto de operações procedimentais para complementar a sequência de passos necessários ao reconhecimento de esboço, permitindo inferir uma representação tridimensional do esboço e, posteriormente, inferir qual a operação procedimental desejada. Para ajustar os parâmetros das operações de forma a que estas produzam geometria semelhante ao esboço, o sistema usa o método de gradiente descendente com uma função baseada na distância entre as linhas 3D e as arestas da geometria.

O sistema apresentado aparenta ser um passo promissor na direcção de um sistema completo de modelação procedimental caligráfica. Um novo conjunto de questões de investigação tornou-se evidente durante o desenvolvimento deste trabalho que, após terem resposta num trabalho futuro, trazem a possibilidade de aumentar a expressividade dos métodos de modelação procedimentais orientados a espaços urbanos, tornando-os mais intu-

itivos e aumentando a sua aceitação por parte dos utilizadores.

Contents

1 Introduction 1

- 1.1 Motivation . . . 5
- 1.2 Hypothesis and Research Questions . . . 6
- 1.3 Objectives . . . 8
- 1.4 Related Publications . . . 9
- 1.5 Problems in Sketch Based Procedural Modelling of Buildings . . . 10
  - 1.5.1 Lack of Direct Control . . . 11
  - 1.5.2 Restrictions of Split-Based Methods . . . 12
  - 1.5.3 Semantic Ambiguity in User Selections . . . 14
  - 1.5.4 Procedural Rule Inference and Parameter Estimation from Sketches 15
- 1.6 Solution Overview . . . 17
  - 1.6.1 Layer-Based Procedural Modelling . . . 17
  - 1.6.2 Semantic Selection of Sets of Shapes . . . 19
  - 1.6.3 Procedural Rule Inference . . . 21
- 1.7 Document Structure . . . 23

2 Procedural Modelling 25

- 2.1 Definition . . . 26
- 2.2 Advantages . . . 27
- 2.3 Disadvantages . . . 28
- 2.4 Procedural Modelling of Virtual Urban Environments . . . 30
  - 2.4.1 Terrain Modelling . . . 32
  - 2.4.2 Road Network Modelling . . . 36
  - 2.4.3 Building Modelling . . . 41
  - 2.4.4 Façade Modelling . . . 53
  - 2.4.5 Modelling Other Content . . . 59
- 2.5 Open Questions . . . 60
- 2.6 Summary . . . 62

3 Sketch Based Interfaces for Modelling 65

- 3.1 Sketching Pipeline . . . 67
- 3.2 Model Creation . . . 71

Evocative Systems . . . 72
3.2.1 Constructive Systems . . . 73
3.3 Modification . . . 77
3.3.1 Augmentative Systems . . . 77
3.3.2 Deformation Systems . . . 78
3.4 Sketch Based Modelling of Urban Entities . . . 79
3.5 Open Questions . . . 82
3.6 Summary . . . 84

4 Layered Shape Grammars 85
4.1 Overview . . . 86
4.2 Layered Shape Grammars . . . 89
4.2.1 Procedural Items . . . 89
VolumetricShape . . . 90
PlanarShapes . . . 92
Layers . . . 94
Shape2D . . . 95
4.2.2 Overloaded Procedural Rules . . . 97
4.2.3 Procedural Operations . . . 99
Layer Creation Operation . . . 99
Segment Operation . . . 101
Vectorial Shape Operation . . . 106
Set Property Operation . . . 108
4.3 Planar Shape Normalisation . . . 110
4.3.1 Layers Merge . . . 112
4.3.2 Conversion to Planar Shapes . . . 115
4.3.3 Deferred Procedural Items . . . 118
4.4 Derivation Process . . . 120
4.5 Limitations . . . 123
4.6 Summary . . . 124

5 Generalised Selections for Improved Direct Control in Procedural Modelling 127
5.1 Overview . . . 129
5.2 Selection Action Paradigm . . . 133
5.3 Semantic Queries . . . 134
5.3.1 Selectors . . . 136
5.3.2 Hierarchical Relations . . . 141

5.4 Selection Action Tree 146
5.4.1 Shorthand notation 148
5.4.2 Procedural Derivation 151
5.5 User Selection Inference 155
5.5.1 Semantic Profiles 159
5.5.2 Initial Population 160
5.5.3 Evaluation 161
5.5.4 Mutation 163
5.5.5 Crossover 165
5.6 Applications 166
5.6.1 Layers of procedural content 166
5.6.2 Generation of level mechanics 170
5.6.3 City Evolution 172
5.6.4 Weathering 174
5.6.5 Application to non-procedural buildings 175
5.6.6 Limitations 175
5.7 Summary 177

6 Sketch Based Procedural Modelling 179
6.1 Overview 181
6.2 User Interface and Workflow 183
6.3 Stroke Filtering and Segmentation 186
6.4 Converting Strokes to World-Space Lines 194
6.4.1 Line Candidate Generation 196
6.4.2 Candidate Line Evaluation 201
6.5 Heuristics for the Inference of Procedural Operations 203
6.5.1 Extrusion Heuristic 205
6.5.2 Offset Heuristic 206
6.5.3 Split and Repeat Split Heuristic 207
6.5.4 Component Split Heuristic 208
6.5.5 Sample Based Generation of Candidate Procedural Operations . . 209
6.6 Parameter Adjustment 210
6.6.1 Error Function 211
6.6.2 Gradient Descent Method for Parameter Adjustment 212
6.7 Tools for Precise Sketching of Architecture 217
6.7.1 Analytical Drawing 217
6.7.2 Guidelines 219
6.7.3 Support Lines 220

6.8 Summary 222

7 Results and Discussion 227
7.1 Layered Shape Grammars 228
7.1.1 Simple Façade Example 228
7.1.2 Expressiveness 231
Complex Windows 231
Modelling the Mikimoto Ginza Building 234
7.1.3 Comparison to Split Based Methods 239
7.2 Generalised Selections for Improved Direct Control in Procedural Modelling 243
7.2.1 Modelling Session Example 244
7.2.2 Usability 246
7.2.3 Selection Examples 251
7.2.4 Comparison to existing methods 252
7.2.5 User Selection Inference 256
7.3 Sketch Based Procedural Modelling 262
7.3.1 Methodology 263
7.3.2 User Characterisation 265
7.3.3 Initial Opinions Regarding the Methodology and Prototype 271
7.3.4 First task: Sketching Interface for Procedural Modelling 278
7.3.5 Second task: Usage of Guidelines and Support Lines 281
7.3.6 Third task: Modelling an Example Building 283
7.3.7 User Opinion After Task Completion 285
7.3.8 Discussion 295
7.4 Summary 297

8 Conclusions and Future Work 301
8.1 Research Questions 302
8.1.1 How can procedural modelling methods be improved to provide enough expressiveness to be better paired with sketch interfaces? 302
8.1.2 What is required from a procedural modelling method to improve the amount of direct control a user can have over the generated models? 304
8.1.3 How can semantics present in procedural modelling rules be used in a sketch based procedural modelling system? 306
8.1.4 How can a sketch based interface for modelling be used in the procedural modelling of buildings? 307
8.1.5 How can procedural modelling rules and parameters be defined through sketch based interfaces for modelling? 309

8.1.6 Can a sketch based interface for modelling improve the workflow of methods for procedural modelling of buildings? 310
8.1.7 How can a sketch based interface for modelling enhance the expressiveness and direct control of procedural methods? 311
8.2 Future Work 313
8.2.1 Interactive Procedural Modelling with Layers 313
8.2.2 Integration of Layers in the Selection Action Paradigm 319
8.3 Conclusions 320

Appendix A Semantic Selections in Procedural Modelling 329

Appendix B Sketch Based Interaction for Procedural Modelling: Initial Questionnaire 337

Appendix C Sketch Based Interaction for Procedural Modelling: Questionnaire After Prototype Presentation 343

Appendix D Sketch Based Interaction for Procedural Modelling: Task 1 349

Appendix E Sketch Based Interaction for Procedural Modelling: Task2 353

Appendix F Sketch Based Interaction for Procedural Modelling: Task 3 357

Appendix G Sketch Based Interaction for Procedural Modelling: Questionnaire After Experimenting with the Prototype 359

References 380

Listing of figures

1.1 Iterative steps in Procedural Modelling Techniques. 6
1.2 From a procedurally generated façade (a) the user selects an architectural element (b), drags it to a new position and the model is regenerated (c). 12

2.1 Cost comparison between procedural and traditional techniques [Dan07] . 29
2.2 Midpoint displacement method for terrain generation [Mar96] 33
2.3 Example based terrain generation [Zho+07] 35
2.4 A terrain generation method capable of representing arches and caves [Pey+09] 35
2.5 Modeling road networks on a large scale terrain [BN08] 37
2.6 Left: maps with water, elevation and population density. Right: resulting road network. [PM01] 38
2.7 Example of execution: for a given node v_d in the existing network (middle) the algorithm searches for a node v_s that has similar topology within a radius and up to a rotation ϑ. New segments are added by copying segments and snapping to intersections [YS12]. 40
2.8 Application of L-Systems in Building Modelling [PM01]. 41
2.9 Execution of a Split Grammar. *START* is the axiom and the lower row represents the results. 43
2.10 Left: the Scope of a Shape. Right: a generated model with three primitives [Mül+06]. 43
2.11 Applying deformation to a straight façade (a) yields a different number of windows on the side parts (b). [Zmu+13] 45
2.12 Modelling a porch with the Node-based approach [Sil+13]. 47
2.13 From a simple example (right), complex cities can be generated (left) [MM08]. 48
2.14 Modelling large environments with procedural extrusions [KW11]. 49
2.15 Real-Time procedural city generation [Gre+03]. 50
2.16 Praça da República in Oporto, Portugal modelled with XL3D [Coe+07]. . 51
2.17 Modelling of the Boavista Roundabout in Oporto, Portugal [Sil10]. 52
2.18 First two steps in the execution. [Mül+07] 54
2.19 Left: a façade image. Right: the generated model [Mül+07]. 55

2.20 *F-Shade* resulting buildings [Hae+10]. 56
2.21 Each pixel is evaluated, tracking down split operations until a terminal shape(a and b). The method either samples from a texture (c) or instantiates a new geometric entity (d) [KBK13]. 57
2.22 Modelling process proposed by Finkenzeller [Fin08]. 58

3.1 The pipeline of sketch based interfaces for modelling [Ols+09]. 68
3.2 Filtering methods [Ols+09]. 70
3.3 Beautification process of a house-like sketch [Ols+09]. 70
3.4 The approach proposed by Igarashi pushes vertices away from the chordal axis [IMT99]. 76
3.5 Results of applying surficial augmentation to generate vein-like features on a hand model [Ols+05]. 78
3.6 From a single user sketch (left) the system extracts the building structure (centre) and provides a detailed textured result (right) [Che+08b]. 81
3.7 Example of a castle sketch being transformed into a 3D model [Ols+11]. . . 81

4.1 The Mikimoto Ginza building in Tokyo is an example of an irregular façade. 87
4.2 A volumetric shape with its scope. 91
4.3 A planar shape with its scope. 93
4.4 Relations between planar shapes, layers and 2D shapes. 95
4.5 Application of the rule in Listing 4.2 to a planar shape. 101
4.6 Application of the grammar in Listing 4.3 to the two layers defined. 105
4.7 Application of the grammar in Listing 4.4 to create an arched door. 108
4.8 Application of the rule in Listing 4.5 to a 2D shape causing it to be generated. 109
4.9 Clipping tree for the final model in Figure 4.5. 110
4.10 Incorrectly generated windows. 119
4.11 Correctly generated windows. 120
4.12 Priority queue for the procedural derivation. *NT* represents the normalisation token. 122

5.1 The user workflow in this system. 130
5.2 The interface presented to the user. 131
5.3 Results of applying the semantic query in Listing 5.3. 140
5.4 Results of applying the semantic query in Listing 5.4. 141
5.5 The descendants operation applied to a given shape in a shape tree. 143
5.6 The children operation applied to a given shape in a shape tree. 144
5.7 The siblings operation applied to a given shape in a shape tree. 145
5.8 An example of a selection action tree and its application sequence. 148

5.9 The application of a selection-action node is constrained to the shapes created by the nodes inside the red border. 151
5.10 Selection action tree illustrating the first step in the generation of a small residential house. 153
5.11 Selection action tree illustrating the second step in the generation of a small residential house. 154
5.12 Selection action tree illustrating the third step in the generation of a small residential house. 155
5.13 Selection action tree illustrating the fourth step in the generation of a small residential house. 156
5.15 Steps in the generation of a simple residential building. 157
5.16 Example of candidate evolution during selection inference. 165
5.17 The several layers that can be conceptualized as layers of procedural content. 167
5.18 Variations of a city block and the selection action trees responsible for such variation. 169
5.19 Scene layers responsible for creating the variation in Figure 5.18b. 170
5.20 NPC Waypoints evenly distributed in sidewalks and cross-walks (top) and the layer that created them (bottom). 171
5.21 A city block before (5.21a) destruction, the geometry affected within the destruction radius (5.21b) and two points during the simulation (5.21c and 5.21d). . . 173
5.22 It is possible to use this system to non procedural building hull (a) and create variations (b and c) using layers of procedural operations. 176

6.1 The user interface presented to the user. 184
6.2 A few steps in a short modelling session. 186
6.3 A stroke provided by a user where the intention was to draw a rectangle. . . 188
6.4 The speed and curvature metrics for each point in the stroke show in Figure 6.3. The values were scaled to the interval between 0 and 1 for illustration purposes. 190
6.5 Orthogonal distance (black) between points in the original user stroke (red) and the respective segmentation line (blue). 193
6.6 A line orthogonal to a planar shape may lose such property when projected onto the scene. 195
6.7 A vertical line on the screen when projected onto existing geometry may lose such property. 196
6.8 An example of a family of lines for an extrusion operation where the red, blue and green lines correspond to extrusion values of 1, 2 and 4. 197
6.9 Using guidelines and support lines to precisely split a face in half. 222

7.1 The example façade (Subsection 7.1.1), on the left, and its decomposition into layers, on the right. 230
7.2 A complex window created with layers. 233
7.3 Variations obtained by toggling on or off some layers in Listing 7.2. 233
7.4 Layers used to place the windows in the Mikimoto Ginza building. 236
7.5 Windows used for the Mikimoto building (Figure 7.6). 238
7.6 An approximation of the Mikimoto Ginza (Figure 4.1) created with layered shape grammars. 239
7.7 Changing the entrance size using the layered approach, 7.7a, leads to better results than with the split based approach, 7.7b. 241
7.8 Intentionally misaligned balconies in the example façade. 242
7.9 Detail of the ground windows in the split-based approach. 243
7.10 A modelling session with this system showing several selections (top row) and the operations applied (bottom row). 244
7.11 The selection-action tree for the example. 245
7.12 Possible user selections in the example in Figure 7.10. User selections are in yellow, results of applying one of the suggested queries. 245
7.13 Example of a building created with the selection-action paradigm. 247
7.14 Summary of some answers to the user evaluation questionnaire. Values are in a *Likert* scale where higher values mean a more positive user opinion for a parameter. 249
7.15 Examples of semantic selections over a simple building. 252
7.16 Building used to compare traditional methodologies with the one presented here. 253
7.17 Comparison between this methodology (a) and the one presented in [Pat12](b) to generate the model in Figure 7.16. 253
7.18 In the left column, for a given user selection (shown in yellow) the system generates a query that selects the shapes shown in yellow and green. For each of the examples, the centre column shows the evolution of the best score and the right column shows the evolution of the difference metric (δ) between the synthetic and the best candidate queries. 259
7.19 Score variation when 10%, 50% and 100% of the shapes are sampled. 262
7.20 User experience with traditional and procedural modelling tools. 265
7.21 Users' opinion about procedural modelling tools. 266
7.22 User's opinions regarding traditional (left) and procedural modelling tools (right) 267
7.23 Tools used in early stages of modelling. 269
7.24 Concept art in the modelling process. 270
7.25 Early opinions about the presented methodology. 272

7.26 Adequacy of the methodology to building modelling 273
7.27 The prototype's sketching interaction. 274
7.28 Comparison with current procedural modelling tools regarding their capabilities. 275
7.29 User opinion about model creation capabilities of the prototype when compared to other procedural modelling tools. 276
7.30 Opinions regarding the benefits the sketching interface brings for procedural modelling. 277
7.31 Early opinions about the prototype. 278
7.32 Initial state of the first task presented to the users. 279
7.33 Desired outcome of the first task presented to the users. 280
7.34 Initial state of the second task presented to the users. 281
7.35 Desired outcome of the second task presented to the users. 282
7.36 Desired outcome of the third task presented to the users. 284
7.37 Participants' opinions about the methodology after task completion. 286
7.38 Participants' opinions about the sketching interaction after task completion. 287
7.39 Participants' general opinions about the methodology when compared with other procedural modelling methods. 288
7.40 Participants' opinions about the model creation capabilities after task completion. 289
7.41 Participants' opinions about the sketching interface after task completion. . 290
7.42 Participants' opinions about the correctness of the suggested procedural operations and parameters. 291
7.43 Participants' opinions about the usage of support lines and guidelines. . . . 292
7.44 Participants' opinions about the prototype after task completion. 293
7.45 Participants' opinions about the prototype features after task completion. . 294

8.1 An example façade where the door is partially occluding a grid of windows. 315
8.2 Schematic decomposition into layers of the shape grammar in Listing 8.3. . 317

D.1 350
D.2 350

E.1 354
E.2 354

F.1 358

Acknowledgments

Firstly I would like to thank my supervisors Professor António Coelho and Professor António Augusto Sousa for all the continuous help, guidance and support from the moment I started this work until now.

I also send my deep thanks to Professor Gustavo Patow for his contagious enthusiasm and kind words which have always kept me motivated to improve my work. It has been a pleasure and I look forward to work again with such extraordinary Professors.

My thanks go to Professor Xavier Pueyo and Professor Gonzalo Besuievsky, everyone at the Visualization, Virtual Reality and Graphics Interaction research group and the *Universitat de Girona* for receiving me for two research periods. Their support and help was incredible and I will always remember dearly how I was welcomed.

This work was partially funded by the Portuguese government by FCT through the PhD scholarship SFRH/BD/92019/2012.

I pour my hear in gratitude for my family and friends who have always been there for me and have made me the person I am today. I am one lucky person to have each and every single one of you in my life.

The most special thanks goes to Raquel who has celebrated with me even the smallest

victories and lifted me up when things didn't look so bright. I would need to write as many pages as there are in this document to begin a proper *thank you*. ∞

1

Introduction

The use of urban environments has found its place in several applications ranging from computer games, cinema, urban planning or, even, virtual tourism. Users are increasingly demanding more detailed and convincing scenery. However, modelling such environments is a complex task that may require several *person-years* worth of labour to achieve.

Urban environments often show traces of historical, economical and cultural changes over time, which leads to an enormous variety of building styles and, even, different overall ambience that grants different cities unique feelings. With a quick look at any urban environment it is possible to identify several distinct elements, even within the same category, which must all be modelled correctly to convey credible results.

Nonetheless, there are also patterns in these environments which stem from urban planning rules and guidelines with which cities are built. City layouts, for instance, are conditioned by roads and it is common to see grid or radial patterns. Buildings also possess repeating elements in their structure, such as windows, doors or ornaments laid out according to some pattern. Without these patterns, city elements would have a chaotic aspect.

Procedural Modelling (PM) techniques exploit such patterns by encoding them into sets of rules with which the geometry and textures are generated, yielding a compact representation of their structure. Rules, in turn, can be parametrised with several attributes, changing the final aspect of the output. One can, for instance, define the number of floors a building has or, even, based on the date of construction, embed a building with a different architectural style. Coupled with randomness it is possible to generate vast amounts of models and, thus, easily create entire urban environments.

Typically, procedural rules are defined as a formal grammar, where symbols indicate which procedural rules are executed on the geometry, starting with an initial shape. The grammar derivation process, iteratively refines the geometry, adding more detail to the result. However, a great visual representation may only provide a part of such realism. Visually stunning graphics and highly detailed models only make the urban environments visually appealing but, unless the virtual world behaves as the real world, it may look unconvincing and break the user's immersion. To imbue more realism into the generated scenes, the procedural techniques can make use of semantic information to create urban environments that make sense.

Despite their immense power, one of the main disadvantages of using grammar based rules is that they can, typically, be unintuitive and daunting for the end user, despite their professional background. The most common development process in the procedural approach is to define the rules, execute the system, analyse the results and modify the rules in order to obtain a better 3D model. This method is not intuitive, lacks predictability and controllability, requires an in-depth knowledge of procedural modelling and resembles programming in an advanced language [Šta+10]. More *user-friendly* approaches are, thus, needed.

Node based approaches to procedural modelling [Pat12; Sil+13; Sil+15] of urban scenes have emerged which provide a more intuitive way of specifying procedural rules. However, the iterative cycle is still present and there is a disconnection between the definition of rules and the generated three-dimensional models. Nonetheless, it is a solid starting point on which to define more intuitive approaches to procedural modelling. In contrast, in shape grammars where related rules can be defined several lines apart, node based approaches provide the user with a visual representation that bridges the gap between rules.

The expressive power of drawing, on the other hand, is enormous and any person, even with average drawing skills, can communicate ideas quickly and easily. Moreover, sketching is often the starting point of any creative work and it is not different in the creation of urban environments. Environment designers in games, for instance, usually create concept drawings of the spaces where the game takes place. These, to be used, need to be modelled in a 3D modelling package, taking time and introducing a bottleneck in the content creation pipeline.

Sketch Based Interfaces for Modelling (SBIM), which have received a lot of research interest recently, try to close the gap between the concept idea and the final result by providing a drawing interface and creating the respective 3D models automatically, therefore, minimising the impact of such bottleneck.

Unfortunately, the most accepted methods for procedural modelling (i.e., shape grammars and node based approaches) of buildings lack the necessary expressiveness to be compatible with a sketch based interface for modelling. This is mainly due to their over-reliance on splitting operations which tend to create rectangular or cuboid shapes. More complex geometries must be modelled externally and imported to the procedural modelling workflow introducing a bottleneck. Increasing the expressiveness of procedural methods to become closer to the one in sketch based interfaces for modelling is also discussed in this thesis.

It is here hypothesized that a good articulation between procedural modelling systems and sketch based interfaces for modelling can empower content creators even further by allowing them to draw urban elements (e.g., buildings) and have procedural rules automatically defined which can, afterwards, be used to model variations.

1.1 Motivation

Procedural Modelling tools have, as target audience, creative professionals whose goals may be to create content for a number of different applications. A common desire, however, is to work with a tool that does not interrupt the creative process by forcing the user to dedicate more thought to how the system works than to the creation process.

The iterative nature of these methods (see Figure 1.1), where the user must first define the rules and parameters, set the parameter values, generate the geometry and analyse the results until a satisfactory output is achieved can be cumbersome to the artist. Furthermore, this process lacks visual feedback as it is not clear how a change in the rule base will affect the generated geometry.

This lack of intuitiveness, immediate feedback, predictability and controllability coupled with the fact that the user must have background knowledge on procedural modelling to

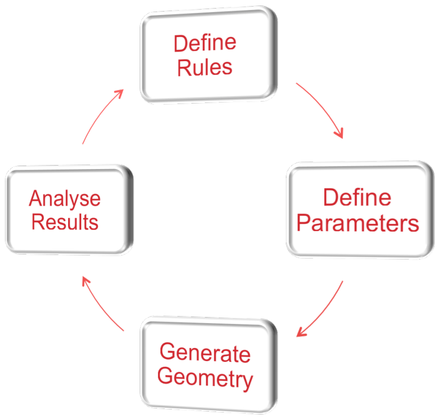


Figure 1.1: Iterative steps in Procedural Modelling Techniques.

be able to use these tools and that they often approach the generation of inherently visual artefacts in a text-based manner resembling a programming language motivates the development of this work.

1.2 Hypothesis and Research Questions

Procedural modelling methods provide, undoubtedly, powerful techniques for the automatic generation of three-dimensional content, requiring only that the user specifies a set of input rules. These should, preferably, be produced by the same person who envisions the end result, most typically a designer or artist. Hence, the procedural system should be as intuitive as possible while also trying not to disrupt the creative process.

This, however, is not necessarily the case as mentioned before. Despite that, recent work has made the procedural generation more user-friendly which has shown good results but the user must still manually produce the rules in some interactive fashion that steers the user away from text-based approaches.

Drawing, on the other hand, is a powerful communication and conceptualisation medium which, if integrated correctly with procedural modelling systems, may provide faster methods for the content generation of urban environments.

This raises the following set of research questions which this thesis aims to answer.

- How can procedural modelling methods be improved to provide enough expressiveness to be better paired with sketch interfaces?
- What is required from a procedural modelling method to improve the amount of direct control a user can have over the generated models?
- How can semantics present in procedural modelling rules be used in a sketch based procedural modelling system?
- How can a sketch based interface for modelling be used in the procedural modelling of buildings?
- How can procedural modelling rules and parameters be defined through sketch based interfaces for modelling?
- Can a sketch based interface for modelling improve the workflow of methods for procedural modelling of buildings?
- How can a sketch based interface for modelling enhance the expressiveness and direct control of procedural methods?

Having this in mind, the two following hypothesis are to be tested in this Thesis:

- Hypothesis 1: The generation of convincing urban models that are similar to their real-world counterparts can be achieved through the use of semantics explicitly or implicitly present in procedural models and rules.

- Hypothesis 2: It is possible to provide more intuitive interaction to specify or to obtain rules for the procedural modelling of buildings, reducing the iterative cycle of current procedural modelling methods, by using sketch based interfaces for modelling.

1.3 Objectives

The main objective of this thesis is to develop sketch based procedural modelling methodologies to generate buildings interactively, without disrupting the creative process. Consequently, they must be as intuitive as possible and should aim not to divert the user's attention to secondary tasks such as searching through the interface for a specific command.

The following, more concrete objectives are to be achieved throughout the remainder of this work:

- Study the current procedural modelling methodologies.
- Study the available sketch based interfaces for modelling techniques.
- Devise methods to increase the expressiveness of procedural modelling techniques.
- Create a method to improve the amount of direct control procedural modelling methods allow over generated models.
- Devise a sketch based interface for modelling aimed at the procedural modelling of buildings.
- Develop a methodology to aid in the interpretation of sketch gestures given the set of procedural rules.

- Analyse how sketch based interfaces for modelling can improve the procedural modelling workflow.

1.4 Related Publications

The work developed throughout this thesis has been documented and subject to the peer-review process in the publications listed in the remaining of this section.

- Jesus, Diego, António Coelho, and António Augusto Sousa. 2015. ”Towards Interactive Procedural Modelling of Buildings.” In Proceedings of the 31st Spring Conference on Computer Graphics, 173–76. Smolenice.
- Jesus, Diego, António Coelho, and António Augusto Sousa. 2016. ”Layered Shape Grammars for Procedural Modelling of Buildings.” The Visual Computer 32 (6): 933–43.
- Jesus, Diego, Gustavo Patow, António Coelho, and António Augusto Sousa. 2018. ”Generalized selections for direct control in procedural buildings” Computers & Graphics 72: 106-121

The first publication ”Towards Interactive Procedural Modelling of Buildings” [JCS15] presented the early contributions of the idea of using vectorial shapes and layers within faces of procedural models to increase its expressiveness. This was a short paper in the 31st *Spring Conference on Computer Graphics*.

Following the previous publication, the ”Layered Shape Grammars for Procedural Modelling of Buildings” [JCS16] paper presented the more mature and final contributions in the same idea which are also expressed in this thesis. This idea was presented at the *Computer Graphics International 2016* and published on the *The Visual Computer* journal.

The contributions towards a direct control in procedural modelling of buildings with user selection inference were presented in the third publication with the ”Generalized selections for direct control in procedural buildings” [Jes+18]. The contributions in this publication are also present in this thesis in Chapter 5.

An oral communication was also presented in the Doctoral Consortium at the 37th edition of Eurographics in 2016. In this communication, the ideas and approaches presented in this thesis were presented and discussed with members of the Computer Graphics field.

The following sections illustrate the open problems in sketch based procedural modelling of buildings and how the contributions within this thesis aim at their solution.

1.5 Problems in Sketch Based Procedural Modelling of Buildings

Despite the immense capabilities of procedural modelling methodologies there are some drawbacks. In this section some of these are highlighted, which were chosen as the ones most relevant for the development of a sketch based procedural modelling system targeted at buildings.

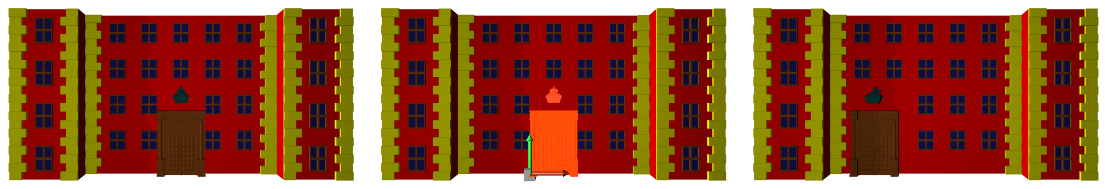
1.5.1 Lack of Direct Control

A severe problem in most procedural modelling systems is the fact that these do not offer the user the possibility of directly manipulating the created buildings or geometries.

In fact, often the only way users have to change the building's aspect is to modify the rule base, by adding or removing rules, or by changing the parameters in such rules. In the best of scenarios, the rules are encoded in a graph or visual language representation [Pat12; BP12; Sil+13; Sil+15] but more often than not, the user must modify a text-based shape grammar [PM01; Won+03; Mül+06; KPK10; SM15; JCS16]. In any case there is a separation between what the user can modify and manipulate and what is, in fact, generated.

The users are, often, limited to defining rules and tuning the respective parameters, after which they must regenerate the geometry to analyse the results. This *define-generate-analyse* cycle is a currently open problem in procedural modelling and is responsible for the lack of direct control over the created models (e.g., dragging elements as in Figure 1.2) and provides very little visual feedback.

This cycle directly affects creativity as the user is often forced to spend more time on modifying the rules searching for the exact configuration that generates the geometry that fits the user intention, than on creating new rules that add more detail to the building models. In other words, more effort is put on tuning the rules and parameters than on improving the models towards more detailed and realistic results.



(a) (b) (c)

Figure 1.2: From a procedurally generated façade (a) the user selects an architectural element (b), drags it to a new position and the model is regenerated (c).

1.5.2 Restrictions of Split-Based Methods

The currently most accepted methodologies for procedural modelling (i.e., those based on shape grammars or graphs) of urban scenes heavily rely on using split operations to refine the geometry. Split operations consist of splitting a shape one or more times along one or more planes defined within the shape's bounding box (also referred to as scope) and orthogonal to one of its axis.

Replacement methods, where in each operation a set of shapes is replaced by a new set of shapes (e.g., shape grammars), are the most notorious example of systems that use this split based approach and often provide (at least) two types of split operations: the regular split and the repeat split. The first splits geometry along a bounding box axis given a set of sizes, generating a new shape for each size. The second repeatedly splits geometries along a bounding box axis given a single size, generating new shapes of equal dimensions.

In typical scenarios, users chain split operations to create several new partitions that are detailed individually, often alternating the split axis. This, in turn, tends to generate rectangular or cuboid shapes, limiting the expressiveness of procedural modelling methods. Even slightly more complex shapes are nearly impossible to create with split base operations. For instance, a simple arched window cannot be created with this paradigm and, therefore, must be externally modelled and imported to the generation. In turn, this consumes more time and may not be easily integrated with the rest of the model as there may be geometry misalignment or compatibility issues.

Moreover, forcing users to focus on the exact order of splitting operations can be rather daunting and may disrupt the creative process by forcing the user to focus on how to describe a building instead of how a building should look.

As stated by Zhang et al. [Zha+13], users don't typically perceive buildings through splitting lines. Humans, when looking at a building façade, often perceive regular patterns of similar elements (e.g., a grid of windows) that may be interrupted by other elements (e.g., an entrance). A representation that is closer to how users perceive the façade of a building can lead to improved creativity. Zhang et al. [Zha+13] tackle this issue by decomposing a façade into layers of elements.

Nonetheless, split-based operations are a powerful way to represent complex patterns in architectural models and do provide an efficient way to partition and place architectural elements within façades. Therefore, this flexibility must not be neglected and should, instead, be complemented with other methods that enhance its expressiveness.

1.5.3 Semantic Ambiguity in User Selections

To enable direct control over a procedural model a user must be able to interact with the generated model. A typical paradigm in 3D modelling is to allow the user to select (via point and click) the set of geometry elements and then perform an operation on such elements.

This scheme is simple and intuitive for the end user who must simply select the set of elements to manipulate. Furthermore, the selection is exact and unambiguous as the user precisely chooses which elements are included in the selection. Systems often provide means to add or remove elements from the selection set.

On the other hand, selection in procedural modelling systems is highly ambiguous due to the data amplification nature of procedural modelling systems. A given selection of shapes in a procedural model may reflect several possible user intentions. In other words, the selection set may be a subset of what the user intended to select.

Moreover, this selection is neither exact nor unambiguous. Even if the user selects all shapes in a given model matching a given criteria, a simple modification in the rule set might invalidate such selection. For instance, if a user selected all floors in a building, simply increasing the number of floors is enough to invalidate the intention of the previous selection. A naive solution is to keep the first few floors selected. However, this does not scale to more complex situations.

Consider a rule that subdivides a façade into its floors. Selecting one of the generated floor shapes may correspond to the user's intent of selecting that exact floor or, on the other hand, selecting all floors. If the user, instead, selects the first and third floor, the intent may, then, be to select all odd floors.

This brief example only considers a single façade and its floors, but it is already possible to see that the amount of possible selections and user intents rapidly increases with the number of shapes.

Considering four façades, each facing a cardinal point, and it is now possible to take the direction a given floor is facing into account in the user selection ambiguity. In other words, the user may want to select floors only facing the north direction, odd floors in all directions among other possible combinations.

A new level of indirection in the user selection in procedural models is required which should take into account not only the user selection set but also capture the semantics behind the user intention.

1.5.4 Procedural Rule Inference and Parameter Estimation from Sketches

Typical procedural modelling methods rely on the user to define rules and parameters. In a sketch based procedural modelling system rules must also be obtained, albeit using a different user interface paradigm.

The purpose of integrating procedural modelling methods with sketch based user interfaces is to allow a more direct control and expressiveness over such systems while, at the same time, diminishing the interruptions during the creative process. It should also minimise the gap between concept drawings and the final 3D models.

However, sketches are extremely ambiguous which makes the recognition process complex. Therefore, a sketch provided by a user could potentially have several meanings. Considering noise during the capture phase and hand jitters makes the inference even more difficult.

A sketch based procedural modelling system should be able to infer both the intended operations and its parameters. Moreover, the inferred operations and parameters must be converted into rules in the underlying procedural mechanism (i.e., shape grammars or nodes) and be correctly associated with other rules in the current rule base. In other words, the new rules should be created with the proper symbols in a shape grammar or linked to other nodes in a node-based system. Failing to correctly produce the rules and their associations may lead to an erroneous output or no output at all.

Urban spaces, in general, and buildings in specific contain many patterns such as repetitions and symmetries which procedural rules often exploit. As such, a sketch based system should provide the means to correctly and easily create these patterns. On the other hand, such systems can take advantage of a limited set of procedural operations to aid in the sketch recognition process.

1.6 Solution Overview

To overcome the previously mentioned issues three distinct methodologies were developed, each aiming at solving one or more of such problems. When combined, these provide a sketch based procedural modelling system.

This section intends to briefly explain each of these contributions, highlighting the ways in which they help to bridge the gap between procedural modelling and sketch interfaces.

1.6.1 Layer-Based Procedural Modelling

To tackle the limitations in split based methods, especially in façade descriptions, the developed solution was to define layers in procedural models to which different sets of rules can be applied, creating different architectural elements. Such layers are then merged into a flat representation while dealing with geometry conflicts.

This allows for a more structured representation of procedural rules since content generation can be grouped into semantically meaningful layers. For example the user can group rules that generate a regular grid of windows into one layer while deferring the generation of ornaments to another layer. Furthermore, this approach easily provides the generation of more variations by turning on or off the execution of some layers, reordering the layers or swapping the rules in one layer with a new set of rules, providing a non-linear modelling workflow. All these changes are contained within a single layer while the structure and resulting architecture of other layers remain equal.

Not only does this provide a more organized way of creating rules but also suits the human perception of buildings. As noted by Zhang et al. [Zha+13] humans tend to perceive façade layouts not through split lines but as a collection of regular patterns that may be interrupted by other elements or groups of elements.

Moreover, during the development of this thesis, it was found that the current level of expressiveness provided by procedural modelling systems was not closely matched to the expressiveness sketch based systems naturally provide. While it is easy for a user to sketch arched windows, it is not trivial for split based methods to generate such elements without importing externally modelled assets.

Layer based procedural modelling also takes a step forward in solving this problem by allowing the specification of irregular shapes (e.g., arches) within layers. In the proposed methodology, these are defined vectorially as closed 2D paths forming the outline of the desired new shape.

Defining several layers within a procedural model inevitably creates geometry conflicts between shapes in different layers. Shapes contained in front layers may overlap shapes in background layers. This is, actually, a desired behaviour as it allows users to define groups of shapes that interrupt regular patterns in other layers, bringing the representation closer to human perception as approached by Zhang et al. [Zha+13].

Nonetheless, merging all layers is required to produce a final output. The solution found was to use Boolean operations (intersection, difference and union) to combine all shapes considering their relative depth within the set of layers. Some fine-grained control is also given to the user who can select if some shapes should be discarded in case of being partially or totally occluded by other shapes.

1.6.2 Semantic Selection of Sets of Shapes

As previously mentioned, user selections over procedural models can be extremely ambiguous due to the data amplification nature of such methods. This thesis describes a solution for this problem that shifts selections from a simple set of selected shapes to semantic queries that describe the user intended selection.

The description is based on simple properties of individual shapes, complex filters and hierarchical relations between shapes in a shape tree. The internal properties of shapes are the symbol (in case of shape grammars) and tags, while more complex conditions such as parity of index or the direction a shape is facing is encoded into filters. Semantic queries can also take advantage of the fact that shape trees normally provide a hierarchical structure of a building. Therefore, it is possible to create semantic queries that select shapes that show some sort of relation (in this system these are the children, descendants or sibling relations) between one another.

Moreover, semantic queries are naturally equipped to handle variations in the rule set as they are descriptions and not a fixed set of shapes. In the case of selection sets, variations to the rule set may invalidate part of the selection if some selected shapes cease to exist or not expand properly to fit new models if new similar shapes are added. On the other hand, semantic queries reduce this issue by describing how selections should be constructed.

The intention behind user selections can also be captured into one of many semantic queries. If, for example, the user selects sets of alternate floors it should be possible to construct a semantic query that represents such intention. However, manually creating such semantic query may be too cumbersome for the user as it would require the specification of each condition and parameter in the query. A better approach is to allow the user to select the shapes within an interactive application and place the burden of generating queries in the system itself.

An approach to achieve such results using genetic algorithms is also described in this thesis. From a set of user selected shapes a few candidate semantic queries are generated which are evaluated according to a fitness function. Mutations and crossing are then applied to the best candidates and the process is repeated for a given number of iterations. The best candidate queries generated throughout all iterations are kept and presented to the user who then selects the one that best matches the intent. The goal is to map the selected shapes in an interactive application to a given semantic query where the system has the greater burden of generating a set of feasible semantic queries and the user has the final decision over which is more suitable for a given domain.

1.6.3 Procedural Rule Inference

The end goal of this thesis is to provide a methodology that allows users to specify procedural rules by sketching directly over procedural models. Therefore, to generate procedural rules from user provided strokes these should be interpreted and correctly inserted into the rule base. A mechanism to infer procedural rules from user strokes has been developed and is thoroughly explained in this thesis.

However, as previously described, sketch based interfaces for modelling are ripe with ambiguity and this is also reflected in the developed methodology. Firstly, from a viewpoint, a set of user strokes can form an infinite number of configurations in 3D space and a mapping between the sketch and the correct configuration must be found. In this thesis such mapping is achieved by using a modified version of the system defined by Schmidt et al. [Sch+09] as it is a good fit for architectural drawings. This system uses the analytical drawing technique developed and widely used by designers [Sch+09] and relies on previously specified strokes and constraints to infer new strokes.

However, where their system provides candidate line segments by creating combinations of starting positions, directions and distance (effectively defining a line segment), the system described in this thesis employs knowledge of the available procedural operations (e.g., extrusions and splits) to generate candidate segments. For example, for an extrusion it generates candidate lines starting at a shape's vertices and are orthogonal to the face while, for a split operation, it generates lines based on the intersection of the face with a split plane. Each candidate line is projected onto the viewing plane and compared with the user stroke and the best match is inserted into the configuration. Moreover, this methodology uses shape edges to serve as a basis for new user strokes.

A second level of ambiguity arises from the fact that a given configuration of lines in a 3D space can be mapped to different procedural operations. The system defined in this thesis analyses the set of 3D line segments by running a heuristic for each procedural operation. The output of such heuristic is a candidate procedural operation with a set of estimated parameter values. Candidate procedural operations are, then, fed into a gradient descent method which will modify the parameter values approximating the edges of the resulting geometries to the set of 3D line segments. The gradient descent uses a distance based function. All operations are then ranked according to this function and presented to the user who, having knowledge of the intended result, is better equipped to make the final decision.

The user first creates a semantic selection (either manually or using the genetic algorithm) specifying the shapes to which subsequent modifications will be applied. From these selected shapes the user chooses one over which the sketching will be performed and, after the user chooses the adequate procedural operation, the system inserts it into the rule base such that it is applied to the shapes selected by the semantic query. The geometry is regenerated and the user can repeat the process.

This mechanism is tightly coupled with the semantic selection system, as it uses it to properly insert new rules in the rule base.

1.7 Document Structure

After the thesis Introduction (Chapter 1) where the motivation, research questions, hypothesis and an overall view of the open problems and the proposed solutions, this document presents the State of the Art on procedural modelling (Chapter 2) and sketch based interfaces for modelling (Section 3).

Posteriorly, Chapter 4 presents the contributions that were made towards extending shape grammars with layers and vectorially defined shapes. This chapter also discusses how such extension increases the expressiveness of procedural modelling of buildings.

Chapter 5 introduces semantic queries and a selection action paradigm for procedural modelling of buildings. Such contribution aims at solving the selection problem within procedural models and shows how queries can be used as flexible way of keeping selections valid throughout rule changes. This chapter presents the selection inference system that allows queries that capture the user intention to be generated from a few user selected shapes.

With the previous contributions explained it is then possible to introduce the contributions towards a sketch based procedural modelling system. These are discussed in Chapter 6 alongside with how the previous techniques and the already extensively studied contributions in the SBIM field are assembled together to form a sketch based procedural modelling system.

The evaluation and results of the entire system are shown in Chapter 7 demonstrating how the conjunction of systems presented in the previous chapters impacts the procedural modelling workflow.

Finally, the Conclusions and Future Work chapter (Chapter 8) describes future avenues of research based on the presented contributions and draws final remarks on the entire work.

2 Procedural Modelling

As in every scientific research, it is important to fully understand what has been done to this point to gain better insight of the subject of the proposed thesis, analysing the applications, requirements and open problems on the research field of procedural modelling and sketch based interfaces for modelling.

This chapter presents the main contributions in procedural modelling approaches, starting with a definition of procedural modelling, advantages and disadvantages of their methods. Then, techniques more specific to the generation of urban spaces will be presented.

Procedural modelling is a term that groups numerous techniques from Computer Graphics that are used in the automatic generation of three-dimensional models or textures, given a set of rules or procedures. Examples of this are L-Systems and Fractals, since they employ an algorithm in the content generation.

This subsection presents procedural modelling definitions, according to several authors, along its advantages and disadvantages.

2.1 Definition

Several definitions for procedural modelling are easily found, some being more strict than others. This way, procedural contents can be considered as a subset of contents created by executing an algorithm considering a set of rules and randomness which can result in a large set of variations, contrasting with fully manually created contents [Sof07]. This definition, however, categorises the generated elements into two different extremes: the content is either completely procedural or completely manual.

On the other hand, a different definition chooses to introduce a continuous scale between these poles. In this case, there is no longer strictly procedural modelling nor strictly manual modelling. There are, however, values in between. Such definition allows measuring how much of an element was created using procedural techniques, i.e., how much did algorithmic methods contribute for its creation. This categorisation then becomes a function of the amount of manual input data given to a certain method. Higher amounts mean less procedural content. At the extremes of such function are the total absence of manual input (completely procedural methods) and the complete manual definition of the content (completely manual methods) [Hal08].

Procedural modelling can, thus, be defined as the programmatic generation of content using a random or pseudo-random process resulting in an unpredictable range of content [Wik10]. Although associated with randomness, procedural methods can be able to generate the same results in different executions, since computers can only generate pseudo-random numbers from a seed. Providing the random generator with the same seed results in the same output throughout executions.

As in every paradigm, procedural modelling has its advantages and disadvantages, which will be discussed in the following.

2.2 Advantages

Procedural modelling has, among others, the following benefits:

- Data Compression: Since procedural techniques are capable of generating large amounts of data (either random or deterministic) from a reduced or, even, empty set of inputs they allow a large data compression rate. This is advantageous both in the storage and transmission of data through a network [Hal08]. Several examples can be found, mainly in the game industry, that make use of such property. An example is the game *.kkrieger* [the04], which only occupies 100 KB.

- User Generated Content: It is possible that the users of a modelling system lack the time or skill to produce high quality content. However, if a procedural modelling system that only requires a small set of intuitive input is available, then it becomes easy and fast to use [Hal08]. A classic example are the character generators in games.

- Programmer Generated Content: Procedural modelling tools allow teams with a few artists and designers to produce high quality content, by developing systems capable of generating an unlimited range of contents from a set of parametrizable or random input data [Hal08].

- Higher Productivity: A procedural system is based on the idea of automating the creation process, decreasing the workload of the designers team. This results in the need of fewer inputs for the same output content, improving productivity [Hal08].

- Cost Reduction: Although procedural modelling systems involve large amounts of development time and costs, they are able to produce a large amount of contents at reduced costs, as seen in Figure 2.1. These contents are also of a more *Agile* nature than the ones manually produced. They can, this way, be inserted more easily in an iterative process allowing faster prototyping [Dan07].

2.3 Disadvantages

Although procedural modelling techniques have several advantages, it is not always feasible or, even, possible their employment. In the following some disadvantages of such systems are presented:

- Implementation Complexity: Since these methods present a large algorithmic component, their implementation requires a highly qualified development team. On the

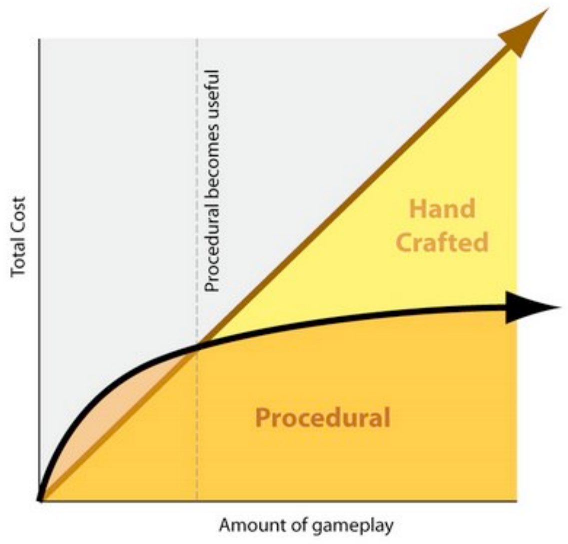


Figure 2.1: Cost comparison between procedural and traditional techniques [Dan07]

other hand, because it is expected that the results have good visual quality it is necessary that the people involved in the content generation have some artistic knowledge [Dan07; Rem08].

- Lack of expressiveness: Assuming that a system produces more expressive results when it accepts more input data, it is easy to conclude that a highly procedural system (and, thus, one that is lowly parametrizable) generates content with a low level of expressiveness [Hal08]. Since systems of such nature are, typically, based on rules the results are often repetitive and little imaginative. On the other hand, drawing is a highly expressive medium and combining procedural techniques with sketch based interfaces for modelling may enhance the expressiveness of procedural modelling methods. This is one of the research questions to be answered in the proposed thesis.

- High waiting times: Procedural modelling systems usually require a more intense processing. For this reason, using such tools for content creation in run time may translate into a higher waiting time, specially in less powerful machines, something the user may not be willing to spend [Sof07].

- Loss of direct control: Given that procedural techniques take some advantage of randomness in their execution, some content parts will be generated without human intervention. Such fact is not necessarily visible until the user wishes to create some more specific content. The more specific the desired content, the more human intervention will be necessary [You09]. This is also related with the lack of expressiveness and the integration with sketching based interfaces for modelling may help to increase the direct control in procedural methods.

- Content testing: Procedural methods can generate content in a large scale from a reduced set of input data and generation rules. However, small changes in the input may lead to errors that, due to the dimension of the result set may be hard to detect [Rem08].

2.4 Procedural Modelling of Virtual Urban Environments

Urban environments are large, highly detailed spaces which have a great importance in a number of applications such as urban planning, virtual tourism, cinema and games. However, given their level of detail, modelling such environments may require several thousand of *person-years* and may involve large costs. On the other hand, cities are man-made spaces and their construction process often follows some sort of pattern.

Urban procedural modelling techniques exploit such patterns by incorporating them in the form of rules and algorithms. Their execution can deliver a great number of new models in a short time when compared to traditional modelling techniques, thus reducing the associated costs and bringing the previously mentioned advantages to this field. Obviously, the trade-off is that they also bring the disadvantages.

These techniques can be further divided into several categories, which will be discussed in the following subsections. These categories can be executed in sequence when creating an entire virtual city as in the seminal work of Parish et al. [PM01]. However, Ridorsa and Patow [RP10] state that, in practice, this model is insufficient when including a broader city model that includes other features like parks, landmarks, streets and others. Therefore, they introduced the skylineEngine that presents a non-linear pipeline using a Directed Acyclic Graph to define the execution order.

- Terrain Modelling: Terrain serves as a solid foundation for every landscape including cities. In the context of Urban procedural modelling, it is important that their virtual representation is as realistic as possible to produce visually appealing spaces.
- Road Network Modelling: Besides providing a means of transportation inside a city, road networks also divide the space into blocks and lots, which serve as a starting point in the creation of building's geometry.
- Building Modelling: This stage employs procedural modelling techniques to create the geometry of the architecture representing the several buildings. Usually, for a full city generation, the buildings are placed in lots created by the previous step.
- Façade Modelling: Normally, a building has specific characteristics that allow us to distinguish it from the rest, such as the layout of elements in their façade. This stage takes on such responsibility.
- Modelling Other Content: An urban environment is not made up of only streets and buildings. Several other elements need to be created and correctly placed to create a convincing virtual city.

2.4.1 Terrain Modelling

Every urban environment requires a solid surface on which to lay its foundations. Such surface grants a base where objects can be positioned and provides a means where habitants can move. It is, then, trivial to conclude that, in urban modelling, such surface plays an important role.

Procedurally creating artificial terrains has already been approached by several authors and commercial tools [Zho+07; Dis09], obtaining interesting results. Some of the proposed algorithms employ noise generation to produce terrains similar to those found in nature [PJ85], while others simulate natural physical processes such as water and wind erosion [MKM89; KMN88; Mar+97; Šta+08].

Fractal-based modelling was pioneered in the work of Mandelbrot [Man83] and, from there, it originated a great number of stochastic subdivision techniques. Fournier et al. [FFC82] proposed the well-known midpoint displacement technique to create fractal terrains. Afterwards Miller et al. described several variants [Mil86]. Given a rectangular grid, the algorithm assigns random height values to its corners. It then proceeds to iteratively subdivide the grid and each midpoint's value is calculated averaging the corners of the square or diamond that surrounds it plus a random offset (see Figure 2.2). This offset is decreased in each iteration, based on a parameter defining the intended terrain roughness.

Another example of fractal terrain generation is the work of Ken Perlin [PJ85] which

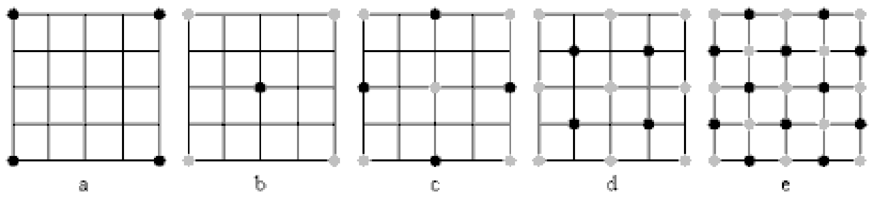


Figure 2.2: Midpoint displacement method for terrain generation [Mar96]

generates noise by sampling and interpolating points in a grid of random vectors. It is possible to add several scaled levels of noise to each point resulting in natural, mountainous-like structures.

Simulating processes found in nature is another alternative to terrain generation. Thermal erosion reduces the sharp changes in elevation by iteratively transporting materials from higher to lower terrain points until a certain criterion that measures material stability is found. Fluvial erosion, on the other hand, transfers material based on the local slope of the terrain [Sme+09]. These methods can, typically, be used as a refinement step, after a coarse height-map has been obtained. Musgrave [MKM89; Mus93] has worked on both types of erosion and Olsen et al. [Ols04] presented some optimisations that enabled real-time terrain generation. Benes and Forsbach [BF01] work presented a terrain structure based on stacked slices of material, each with an associated height and material parameters, such as density. This structure provided more realistic terrain erosion algorithms.

Although adding much to the realism of the results, these algorithms are also known for being time-consuming, often having to run for hundreds or thousands of iterations

[Sme+09]. Recent work has been put into porting the methods to the GPU, which has resulted in interactive erosion algorithms [Šta+08; ASA07].

Yet another alternative approach to terrain modelling is using constraint-based techniques, user sketches and sample terrain. These methods tend to be simpler to use than those presented so far, since the previous ones often rely heavily on using parameters (sometimes unintuitive) and their results are fairly random. Users are, typically, only able to control the outcome on a global basis.

Stachniak and Stuerzlinger [SS05] propose a method that uses constraints (in the form of image masks) in the terrain generation process. The algorithm then proceeds to search for an acceptable set of deformations operations to apply to the terrain to obtain a result that conforms to the constraints specified. Zhou et al. [Zho+07] described a technique that takes an example terrain model as input and the user is allowed to sketch desired features such as mountain ridges. Features are extracted from the example and matched against the sketch and seamed together to form the final result. An example can be seen in Figure 2.3. De Carpentier and Bidarra [CB09] introduced the concept of procedural brushes where the users *paint* the height-map directly in 3D with simple terrain raising brushes and the GPU introduces several types of noise generated in real-time.

A known limitation of the height-field based methods is that they embody the altitude as a function of the x and y coordinates and, as such, can only contain one value for each loca-

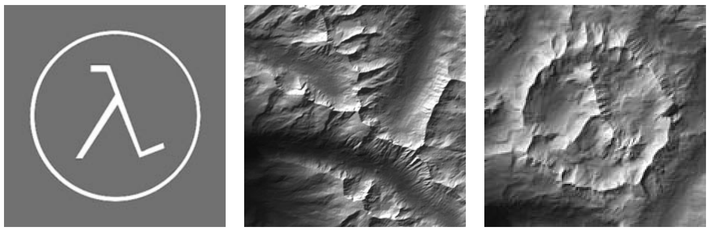

(a) User sketch (b) Terrain Example (c) Final Result

Figure 2.3: Example based terrain generation [Zho+07]

tion. Caves, arches or overhangs cannot be modelled using these systems. Gamito and Musgrave [GK01] proposed a terrain warping system resulting in regular artificial overhangs. One method that provides interesting results, visible in Figure 2.4 is the one presented by Peytavie et al. [Pey+09]. Here, a more elaborate structure with support for different layers of materials is able to model rocks, arches, overhangs and caves.

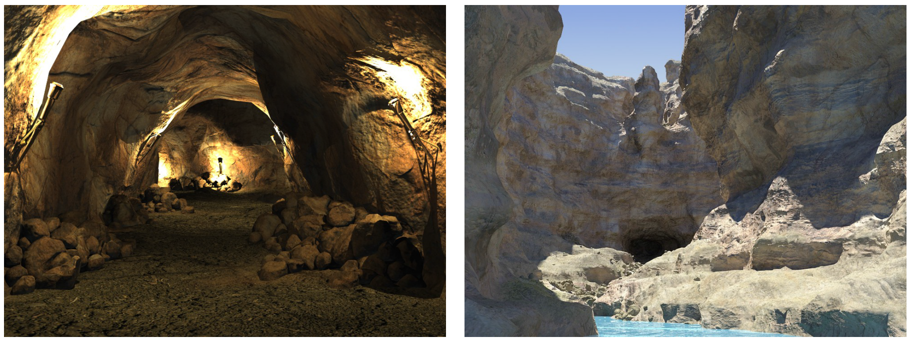


Figure 2.4: A terrain generation method capable of representing arches and caves [Pey+09]

More recently, Santamaria-Ibirika et al. [San+14], proposed a new approach to procedural volumetric terrains, which generates customisable terrains with layered materials and other features such as mineral veins, underground caves, material mixtures and underground material flow. The method allows the user to specify the terrain's characteristics by using a set of intuitive parameters.

Despite being able to synthesise interesting terrains, the fact that the methods presented thus far rely on some amount of randomness does not allow the reproduction of existing terrains. To overcome this challenge, it is usually preferable to use data from the real world such as photographs or Geographic Information Systems (GIS) which provide terrain information in a Digital Elevation Model (DEM). In this case, the surface is represented as grid of equally sized cells representing terrain areas. The fact that cells are laid out regularly allows performing spatial queries such as which zones are visible from a given point or the ones that have sharper slopes.

2.4.2 Road Network Modelling

A city's road network provides a means of transportation between its different areas and, also, divides the space into blocks and lots, defining the building's distribution. It serves as an organisational entity of the city space and is responsible for the overall aspect of an urban model. Thus, it is important that the user of a procedural modelling system can have the most control over this step's execution.

One of the simplest techniques for procedurally generating a network of streets is to create a dense square grid [Gre+03]. To make it less repetitive, some displacement noise can be added to the vertices. However, the results still lack the realism.

Bruneton and Neyret [BN08] proposed a system enabling detailed road modelling in large environments, integrating information of rivers, roads and bridges with the digital elevation model of an irregular terrain. Taking into consideration that the method operates on scenes of large scale the authors implemented the system with a dynamic level of detail based on the distance of an element to the camera. Figure 2.5 shows an example output of this work. Although producing interesting and visually appealing results, this method is not well suited for urban areas.

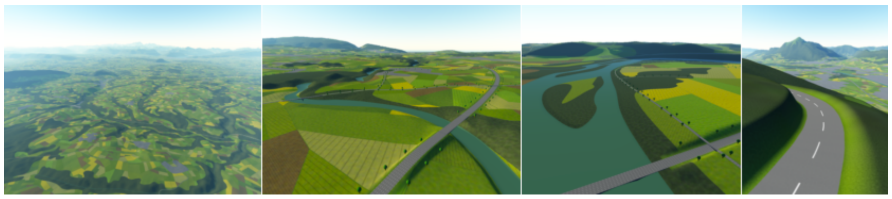


Figure 2.5: Modeling road networks on a large scale terrain [BN08]

The use of template based modelling as proposed by Sun et al. [Sun+02] provides more elaborate results. This approach is based on the premise that there are several frequent patterns in real road networks and the authors aimed at their reconstruction. There is one template for each pattern: a population-based template (implemented as the Voronoi diagram of population centres), a raster and radial template, or a mixed template. Firstly, main roads (highways) are generated using these templates and are, then, curved to avoid large elevation gradients. Afterwards, the spaces between these roads are filled with smaller, secondary streets in a grid pattern.

L-Systems [Lin68] have also been used to procedurally create road networks. Parish and Muller [PM01] used extended L-Systems for this purpose. Their implementation was driven by the goal of connecting areas with high population density while creating roads following a specified pattern (e.g., raster or radial). The system was extended with generation rules that tended to connect newly proposed road segments to existing intersections and to check for road validity with respect to impassable areas (for instance, water) and elevation constraints. The inputs are maps with geographical information like those seen in Figure 2.6. In this work, the authors have also presented the urban procedural modelling tool CityEngine.

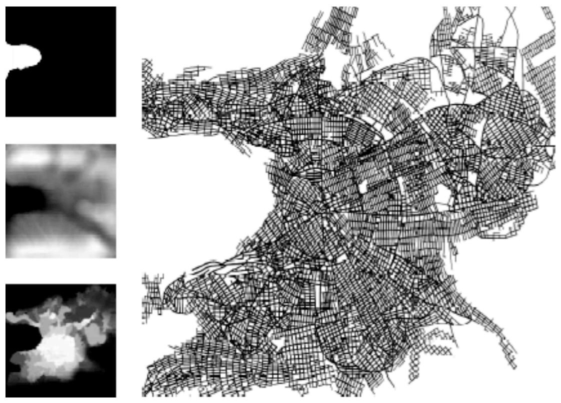


Figure 2.6: Left: maps with water, elevation and population density. Right: resulting road network. [PM01]

Kelly and McCabe [KM07] introduced the interactive city editor tool CityGen on which the user can specify the main street layout by placing nodes in the 3D terrain. The spaces in between are filled with one of three road patterns: Manhattan-style grids, industrial grown roads with dead-ends and organic roads.

Chen et al. [Che+08a] propose an interactive road generation tool using tensor fields.

Main streets are generated by tracing streamlines starting at seed points along the major eigenvector direction until some stopping condition is met. Then, some new seed points are placed along this curve and new, secondary streets are traced in the perpendicular direction (along the minor eigenvector). Users are allowed to influence the tensor field in order to produce more plausible results.

This method, however, is fairly restricted in terms of user interaction. Although it is possible to edit the location of the seed points, the resulting urban layout may not be valid, i.e., the road network may contain self intersections or may intersect buildings. Also, whenever the need to regenerate the tensor field arises, all user editions are lost.

To overcome this challenge, Lipp et al. [Lip+11] proposed a method allowing the user to perform persistent editions that are in the valid domain. A valid urban layout is one where all intersection of roads form a valid road junction and building parcels don't intersect each other or nearby roads. The authors defined user operations that, together with graph cuts, maintain such validity.

Example-based methods have also been used to procedurally create roads. In their work [YS12], Yu and Steed presented a technique to automatically synthesise networks that are perceptually similar to a given example of road network, starting from an initial node or by expanding an existing network. For every unfinished node, the algorithm searches for candidate matches in the example that have a similar node topology. Furthermore, the method can also blend different styles in an intuitive way. An example of the algorithm's execution can be seen in Figure 2.7.

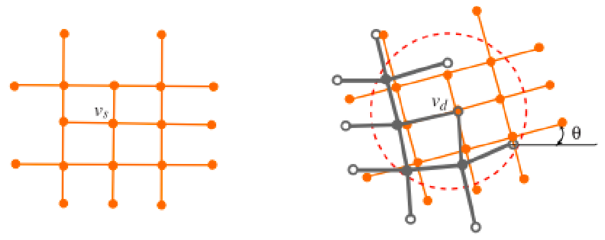


Figure 2.7: Example of execution: for a given node v_d in the existing network (middle) the algorithm searches for a node v_s that has similar topology within a radius and up to a rotation ϑ. New segments are added by copying segments and snapping to intersections [YS12].

Beneš et al. [BWK14] introduce a method to simulate the development of a city's road network taking into account neighbouring cities. The resulting cities are formed by allowing several smaller settlements to grow together and to form a rich road structure. This method also focus on reducing the amount of user interaction to a minimal well-defined set of user parameters.

A completely different approach was taken by Lechner et al. [LWW03] where the authors used agents to grow roads. The city is divided into areas like residential, commercial or industrial and special areas like government buildings or squares. They define two agents: the *extender* and the *connector*. The extender searches for unconnected areas close to the road network and finds the best path to connect both. The connector starts in a random location in the network and, randomly, picks another spot within a given radius. It then calculates the shortest path and, if the travel time is considered to be too long, creates a direct road connection.

2.4.3 Building Modelling

The next step in the procedural modelling of urban spaces is, typically, the generation of buildings themselves. The blocks encircled by the roads created in the previous stage, are subdivided into smaller areas making up the building lots. These, in turn, serve as an initial shape (an axiom) for the processes discussed in this section.

A first approach to this problem was presented in [PM01] where the authors proposed using L-Systems [Lin68] in the modelling of buildings. The execution of such systems consists in the successive application of transformations on a given axiom until the desired geometry is achieved. The set of transformations include translations, extrusions or rotations for instance. Figure 2.8 shows the initial steps of this process.

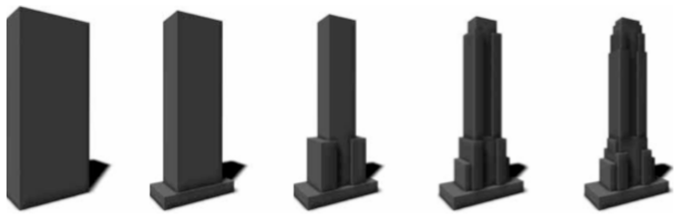


Figure 2.8: Application of L-Systems in Building Modelling [PM01].

Although capable of generating a wide variety of building models, Wonka and Muller [Won+03; Mül+06] consider that L-systems are not well suited for such task, since they are oriented towards modelling growth processes in an open space (i.e., modelling plants).

Buildings, however, have stricter spatial restrictions [Won+03; Mül+06].

In his work [Won+03], Wonka introduced a new grammar targeted at solving this problem, which was called *Split Grammar*. In this method, the process starts with a simple, non-terminal shape to which several splits are applied, resulting in smaller shapes until individual elements (such as windows or other ornaments) are reached. The procedure is illustrated in Figure 2.9.

A large database of rules is available to this method which are applied in the modelling of buildings of different architectural styles making the need to develop a set of rules for each building unnecessary. However, this presents a new problem: in each grammar derivation step it may be possible to apply a wide number of rules resulting in buildings with a chaotic aspect [Won+03]. This problem has more impact with a larger database. The authors proposed another grammar (the *Control Grammar*) with the goal of controlling the elements' spatial disposition in an orderly manner.

To generate a complex building, this approach would require too many splits which is a limitation. To overcome this, Muller in [Mül+06] proposed yet another grammar: *CGA Grammar*. *CGA* stands for Computer Generated Architecture and is currently implemented in the CityEngine tool.

This grammar works with the concept of shapes which consist of a symbol (a string of characters), geometrical attributes and numerical attributes. Shapes are identified by sym-

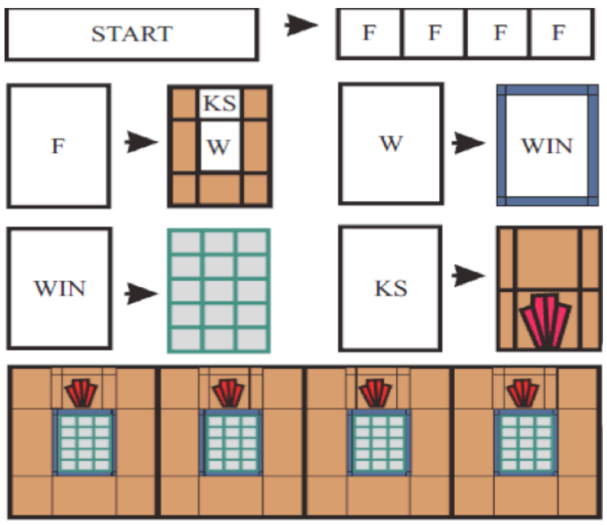


Figure 2.9: Execution of a Split Grammar. *START* is the axiom and the lower row represents the results.

bols which may be terminal or non-terminal, corresponding to terminal or non-terminal shapes respectively. The geometric attributes contain a position vector, three orthogonal vectors, making up a coordinate system, and a size vector. These elements form a bounding volume called the Shape's Scope. Figure 2.10 illustrates this.

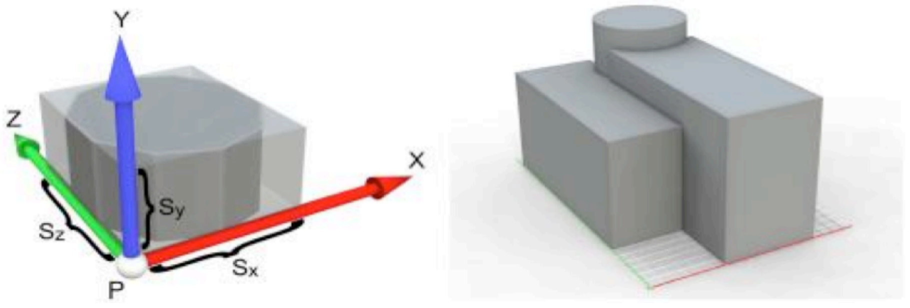


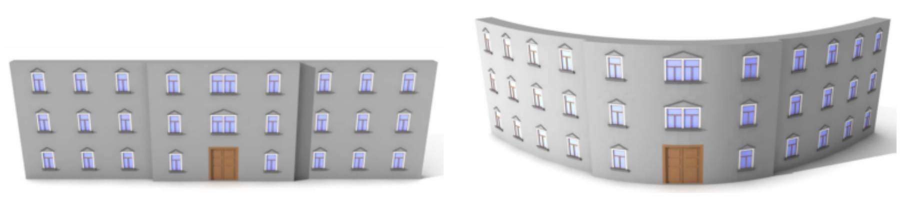
Figure 2.10: Left: the Scope of a Shape. Right: a generated model with three primitives [Mül+06].

The generation process starts with a non-terminal shape — the axiom — to which is applied one of the rules. This generates a new set of shapes (terminal or non-terminal) which are introduced in the result. The process is repeated until there are no more non-terminal shapes in the set. Each rule has an associated derivation priority so that it occurs with a growing, but controlled, level of detail. In each step of the derivation the highest priority rule is chosen.

Two more methods to increase the detail are presented here: the *occlusion test* and *snap lines*. The former indicates whether or not there is partial or total occlusion between several of the building's shapes. Such test can include all shapes, shapes with a given symbol or all shapes except the predecessors, being up to the user to select the criterion. The latter method allows to slightly change the geometry, snapping edges or faces to coincide with a nearby plane or line, thus preventing situations that do not occur in real environments (e.g., partial intersection between a window and a wall).

The previous methods present one common drawback: they aren't capable of generating curved architecture. This stems from the fact that split grammars use split planes to divide the shapes and the rules only apply to planar surface parts. Round elements can only be approximated by static pre-modelled assets.

Zmugg et al. [Zmu+13] introduced an extension to current split grammar systems, where the user can introduce free-form deformations that can be introduced at any level in a grammar. The further subdivisions take these deformations into account by adapting to the new surface dimensions and repetitions can adjust to more or less space. Figure 2.11 illustrates an example of a building façade modelled with this method. Note that, while there are only three columns of windows on the side parts of the original façade, the deformed façade contains four to compensate for the larger length of the curved surface. However, the windows' width remain the same.



(a) Original Façade **(b)** Deformed Façade

Figure 2.11: Applying deformation to a straight façade (a) yields a different number of windows on the side parts (b). [Zmu+13]

The previously mentioned works relied on a shape grammar approach, in which rules were text-based. There are some inherent interaction problems to approaches of this type, namely the fact that they require the user to write large amounts of rules to achieve detailed results. Furthermore, understanding rules created by other users is, sometimes, not a trivial process, as there is a natural disassociation between the rules used and the generated product tree [Pat12].

Patow [Pat12] presented a method to procedurally generate buildings using a graph based approach, allowing users to create rules from scratch without the need to edit text files. The method also bridges the disassociation between rules, their interrelationships and their applications. Extending this method, Barroso et al. [BBP13] introduced a visual copy and paste for procedurally modelled buildings with rule set rewriting. Besuievsky and Patow [BP13] further enhanced this method by allowing the user to customise the level-of-detail of the generated models based on different user criteria (e.g., distance to viewer or screen-size projection) through a scripting interface.

Another, recent approach was presented by Silva et al. [Sil+13] who proposed a graph based visual language for procedural models. The nodes in the graph have one input port and one or more output ports to which links are connected. Links conduct shapes and parameters between nodes that, in turn, execute operations on the shapes and output the results. Several links can be connected to the same port, allowing to of nodes.

There is also the possibility of grouping several nodes into a larger component node. This also enhances re-usability but, mostly, improves the semantic of the modelling process. One can create architecturally significant elements such as building parts, façade styles or windows. Moreover, components define an interface that other nodes can use which helps specifying how components are intended to be used. Figure 2.12 illustrates this approach to model a porch, which generates over 200 lines of text grammar rules.

Like in terrain generation and street network modelling, there are, also, methods taking example buildings as input and procedurally generating new buildings. Merrell [Mer07] used a three-dimensional grid to partition a small example which automatically defines that two parts can only be adjacent in the resulting model if they are adjacent in the exam-

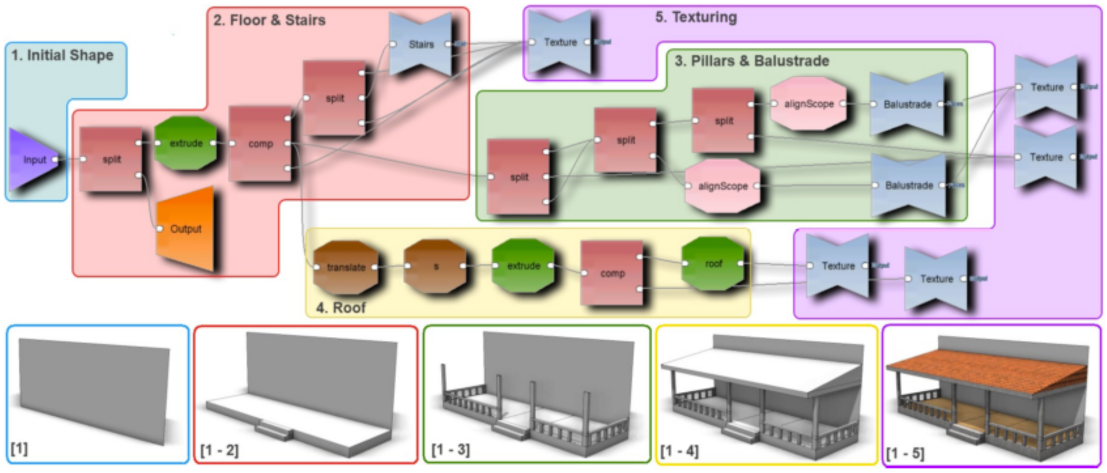


Figure 2.12: Modelling a porch with the Node-based approach [Sil+13].

ple model. Such approach can be used when modelling symmetric buildings, models that change overtime or models fitting soft constraints. The results can be quite elaborate despite its grid like structure.

Subsequently, this method was improved [MM08; MM11] by adding several geometric constraints. A dimensional constraint states that a given object should have a specific size; algebraic constraints introduce relationships among numeric properties (e.g., the height of an object should be twice its width); incident constraints allows to manage the incidences of vertices to faces; connectivity states how objects should be connected to each other; and large-scale constraints allowing the user to specify general ideas about how the model should look. These improvements allow a better control of the model synthesis (See Figure 2.13).

More interactive approaches have been presented, where the user directly manipulates

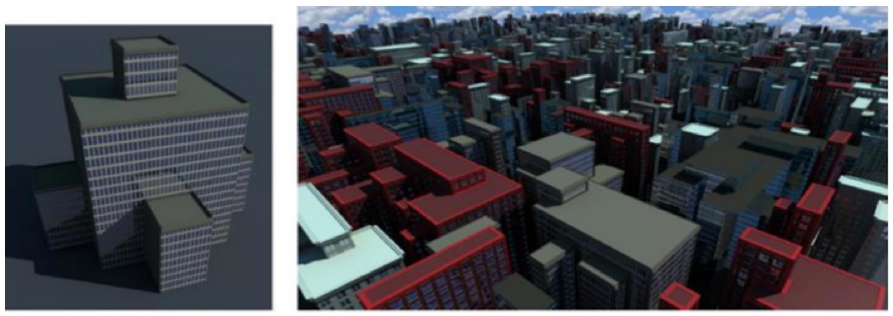


Figure 2.13: From a simple example (right), complex cities can be generated (left) [MM08].

the geometry at some level and the system generates procedural rules. Such is the case in the work presented by Lipp et al. [LWW08] where the authors proposed an interactive visual method for grammar edition, giving the user direct control over each grammar aspect. Given the possibility that a user could desire to change the aspect of a specific façade element, or even all elements in a façade column or row, the text-based approach would require the user to manually create more rules in order to precisely state the intent. This is also addressed in this work with the introduction of the exact and semantic locators concepts. These give the user the possibility of applying changes to a specific element or based on some semantic attribute (e.g., façade number or floor) respectively, by directly selecting the element or elements on the visual interface. Nonetheless, the underlying paradigm still employs grammar rules.

A different approach was introduced by Kelly and Wonka [KW11] who presented a method to procedurally model building exteriors through procedural extrusions. The technique started with a building footprint outlined by the user which, afterwards, defines one vertical profile line for each building façade. Applying a sweep algorithm produces complex, detailed buildings with interesting features such as curved or overhanging roofs, buttresses or chimneys. In this work, the authors present, also, results integrating floor plans from GIS databases, allowing modelling larger environments (See Figure 2.14).

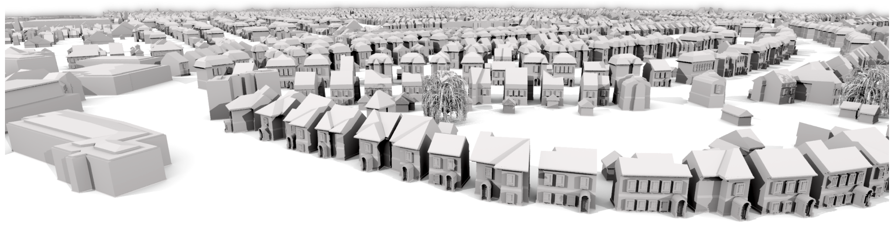


Figure 2.14: Modelling large environments with procedural extrusions [KW11].

Procedural modelling can be used to generate immense urban spaces with some approaches executing in run-time. Greuter et al. [Gre+03] presented a system to generate *pseudo-infinite* areas of urban area in run-time. The method proposed obtained results corresponding to an extremely large city, consisting of $4 * 10^8$ different buildings. This is achieved by generating new buildings that lie inside the camera's view frustum. These buildings are cached in video memory so that their generation is not required in following frames. Once the camera is sufficiently distant, the buildings are disposed from cache in order to save memory. To guarantee that the exact same building is generated in the same position every time it is visible, the whole process is guided by a global seed value and some additional parameters. Figure 2.15 shows a city output by this method.

Despite the great amount of detail and high visual quality the methods above produce,

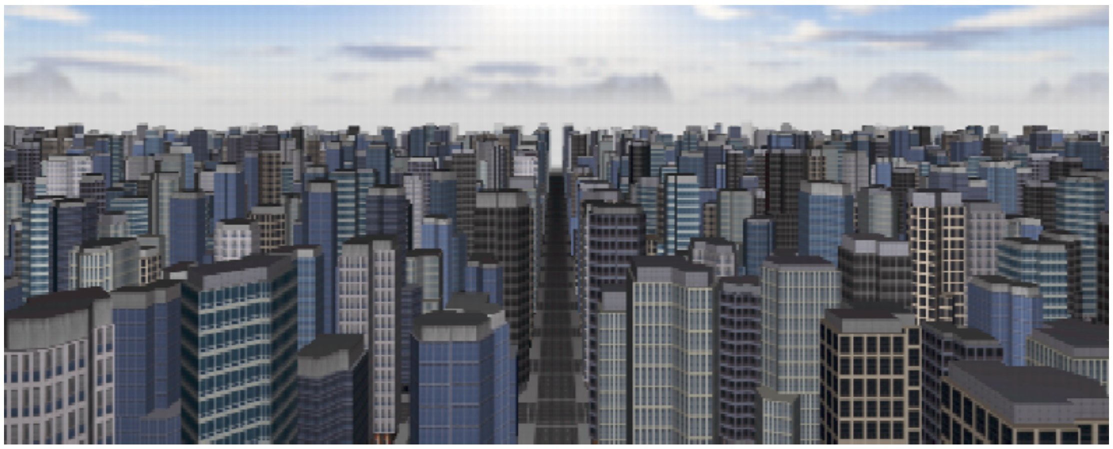


Figure 2.15: Real-Time procedural city generation [Gre+03].

they target mainly the generation of fictitious urban spaces. To create scenes that closely resemble real areas, the system needs to incorporate a high level of information in order to generate enough detail. This means that, using these methods, the user would have to write a large amount of production rules, thus requiring a high level of human interaction. This goes against one of the purposes of procedural modelling: automatic generation.

Several authors [DB05; Coe+07; Jes+12; PP13] suggested the integration of such techniques with Geographic Information Systems (GIS) since these contain great quantities of geo-referenced urban areas which can be automatically extracted. This way, it becomes possible to generate building models in their correct position, along with several other parameters such as the height, number of floors or, even, its age allowing the procedural rules to choose a correct architectural style.

Moreover, the existence of geo-referenced data provides the capability to infer spatial relations between the distinct urban objects, creating the concept of geospatial awareness [Coe+07]. With this, it is possible to implement a system capable of generating models according to some basic construction rules. For instance, bus stops must be placed near a road and oriented towards them.

In [Coe+07] Coelho presented the Geospatial L-Systems, a new extension to parametric L-Systems, incorporating geospatial awareness. The development sequence in these systems is supported by a GIS which stores and analyses the geospatial modules. The system is capable of creating virtual environments with a great level of visual fidelity. Nonetheless, the obtained realism depends on the quality of the stored data. This system is implemented in the XL3D Procedural Modelling tool and Figure 2.16 shows an example of an urban area created with this tool.

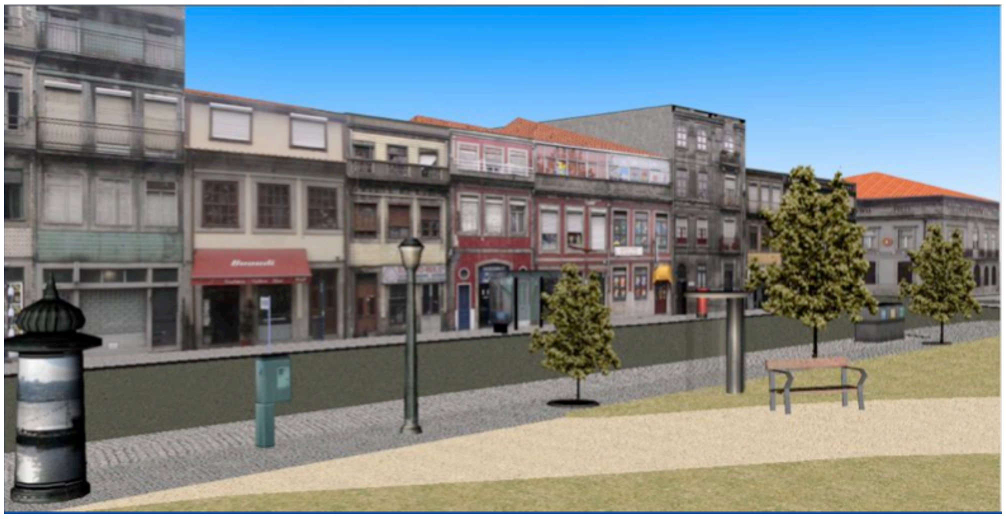


Figure 2.16: Praça da República in Oporto, Portugal modelled with XL3D [Coe+07].

A new grammar, called PG3D grammar, was introduced in [Sil10]. The main characteristic in this work lies on the fact that it is implemented in a spatial database engine introducing fast data access, large data manipulation and complex query capabilities. The results of the grammar execution can, afterwards, be extracted and incorporated into any digital game or custom modelling tool [Sil10]. Figure 2.17 shows an output scene.

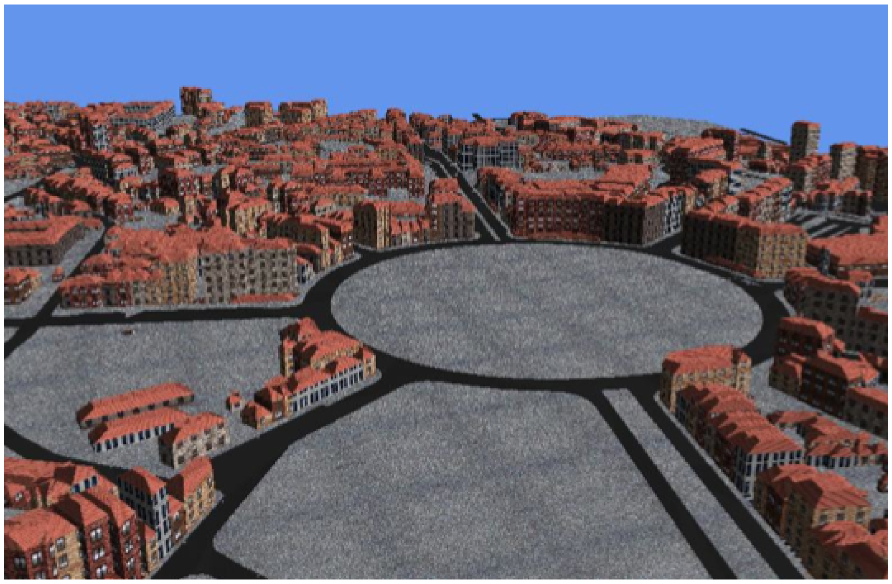


Figure 2.17: Modelling of the Boavista Roundabout in Oporto, Portugal [Sil10].

Robles-Ortega et al. [Rob11; ROF13] used genetic algorithms for the generation of textured buildings from 2D GIS data using only a small set of images. The authors proposed two different genetic algorithms, where one is targeted at generating the lower part of a building and the other generates the upper part. Because the algorithm generating the lower part takes into account the street slope, the method can be used in cities with both horizontal and steep streets.

However, data present in GIS systems often contains inaccuracies. Pueyo and Patow [PP13] proposed a system for the robust generation of blocks and building layouts based on a repairing process applied when the data is not correct.

2.4.4 Façade Modelling

There are several characteristics that distinguish a building from all others. Among these are the building height, number of floors but, most importantly, the layout of the different elements in its façade. These are, most certainly the aspects of a building that allow an observer to identify a specific building when taken from its context such as in a photograph or diagram.

To enhance the level of detail of the produced buildings, giving a higher degree of familiarity to the user, several authors have oriented their work towards more detailed procedural modelling solutions, focusing on the detailed representation of façades.

To enhance previous work, Muller et al. [Mül+07] presented a system that, from a façade image creates production rules that generate detailed three-dimensional models of the given façade. The author describes the process as a sequence of four different stages:

- Façade Structure Detection: Splits the façade image vertically into floors and then into tiles through the use of mutual information. A tile is an important concept, representing an architectural element, such as a window or door, and the surrounding wall.

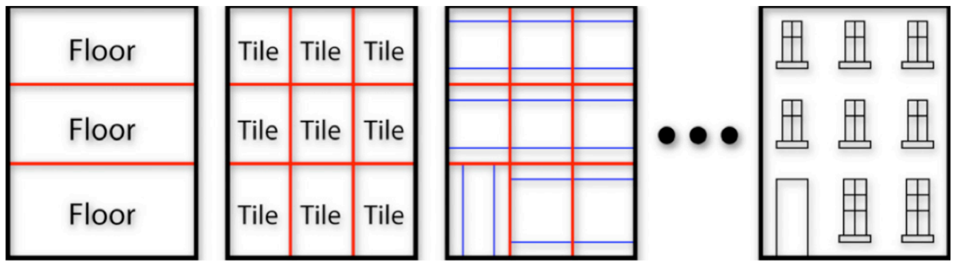


Figure 2.18: First two steps in the execution. [Mül+07]

- Section Refinement: The method further splits the tiles into smaller rectangles using the division obtained in the first step, where similar tiles are grouped together. This step derives from split grammars. The first two steps are illustrated in Figure 2.18.
- Element Recognition: A match between the small rectangles and three-dimensional architectural elements in a database is conducted. Next follows the generation of three-dimensional textured models together with the semantic structure in a shape tree.
- Edition and Extraction of rules: The semantic interpretation of façades can reveal useful in various editing operations, including the extraction of grammar rules from the shape tree previously built.

As is possible to see in Figure 2.19, the result of executing the proposed method yields models with a high level similarity between the generated façade and the input image. Despite that, this method is very sensitive to noise in the images turning its use difficult in certain cases.

Extending this work, Musialski et al. [MWW12] introduced a workflow mixing manual and automatic splitting of façade images. Starting from an orthogonal input image, the user interactively splits the façade into shapes, with the so called guillotine cut. The system automatically finds most split lines and symmetries, while the global façade structure is de-

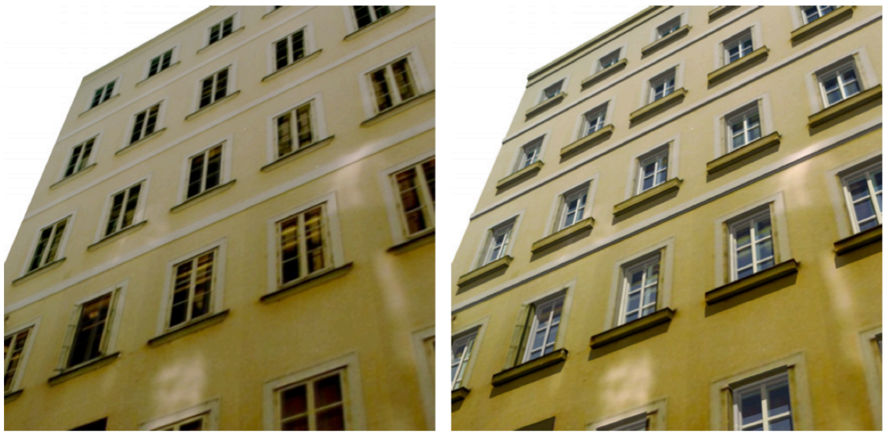


Figure 2.19: Left: a façade image. Right: the generated model [Mül+07].

termined by the user. The authors also introduce the concept of *coherence-based modelling*, where coherence is defined by regions that contain architectural elements (e.g. windows) that can be separated by a common guillotine cut.

A new grammar, named *F-Shade* and oriented towards façade modelling, has been presented by Haegler et al. [Hae+10]. The goal of this work was to present a grammar to compactly represent a façade structure while permitting fast random access to its data given the coordinates of any point in the surface of the building.

This grammar encodes the structure through the disposition of rectangular regions in a base polygon. In turn, each region contains information about the texture to be applied, a depth value and material characteristics.

Another contribution from this work is a renderer prototype to visualise the generated models from a compact representation of the grammar directly in GPU memory. For this, the system was implemented using a technique known as *deferred shading* and split into three phases.

In a first step, the building geometry is drawn to a temporary memory. Afterwards, a pixel-based evaluation of the grammar rules stored in GPU memory is executed, from which the texture to apply to the pixel is obtained. The last step presents the resulting model using a pixel shader and the data obtained in previous steps. The results can be seen in Figure 2.20.

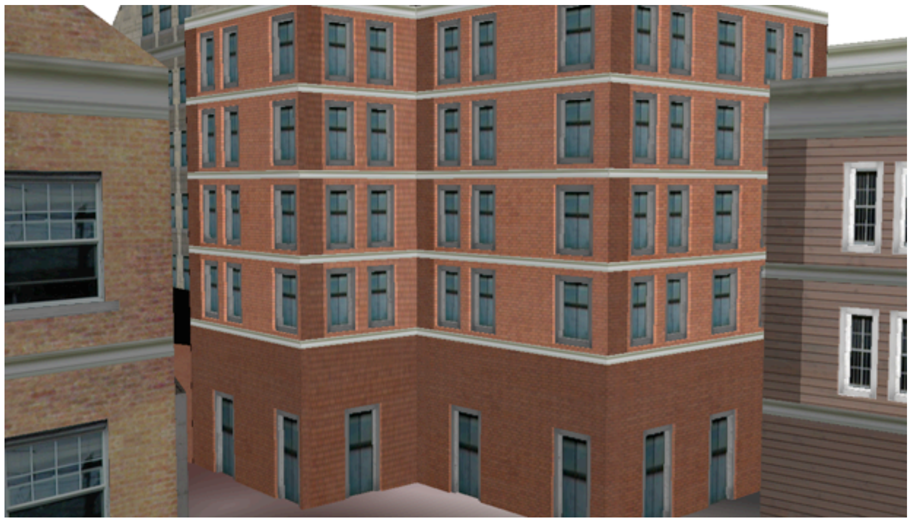


Figure 2.20: *F-Shade* resulting buildings [Hae+10].

More recently, this work has been extended by Krecklau et al. [KBK13], who introduced a view-dependent real-time rendering of procedural façades with high geometric detail, by taking the idea further by introducing 3D geometric detail in addition to flat textures. The authors introduced a two-pass procedure that first renders a flat procedural surface, much like in [Hae+10]. During the rasterization, the method triggers the generation of a detailed asset whenever a geometric façade element is potentially visible. The set of generated models are then rendered in a second pass. Figure 2.21 illustrates this.

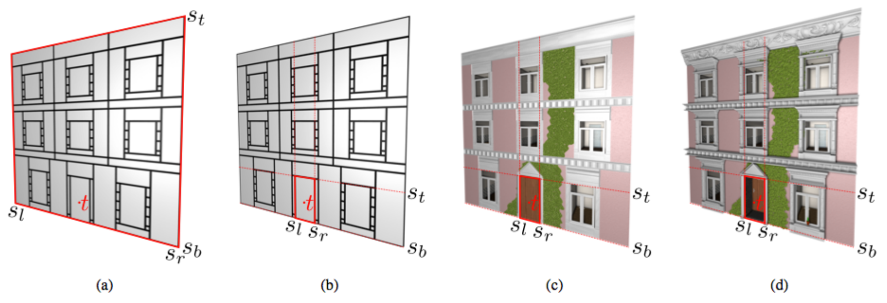


Figure 2.21: Each pixel is evaluated, tracking down split operations until a terminal shape(a and b). The method either samples from a texture (c) or instantiates a new geometric entity (d) [KBK13].

An approach capable of generating more complex façade models was proposed by Finkenzeller [Fin08] (Figure 2.22, c). The system presented in this work can generate frames, borders and ornaments which can, easily, be found in buildings (Figure 2.22, b). However, the method needs more user intervention than previous methods, since it requires a coarse building model to be created. Nonetheless, this is intended to be fairly simple serving as a way to establish the contours of the building, as well as its purpose and intended styles (Figure 2.22, a).

The first stage of the process requires the user to create a planar representation of the ground floor, consisting of one or more convex polygons, called *Floor Plan Module* (*fpm*). The above floors are modelled based on the previous floor by keeping, removing or splitting the existing *fpm*.

Afterwards, the system generates a representation of the several façade elements. A wall is generated for each exterior *fpm* edge, which is vertically subdivided, given the chosen architectural style, creating partitions. Each partition, in turn, contains other architectural elements and, at most, a door or a window.

The last step generates the roofs taking into account the building structure. The algorithm defines the aspect of the entire roof based on the individual structure of each *fpm*. For this, individual roofs are generated for each *fpm* in the top of the building which are, afterwards, combined by adapting them in the intersection zones.

It is possible, at any given instant, to modify the parameter of any element and, thus, the building aspect. This allows the user to change the style of the entire building, a floor or even windows or doors individually. As a consequence, the system permits the generation of more complex structures in a shorter amount of time.

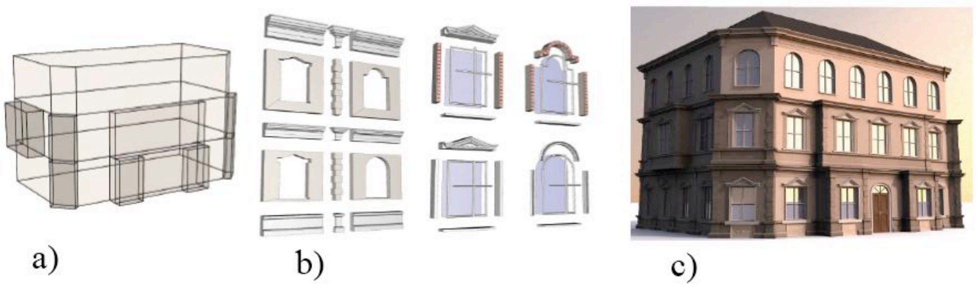


Figure 2.22: Modelling process proposed by Finkenzeller [Fin08].

2.4.5 Modelling Other Content

There are several more elements that build up an urban scene other than roads and buildings. Examples of these elements are electrical posts, street lamps, or garden benches which are called urban furniture. To enhance the realism of the generated virtual cityscapes, it is also important to model such entities. The introduction of green spaces and trees is also an important element to include in urban areas [Coe+07].

Usually, there is a low level of diversity inside a category of urban furniture throughout the city. This is due to the fact that these items are, usually, bought in bulk for economic and aesthetic reasons which results in a homogeneous aspect throughout the city. This allows one to store the models or the procedures to generate such models in a database and re-use them whenever necessary.

On the other hand, trees and green spaces present a natural growing process and, thus, have a larger variety inside one category requiring more dedicated modelling techniques. The employed methods should, preferably, be procedural and introduce some degree of randomness in the generation, giving the notion of diversity. This problem has already been intensively studied, and typically employ the use of L-Systems [MP96; PJM94; PL90].

Although the modelling process of these elements is considered to be trivial, their correct displacement and orientation around the city constitutes a problem, since in urban environments some rules need to be followed. For example, trees cannot be placed on driving lanes and bus stops must be oriented towards the street. As a consequence, the placement must not be purely random but, instead, should be guided by a set of rules.

In the approach followed by Coelho et al. [Coe+07], the authors use geo-referenced information stored in databases and Geographical Informations Systems. In this work such data is employed to correctly position the elements in the space, as well as to create relations between such elements (geospatial contextualisation) allowing to correctly set their orientation. In the case the existing information is not enough, it is amplified using the Geospatial L-Systems, proposed by Coelho [Coe+07], through the definition of new rules and the introduction of randomness.

2.5 Open Questions

The area of procedural modelling of urban environments is in great expansion and several works have been published by different authors presenting innovative solutions and interesting results. Such approaches have been proving that the employment of procedural methods in the most different areas is becoming a very viable alternative.

Nonetheless, some issues remain open, leaving space for further research targeted at improving their execution and enhancing the results. Examples are:

- Automatic Extraction of Rules: Since many of the presented approaches rely on formal grammars, there is also the inherent challenge of creating the production rules leading to the desired models. Muller has already tackled this problem regarding façades [Mül+07], yet, research is still needed in order to improve this aspect. The automatic extraction of rules or their parameters is an area called *inverse procedural modelling*.

- Library of Rules: Since, most often, the production rules are in text form, they are easily re-utilised. Normally, it is considered easier to change a given rule to correspond to new requirements than it is to analyse a design and redefine it from start. Creating a library of production rules and making it publicly available may be a great leverage in creating a new virtual city.

- Defining rules from natural language: Using formal grammars has the consequence of bringing a strict syntax into the creation of production rules, reducing its intuitiveness. For such reason, defining rules in natural language could make procedural generation more available to all types of audience. Another advantage would be the possibility of using other types of sources as book and other architectural reports. Such approach has already begun to be explored in [RCR09].

- Definition of Graphical User Interfaces: The use of text-based production rules in the generation process of procedural methods has the disadvantage of not being intuitive, reducing the acceptance by users of an artistic background (e.g., designers and architects). To counteract this, the edition process should be performed at a higher, simpler level. The creation of graphical interfaces may solve this problem. The work proposed by Lipp et al. [LWW08] presents a system where the edition of rules is done directly on the generated model and there is no need to write the production rules. The proposed thesis also aims at creating more intuitive and interactive ways to define procedural rules, namely through the use of Sketch Based Interfaces for Modelling. Other approaches employ a node-based approach [Sil+13] providing a visualisation of the flow of data being processed.

- Integration of methods: Most procedural modelling techniques specialise in one content or feature and there has been little research attention to the cooperation between distinct procedural methods [ST14]. To successfully create a virtual world requires the manual integration of all different features once they have been generated. However, there is an additional challenge in the fact that there is, most often, interactions between different elements. The *Sketchaworld* [Sme+11] system is a step further in the direction of solving this problem.

- Complex Building Shapes: Most complex architectural features, including overhangs, dormer windows, roof constructions with vertical walls, buttresses, chimneys and bay windows are not easily modelled by common procedural modelling techniques based on split or shape grammars. This effectively limits the user's expressive power by restricting the features that can be included in an architectural model. Procedural extrusions [KW11] have been used to overcome this problem but few systems employ similar approaches.

- Semantic Information: The visual part of an urban environment is not enough to convey the feeling of realism. If a greatly detailed three-dimensional model behaves or is structured incorrectly it can, easily, look unreal and break the immersion. A solution is to embed both the modelling and the execution with semantic information, i.e. what the objects are, how they behave and how they are related to other objects and the world. Tutenel et al. [Tut+08; Tut+09; Tut+11a; Tut+11b] have extensively explored this issue.

2.6 Summary

This chapter has presented the current state of the art on the field of procedural modelling of urban environments. These methods can be grouped by the type of output they produce (e.g., road networks or buildings). To create complete environments it is, typically, necessary to generate several of these types of models which is usually done in the order the methods were presented in Subsection 2.4 (Terrain, Road Networks, Building and Façades).

Since the seminal work of Parish and Muller [PM01] which introduced L-Systems as a means to procedurally generate road networks and buildings, several methodologies for procedural modelling of urban areas have emerged. The most recent is the node-based approach [Pat12; Sil+13] which provide a more intuitive interface than text-based approaches. However, the iterative cycle is still present. The proposed work aims at developing a sketch based interface for modelling methodology on top of such techniques for procedural modelling.

3

Sketch Based Interfaces for Modelling

This chapter presents sketch based interfaces for modelling. Among all interaction metaphors, this one was considered of importance due to the fact that it resembles one of the communication mediums of choice of artists: drawing.

One common problem of several procedural modelling systems is the low level of user interaction they provide, often resulting in a lack of expressiveness or direct control over the generated models, as mentioned in Subsection 2.3. Most of the approaches rely on a grammar-based approach to generate the geometry which, by itself, represents a challenge to users of artistic background since the systems require an in-depth knowledge of how it operates. Moreover, this kind of approach has an iterative cycle where the user must first write the rules, generate the procedural model, evaluate the result and enumerate the changes or refinements to be made and, finally, rewrite the rules. The editing operations are greatly disconnected from the visual feedback. This is a challenge to be tackled during the development of the proposed thesis, using Sketch Based Interfaces for Modelling (SBIM).

User interfaces in modelling have traditionally followed the WIMP (Window, Icon, Menu, Pointer) paradigm, which can be functional and powerful but also cumbersome and daunting for a novice user. More accessible and natural user interfaces have emerged, bringing modelling systems more intuitive interaction metaphors. An example, which will be reviewed here, are the sketch based interfaces for modelling. The goal in these interfaces is to allow sketches to be used in the modelling process in several levels of detail, ranging from the rough model creation to the fine detail construction. The user progressively refines a coarse model until the desired result is achieved. Mapping a 2D sketch to a 3D modelling operation is a difficult task, rife with ambiguity [Ols+09].

Furthermore, drawing has always been an extremely effective means of communication and a tool that has always been present in mankind. Sketching is a natural way to communicate ideas quickly, since with a few strokes complex shapes can be evoked in viewers. It is also the medium where artists, designers and architects work on. Since these are the end users of a modelling system, whether traditional or procedural, it is important, therefore, to study how a sketch based interface may help on the task of procedural content generation.

Drawing also constitutes an early phase of prototyping a design before it is manually translated to a 3D model by a trained 3D artist. Because of this, model creation is a major bottleneck in a production pipeline, requiring human effort to create the complex and diverse shapes and intricate relationships [Ols+09]. The goal with SBIM is to merge the complex, traditional modelling systems with the natural interaction of sketching allowing users to construct and edit models in a progressive way from an initial high level concept to a more detailed and accurate end-model.

3.1 Sketching Pipeline

Typically, sketch based interface for modelling systems have a three stages pipeline transforming the user input strokes into complete three-dimensional models as suggested by Olsen et al. [Ols+09]. The first stage is responsible for acquiring the raw user sketch, which is subsequently filtered in a following step in order to facilitate the processing. The last stage is to interpret the sketch in the context of the specific application. This pipeline is illustrated in Figure 3.1.

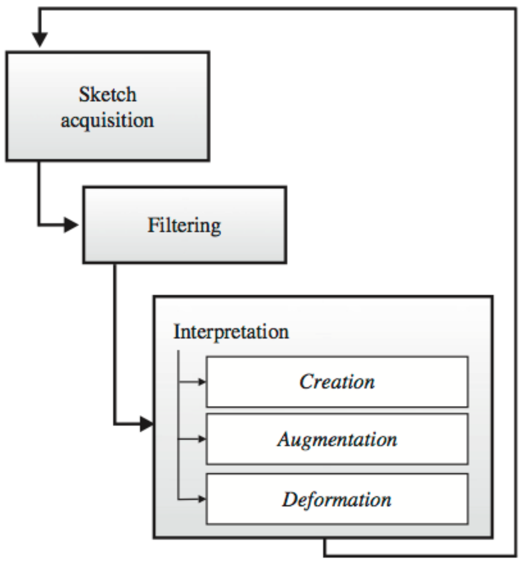


Figure 3.1: The pipeline of sketch based interfaces for modelling [Ols+09].

The most basic operation in any sketch based interface is the acquisition which is typically conducted with a freehand input device. However, devices that mimic the drawing on paper process are better able to exploit a user's ability to draw (e.g., a tablet display). After collecting user input, a SBIM system must store the sketches in some sort of representation format. At the bare minimum, the hardware provides the 2D coordinate system of the strokes.

The strokes are comprised of samples which are spaced irregularly, depending on the drawing speed. Therefore, they tend to be closer near curves, where the user is more careful with the drawing [Ols+09], which can be exploited to identify "important" parts [SSD06]. Also, the samples can be augmented with more information such as time samples, pressure or pen orientation, depending on the target application and available hardware [Ols+09].

The large body of work in image processing has had an impact in the decision of the representation format of samples, which turned to the use of an image-based stroke representation, in which the stroke is approximated with a pixel-grid. This representation has the advantage of fixed memory usage as well as automatic blending of multiple strokes [Ols+09]. However, any other information besides the sample positions is lost.

However, both the input medium and the user may introduce errors in the intended sketch and, as such, before interpreting a sketch it must be filtered. Poor drawing skills or slight hand jitters have an impact on the drawing of both straight lines and curves. Also, sometimes traditional digitising tables may have resolutions as low as 4-5 dots per inch opposed to the 1200-1400 dots per inch of scanned drawings. This is because sometimes users draw so fast that even with high sampling rates such as 100 Hz only a few points per inch can be sampled [SSD06].

The filtering process can be done by discarding any sample within a threshold and by interpolating between samples separated by more than a threshold [Ols+09]. Another approach is polyline or polygon approximation, reducing the complexity of a stroke to just a few samples [IMT99]. It can also be done by fitting the strokes to a parametric curve such as Bézier [Pie87] or B-Splines [BC90]. It is also possible to combine the polyline approach with the curve fitting yielding a segmented approach. Figure 3.2 shows examples of these methods.

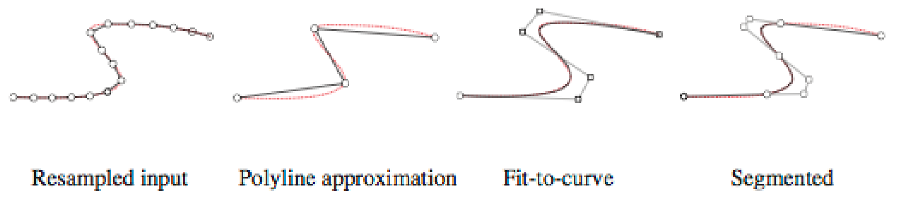


Figure 3.2: Filtering methods [Ols+09].

The *Beautification* process [Iga+97] is a technique for inferring geometric constraints between strokes at a high level, such as linearity, co-location, parallelism, perpendicularity and symmetry. Figure 3.3 shows an example of this.

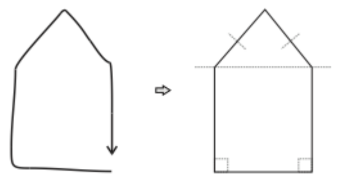


Figure 3.3: Beautification process of a house-like sketch [Ols+09].

After the introduced sketch is sufficiently filtered, it must be interpreted in order to be given a meaning in the application context, i.e. mapped to a modelling operation. The main problem of Sketch Based Interfaces resides in the interpretation of user sketches. In other words, the system must infer what the user intended to draw, and how can it be mapped into a modelling operation. Olsen et. al [Ols+09] proposed a categorisation of SBIM into three primary methods: to create a 3D model, to add details to an existing model and to deform and manipulate a model. Although the authors define the previous categories, a complete SBIM system should contain methods from all these categories in order to allow total control of the modelling process.

3.2 Model Creation

A model creation system tries to reconstruct a 3D model from the 2D sketched input. Olsen et al. [Ols+09] divide this further into two categories. One includes *evocative* systems (Schmidt et al. also name this *suggestive* systems [Sch+06]) where user sketches are used to instantiate a built-in model similar to the input. Here, the systems can also fetch models from a database and use a metric to compare such models to the user input and choose the best candidate [YSP05; Fun+03; SI07]. These methods usually use *expectation lists* which allow the user to resolve ambiguous situations. Such is the case in SKETCH [ZHH96], Chateau [IH01] and GiDES++ [Per+03].

On the other hand there are *constructive* systems (Schmidt et al. refer to these as *literal* systems [Sch+06]) that attempt to map the user strokes directly to the output model. A fundamental operation in these systems is *inflation* where user-sketched closed 2D contours become the 3D silhouettes of round shapes [Sch+06]. A pioneering example is Teddy [IMT99] where the user can easily create rotund shapes such as stuffed animals.

This categorisation neatly aligns with the classical distinction between reconstruction and recognition [Ols+09].

Evocative Systems

Olsen et al. [Ols+09] further subdivide *evocative* systems into *iconic* and *template retrieval* systems. *Iconic* systems infer a final 3D shape from a set of simple iconic strokes. The SKETCH system [ZHH96] is one example where simple groups of strokes are used to define a primitive 3D object. Three stroke lines meeting at a point are used to instantiate a cuboid shape whose dimensions are taken from each of the strokes. The GIDeS system [Per+03] also follows a similar design providing a larger rage of primitives.

Template retrieval systems, on the other hand, fetch models from a database of template objects, evaluating their similarity with the user sketch. The objects are typically more complex than the simple primitives instantiated in *iconic* systems and, as such, both the user sketch and the algorithms needed to infer the user intentions are more complex. The similarity evaluation is typically done in 2D space since such evaluation in 3D would require the reconstruction of the 2D sketch, which is the problem to be solved.

Funkhouser et al. [Fun+03] use the projected contour from 13 viewpoints defining a shape descriptor of an object. By applying image-based transformations to each contour, it is possible to extract a fixed-length, rotation-invariant feature which is then compared to the user sketch after applying the same transformation.

Shin and Igarashi's Magic Canvas [SI07] extends this concept by using 16 contours from each template object but, also, use a Fourier-based method for sketch matching. This system enables the user to sketch an entire 3D scene and, so, besides instantiating template objects, Magic Canvas must also place them correctly in the 3D scene. This is accomplished by scaling and rotating the objects to match the input sketch.

Yang et al. [YSP05] use procedurally described models rather than mesh template objects. These models describe how to make a specific model (e.g., a mug) instead of having a static mug mesh. This has the added benefit of allowing the template to be deformed to conform to the user's sketch. However, adding new templates to this system requires more labour.

Instead of instantiating a complete model at once, the approach suggested by Lee and Funkhouser [LF08], allows composing models out of template parts. The system provides a set of templates for object parts which the user can instantiate separately and iteratively compose a larger object. For example, the user would, in a first moment, draw the body of an aeroplane, then the wings, engines, missiles and so on. The system automatically places the parts relatively to other elements. This allows a broader range of variations in any given class of objects.

3.2.1 Constructive Systems

While the previous methods used some sort of knowledge in the interpretation of user sketches, diminishing the domain of possible interpretations, *constructive* systems must rely solely on rules. This represents a greater challenge which has received a great deal of attention, yielding several interesting approaches.

These systems can be, according to Olsen et al. [Ols+09], grouped into three different categories: *engineering design systems*, *free-form design systems* and *multi-view systems*.

Engineering objects are, often, characterised by being mostly planar and hard-edged and systems aiming at constructing this type of objects can use this domain knowledge to cut down the number of possible interpretations.

Lipson and Shpitalni [LS07] presented an optimisation-based approach for reconstructing geometry from user sketches performed in parallel projection. Each stroke in the drawing corresponds to an edge in the 3D model and each endpoint to a vertex. The system detects relationships, such as corners, perpendicularities and so on, between the strokes and then performs an optimisation on a linear equation where the depths of the vertices are the unknowns. This system is an example where the user first sketches the entire model and then the system computes its 3D geometry.

Interactive systems have been proposed where the user can iteratively sketch parts of the object and the system automatically reconstructs them. This gives the user the opportunity to correct or refine the results more quickly. The system also benefits from iterative approaches since it can use simpler reconstruction techniques. The most common approach is *extrusion* [Sch+06; SC04; Per+00] which is well-suited to creating models with hard edges such as cubes and cylinders [Ols+09].

Inspired in analytical drawing where artists use image space scaffolding, defining the objects' coarse structure, to incorporate details into the sketched elements, Schmidt et al. [Sch+09] proposed an approach to inferring 3D lines and curves from perspective drawings in an interactive design tool. The system creates 3D lines and curves based on a pure-inference sketch interface [Sch+09] that generates sets of candidate curves and compares them to the original user stroke. There are also useful interactive drawing aids that allow the user to sketch 3D curves with precision.

In *free-form design*, in contrast to engineering objects, models tend to be smoother and, as such, the approach must be different. Typically, systems that are contained in this category, let the user sketch the contour of the objects and then use the contour skeleton (the line from which the closest contour points are equidistant) to infer a surface. The contours are, simply, the lines that separate surface parts that face the viewer from those that don't. This contains the silhouette and other interior features. The simplest case, where the skeleton is a straight line and is the contour's symmetry axis, creates a surface of revolution by revolving the contour around the skeleton, such as in Schmidt's ShapeShop system [Sch+06].

For more complex, non intersecting contours a typical approach is inflation. The Teddy system [IMT99] pushes vertices away from the skeleton by the distance to the contour (Figure 3.4). Alexe et al. [AGB04] use an implicit surface representation and place spherical implicit primitives at the skeleton vertices. The spheres are blended together and the generated geometry matches the given contour.

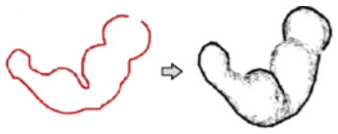


Figure 3.4: The approach proposed by Igarashi pushes vertices away from the chordal axis [IMT99].

Intersecting contours are indicative of surfaces and other contour lines that face the viewer but, due to occlusions, are invisible. To infer such hidden contour lines, Williams [WH94] presented a method that has been, subsequently, used by several approaches on 3D reconstruction from sketches [KH06; CS07]. Karpenko and Hughes [KH06] obtain a smooth shape by creating a ”topological embedding” and constructing a mass-spring system and finding a smooth equilibrium state. However, this approach requires a careful parameter tuning.

The ShapeShop system [Sch+06] also inflates 2D contours into rounded 3D objects. Since Hierarchical Implicit Volume Model is the underlying shape representation in this system, nearby primitives naturally blend together to form more complex shapes.

Schmidt and Singh [SS08] extended the previous work with a connection between sketch based interfaces for modelling and procedural surface modelling. Surfaces are generated by traversing a tree whose internal nodes are operations on surfaces, while leaf nodes can be surfaces that have been sketched previously. This way, altering an internal node has repercussions on the sub-tree rooted at that node.

Lastly, *multi-view* systems are those that allow the user to elaborate sketches from several drawing planes. As an example, Karpenko et al. [KHR04] presented an interactive system that used epipolar lines to fix the depth component of each stroke vertex. After drawing a stroke, the user can rotate the view and see lines extending along the depth direction of the previous viewing point. When the user draws another stroke, it is projected onto such lines which defines the depth of each vertex. The authors, however, state that this method is not very intuitive.

3.3 Modification

In order to be fully useful, sketch based interfaces for modelling must, also, allow the modification of the created 3D models. This can be done through the addition of more detail or by deforming what has already been modelled. In turn, these can be grouped into Augmentative or Deformation Systems respectively.

3.3.1 Augmentative Systems

Augmentative systems are those that add detail or new parts to an already existing model. Creating more elaborate detail on an existing model is an easier task, since the model serves as a 3D reference for mapping strokes into 3D [Ols+09]. The same authors further classify these systems as *surficial* or *additive*. Surficial augmentation allows users to embed fine-grained details into the surface of the model, normally displacing the surface along its normal wherever the stroke intersects the surface. This is ideal to create sharp creases on the surface as visible in Figure 3.5.

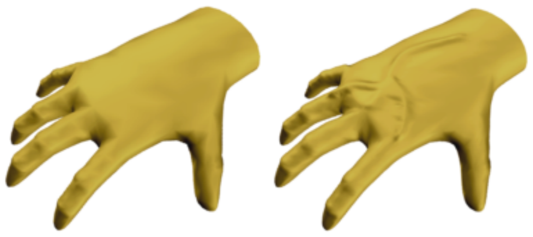


Figure 3.5: Results of applying surficial augmentation to generate vein-like features on a hand model [Ols+05].

Additive augmentation constructs new parts of a model, such as a limb or outcropping, from the user sketches, attaching them to the model in a location also specified by the user. As an example the extrusion operator in Teddy [IMT99] uses a circular stroke to initiate operation and define the connection zone. The next sketch defines the contour of the new part, which is inflated and attached to the original model in the connection zone.

In the ShapeShop system, Schmidt et al. [Sch+06] used implicit surfaces that provide blending capabilities in neighbouring surfaces to implement additive augmentation. The user sketches a new surface in the proximity of the existing model and the implicit surface representation naturally blends both together.

3.3.2 Deformation Systems

Lastly, deformation systems are those that support operations such as cutting, bending, twisting, creating holes through the model, segmentation, free-form deformation and affine transformation.

Most complex deformations are based on the idea of oversketching, where the user draws over an existing stroke with the intention of refining or deforming it. Bending and twisting can be achieved by matching a reference stroke to a target stroke and deforming the model in a similar fashion. An example is the system proposed by Cherlin et al. [Che+05] where the user can generate variations of existing models using orthogonal deformation strokes, that take as reference a previously drawn stroke. A different oversketching approach uses contour lines extracted from the model as reference [Nea+05].

The FiberMesh system presented by Nealen et al. [Nea+07] allows the user to elastically manipulate 3D curves that deform the model by changing the constraints in the optimisation algorithm that generates the surface.

Free-form deformation, where a lattice is created surrounding the model and modifications to lattice points are translated to the model, has also been used. Draper and Egbert [DE03] have proposed a sketch based interface for manipulating these lattices. This is an extension to the functionalities of Teddy [IMT99], allowing bending, twisting and stretching.

3.4 Sketch Based Modelling of Urban Entities

Sketch based interfaces for modelling have also been applied to the field of architectural model generation. The Stretch-A-Sketch system [Gro94] allowed the user to sketch building plans by identifying and maintaining spatial relations in hand drawn diagrams while the user moves and resizes the elements. The system is aimed at bridging the conceptual design phase to the more precise CAD-like drawing and editing.

Juchmes and Leclercq [JLA04] introduced an agent-based approach to the interpretation of architectural sketches of floor plans, where each agent is aims at recognising a different kind of element from the user strokes. Different agents, however, can claim that one stroke is part of a different element, resulting in a conflict. To resolve this, agents communicate and reach an agreement based on recognition probability rates.

Chen et al. [Che+08b] presented a system to generate 2.5D models of buildings from a single user drawn sketch. Not only can the user sketch the overall building structure, but is also able to introduce detailed geometry and textures in the final result. This is achieved by using strokes to fetch these elements from a database. However, the system assumes that the building consists of three groups of edges that are mutually perpendicular in 3D space. Another drawback is that the results are only 2.5D and do not constitute a complete model. Nonetheless, the system is capable of generating very detailed and aesthetically pleasing models as seen in Figure 3.6.

Olsen et al. [Ols+11] conducted a user study to learn how inexperienced users draw several types of buildings and, based on the results, proposed an algorithm which analyses the sketch input, extracts shape and detail information, predicts the building type, creates 3D

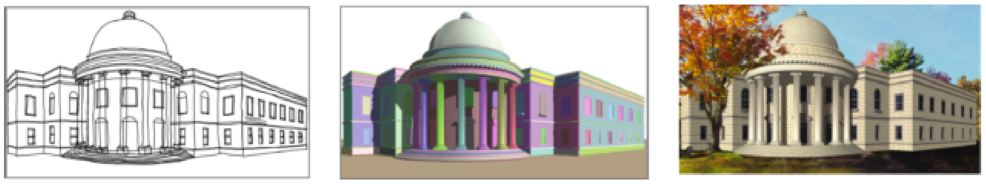


Figure 3.6: From a single user sketch (left) the system extracts the building structure (centre) and provides a detailed textured result (right) [Che+08b].

models using three different reconstruction techniques, and adds building details with a displacement texture. However, the results appear quite simplistic as can be seen in Figure 3.7.

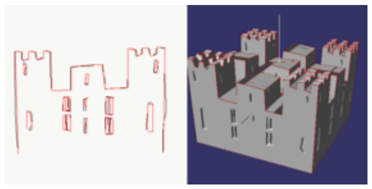


Figure 3.7: Example of a castle sketch being transformed into a 3D model [Ols+11].

More recently, the main contribution has been the work by Nishida et al. [Nis+16]. The presented system uses convolutional neural networks to recognise a user sketch and estimate the respective parameters in order to translate the user intention into a CGA rule. As any neural network requires training before being able to recognise any given input, the system is only able to translate sketches for which it has been trained.

3.5 Open Questions

Although research has shown several interesting results, sketch based interfaces for modelling still have several open questions which need further research. They are still far from being a total replacement of current modelling systems despite the fact that the underlying metaphor (i.e. paper and pencil) has a great level of expressive power and accessibility. The main problem lies in the amount of ambiguity a given sketch has. To one level, we have the depth ambiguity which is also a recurring problem in 3D reconstruction from images. In another, there is also ambiguity in inferring the user intention from the drawn strokes. This subsection enumerates some open questions in the SBIM paradigm.

Sketch based interfaces for modelling, obviously, originate from the paper and pencil medium. However, while on the traditional medium the user can freely draw strokes in any arbitrary order, most current interfaces require the user to draw in a specific manner, reducing the immersion and ease of use [Ols+09].

Also, sketch based interfaces for modelling often rely on guessing and inferring the user intentions and, when the system fails to acknowledge the correct operation, it can cause frustration in the user. Implementing interfaces that provide stable and predictable interactions is still a large challenge [Ols+09]. Usually, current systems provide expectation lists to help the user in the resolution of this kind of problems.

Lack of self-disclosure is another problem in sketching interfaces. While in WIMP interfaces the user can explore the interface to discover what the several menus and buttons do, SBIM systems often present only a white canvas, not disclosing any clues about how to use it. Also, these interfaces typically need the user to recall, from memory, how to execute an operation. In contrast, by navigating a WIMP interface the user can recognise the operation he wishes to perform.

Contrasting with traditional modelling systems, where the user can accurately define the positioning and orientation of 3D surfaces, SBIM systems may lack some precision. However, precision can be introduced if the user can specify geometric constraints such as parallelism, perpendicularity or line congruency. This is the case in most systems dedicated to engineering design systems while, on the other hand, free-form design systems precision is sacrificed in favour of simplicity [Ols+09].

Despite the several works presented here, there has been, so far, no connection between the procedural modelling techniques for urban spaces reviewed previously and sketch based interfaces for modelling. Given that some disadvantages (namely the lack of expressiveness and direct control) that procedural techniques have are characteristics in which SBIM excel, merging both concepts may prove to be an interesting research path.

3.6 Summary

Sketch based interfaces for modelling are based on a, much appreciated by artists, pen and paper metaphor which can enhance the creative process by alleviating the bottleneck between concept art sketches and the final three-dimensional model. However, the interpretation of the 2D sketches performed by users is a difficult task with a high level of ambiguity.

Buildings, on the other hand, have strict geometric constructive rules which coupled with the semantic information present in procedural rules can be used to better interpret the user sketches by reducing the number of possible interpretations.

This way, the interaction problems present in current procedural modelling techniques can be alleviated with the use of sketch based interfaces for modelling. Nonetheless, only a few works have mixed these concepts (e.g., [SS08; Lon+12]) and the connection between both worlds remains largely unexplored. An urban modelling system where procedural modelling rules can be specified with a sketch based interface for modelling can be seen as an improvement in the procedural modelling workflow. The proposed thesis aims to bring closer these methodologies, leveraging their advantages, in order to facilitate the expeditious modelling of urban environments.

4

Layered Shape Grammars

One of the identified problems with procedural modelling of buildings is their level of expressiveness which is mismatched with the level of expressiveness of sketch based interfaces for modelling. While in the latter there is a tendency to provide free-form modifications to a given model the former typically only provides modifications based on split lines and planes, limiting the user creativity.

This chapter describes a solution to shorten the expressiveness gap between procedural modelling and sketch based interfaces for modelling by introducing two extensions to shape grammars: description of layers within planar surfaces (e.g., façades) and the possibility to vectorially define shapes within these layers allowing, for example, arched elements. By doing so, the methodology is capable of generating non-trivial layouts, going beyond the regular grid structure, and seamlessly integrate more complex architectural elements such as arched doors and windows.

4.1 Overview

Procedural modelling of buildings is often achieved with shape grammars ([PM01; Won+03; Mül+06; KPK10]) or with graph based approaches ([Pat12; Sil+13; Sil+15]). In both cases, procedural methods encode the building generation in rules which are iteratively applied and replace one shape with a new set of shapes, incrementally adding detail to the models. The paradigm, however, is based on splitting operations that divide the geometry into parts that will be detailed separately.

Split-based approaches, therefore, impose too strict limitations on the geometry that can be procedurally generated. For example generation through split planes constrains the created shapes to be mostly rectangular and typically enforce a grid like structure. Some façades, however, don't possess such a trivial structure and, therefore, become quite difficult to encode with this paradigm without importing externally modelled assets. An example of this sort of façades is the Mikimoto Ginza building in Tokyo as seen in Figure 4.1.

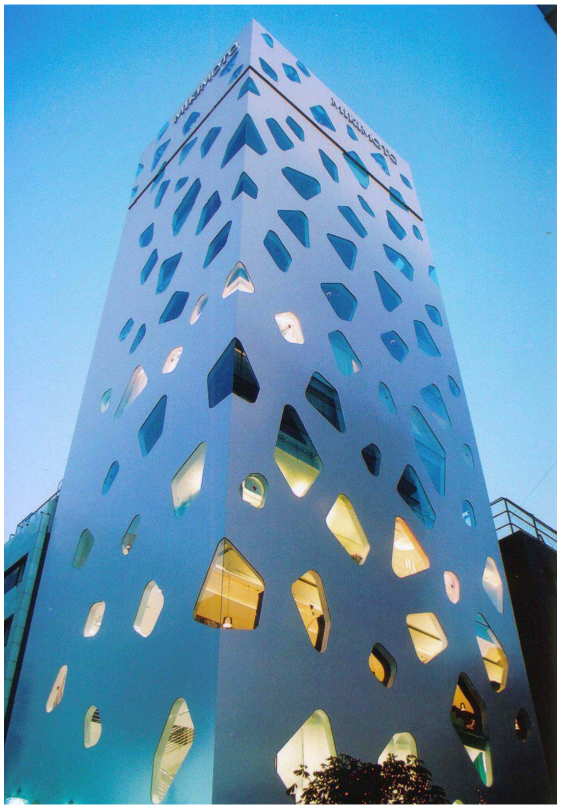


Figure 4.1: The Mikimoto Ginza building in Tokyo is an example of an irregular façade.

Introducing layers in planar surfaces allows more complex layouts and more compact and natural representations of façades. ([Zha+13]). Also, layers are common concepts in many 2D or 3D content generation tools, with which most users have some degree of familiarity. Moreover, the ability of reordering layers or toggling the visibility of their contents can lead to different results and a non-linear creative workflow allowing the exploration of greater variations of procedural models.

In split-based methods, moreover, complex geometry can often only be imported as externally modelled assets which are, sometimes, not easily integrated with the rest of the model. Allowing the vectorial definition of two-dimensional shapes within a model's planar surfaces has the capability of improving the expressiveness of procedural modelling methods. The paradigm is now capable of providing a means to specify more complex geometry within the shape grammar instead of requiring it to be imported. Also, it has the potential to form the basis of a more interactive procedural modelling system as the user can draw more complex shapes in the planar surfaces and the system can infer its vectorial representation (by means of, e.g., least squares minimization).

The system is based on shape grammars, similar to [Mül+06] and [KPK10], which were extended to introduce support for the creation of layers and the vectorial definition of shapes within layers. This allows the creation of more complex shapes within façades, and more variations with less modelling effort.

Firstly, the system achieves this by considering several types of procedural items: *volumetric shapes*, *planar shapes*, *layers* and *2D shapes* (Subsection 4.2.1). Planar shapes contain two-dimensional scenes where 2D shapes can be organised in layers and ordered by depth. In this system, the procedural operations are overloaded, meaning that their execution and output depends on the type of the input procedural items (Subsection 4.2.2).

Unlike described by Müller et al. [Mül+06], where the procedural rule execution scheduling is controlled by user defined priorities, this system first executes all rules that add two-dimensional detail to planar shapes, using the operations described in Subsection 4.2.3.

Once all 2D shapes have been created, the system merges all layers and 2D shapes in the planar shapes, taking into account occlusions between 2D shapes, creating a set of disjoint shapes, which are then converted into new planar shapes in three-dimensional space. This process is called *planar shape normalisation* and is described in Section 4.3.

Operations that lead to volumetric shapes can, then, be executed, which add 3D detail to the façades. This process is repeated until there are only terminal shapes.

4.2 Layered Shape Grammars

This section describes the extensions to shape grammars that allow the specification of layers and the vectorial definition of shapes. Specifically, it describes the available procedural item types, how rules in the system are overloaded and the introduced operations.

4.2.1 Procedural Items

Procedural rules in this technique operate on Procedural Items (PI) which are identified by a symbol and, as in other procedural modelling systems ([Sil+15; KPK10]), our methodology supports several types of procedural items. The set of all available procedural item types is named τ and is defined as in Equation 4.1.

$$\tau = \{VolumetricShape, PlanarShape, Shape2D, Layer\} \tag{4.1}$$

The remainder of this subsection defines each of these types in great detail.

VolumetricShape

Volumetric shapes represent shapes in three-dimensional space whose bounding box has a volume superior to zero. In other words, these are 3D shapes that may or may not represent a closed volume.

Each volumetric shape has its own frame of coordinates and size, which is named scope, similarly to the one defined by [Mül+06]. The scope is used to define sizes and axes which are used in the execution of procedural operations. Figure 4.2 illustrates a volumetric shape with its scope.

A volumetric shape V is, thus, defined by the tuple V as in Equation 4.2.

$$V = \langle G, M, P, R, S \rangle \tag{4.2}$$

Where G represents its geometry and M its material properties (e.g., color, texture, etc.). The scope's position is given by $P = (p_x, p_y, p_z)$ defining a point in 3D space, its size along

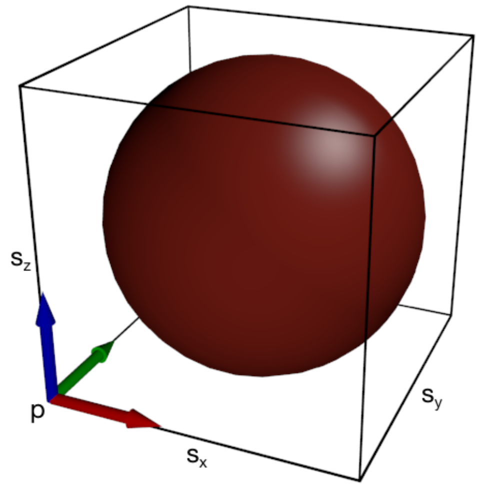


Figure 4.2: A volumetric shape with its scope.

each of its axis is represented by $S = (s_x, s_y, s_z)$ and the direction of each axis is defined by the R matrix in Equation 4.3. In this matrix, $X = (X_x, X_y, X_z)$, $Y = (Y_x, Y_y, Y_z)$, $Z = (Z_x, Z_y, Z_z)$, represent respectively, the direction of the scope's x, y and z axis.

$$R = \begin{bmatrix} X_x & Y_x & Z_x \\ X_y & Y_y & Z_y \\ X_z & Y_z & Z_z \end{bmatrix} \tag{4.3}$$

The matrix R defines a rotation from the world's frame of coordinates to the scope's frame of coordinates. Therefore, the matrix T, defined in Equation 4.4 is used to transform a point in the world space to a point in scope space:

$$T = \begin{bmatrix} X_x & Y_x & Z_x & 0 \\ X_y & Y_y & Z_y & 0 \\ X_z & Y_z & Z_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} X_x & Y_x & Z_x & p_x \\ X_y & Y_y & Z_y & p_y \\ X_z & Y_z & Z_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

The scope transform matrix (T) will be used to convert 2D shapes to planar shapes in the normalisation phase (see Subsection 4.3.2).

Volumetric shapes are used to define volumes in procedural modules, such as the building volume, columns, roofs and other shapes that cannot be contained in a plane. Volumetric shapes are often created by extruding a planar shape or 2D shape or by inserting premodelled assets. Splitting a volumetric shape also yields a set of volumetric shapes. This type of shapes can also be decomposed into planar shapes corresponding to each of its faces by using the component split operation, allowing each face to be further detailed.

PlanarShapes

In contrast, planar shapes, are shapes that have a single face contained in an arbitrary plane. As with volumetric shapes, planar shapes also have an associated scope. However, its size along the z axis is zero and its xy-plane is coplanar with the shape. Figure 4.3 illustrates a planar shape and its scope.

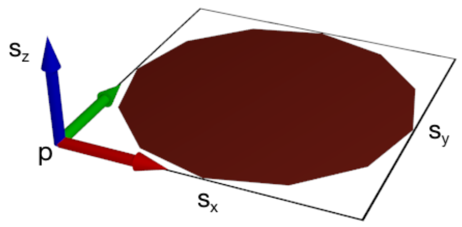


Figure 4.3: A planar shape with its scope.

Planar shapes are defined by the tuple

$$P = \langle G, M, P, R, S, Scene \rangle \tag{4.5}$$

Where G, M, P and R are similar to the volumetric shape definition and $S = (s_x, s_y, 0)$.

There is a 2D scene *Scene* associated with each planar shape, whose x and y axes are mapped to the scope's x and y axes.

This scene consists of a stack of layers, initially containing one layer with a 2D shape (the background shape), created by mapping the planar shape's 3D geometry into the 2D scene.

More formally, *scene* is defined by

$$scene = \langle L_0, L_1, ..., L_n \rangle \tag{4.6}$$

Where $L_i, i \in [0, n]$ represents a layer in the scene and $depth(L_j) > depth(L_{j+1}), j \in [0, n[$. In other words, layers within the scene are sorted by decreasing order of their depth.

Layers

Layers, as mentioned, are stacked by their depth within a 2D scene, where the bottom-most layer represents the deepest layer in a 2D scene. Each layer contains a list of 2D shapes sorted by depth. Figure 4.4 illustrates the relations between planar shapes, layers and 2D shapes.

Similarl to planar shapes, a layer is defined by a stack of 2D shapes sorted by decreasing order of depth:

$$layer = \langle S_0, S_1, ..., S_n \rangle \tag{4.7}$$

Where $S_i, i \in [0, n]$ is a *Shape2D* and $depth(S_j) > depth(S_{j+1}), j \in [0, n[$.

Layers are considered procedural items since they are created and manipulated by procedural rules.

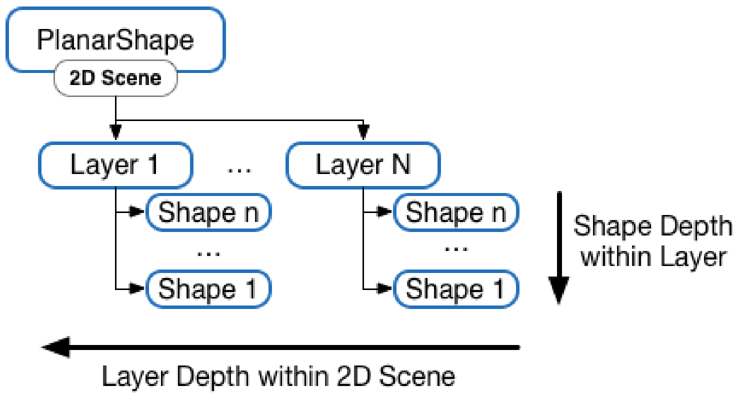


Figure 4.4: Relations between planar shapes, layers and 2D shapes.

Shape2D

Finally, a 2D shape is a bidimensional shape contained in a layer of a 2D scene. All shapes are clipped inside another shape (except for the background shape), which must be of a higher depth, and are possibly occluded by shapes of lower depth. The clipping and occlusion process is described in Section 4.3.

These procedural items have two boolean attributes that can be set independently. These are *placeholder* and the *fixed geometry* flags.

More formally, a 2D shape is defined by the tuple:

$$shape = \langle G, placeholder, fixed, clip \rangle \tag{4.8}$$

Where G is the shape's geometry (simply given by an ordered list of points), *placeholder* and *fixed* are the previously mentioned flags and *clip* is a list of 2D shapes that are to be clipped by a given shape, defining the clipping tree. Moreover, for a shape s and each shape $s_c \in s.clip$, $depth(s_c) < depth(s)$.

All 2D shapes are created as placeholder shapes, which means that they are derived normally but they will not be transformed into planar shapes during the planar shape normalisation (Section 4.3). Before the normalisation process, 2D shapes are automatically marked as non-placeholders if their derivation requires a planar shape.

For example, if the next operation to be applied to a 2D shape is an extrusion, then the shape is marked as non-placeholder. The shape is, consequently, converted into a planar shape which can be correctly extruded.

Placeholder shapes are used to take advantage of shape grammar's flexibility in representing complex architectural patterns by dividing, partitioning and positioning architectural elements in façades and other planar surfaces while, at the same time, not enforcing hard splits in the geometry. This has been identified as an opportunity to improve procedural modelling's expressiveness in Subsection 1.5.2.

The *fixed geometry* flag is used to control how the occlusion process affects each shape. During occlusion calculations (Subsection 4.3.1) if a planar shape with fixed geometry is occluded, then its geometry is discarded. If the flag is not set, the final geometry is calculated by subtracting the occluding geometry from its original geometry.

4.2.2 Overloaded Procedural Rules

The rules in the system are similar to those in regular shape grammars in the sense that they define the replacement of a predecessor shape with a set of successor shapes through the sequential execution of a list of shape operations, *RuleOps*. A rule *R* in this system is defined by

$$R = \langle Symbol, RuleOps \rangle \tag{4.9}$$

where *Symbol* is the rule's predecessor symbol and *RuleOps* is the list of procedural operations applied to each input shape.

However, because the system supports several types of procedural items, the operations are overloaded. This means that the same operation can have a different execution and can output procedural items of different types, depending on the type of input, while maintaining the operation semantics. Furthermore, it can restrict the application of an operation to some types.

The *split* operation, for example, can be applied to volumetric and planar shapes, outputting volumetric or planar shapes respectively. The *extrude* operation always outputs volumetric shapes, although it only accepts volumetric and planar shapes. The *create layers* operation (Subsection 4.2.3) only accepts planar shapes and outputs layers.

More formally, an operation O is a function

$$\begin{gathered} O : t_{in} \rightarrow t_{out}* \\ t_{in}, t_{out} \in \tau \end{gathered} \tag{4.10}$$

that acts on procedural items of type t_{in} and outputs zero or more procedural items of type t_{out}. Each operation has an associated type mapping $\omega_O : \tau \mapsto \tau$ that describes the type transformations the operation performs. The output type of an operation for a given input procedural item with type t is given by $\omega_O(t)$. If t is not in the domain of ω_O, then $\omega_O(t) = \varnothing$. Likewise, $\omega_O(\varnothing) = \varnothing$.

The output type of a procedural rule R given an input procedural item of type t is, then, given by the operator $out\_type(R, t)$ defined by the algorithm depicted in Listing 4.1.

```
function $out\_type(rule, item\_type)$:
  current_type $\leftarrow$ item_type
  for op in rule.RuleOps:
    current_type = $\omega_O$
  return current_type
```

Listing 4.1: Algorithm to calculate the procedural item type of a rule given a procedural item

Before applying a rule to a procedural item, the system runs the above algorithm and checks its result. A return value different from $\varnothing$ indicates that the rule can be executed with the given procedural item, since each operation $op_{i+1} \in RuleOps$ is able to manipulate the results of the operation executed before ($op_i \in RuleOps$). Otherwise, an error is raised and the execution is halted.

4.2.3 Procedural Operations

In addition to the operators introduced by Müller et al. [Mül+06] , a new set of operations that act on procedural items have been defined. The following sub sections describe their operation and define their syntax, loosely following the syntax presented by Müller et al. [Mül+06].

Layer Creation Operation

To create layers within planar shapes, the layer creation operation has been defined. Such operation accepts a series of successor rule symbols, creating a new layer for each symbol in decreasing order of depth. Successor rules can, afterwards, add further detail to each layer separately. The layer operation only accepts planar shapes as its input type and only outputs layers and, therefore, its mapping function ω_{layers} is given by the following:

$$\omega_{layers}(t) = \begin{cases} layer & \text{if } t = planarshape \\ \varnothing & \text{if } otherwise \end{cases} \tag{4.11}$$

For example, given a planar shape with the *Facade* symbol and representing a building façade, the following example shows the creation of two layers: one that will contain the façade's floors description and another defining the entrance.

```
Facade → layers("floors","door"){Floors | Door}
```

Listing 4.2: Rule to create two layers in a planar shape.

The n^{th} parameter of this operation defines the name of the n^{th} layer to be created in the planar shape. For each layer, a rule name must be given between the { and } delimiters, separated by a | character. Although it seems redundant, having a layer name and a rule name adds a level of semantics to this operation. The user can, for example, swap the rule for the *door* layer with another one (e.g., *GothicDoors*), while the semantics is still retained. Furthermore, it allows to externally turn on or off the generation of the layer via its composite name: *Facade.door*, increasing the variation of results without modifying the rule base.

Figure 4.5 illustrates the application of the previous rule to a planar shape with *Facade* symbol. It is possible to see the decomposition of the *Facade* planar shape (on the left) into

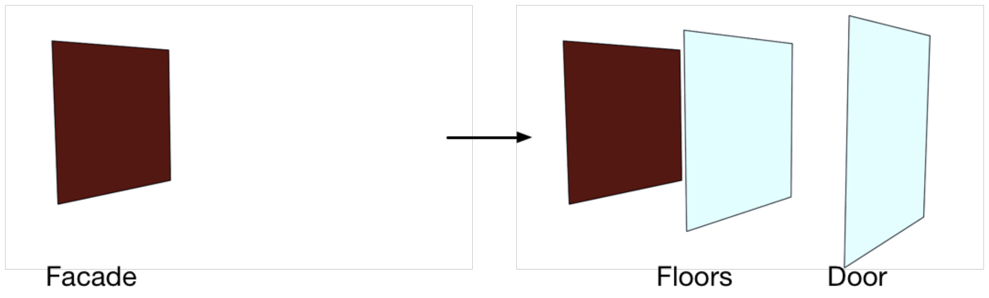


Figure 4.5: Application of the rule in Listing 4.2 to a planar shape.

the *Door* and *Floors* layers (on the right). As the layers are empty, there is no actual change of the geometry and the final result is similar.

Segment Operation

The segment operation is used to partition a planar shape, 2D shape or layer into 2D shapes along one of its axis (x or y), placing the resulting shapes into one of the 2D scene's layers and defining its depth within the 2D scene. Therefore, its type mapping function $\omega_{segment}$ is given by the following:

$$\omega_{segment}(t) = \begin{cases} Shape2D & \text{if } t \in PlanarShape, Shape2D, Layer \\ \varnothing & \text{if } otherwise \end{cases} \tag{4.12}$$

This operation is similar to the split operation introduced by Müller et al. [Mül+06]. However, the segment operation does not force a split in the geometry at the moment of its execution. Instead, it creates placeholder 2D shapes partitioning the given procedural item while the modifications to the final geometry are deferred to the planar shape normalisation phase (see Section 4.3).

Intuitively, this operation allows the same flexibility in architectural elements placement without incurring in an excessive number of splits and, therefore, semantically meaningless shapes. Moreover, keeping the geometry intact until a later moment allows the shapes contained in different layers to be correctly merged.

The segment operation defines each created 2D shape's depth within the scene as $d = depth_{pred} - 1$ where $depth_{pred}$ is the predecessor procedural item's depth, effectively creating 2D shapes that are in a less deep level than their predecessor procedural items.

As this operation accepts procedural items of three different types (PlanarShape, Shape2D, Layer), there are also three different execution types and the resulting shapes may be inserted into different layers and at different depths depending on the type of the predecessor procedural item.

If the predecessor is a 2D shape, then $depth_{pred}$ is that shape's depth, and the resulting shapes are placed within the same layer. This effectively creates 2D shapes above (i.e., at lower depth) than the predecessor.

Whenever the predecessor is a planar shape, $depth_{pred}$ is the 2D scene's background shape's depth and the shapes are inserted in the deepest layer. This situation represents the case when a planar shape is being segmented directly and, as a result, all shapes are directly above the background shape. Moreover, since the planar shape will be consumed from the list of non-terminals when the given rule is applied, this case also implies that the planar shape will have a single layer (the default layer) where shapes will be inserted.

If, however, the predecessor is a layer, then the shapes are included in that layer and $depth_{pred}$ takes the value of the minimum depth of the 2D shapes in the layer underneath or, if the predecessor is the bottommost layer, the background shape's depth. Intuitively, this case calculates the 2D shape's depth depending on the layer's position within the 2D scene.

This operation is also responsible for creating a clipping tree of a 2D scene's shapes, where nodes represent the 2D shapes and edges represent a clipping relation such that the geometry of children shapes must be contained inside the parent shape. In other words, geometry that lies outside the parent's shape will be clipped and discarded.

The clipping tree root node is the 2D scene background shape, which is guaranteed to exist for every 2D scene. When executing the segment operation, if the predecessor procedural item is a planar shape or a layer, then the 2D shapes output by the segment operation become children of the background shape. On the other hand, if the predecessor is a 2D shape, then the created shapes become children of the predecessor shape and are, thus, clipped within its geometry. The clipping tree will be used in the planar shape normalisation step (Section 4.3) to calculate the final geometry of each 2D shape and, as consequence, the planar shapes generated.

By default, this operation creates only placeholder shapes, meaning that they will not be converted into planar shapes unless their derivation leads to a procedural item of a type other than *Shape2D*. In other words, if next rule to be applied to a given 2D shape only produces results within the bidimensional domain then the shape is kept as a placeholder. In contrast, it will be marked as non-placeholder if the next rule creates a volumetric or planar shape, and the shape will be converted into a planar shape in the shape normalisation phase.

This keeps the background geometry intact for as long as possible allowing the correct and seamless integration of complex geometry into planar shapes and 2D shape occlusion during the planar shape normalisation step (Section 4.3).

As with the operations described by Müller et al. [Mül+06] , there is also a repeat segment operation. This is indicated by appending an asterisk character to the operation body (after the last } character). Moreover, segment sizes can also be defined as relative to the predecessor size (by prepending the value with a ' character) or as a float size (by using a ~ before the value) proportionally distributing the remaining size by all float segments. This is also present in the CGA grammar [Mül+06].

```
Floors → segment("Y", 3,2,2)
  { ε | Floor | ε }
Floor → segment("X", 3,2,3)
```

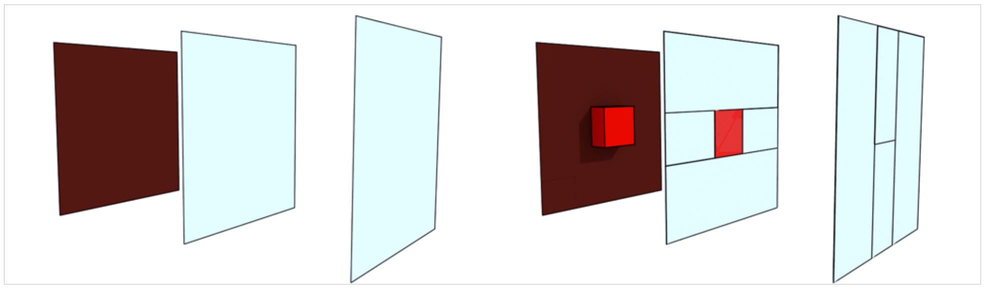


Figure 4.6: Application of the grammar in Listing 4.3 to the two layers defined.

```
  { ε | Balcony | ε }
Balcony → extrude(0.1)
Door → segment("X", 3,2,3)
  { ε | DoorVSegment} | ε }
DoorVSegment → segment("Y", 3.5, 3.5)
  { DoorExtrusion | ε }
```

Listing 4.3: Creates several segments in each layer.

Consider the Layered Shape Grammar present in Listing 4.3 that further details the previous example. This results in a façade with a single square extrusion centred in the middle floor. Note that the segment operation syntax is similar to the split operation described before and ε is the empty shape. No practical consequences result from including empty shapes since only non-empty, non-placeholder 2D shapes have influence on the final geometry (the reader is referred to Section 4.3 for more details). The results are illustrated in the right image in Figure 4.6 where the image in the left corresponds to the previous derivation state (right image in Figure 4.5).

As it is possible to see, the *Floors* layer has been vertically segmented into three different zones where the middle has been given the *Floor* symbol and the remaining were created as empty shapes. The *Floor* shape was then segmented horizontally three times and, similarly, the first and last segments are empty shapes whereas the middle shape has the *Balcony* symbol. The *Door* layer was segmented horizontally three times, where the second segment (the *DoorVSegment* shape) was again segmented into the *DoorExtrusion* shape and an empty shape.

In this scenario, only the *Balcony* shape results in modifications to the final model since its derivation leads to a volumetric shape via the extrude operation and the remaining empty shapes are used as padding to correctly place the balcony.

Vectorial Shape Operation

The vectorial shape operation is used to create complex 2D shapes within other shapes, taking as input the definition of a vectorial shape representing a closed path, allowing specifing complex architectural elements such as arched windows or doors. In this implementation the *Scalable Vector Graphics* (*SVG*) was chosen . The SVG specification creates a series of control points which are then interpreted either as vectorial curves (e.g., quadratic or cubic curves) or as straight lines connecting two control points.

Afterwards, the vectorial definition is sampled down to a polygon P using an user specified sampling factor s, allowing to control the level of detail. The sampled points $p_i(x, y) \in$

P, where $x, y \in [0, 1]$ and $i \in [0, s[$ are used to create a polygonal 2D shape by interpolating them inside the predecessor shape's bounding rectangle, thus the same vectorial shape definition aligns with any predecessor shape's bounding box.

The vectorial shape operation, like the segment operation, takes as input procedural items of type planar shape, layer and shape 2D and, therefore, its type mapping $\omega_{vectorial}$ is defined as

$$\omega_{vectorial}(t) = \begin{cases} shape2D & \text{if } t \in \{PlanarShape, Shape2D, Layer\} \\ \varnothing & \text{if } otherwise \end{cases} \tag{4.13}$$

The layered shape grammar in Listing 4.4 illustrates the creation of an arched door in the previous examples. The first argument to the vectorial operation is the SVG path specification and a second, optional, argument is the sampling factor indicating how many points are to be sampled from the path.

As depicted in the right image in Figure 4.7, the extrusion is no longer generated since, by default all 2D shapes are created having fixed geometry meaning that in case of being occluded by a shape with a smaller depth they are discarded. Because the door is generated in a less deep layer, it takes precedence. Note, however, that the respective 2D shape is still generated in the *Floors* layer.

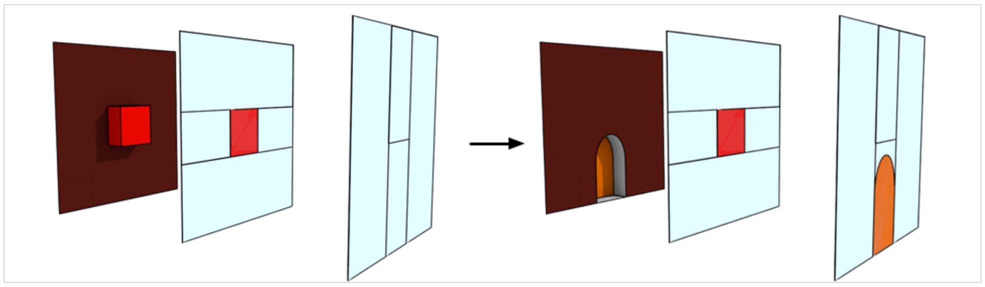


Figure 4.7: Application of the grammar in Listing 4.4 to create an arched door.

```
Door → segment("X", 3,2,3)
  { ε | DoorTile | ε }
DoorTile → segment("Y", 4,~1)
  { ArchedDoor | ε }
ArchedDoor →
  vectorial("M0,0 L0,0.5 C0,1 1,1 1,0.5 L1,0Z", 50)
  extrude(-0.1)
```

Listing 4.4: Demonstration of the vectorial operation use.

Set Property Operation

Similar to other procedural modelling frameworks, layered shape grammars provide an operation to set properties on procedural items. This is especially useful, in the context of layered shape grammars, to give the user the possibility to define 2D shapes with non fixed geometry. Thus, the user gains control over which shapes are to be discarded in case of being occluded by another shape and those that are to be modified to accommodate to the occlusion.

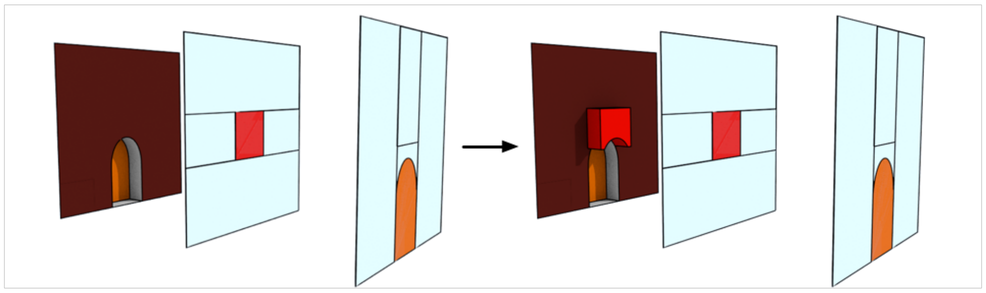


Figure 4.8: Application of the rule in Listing 4.5 to a 2D shape causing it to be generated.

Regardless of the input procedural item's type, this operation always outputs the same type and, therefore, its type mapping $\omega_{set\_property}$ is the identity mapping as given by:

$$\omega_{set\_property}(t) = t \tag{4.14}$$

The layered shape grammar extract in Listing 4.5 sets the *fixed_geometry* property to false in the *Balcony* rule before the extrusion operation is applied, causing the balcony's geometry to be modified adapting to the arched door (see right image in Figure 4.8).

```
Balcony → set_property("fixed_geometry", false)
  extrude(0.1)
```

Listing 4.5: Rule to set the fixed geometry flag in a 2D Shape.

The generated clipping tree for the final model in Figure 4.5 is given in Figure 4.9. As it is possible to see, the depth of the arched door (*ArchedDoor*) is lower than the balcony shape (*Balcony*). Therefore, and since the balcony's fixed geometry flag has been unset, the balcony adapts to the arched door's geometry.

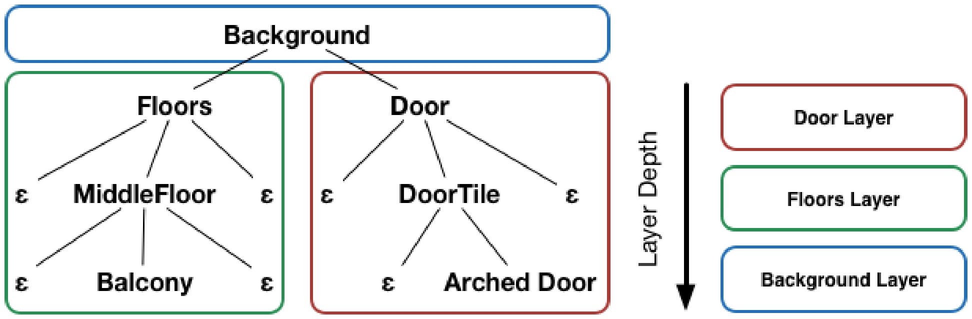


Figure 4.9: Clipping tree for the final model in Figure 4.5.

4.3 Planar Shape Normalisation

The normalisation of a planar shape is a process that has the goal of converting the 2D scene associated with each planar shape to a new set of planar shapes which will, afterwards, be available to further derivation. In the process, each 2D shape that is non-empty and non-placeholder is also transformed into a planar shape. This way the procedural derivation can continue to add volumetric detail, as the shapes that were solely within the bi-dimensional domain now have a tridimensional representation. Operations such as extrusion or splits require procedural items which have a 3D geometry (i.e., planar shapes and volumetric shapes).

In this methodology, the normalisation process is triggered whenever the derivation process encounters a special non-terminal procedural item whose type is *NormalisationToken*. By carefully scheduling the execution of non-terminals (as described in Section 4.4), this token separates the processing of 2D shapes from the remaining procedural items.

This procedural item is automatically created as soon as a 2D shape is created within any planar shape and the system guarantees that there is at most one present at a time, which is discarded after the normalisation process. The system creates as many 2D shapes as possible before normalising the planar shapes (see Section 4.4). Afterwards, the derivation of the remaining procedural items may lead to new 2D shapes and the process is repeated.

The process consists of two stages for each planar shape. The first is to merge all layers in the 2D scene, calculating occlusions between shapes. This process can, usually, be found in 2D graphics software. The second stage is to convert these 2D shapes to planar shapes in three-dimensional space.

More formally, the normalisation process is a function such that:

$$normalise : PlanarShape \rightarrow \langle Bg_{shape}, Front_{shapes} \rangle \tag{4.15}$$

which takes a PlanarShape p, along with its 2D scene and shapes, and transforms it into a disjoint set of shapes such that $p \equiv Bg_{shape} \cup Front_{shapes}$. The Bg_{shape} represents the converted background shape in the 2D scene while $Front_{shapes}$ is a set containing the remaining planar shapes.

After the normalisation process the background shape is marked as a terminal shape while the remaining shapes are considered non-terminals. The reasoning being that background shape represents the original planar shape which has already been derived. On the other hand, the remaining shapes were created afterwards and still require further derivation (i.e., the goal of the normalisation process).

4.3.1 Layers Merge

The first step in the process of merging all layers is to sort all 2D shapes by increasing order of depth, taking into account the shape's depth within a layer and the relative position of such layer. This results in a list L where the last element is the background shape:

$$\begin{gathered} L = (s_0, s_1, \ldots, s_n), n \in [0, |shapes|] \\ depth(s_j) < depth(s_{j+1}), j \in [0, |shapes|[\end{gathered} \tag{4.16}$$

All non-terminal 2D shapes are set as non-placeholder shapes if the rule that will immediately derive it produces a procedural item other than *Shape2D*, guaranteeing that only shapes that will be derived further into procedural items with a volumetric geometry are converted into planar shapes. In other words, given the FutureType function of a procedural item (defined in Listing 4.6) then $s_j.placeholder = false$ if, and only if, $FutureType(s_j) \neq Shape2D$.

```
function FutureType(PI):
  rule ← rule whose symbol matches PI.symbol
  type ← PI.type
  return out_type(rule, type)
```

Listing 4.6: Pseudo-code for the FutureTypefunction.

The clipping tree is traversed in depth first order, starting with the Bg_{shape} and the 2D shape of each node is clipped inside its parent shape. Enforcing shapes to be contained within its parent shape is a requirement since some operations may allow the creation of geometry outside this boundary. Most notoriously, the vectorial operationcan easily create such situations and, while a solution could be to constrain all control points to be within the boundaries (i.e., enforcing all $c_i(x, y)$, where $x, y \in [0, 1]$ and c_i is a control point), this would severely limit the geometries that could be defined. The employed solution was, therefore, to clip the geometry after it has been defined. The clipped geometry g' for each shape s is, then, given by:

$$\begin{aligned} g' = geometry(s) \cap geometry(parent(s)) \\ s \neq Bg_{shape} \end{aligned} \tag{4.17}$$

The final 2D geometry of each shape is, then, calculated by iterating L, performing occlusion calculations with a mask shape M, while considering the *placeholder* and *fixed geometry* flags. The final shape is calculated as $s'' = occlude(s', M)$, where *occlude* is a function defined in Equation 4.18.

$$occlude(s, M) = \begin{cases} \varnothing & \text{if } placeholder(s) \\ geometry(s) & \text{if } geometry(s) \cap M = \varnothing \\ \varnothing & \text{if } fixed\_geometry(s) \wedge geometry(s) \cap M \neq \varnothing \\ geometry(s) \setminus M & \text{if } \neg fixed\_geometry(s) \wedge geometry(s) \cap M \neq \varnothing \end{cases} \tag{4.18}$$

The mask shape is constructed by accumulating the occlusion results such that, for each $s_i \in L$, the resulting normalisation mask is given by $M_{i+1} = M_i \cup occlude(s_i, M_i)$.

Since placeholder shapes do not have any impact on the final model their geometry can be safely discarded. If the shape does not intersect the mask then there is no occlusion and the shape's geometry is not modified. If there is an occlusion the geometry is discarded if the shape is marked as fixed geometry. On the other hand, the geometry of shapes not marked as fixed geometry is calculated by subtracting the mask from its original geometry.

The clipping and occlusion process can, eventually, lead to shapes being split into two or more shapes. In these situations, the resulting shapes are considered separate shapes, and will lead to different planar shapes with the same symbol.

4.3.2 Conversion to Planar Shapes

The conversion of a 2D scene associated with a planar shape P into a set of new planar shapes, is achieved by noting that the scene is actually embedded in the plane defined by the X and Y axis of P's scope.

There is, thus, a trivial mapping $\varphi_P : \mathbb{R}^2 \mapsto \mathbb{R}^3$ that transforms points in a 2D scene in a planar shape P into points in the 3D scene. Given a 2D point within the planar shape's associated 2D scene $p(x, y) \in scene2D(P)$, its world position is calculated with the φ_P function in Equation 4.19, where X, Y and Z are the scope´s axis as defined in Equation 4.3.

$$\varphi_P(p) = \begin{bmatrix} X_x & Y_x & Z_x & P_x \\ X_y & Y_y & Z_y & P_y \\ X_z & Y_z & Z_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_x \\ p_y \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} X_x \cdot p_x + Y_x \cdot p_y + P_x \\ X_y \cdot p_x + Y_y \cdot p_y + P_y \\ X_z \cdot p_x + Y_z \cdot p_y + P_z \\ 1 \end{bmatrix} \tag{4.19}$$

For each 2D shape s in a planar shape P that is suitable to be converted into a new planar shape (i.e., non-empty, non-placeholder), the system converts its 2D polygon (i.e., $P_{2D} = polygon(s)$) into a 3D polygon P_{3D} by using φ_P as defined in Equation 4.19. Having a 3D polygon it is then possible to create a new planar shape P' such that $geometry(P') = P_{3D}$.

$$\begin{gathered} P_{3D} = \langle \varphi_P(p_i) | p_i \in P_{2D} \rangle \\ i \in [0, |P_{2D}|[\end{gathered} \tag{4.20}$$

For each new planar shape a new scope must also be created. This is achieved by taking a copy of the original planar shape's scope such that $scope' = scope(P)$. This scope is then translated to the point that is nearest to its origin using vector $\vec{t}$ as given in Equation 4.21.

$$\begin{gathered}\vec{t} = \operatorname*{argmin}_{p} \textit{length}(\textit{position}(\textit{scope}') - p_i), p_i \in P_{3D} \\ \textit{position}(\textit{scope}') \leftarrow \textit{position}(\textit{scope}') + \vec{t}\end{gathered} \tag{4.21}$$

The size of the new scope must also be adjusted to tightly fit the new planar shape's geometry. Therefore, the sizes along the *x* and *y* axis are defined by measuring the distance to the farthest point in each direction. As, by definition, the planar shape is contained within a plane orthogonal to its scope *z* axis, the size along such direction is always zero. The size for each axis is defined in Equation 4.22.

$$\begin{aligned}\textit{size}_i &= \operatorname*{argmax}_{p}(\textit{position}(\textit{scope})_i - p_i), i \in x, y \\ \textit{size}_z &= 0 \\ \textit{size}(\textit{scope}') &\leftarrow (\textit{size}_x, \textit{size}_y, \textit{size}_z)\end{aligned} \tag{4.22}$$

As a final step, the 2D shape's symbol is copied to the new planar shape. The planar shape can now be further detailed through the normal derivation process.

4.3.3 Deferred Procedural Items

Having the system being responsible for automatically deciding when is the best moment to normalise planar shapes frees the user of the burden of crafting the shape grammar such that all 2D shapes are created before the generation of volumetric procedural items takes place. However, this comes at the expense of some loss of control over the generation process.

To achieve a greater control of when the normalisation process takes place, it is possible to circumvent the normal rule execution scheduling (see Section 4.4) and defer the derivation of a procedural item until after a normalisation has occurred. These items are, thus, called deferred procedural items.

This can be encoded in the grammar by prepending the @ character before a successor symbol. After the normalisation process occurs, all deferred items are unmarked as such and their derivation can continue as normal. Deferring a procedural item is particularly useful when segment operation sizes must consider the final size of a planar shape instead of its size before normalisation.

Consider, for example, the case of a simple façade being decomposed into two layers (as described in Listing 4.7). The deepest layer (rule *Windows*) generates the windows in such way that their layout corresponds to a regular grid. However, the top-most layer (rule *Frames*) generates the façade frames which partially overlap the first and last column of windows. A single floor in this façade is illustrated in Figure 4.10 which shows the leftmost and rightmost windows incorrectly generated.

```
Start → layers("windows", "frames")
  { Windows | Frames }
Frames → segment(x,1,~1,1)
  { extrude(0.1) | ε | extrude(0.1) }
Windows → segment(x,0.5,2,0.5)
  { ε | RepeatV | ε }*
RepeatV → segment(y,~1,2,~1)
  { ε | extrude(-0.1) | ε }
```

Listing 4.7: Layered Shape Grammar demonstrating the use of deferred procedural items.

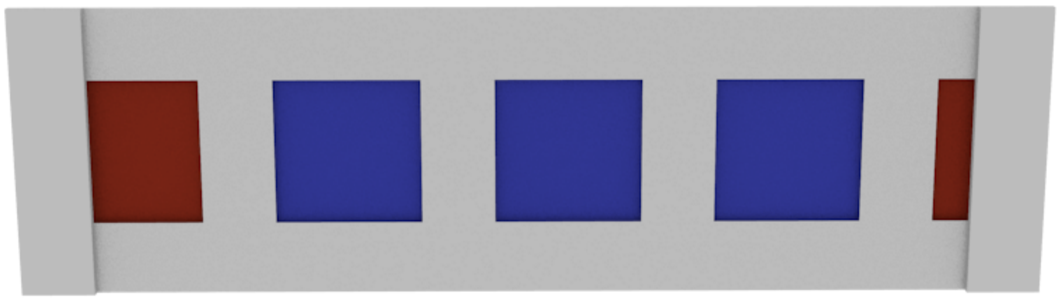


Figure 4.10: Incorrectly generated windows.

By simply inserting a @ character before the *Windows* successor rule in the *Start* rule (Listing 4.8) delays the derivation of the *Windows* rule to after the normalisation process. Therefore, the procedural item passed as input to this rule is no longer a 2D shape (as in the previous scenario) but is a planar shape which has been occluded by the frames. Thus, this shape's size correctly corresponds to the part of the façade where windows should be placed.

As can be seen in Figure 4.11, the windows are now generated taking into account the size of the planar shape, avoiding the undesirable intersections.

```
Start → layers("windows", "frames")
  { @Windows | Frames }
```

Listing 4.8: Layered Shape Grammar demonstrating the use of deferred procedural items.

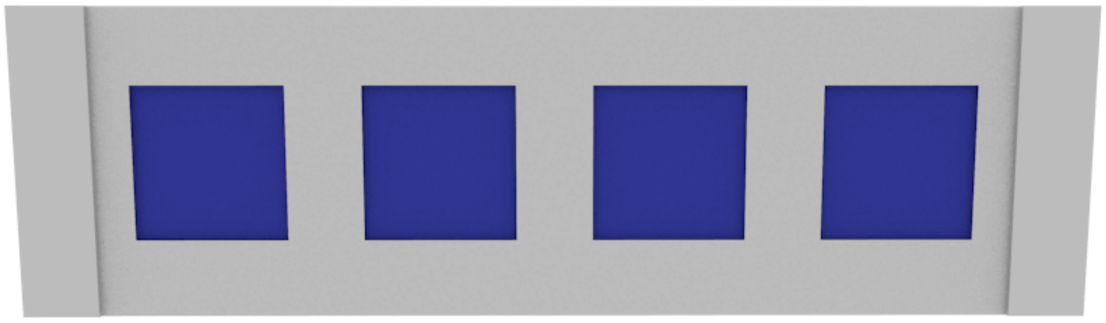


Figure 4.11: Correctly generated windows.

4.4 Derivation Process

As with other shape grammars, the derivation process starts with an arbitrary configuration of procedural items. The shapes present in such configuration are, necessarily, planar or volumetric shapes, since they have a three-dimensional representation. Most often, however, they are planar shapes representing the building outlines.

The main scheduling of the derivation of non-terminal procedural items is based on a priority queue which sorts the items based on the characteristics of each non-terminal.

Intuitively, the priority queue gives more priority to procedural items whose derivation yields a 2D shape.

The procedural items on the initial configuration are, then, inserted into such priority queue Q of non-terminals, which sorts the items in decreasing order of their priority as given by the *priority* function (defined in Equation 4.23), where *FutureType* indicates the type of the resulting items after *item* has been derived.

$$priority(item) = \begin{cases} 3 & \text{if } FutureType(item) = Shape2D \\ 2 & \text{if } item.type = NormalisationToken \\ 1 & \text{if } item.deferred \\ 0 & \text{if } otherwise \end{cases} \tag{4.23}$$

Therefore, procedural items whose *FutureType* equals *Shape2D* have a higher priority, followed by the *normalisation token*, the deferred shapes and finally the remaining procedural items. Figure 4.12 illustrates this queue. Therefore, the system creates all 2D shapes possible, ensuring that 2D scenes are fully detailed and the normalisation process has all available clipping and occlusion data to generate correct results. The generation of geometry evolves naturally from less detailed to more detailed models.

The derivation algorithm is detailed in Listing 4.9 which defines the *derivation* function. In the listing, the third line retrieves and removes the next non-terminal procedural item

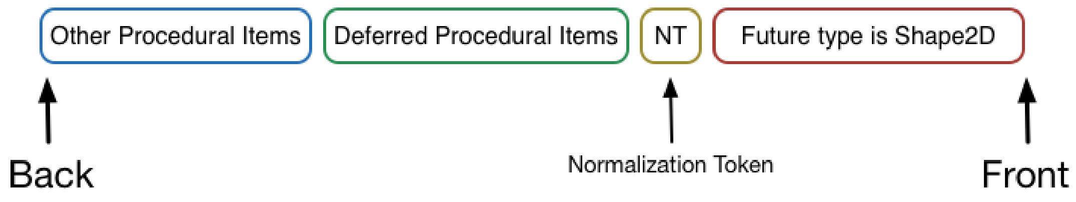


Figure 4.12: Priority queue for the procedural derivation. *NT* represents the normalisation token.

S from the priority queue *NT* and, afterwards the procedural rule that matches the item's symbol is fetched (line 4). Lines 5 and 6 check if the rule can be executed with the given item type, in which case, the rule is executed resulting in a new list of items (line 7). This list is iterated and the algorithm checks if each item is a terminal or not, by querying the rule base for a matching symbol on the left side of a rule. If an item is non-terminal, it is inserted into *Q* for further processing (line 10). A normalisation token is created as soon as a procedural item whose type is *Shape2D* is generated and there is no normalisation token present in *Q* (lines 11 and 12).

```
1  function derivation($Q$):
2   while($Q$ not empty)
3     S ← $Q$.pop()
4     rule ← fetch_rule(S.symbol)
5     if(out_type(rule,S.type) = ∅)
6       error_and_exit()
7     newItems ← rule.execute(S)
8     for(item in newItems)
9       if(non_terminal(item))
10        $Q$.push(item)
11        if(¬ has_normalisation_token($Q$) and type($item$) is Shape2D)
12          $Q$.push(new normalisationToken)
```

Listing 4.9: Derivation process algorithm.

Including the normalisation token as a non-terminal procedural item, guarantees that the planar shape normalisation process occurs in the right moment, while keeping the derivation process simple and similar to other approaches (e.g., [Mül+06]). The main difference between the derivation scheduling in layered shape grammars and regular shape grammars lies in the fact that, for example in the CGA grammar [Mül+06], the user assigns a priority to each rule, whereas in this system the priorities are inferred from the characteristics of the procedural items.

4.5 Limitations

Although layered shape grammars allow a greater flexibility in placing geometry within planar shapes, this system currently does not offer any explicit way to constrain alignments between elements in different layers. The current solution is to parametrise the rules to encode these alignments. However, the normalisation process provides a synchronization point during the procedural derivation where 2D shapes can be modified (e.g., translated or resized). This would allow the system to perform alignments with procedural items already created.

The manual input of a SVG path to vectorially define shapes is an error prone task. However, this can be alleviated in a number of ways. One is to externally define and import these paths, resembling an asset import operation. A more elaborate solution is to provide the designer with tools to sketch the shapes over the models in an interactive procedural modelling application. As the end goal of this thesis is to provide an integration between the procedural modelling of buildings and sketch based interfaces for modelling this represents the connection between layered shape grammars and the sketch based interface defined in Chapter 6.

As with any shape grammar approach to the procedural modelling of buildings, this methodology suffers from the same drawbacks. For instance, these are text-based which is impractical for artists [Pat12] and have limitations in its readability and manageability [Sil+15]. The grammar could be translated to a visual language as presented by Patow et al. [Pat12] or by Silva et al. [Sil+13].

4.6 Summary

In the procedural modelling of buildings, most current methodologies rely on a split-based paradigm, imposing too strict limitations in the generation of geometry such as mostly rectangular shapes and grid-like façades. To model more realistic and more interesting buildings, the current split-based methods may require several more rules and attention in their definition as modifications in one part of the rule base may have repercussions on unrelated areas of the model.

Therefore, this chapter presented an extension to shape grammars that allows the specification of layers within planar surfaces of buildings. Splitting the definition of rules into different layers helps minimise this problem as each layer is self contained and layer merge process produces well expected results.

Moreover, the layered shape grammar approach permits more complex layouts that go beyond regular grid structures. Meanwhile, this concept can, easily, provide a non-linear design workflow and produce more variations with less effort. Such workflow can be achieved by toggling on or off the execution of specific layers or reordering them within the planar shape.

In most procedural modelling methods, complex geometry must be imported as a pre-modelled assets. Layered shape grammars take a step further in procedural expressiveness by allowing the vectorial definition of 2D shapes within layers. Such vectorially defined shapes can be used to create non-rectangular structures within a bounding rectangle and can easily adapt to the geometry produced by less deep shapes.

5

Generalised Selections for Improved Direct Control in Procedural Modelling

As seen, procedural modelling techniques reduce the effort of creating virtual models of urban spaces but, as highlighted in Subsection 1.5.3, current methodologies are not prone to applications with direct user control over the generated models. Direct control in this scenario implies that a user should be able to directly perform operations on the resulting model, as in traditional modelling tools (e.g., Blender, Maya, 3D Studio Max). To achieve this, being able to select shapes and geometries is, more often than not, required.

However, in procedural modelling techniques establishing selections of shapes in the resulting models becomes rather difficult due to the problem of semantic ambiguity of user selections. Due to the data amplification nature of procedural modelling techniques, several configurations of shapes can be achieved given the same rule base by varying its parameters. This implies that given two configurations of shapes, a set of user selected shapes in one configuration may not be easily achieved in the other configuration.

Specification of user selections must, then, be defined based on the semantics behind the user intention instead of being a simple set of shapes. To achieve this end, this chapter describes a selection mechanism based on semantic queries over shape trees which can be applied to several shape configurations resulting in semantically coherent selections. This chapter also shows how it is possible, based on an initial shape tree, to iteratively apply a semantic queries and procedural operations interleaved to create procedural models. Such a procedural modelling paradigm is named *selection action* throughout this thesis.

However, the specification of such semantic queries is not always trivial (and is currently text based resembling a query language) and poses a subset of the problems this thesis aims to solve (e.g., achieve more interactive procedural modelling techniques). Therefore, a mechanism to infer semantic queries from user selections over a given procedural model is also presented in this chapter. Such mechanism uses genetic algorithms to drive a population of semantic queries towards what may be the user intention.

5.1 Overview

The main idea behind the selection-action paradigm is to help users select specific parts of the shape tree (and therefore buildings), and operate on them as input for procedural operations. Selections are defined by *semantic queries* (i.e., the *selections*), which result in sets of selected shapes to which procedural operations are applied (i.e., the *actions*). The system allows users to define rules which apply this process iteratively, starting with an initial configuration of shapes, and incrementally improving the detail of the created models.

Users are offered a quite simple interaction metaphor which is illustrated in Figure 5.1. First, users simply have to *select* a subset of the shapes to modify. The system, then, presents several possible selection sets (i.e., a set of queries that result in selections that contain the one provided by the users) generated from the initial user selection with the genetic algorithm. From these sets, users choose the one that bests suits the present needs. Having selected a query, users have to apply an *action* on the selected geometry by choosing a procedural operation to perform, effectively creating a selection-action pair. Finally, users can change operation parameters to adjust the output's form and appearance. All this is done in an iterative process, where the users interact with the system until the building is finished.

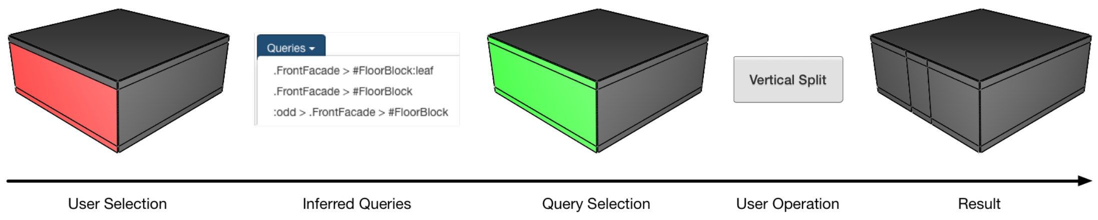

Queries
.FrontFacade > #FloorBlock:leaf
.FrontFacade > #FloorBlock
:odd > .FrontFacade > #FloorBlock
Vertical Split
User Selection
Inferred Queries
Query Selection
User Operation
Result

Figure 5.1: The user workflow in this system.

The user is presented with the interface shown in Figure 5.2, which contains an interactive view of the current model allowing selecting multiple shapes (a). It is also possible to manually create or refine semantic queries (b). The selection-action tree is shown in the bottom left (c) while the query and parameters for the selected node can be edited in the top right (d and e, respectively). The available procedural operations and templates are shown in the bottom right (f). The interface also features a suggestive interface (g) allowing the user to select candidate queries based on the selected shapes.

The basic semantic queries, here called *selectors*, are combinations of atomic query conditions capable of selecting shapes based on a set of given criteria. Different selectors can also be combined to exploit a building's hierarchical structure. These queries represent semantic selections over the shape tree, selecting shapes based on the type of procedural operation that resulted in its generation, their symbols, tags or more complex conditions (named filters) such as their facing direction or whether they are inside a given geometry or

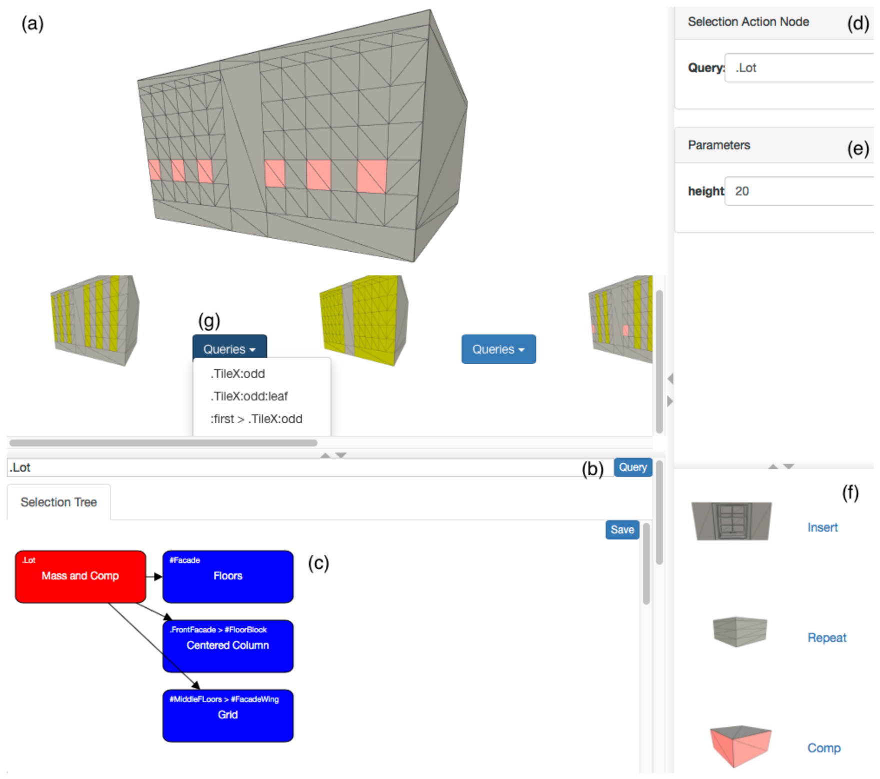


Figure 5.2: The interface presented to the user.

not. Moreover, the user can express conditions based on the hierarchical structure of the shape tree. Semantic selections allow the user to express complex selections such as selecting all windows in the north façade, the windows that are in even floors, or those that are inside a given geometry.

The selection-action system, therefore, makes extensive use of shape trees. This concept is often present in procedural modelling methodologies based on shape grammars and represents the derivation tree of a given grammar. Shape trees also encode a hierarchical structure of buildings as, for example, it is possible to discern the windows in a given floor by identifying the desired floor shape and collecting all window shapes in that sub-tree. Typically, shapes in shape trees only carry an identifying symbol, its geometry, material information and a scope (i.e., an oriented bounding box). In this system they are extended with tags and a reference to the procedural operation that created it.

In a given configuration of shapes, however, different semantic queries may result in the same set of selected shapes, leading to another level of ambiguity. Moreover, the creation of these semantic queries may not always be the easiest task as queries are defined in a text based manner which resembles a query language. To cope with these problems the system also incorporates a genetic algorithm to infer the semantic query that best matches the user intention based on a set of user selected shapes in an interactive application. This helps users interactively find the most suitable semantic query for the intended goal.

The shape tree is traversed to generate the initial population with a genotype composed by the selectors. Crossover and mutation operations are applied iteratively and the population evolves based on a fitness function that selects the semantic queries that are simpler and result in sets of selected shapes that match the user selected shapes as closely as possible. Genetic algorithms also have the added benefit of producing a great number of candidates exploring well the ambiguity of user selections.

Once the semantic query is selected, the user instantiates a procedural operation (or a template) to add further detail to the previously selected shapes (i.e., apply the action). Templates are predefined sets of procedural operations which can be parametrized. Selection-action pairs are organized in a tree structure which is traversed in a depth first order, resulting in the final procedural model.

5.2 Selection Action Paradigm

In shape grammars, or in general for node-based approaches, each geometric shape is tied, since its creation, to a specific procedural rule via symbols or connections between nodes. In contrast, the proposed selection-action paradigm achieves a separation between the geometry and the rules that are used in its generation. The shape tree becomes merely an artefact of the procedural generation and each existing, individual shape can be selected or referenced at any point in the derivation process. The selections are achieved by means of queries that exploit explicit user defined semantics, via symbols and tags; or semantics that are implicitly defined within the resulting shape tree, via filters and hierarchical relations.

This allows for a semantic layer over procedural models, which can be further used to add detail to specific parts of the geometry by applying actions on selected shapes. As a result, it is now possible to define selections that propagate and remain valid even if a model changes (within a reasonable range).

These conditions contribute to a system that is capable of inferring selections given a small set of user selected shapes. As the goal is to allow the user to select an arbitrary set of shapes to modify, having the geometry too tightly coupled with the rules would involve too many complicated modifications to the rules used. In the proposed system, adding a single rule that selects and acts on a subset of shapes is sufficient. Therefore, the pairing of a query and an action is at the core of this paradigm.

In this system, semantics are assigned to geometries by parametrising the procedural rule that generates them. Discovering or learning the semantics of architectural models is extremely dependent on the user's intentions and the specific building being modelled. As such, manually inserting this data is, at this moment, required. Automatic tag assignment is left as an avenue for future work.

As mentioned, this system is (theoretically) agnostic to the underlying procedural framework used as long as it exposes the concept of shape tree. Nonetheless, shape trees should possess symbols and tags to take full advantage of queries. However, if a shape tree lacks any of these concepts, geometric queries should still be able to produce interesting results.

5.3 Semantic Queries

The selection action paradigm relies heavily on the capability of selecting shapes from a shape tree. Shapes, in the implementation of this system, already integrate semantics by associating tags, as each shape is associated to a specific concept in the designer's taxonomy.

As these tags are integrated in the hierarchical process description and associated with the topological and geometric properties of the objects, this creates a semantic layer that supports the overarching concept of semantic queries.

This means that shape trees express the semantic structure of a building. For example, a set of shapes with the *Floor* symbol whose parent is a shape with the *Facade* symbol represent the building's floors within such façade. By explicitly selecting shapes using semantic information, it is possible to apply different derivation paths which result in distinct geometry.

To this end, the concept of semantic queries over a shape tree has been defined. Semantic queries provide, with a concise notation, a multitude of conditions on shape trees. The hierarchical nature present in these trees is also exploited allowing the user to perform selections even based on hierarchical relations between shapes.

Moreover, simple extensions to the semantic query system also provide the capability of performing geometric queries. For example, it is possible to select shapes that are inside or intersect a given volume or perform selections based on the orientation of the shapes.

The user specifies the semantic queries as a list of conditions encoded in a string. A query q matches a shape s if the shape verifies all conditions expressed in the query which is expressed as $match_q(s)$. The $product_q(\mathcal{ST})$ of a semantic query q over a shape tree $\mathcal{ST}$ is an operation that returns all shapes that match the query and is defined in Equation 5.1

$$product_q(\mathcal{ST}) = \{s \,:\, s \in \mathcal{ST} \wedge match_q(s))\} \tag{5.1}$$

The semantic selection system relies on two components: selectors and hierarchical queries. Selectors express a set of semantic conditions that allow the selection of sets of shapes. Hierarchical queries, on the other hand, restrict the selection to shapes that match the hierarchical relations between two or more selectors.

The selectors and hierarchical relations are defined in a syntax to represent semantic selections which will be explained in the following sections.

5.3.1 Selectors

Selectors provide a way of selecting shapes given their intrinsic properties based on a combination of several atomic query components. A *selector* is defined as a tuple $\langle sym, o, T, F \rangle$ of atomic components where o is a type of procedural operation (e.g., subdivision, repeat, split), sym is a symbol for a shape, T is a set of tags and F is a set of filters.

For every component the system also defines a *null* component that always evaluates to true which symbolizes its absence from the selector. Every atomic component is optional, as long as the query is not empty (i.e., at least one of its components is not null).

Selectors are encoded in a string where the different types of components are specified with a well-defined order and syntax defined in the grammar in Listing 5.1.

```
<query> ::= <selector>(<hierarchy_op><selector>)*
<selector> ::= <op_name><symbol><tag>*<filter>*
<op_name> ::= <identifier>
<symbol> ::= "'.'"<identifier>
<tag> ::= "'\#'"<identifier>
<hierarchy_op> ::= "'>'" | "'/'" | "'~'"
<filter> ::= "':'"<identifier><args>
<args> ::= "'('"<arg_list>"')'"
<arg_list> ::= <arg> ("','" <arg>)*
<arg> ::= "'{'" <expression> "'}'"
```

Listing 5.1: Grammar for the selectors in the selection action paradigm

More concisely, a selector can be defined as shown in Listing 5.2.

```
operation.symbol#tag_n:filter_n(...)
```

Listing 5.2: Concise example of a selector.

Here, the operation type component expresses the condition that a shape must have been created by a specific procedural operation type. When present, this condition must be at the beginning of the selector string. This query component can be defined as in Equation 5.2. As seen, for a given selector $q = o$ and a shape s, $match_q(s)$ is true if, and only if, the type of the procedural that generated the shape is o.

Since a shape can only be generated by a single procedural operation, there can only be at most one operation type query component in a given selector.

$$match_o(s) \iff type(operation(s)) \equiv o \tag{5.2}$$

The *symbol* query component, preceded by a '.' character, indicates that a shape must possess a given symbol to be selected. Note that, although it is here assumed that shapes in a shape tree contain symbols, this is not the case for some procedural modelling methods (e.g., some graph based approaches [Sil+15]). In such cases, the *rule symbol* condition can be omitted from the system altogether. Similar to the operation type component, there can only be one symbol component in a selector.

The symbol query component can be formally defined as in Equation 5.3 which, as in the operation type component definition, states that a shape can only be matched by this type of query component if the shape's symbol is the same as defined in the query.

$$match_{.sym}(s) \iff symbol(s) \equiv sym \tag{5.3}$$

In contrast, a selector can have multiple tag and filter conditions. Tag selections are possible by representing each with a preceding '#' character. For a shape to match the tag query component it must have such tag within its set of tags as defined in Equation 5.4.

$$match_{\#tag}(s) \iff tag \in tags(s) \tag{5.4}$$

Finally, filters allow the specification of more complex queries. Note that filters may or may not have parameters. Filters are the entry point for any extension to this system that provides new types of conditions. For example, filters can encode conditions such as the direction a given shape is facing, if it is in an even or odd position relative to its shape tree siblings or even if a shape is inside a given geometry.

Therefore, a filter is a function that takes a shape and zero or more arguments and returns a Boolean value indicating whether a shape matches its internal condition. This is defined in Equation 5.5.

$$match_{:filter(arg_0,\cdots,arg_n)}(s) \iff filter(s, arg_0, \cdots, arg_n) \tag{5.5}$$

The example in Listing 5.3 selects all shapes created with an *Insert* operation that have the *Window* symbol and *small* tag and that are facing the north direction. The results of applying this query over an example building can be seen in Figure 5.3.

```
Insert.Window#small:north
```

Listing 5.3: Example of a selector that selects shapes generated by an Insert operation having the Window symbol, the small tag and are facing north

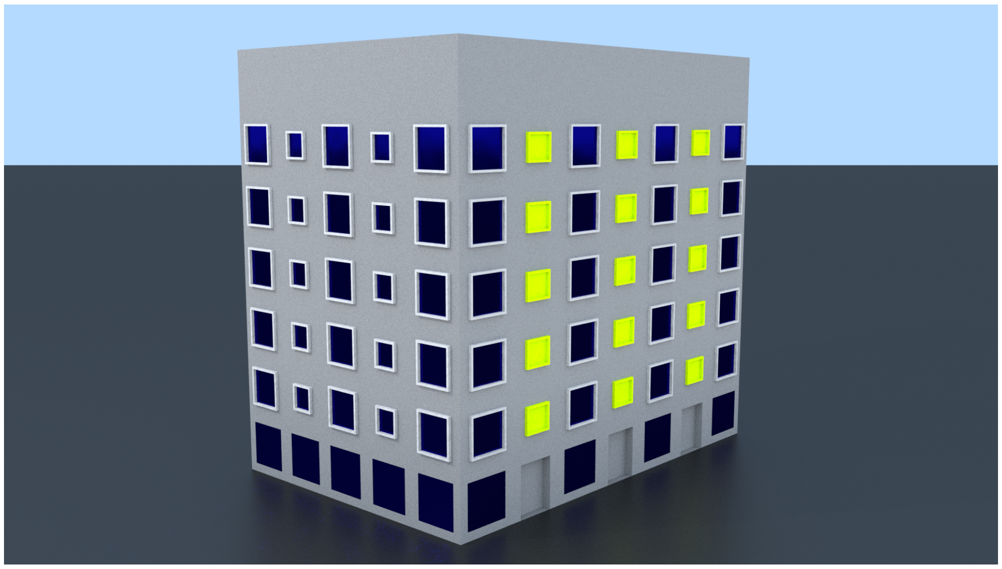


Figure 5.3: Results of applying the semantic query in Listing 5.3.

The next example in Listing 5.4 allows to select all shapes tagged as wall that are inside the sphere with radius 5 and centred at $(3, 1, 5)$.

```
#wall:inside(sphere(3,1,5,5))
```

Listing 5.4: Example of a selector matching shapes with the wall tag within the volume of a sphere of radius 5, centred at (3,1,5).

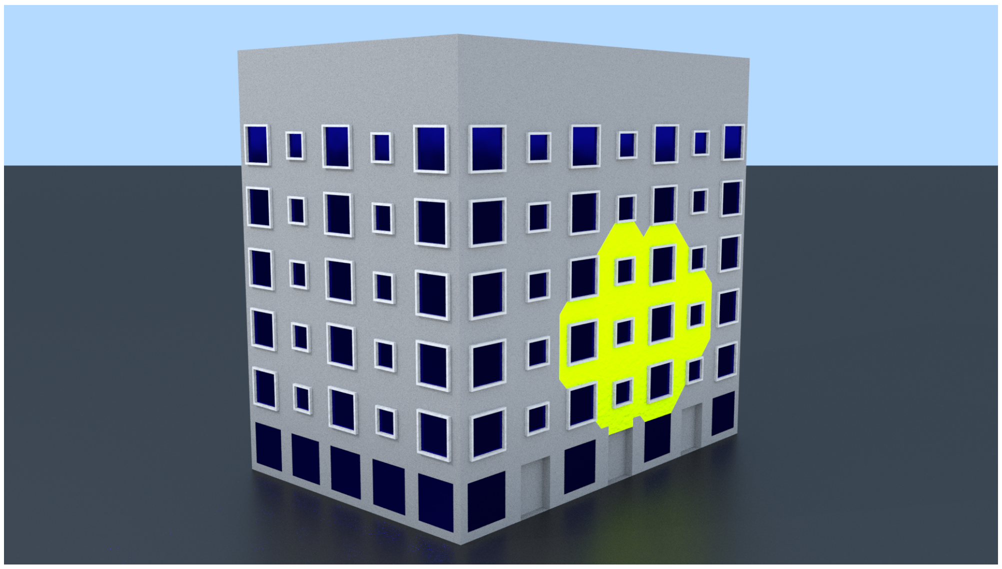


Figure 5.4: Results of applying the semantic query in Listing 5.4.

5.3.2 Hierarchical Relations

Although selectors by themselves can encode numerous conditions, they do not provide a way to express hierarchical relations between shapes within the shape tree. Therefore, this section introduces three operations that chain selectors to provide semantic queries with the capability of expressing selections based on hierarchical relations between sets of shapes. The operations are the *descendants*, *children* and *siblings*. Chaining is achieved by introducing a relation between two selectors as given by Equation 5.6.

$$selector_1\ relation_1\ selector_2\ (\ldots) \tag{5.6}$$

The *descendants* hierarchical relation (expressed by the $>$ symbol) constrains the shapes matched by a selector sel_2 to those shapes that belong to the sub-trees rooted at the shapes matched by sel_1. For this, the $descendants(S)$ operation has been defined which returns the set of all shapes that are in the sub-trees rooted at each shape $s \in S$. More formally, the descendants operation for a shape s is defined as in Equation 5.7 and it is illustrated in Figure 5.5.

$$descendants(S) = \{s : \exists_{s_a} \in S,\ s_a \in ancestors(s)\} \tag{5.7}$$

The results of chaining two selectors with the descendants hierarchical operation is defined in Equation 5.8, using the descendants operation as defined above.

$$product(sel_1 > sel_2) = descendants(product(sel_1)) \cap product(sel_2) \tag{5.8}$$

Similarly, the *children* relation (/ symbol in this notation) constrains the shapes matched by a selector s_2 to the shapes that are children of the ones matched by s_1. As in the previous case, the $children(S)$ operation is defined, returning the set of all children shapes of each

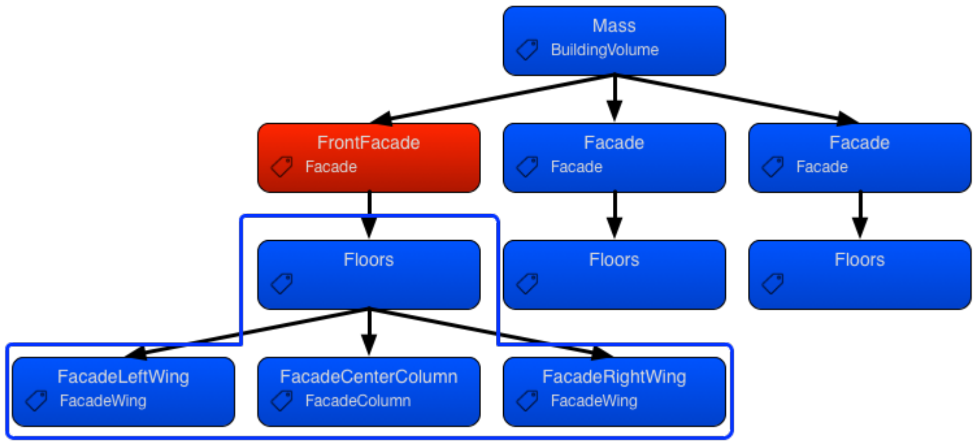


Figure 5.5: The descendants operation applied to a given shape in a shape tree.

$s_i \in S$. Equation 5.9 defines the children operation for a given shape S belonging to a shape tree, which is illustrated in Figure 5.6.

$$children(S) = \{s : \exists_{s_p \in S},\ s_p = parent(s)\} \tag{5.9}$$

Equation 5.10 defines the results of using this relation to chain two selectors which, similarly to the previous case, uses the children operation.

$$product(s_1/s_2) = children(product(s_1)) \cap product(s_2) \tag{5.10}$$

Finally, the *siblings* relation (represented by $\sim$) constrains the shapes matched by sel_2

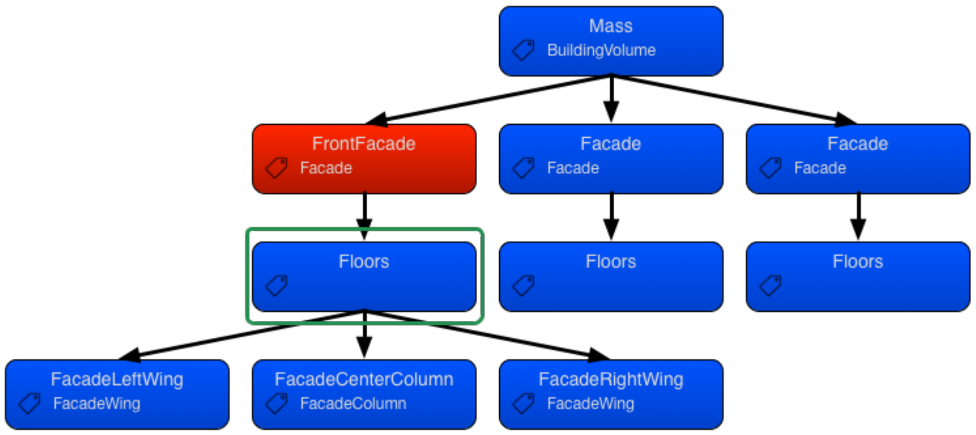


Figure 5.6: The children operation applied to a given shape in a shape tree.

to the shapes that are siblings of shapes matched by sel_1. The $siblings(S)$ operation returns the list of all shapes that share the same parent with a shape $s_i \in S$ but are not in list S. Figure 5.7 illustrates the results of applying this operating to a shape. The siblings relation is defined as in Equation 5.11.

$$siblings(S) = \{s : \exists_{s_s \in S}, parent(s) = parent(s_s) \wedge s_{\notin} S\} \tag{5.11}$$

The product of applying a query chaining two selectors with this relation relies on the use of the siblings operation and is defined in Equation 5.12.

$$product(sel_1 \sim sel_2) = siblings(product(sel_1)) \cap product(sel_2) \tag{5.12}$$

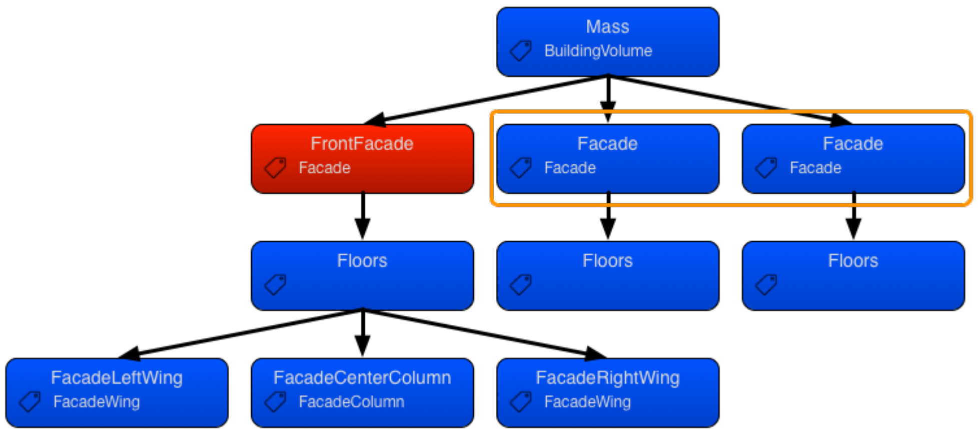


Figure 5.7: The siblings operation applied to a given shape in a shape tree.

It is possible to chain several hierarchical relations, resulting in complex semantic queries that take advantage of the tree structure created by many procedural modelling methodologies. The final result of several chained selectors is defined as in Equation 5.13, where $op_i, i \in [0, n]$ is one of the hierarchical relations defined above.

$$product(sel_1, op_1, \cdots, op_n, sel_n) = op(product(sel_1, op_1, \cdots, op_{n-1}, sel_{n-1})) \cap product(sel_n) \tag{5.13}$$

This flexibility allows users to express complex queries that can be rather difficult with current procedural modelling methodologies. The following semantic query, for example, selects the windows on even floors in the façade that is facing north.

```
.Facade:north / .Floor:even > #Window
```

Listing 5.5: Selects windows on even floors in façades facing north

There is, usually, a multitude of semantic queries that select the same set of shapes, which are named query sets in this document. In the previous example, it should be possible to remove the first selector and append the *north* filter to the second selector:

```
.Floor:even:north > #Window
```

Listing 5.6: Alternative query similar to the one found in Listing 5.5

It is up to the user to choose the best semantic query taking into account criteria such as readability or conciseness.

5.4 Selection Action Tree

To use the semantic queries in the generation of procedural models procedural operations must be applied to the product of such queries in a systematic way. In this system this is achieved by structuring the rules in a tree form such that each node contains a semantic query (the selection) and a procedural operation (the action).

Therefore, upon specifying a semantic query, the selection action system allows the user to directly apply operations to the selected shapes. The semantic query and operation are encoded into a tuple $\langle Q, O, C \rangle$, where Q is the semantic query, O is the operation to perform on the product of Q (its *selection set*) and $C = \langle c_1, \cdots, c_n \rangle$ is the ordered list of the node's children. This pair is referred to as a *selection-action node*.

The selection action system adds detail to the geometry by allowing the iterative application of a sequence of *selection-action nodes*, which will be referred to as application sequence. The application of these nodes is not commutative, since the selected shapes depend on the current shape tree configuration. Intuitively, it would not be possible to use a semantic query to select shapes that have not yet been created.

Selection-action nodes are organized in a tree structure which is called *selection-action tree*. The application sequence is defined as the depth-first traversal of such tree and the semantic selection Q_i at a given *selection-action node i* is only applied to the sub shape trees whose root is contained in its parent's *selection set*, S_p. The selection set of the root selection action node S_{root} is given by its own semantic query Q_{root}. The *selection set* S_i for a node i is defined as in Equation 5.14.

$$\begin{aligned} S_i &= descendants(S_p) \cap product(Q_i) \\ S_{root} &= product(Q_{root}) \end{aligned} \tag{5.14}$$

Figure 5.8 illustrates a simple selection action tree with its application sequence. Within a given selection action node children nodes are ordered from top to bottom and are, therefore, visited in such order when being traversed in depth first order. Note that, for example and without altering the end result, node 6 could become a child of node 5 instead of its sibling as the application sequence would still be the same. This illustrates the decoupling between procedural rule structure and the application of procedural operations and serves as a basis for layers of procedural content (Section 5.6.1), which can be used to separate, e.g., building structure from its appearance.

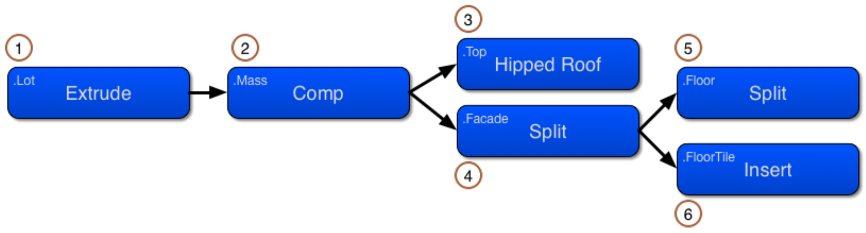


Figure 5.8: An example of a selection action tree and its application sequence.

5.4.1 Shorthand notation

Considering a sequence of N *selection-action nodes* representing the path $p = (n_0, \ldots, n_N)$ from the root node n_0 to a given node n_N, where $n_j = parent(n_{j+1}), 0 \leq j < N$, we can apply Equation 5.14 recursively to find the selection set S_j for the selection action node n_j. Equation 5.15 demonstrates this.

$$\begin{aligned} S_j &= descendants(S_{j-1}) \cap product(Q_j), j \in [1..N] \\ S_0 &= S_{root} = product(Q_{root}) \end{aligned} \tag{5.15}$$

Using Equation 5.13 it is possible to demonstrate that the resulting selection set for the node Q_N is, in fact, equivalent to the semantic query obtained by placing a descendants operation ($>$) between every query Q_j (i.e., $Q_0 > \cdots > Q_{N-1}$) in the path between the root node and Q_N.

By definition, the selection set S_0 of the root node Q_0 is the set of shapes that match its query, i.e., $product(Q_0)$ as defined in Equation 5.15. It is possible to apply the definition in Equation 5.13 to the $Q_0 > Q_1$ semantic query. The resulting selected shapes of such semantic query are, in fact, equivalent to the selection set S_1 for the second selection action node Q_1 as can be demonstrated by Equation 5.16.

$$\begin{aligned} product(Q_0 > Q_1) &= descendants(product_)(Q_0) \cap product(Q_1) \\ &= descendants(S_0) \cap product(Q_1) \\ &= S_1 \end{aligned} \tag{5.16}$$

Extending the previous to the remaining selection action nodes $S_j \in p$ by induction we arrive at the conclusion that the selection set for each node S_N is equivalent to the product of the $Q_0 > \cdots > Q_N$ semantic query as we wanted to demonstrate. This is illustrated in Equation 5.17.

$$\begin{aligned} product(Q_0 > \cdots > Q_N) &= descendants(product(Q_0 > \cdots > Q_{N-1})) \cap product(Q_N) \\ &= descendants(S_{N-1}) \cap product(Q_N) \\ &= S_N \end{aligned} \tag{5.17}$$

Intuitively, this means that edges in the *selection action* tree are a shorthand notation for the descendant hierarchical relation, while root nodes provide context-free selection over the entire shape tree. Moreover, the results of the application of a node are limited to the shapes generated, so far, by the sub-tree rooted at its parent due to the application sequence being defined as a depth first traversal of such tree. Figure 5.9 illustrates the scope of application for a given *selection-action node*.

Note that is possible to convert any *selection-action tree* to a flat list of selection action nodes, without affecting the end result. However, the required semantic queries would require, in turn, a corresponding conversion, becoming rather long as they do not benefit from the hierarchical composition any more. It is, then, up to the user to select the best representation.

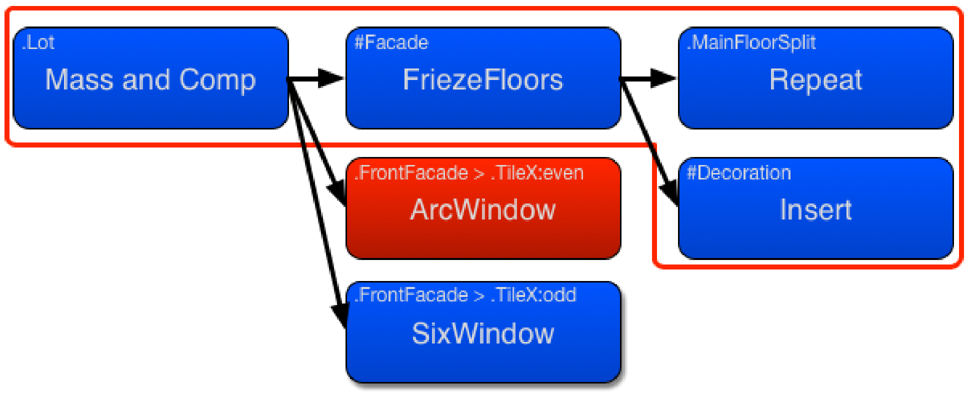


Figure 5.9: The application of a selection-action node is constrained to the shapes created by the nodes inside the red border.

5.4.2 Procedural Derivation

In order to generate procedural models with this paradigm the semantic queries need to be applied interleaved with the procedural operations. In other words, the system uses a semantic query to select shapes from an existing shape tree configuration and, then, executes a procedural operation on these shapes. The newly generated shapes are inserted into the shape tree as children of the shape to which the operation was applied. This procedure is similar to what is performed in other urban procedural modelling approaches (e.g., shape grammars).

As previously mentioned, the application sequence for a given selection action tree is given by the depth first traversal of such tree. Therefore, for each node $n_i = \langle q_i, o_i, C_i \rangle$ visited during the traversal, the respective query Q_i considering the shorthand notation is applied to the current shape tree configuration yielding the selection set S_i for this node.

Afterwards, the system applies the procedural operation o_i to each of the selected shapes $s_j \in S_i$ and appends the set of resulting shapes to the list of children of shapes s_j. The pseudo-code in Listing 5.7 shows the algorithm for the procedural derivation in the selection action paradigm.

```
function ProceduralDerivation(shapeTree, selTree):
 let stack ← [$root(selTree)$]
 while stack is not empty:
  n ← stack.pop()
  Q ← fullQuery(n.query())
  op ← n.operation()
  S ← $product_{shapeTree}(q)$
  for s in S:
   newShapes ← op(s)
   s.children ← s.children + newShapes
  for c in n.children:
   stack.push(c)
```

Listing 5.7: Procedural derivation for the selection action paradigm

Note that the algorithm requires any given configuration of a shape tree. This implies that it is possible to apply selection action trees to a single initial shape (i.e., an axiom) or a completely or partially generated shape tree. Although it is possible to achieve the same functionality with shape grammars, the semantic queries' added flexibility in specifying to which shapes procedural operations are to be applied grants this paradigm an easier way to aggregate procedural rules based on their functionality. It is, therefore, trivial to define several selection action trees that generate different coarse building structures, another set which applies architectural elements of distinct styles and, yet, another set for different ornaments and decorations. This functionality is called layers of procedural contents and is detailed in Section 5.6.1.

As an application example of a selection action tree, let us consider a simple residential building. This example will illustrate the sequential application of several selection action trees, starting from an initial building lot. The first tree constructs the coarse building volume as can be seen in Figure 5.14a.

Such coarse building structure was obtained simply by extruding the shape corresponding to the building outline which was, posteriorly, decomposed into the several shapes that corresponded to its faces. Each façade, identified by the *Facade* symbol was subdivided into floors with the *RepeatSplit* node. To the top shape the *Roof* node was applied which generates (as the name implies) the building's roof. This selection action tree can be seen in Figure 5.10.

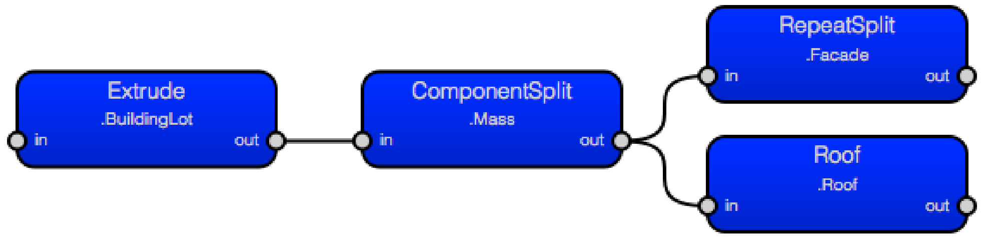


Figure 5.10: Selection action tree illustrating the first step in the generation of a small residential house.

The next step, detailed in Figure 5.11, details the side façade. Firstly, the *Repeat-Split* node divides all floors facing north, by using the *.Floor:north* query, into shapes with the *Floor-Tile* symbol. A floor tile is, then, processed differently based on whether it is the first or last position (defined with the *.FloorTile:ends* query) or if it is one of the middle floor tiles (described with the *.FloorTile:middle*). This partitions the floor tiles differently, creating different spaces where windows will be placed in following steps.

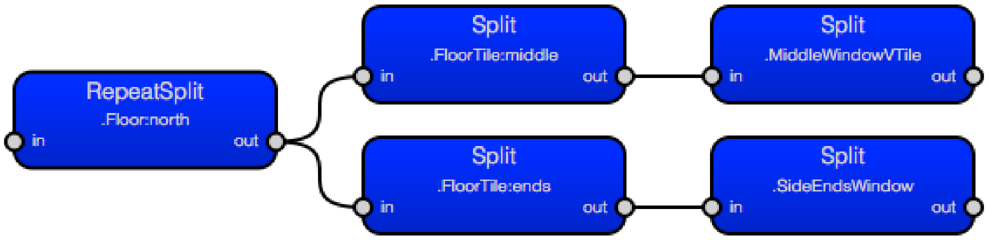


Figure 5.11: Selection action tree illustrating the second step in the generation of a small residential house.

Similarl to the second step, the third step (which is illustrated in Figure 5.12) also uses a direction filter, in this case in the *.Floor:east* query, to split the east facing façade. However, this time each floor is split into a smaller, centred shape and two, equally large shapes on each side. The centred shapes are also subdivided differently depending on whether it is on the ground (specified with the *.Floor:east:first* > *.FloorCenter* query) or the last floor (using the *.Floor:east:last* > *.FloorCenter*). In the first case, the shape is split vertically, creating the shape that will be derived into a door in later steps. On the latter case, the shape is also divided vertically resulting in a shape that will result in a window and a shape that is, afterwards, extruded. The shapes on each side are also processed differently. While the shape to the left is kept unmodified, the shape to the right is subdivided several times resulting in a shape that will also yield a window. To achieve the different derivation paths, the example used the *.FloorFrontSide:last* query.

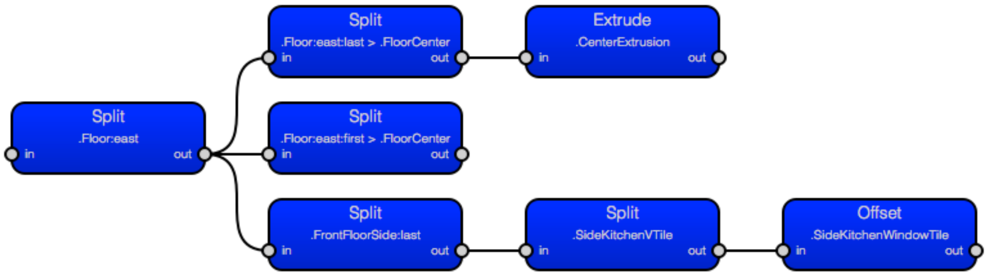


Figure 5.12: Selection action tree illustrating the third step in the generation of a small residential house.

The fourth and last step is responsible to add more detailed assets to the windows and doors generated previously. To this end, four different tags (*LargeWindow*, *MediumWindow*, *SmallWindow* and *Entrance*) were associated to different shapes throughout the building generation to this point, each reflecting a different type of window or door. The different nodes in Figure 5.13 illustrates that to each tag, a different asset was applied. This last step is illustrated in Figure 5.13.

5.5 User Selection Inference

Semantic selections provide a flexible way to select shapes in procedural models based on the implicit semantics in a shape tree and on their geometric properties. As previously seen,

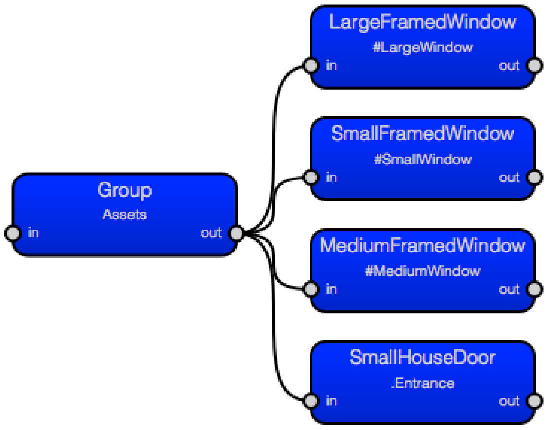
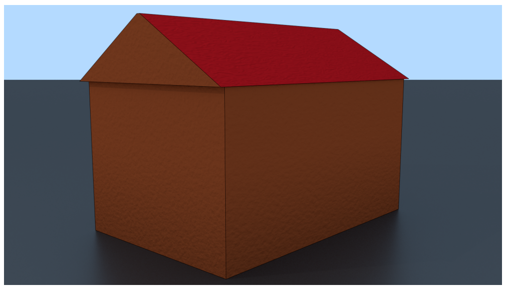
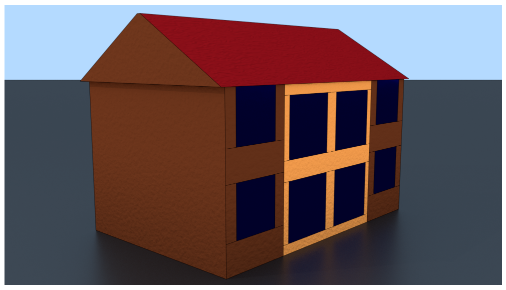


Figure 5.13: Selection action tree illustrating the fourth step in the generation of a small residential house.



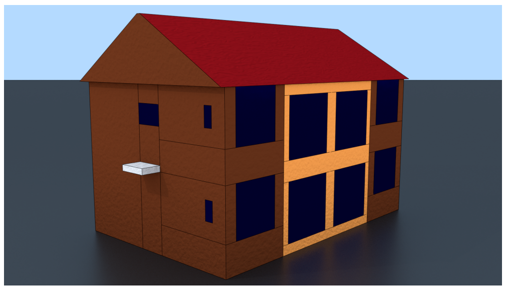
(a) Step 1 as detailed in Figure 5.10.



(b) Step 2 as detailed in Figure 5.11.

this is required to provide an extra level of indirection such that the intention behind user selections via point and click is kept even if the procedural model is modified.

Semantic queries can express complex conditions which are used to select shapes. Nonetheless, the multitude of possible queries in a shape tree poses a burden to the user who must have the knowledge of the structure and semantic features of the present shape tree. Moreover, given a user selection, there are several different expressions that would result in the same selection for a particular building. Therefore, the process of finding a selection expres-


(a) Step 3 as detailed in Figure 5.12.

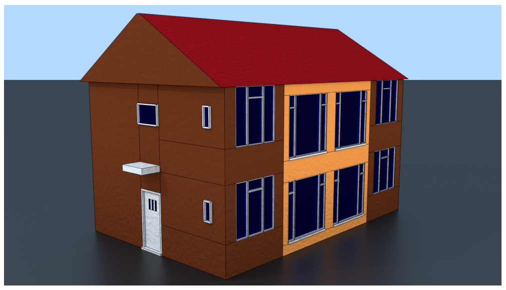

(b) Step 4 as detailed in Figure 5.13.

Figure 5.15: Steps in the generation of a simple residential building.

sion that matches the user needs is, at least, highly ambiguous. Thus, a visual mechanism that is able to generate several suggestions for the users, who are ultimately responsible for the final selection is required as they are the ones who have in mind the desired final outcome.

To alleviate this, the system makes use of a genetic algorithm to generate and rank a set of candidate queries based on a few user selections in an interactive application. Genetic algorithms are ideal for this purpose as they work on populations that are evolved to find the most suitable individuals for a given goal. Thus, they are a suitable match as an optimization tool for the semantic queries. It is possible to consider the mutations and population growth as a motivation for decomposing queries into selectors and hierarchical relationships.

The selection inference starts as soon as the user has selected at least n shapes ($n = 2$ in the implementation). In a first step, the genetic algorithm creates an initial population of semantic queries which is then mutated and crossed in order to obtain a population that closely matches a fitness function by iterating over M iterations (in the implementation $M = 100$). Then, the algorithm outputs the best distinct individuals that were generated throughout the entire process (20 in the system).

These semantic queries are grouped into query sets such that each query in the query set produces the same set of selected shapes. Each query set is given a score that is calculated based on the average score of the queries and the sets are presented to the user in decreasing order of its score within a suggestive interface. The user can, then, navigate the interface and choose the semantic query that better matches the intention.

This is an optimization problem where the system tries to find semantic queries that best match a fitness function. Genetic algorithms were chosen as they naturally create and explore large numbers of candidates. It is an intrinsically exploratory algorithm that searches the solution space for the best candidates without knowing a priori a general direction of search. It is also possible to consider the mutations and population growth as the motivation for decomposing semantic queries in the selectors and hierarchical relationships as these semantic query components are the genetic material that will be evolved with the presented methodology.

The probability values presented in the following sections are the ones that showed the best results in preliminary tests.

5.5.1 Semantic Profiles

Both the generation of an initial population and the evaluation components of the selection inference subsystem rely on the concept of semantic profiles. Each of these merely is a histogram of the number of shapes in a set that match a query composed of a single component (i.e., an operation, symbol, tag or filter) which, here, will be named as a feature.

As a first step, the shape tree is traversed to collect all symbol and tag features as these are created by the user. Operation types and filters are built into the system and are automatically included as a feature. To limit the search space to a discrete space, filters that take parameters are not considered. This step results in a list $\mathcal{F} = [f_0, \cdots, f_n]$ of features.

Considering a set of selected shapes (defined either by point and click selection by the user or by a query), three types of semantic profiles are introduced. The *horizontal profile* aims to give a semantic description of the set. The *vertical profile* describes the hierarchical structure of the set, i.e., the profile is constructed given the paths of shapes from the shape tree root to each of the selected shapes. Finally, the *siblings profile* gives a description of the shapes that are siblings of each of the selected shapes, exploiting locality within the shape tree. A profile p is identified by $\mathcal{P}_p$ and the number of shapes that present feature f_i in such profile is given by $\mathcal{P}_p^i$.

An example of such profiles for several sets of shapes selected by the presented queries can be seen in Table 5.1. These queries were issued on the model shown in the third step of

Figure 7.10.

Query	*.Facade*	*.Floor*	*.WindowTile*	*:even*	*:odd*	Generated Queries
.WindowTile:even						
Horizontal	0	0	50	50	0	*:even*, *.WindowTile*
Vertical	4	20	0	14	12	*.Floor*, *:even*
Siblings	0	0	70	0	70	*:odd*, *.WindowTile*
.Floor:even > *.WindowTile:even*						
Horizontal	0	0	30	30	0	*.WindowTile*, *:even*
Vertical	4	12	0	14	4	*:even*, *.Floor*
Siblings	0	0	42	0	42	*.WindowTile*, *:odd*

Table 5.1: Semantic profiles for two shape selections.

5.5.2 Initial Population

Any genetic algorithm starts with a given initial population, making up the initial genetic pool, which will be evolved as the algorithm progresses. Each individual in the population will be mutated and crossed for several iterations selecting the best candidates with a fitness function which drives the population towards the goal.

Therefore, as a first step in the generation of the initial population, the horizontal profile $\mathcal{P}_h$ of the user selection is computed. This is then used to generate a set of N candidates (in this implementation $N = 20$) by iteratively sampling a feature from the profile with probability $P(sample) = \mathcal{P}_h^i / \sum \mathcal{P}_h^n$. Features that have a higher representation in the profile also have a higher probability of being selected. Features are, posteriorly, converted into queries composed of a single component. This results in a genotype of queries that tries to match, as closely as possible, the selection profile. Furthermore, since it was derived from features that are already present in the selection, it is assured that all candidate queries produce a valid selection. In Table 5.1 the *Generated Queries* column represents some queries generated from the features. An example of initial candidates can be seen in Figure 5.16.

5.5.3 Evaluation

Most likely, the most important component in a genetic algorithm is the evaluation function. This function has to be designed such that it drives the evolution towards the goal. It should also encode the logic of what constitutes an ideal solution. There are three criteria to evaluate the candidate queries.

The evaluation of candidate queries is achieved by finding a normalized ratio between the profiles of the user selected shapes and the set of shapes selected by candidate queries. Given a semantic profile $\mathcal{P}_{sel}$ (horizontal, vertical or siblings) of the user selection and of the candidate query $\mathcal{P}_{cand}$, the system calculates the *Semantic Feature Ratio* (SFR) between feature f_i in both sets as defined Equation 5.18.

$$SFR_i(\mathcal{P}_{sel}, \mathcal{P}_{cand}) = \frac{min(\mathcal{P}^i_{sel}, \mathcal{P}^i_{cand})}{max(\mathcal{P}^i_{sel}, \mathcal{P}^i_{cand})}, \mathcal{P}^i_{sel} \neq 0 \wedge \mathcal{P}^i_{cand} \neq 0 \tag{5.18}$$

A value of zero for feature f_i means that such feature is present in either the candidate profile or the selection profile but not in both. On the other hand, a value of one means that exact same amount of shapes with such feature are present in both profiles. Other values measure how much the sets of shapes differ for the specific feature f_i.

Having the semantic feature ratio for each feature, the system proceeds to find a matching score between the two profiles. This score is, again, obtained as a ratio by computing the profile score $\mathcal{PS}(sel, cand)$ between the two profiles. This is defined in the following equation, where $|\mathcal{P}_{sel}|$ is the number of features in *sel* that have a value greater than zero and $|\mathcal{P}_{cand}| = \sum SFR_n(sel, cand)$ is the sum of the *SFR* of each feature.

$$\mathcal{PS}(sel, cand) = \frac{min(|\mathcal{P}_{sel}|, |\mathcal{P}_{cand}|)}{max(|\mathcal{P}_{sel}|, |\mathcal{P}_{cand}|)}, |\mathcal{P}_{sel}| \neq 0 \land |\mathcal{P}_{cand}| \neq 0$$

Based on the above, the system calculates the candidate's final score by finding the profile score between the horizontal, vertical and siblings profile of both the user selected shapes as well as of the shapes selected by the candidate query. These, respectively, yield the $\mathcal{H} = \mathcal{PS}(sel_h, cand_h)$, $\mathcal{V} = \mathcal{PS}(sel_v, cand_v)$ and $\mathcal{S} = \mathcal{PS}(sel_s, cand_s)$ scores.

Furthermore, based on the fact that the user will always select leaf shapes, there is an additional constraint on the queries, which guides the genetic algorithm into producing candidates that select only leaf shapes. This is modelled by introducing another score $\mathcal{L}$, such that $\mathcal{L} = 0$ if a given candidate selects an internal shape and $\mathcal{L} = 1$ otherwise.

The final score for a query is then given by the following equation:

$$score = \mathcal{L} \cdot \frac{\mathcal{H} + \mathcal{V} + \mathcal{S}}{3} \tag{5.19}$$

5.5.4 Mutation

In genetic algorithms evolution occurs, in part, through mutations in the genetic material in the offspring of two individuals.

As described earlier, queries in the initial population do not possess any hierarchical relation which would limit the type of queries generated by the selection inference system. Therefore, to counteract this and generate candidate queries with hierarchical relations, one of the mutations defined in the system is to prepend a *selector* to a query.

This mutation is performed in an individual *a* in the current population. Firstly, the system starts by generating a *selector b* by randomly sampling one feature from the feature table. It then constructs a query such that *b op a*, where $op \in \{/, >, \#\}$ is a hierarchical relation sampled from the ones defined. If, for example, an individual represents the $\#Window$ query, the system might perform a mutation by prepending a *.Floor* selector with a children relation yielding the *.Floor / #Window* query. This example is shown in Figure 5.16 in the first two images. This mutation is performed with probability $P(add\_sel) = 0.2^{|selectors|}$, where $|selectors|$ is the number of selectors in the candidate query, effectively avoiding queries that are too long.

The system also mutates (with probability $P_{mut\_hier} = 0.1$) an individual by switching one of the hierarchical relations to a different one. In the previous example, a possible mutation would be to switch the children relation to a descendants relation resulting in the *.Floor* $>$ *#Window* semantic query. This example is shown in Figure 5.16 in images 2 and 3. This allows the system to explore different combinations of the hierarchical relation without requiring a new candidate to be generated with the previous mutation.

Another mutation is to add or remove components to the selectors. Therefore, this mutation adds or removes an operation type, symbol, tag or filter to the query with probability $P(add\_comp) = 0.1^{|comps|}$, where $|comps|$ is the number of query components in the candidate. Continuing with the previous example, a possible query would be *.Floor:south* $>$ *#Window:last*.

The system mutates each of the selectors (with probability $P(mut\_sel) = 0.5$) in an individual by modifying one of its query components. This mutation replaces one of the components with another of the same type. Based on the previous query, it is possible to generate the *.Floor:north* $>$ *#Window:even* query as can be seen in Figure 5.16 in images 3 and 4.

By making the probabilities of adding a selector or a component dependent on the size of the query and larger values for the remaining probabilities, the genetic algorithm generates larger queries at a more controlled pace while exploring more solutions within the same size.

Each type of mutation happens independently of each other, meaning each candidate may be subject to zero, one or more mutations. Thus, their probabilities do not need to add up to 1.

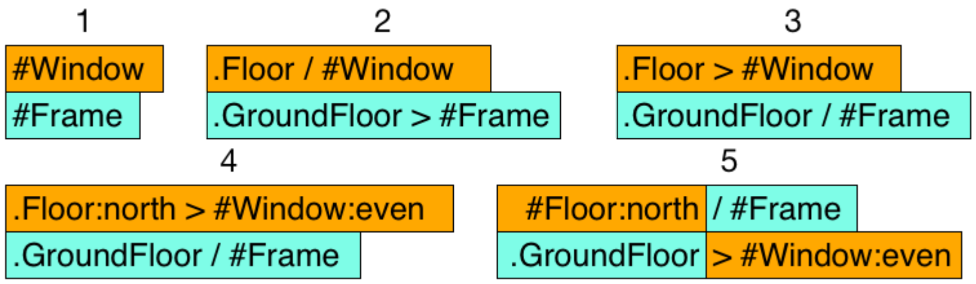


Figure 5.16: Example of candidate evolution during selection inference.

5.5.5 Crossover

The other mechanism to evolve the population in genetic algorithms is to cross the genetic material of two individuals in order to generate two new distinct individuals.

The system, therefore, crosses two individual queries by swapping the last $m = random(0, l_{min})$ selectors of each query, where l_{min} is the number of selectors present in the smaller query. The crossing process happens with a probability of $P(cross) = 0.3$.

For example, from the *#Floor:north* $>$ *#Window:even* and *.GroundFloor / #Frame* queries, the system can generate the *#Floor:north / #Frame* and *.GroundFloor* $>$ *#Window:even*.

Note that the hierarchical relation is kept with the selector to its right. This example can be seen in Figure 5.16 between images 4 and 5.

5.6 Applications

The following subsections demonstrate some applications of this methodology. Observe that these applications have not been implemented, but their implementations should be rather straightforward as a consequence of the proposed selection action methodology, as preliminary tests demonstrate.

On the other hand, the presented methodology, although currently only implemented as a tool, can be executed in runtime within a game (albeit requiring some optimization) to create entire urban levels and their mechanics from scratch or to modify such scenes as game levels evolve.

5.6.1 Layers of procedural content

Since this methodology is based on selecting shapes based on its semantics, it contrasts with most current procedural modelling methodologies where there is an implicit flow of geometry from one rule to the next. In shape grammars, for example, every non-terminal shape has an associated symbol which already defines the next rule to be applied. Similarly, in graph-based approaches, the rules to be applied to a shape are defined by the connections between nodes.

In this methodology any shape can be selected by a semantic query and, thus, transformed at any point in the application process, resulting in a separation between the created models and the rules. Therefore, this methodology naturally supports the concept of layers which can be described as the successive application of different selection-action trees.

As an application of layers of procedural content, it is possible to conceptually define three types of layers in this methodology, which guide the procedural modelling steps from an empty scene to the generation of an entire urban level and possible variations. Note that the only difference between the types of layers is to which shape trees the selection-action trees are applied. At their core, each layer is an individual selection-action tree. Figure 5.17 illustrates these types of layers.

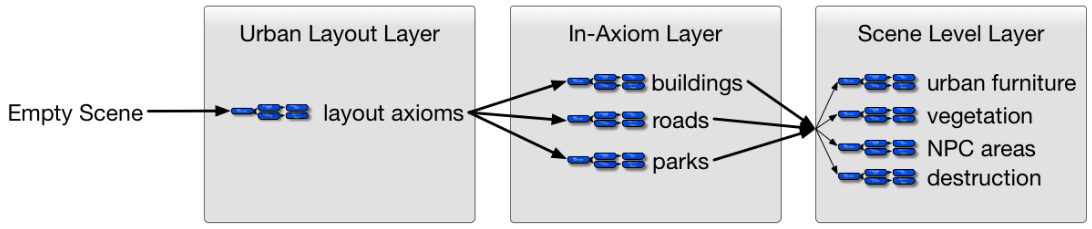


Figure 5.17: The several layers that can be conceptualized as layers of procedural content.

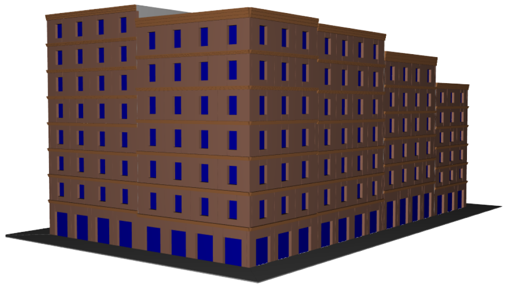
The first type of layer is the *urban layout* layer. This is executed in the first stage of the pipeline, and is responsible for creating the layout for the urban environment and axiom shapes, which will serve as the starting geometry for buildings, roads and other urban elements. Axioms are created by spawning a new shape tree and associating it with a sequence of layers of the following types.

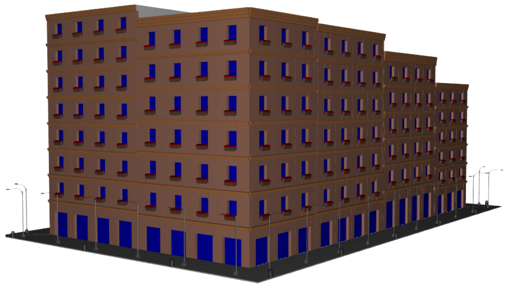
The geometry for each axiom shape is created in the second stage of the pipeline where, to each axiom, a sequence of selection action trees are applied. Each of these trees can be thought of a different, interchangeable layer. Therefore, the second type of layer is the *in-axiom* layer.

Finally, the *scene level* layer is represented by selection-action trees that are applied in sequence to the entire scene in the third stage of the pipeline. These are mainly responsible for creating variations of the geometry created in previous layers.

Using layers of procedural content, it is rather straightforward to create a base urban model and apply variations depicting different states or ambiances of a city. The user could, for example, create a city by defining several *urban layout* and *in-axiom* layers. Afterwards, the user could create sets of *scene level* layers to create a festive ambiance in the city or apply some damage and destruction to represent a post-war scenario.

As a simple example, Figure 5.18a shows a base city block and Figure 5.18b shows the same block after some ornaments were applied on the windows and urban furniture was placed on the sidewalk. In Figure 5.19 it is possible to see the three layers that created the variations. The first layer simply creates a vase in every shape with the *Window* symbol. The second and third layers are rather similar and create and position the lamps and bins respectively in the sidewalks (selected with the *.Sidewalk* query). The *Meta Info* nodes add meta information to the created shapes which, in this case, allows both the lamps and bins to have a dynamic body during the physics simulation.


(a) The base city block.


(b) Ornaments and urban furniture.

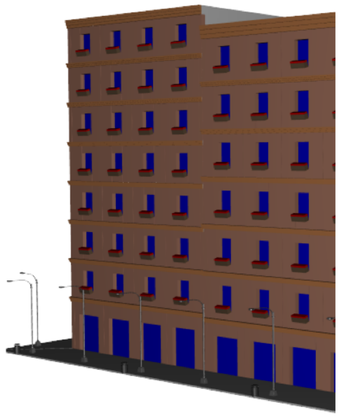

(c) Closer view on the modifications.

Figure 5.18: Variations of a city block and the selection action trees responsible for such variation.

Using interchangeable layers promotes a non-linear workflow of procedural content generation allowing re-usability of procedural assets and to easily create variations simply by interchanging layers while guaranteeing consistency throughout all procedurally generated levels.

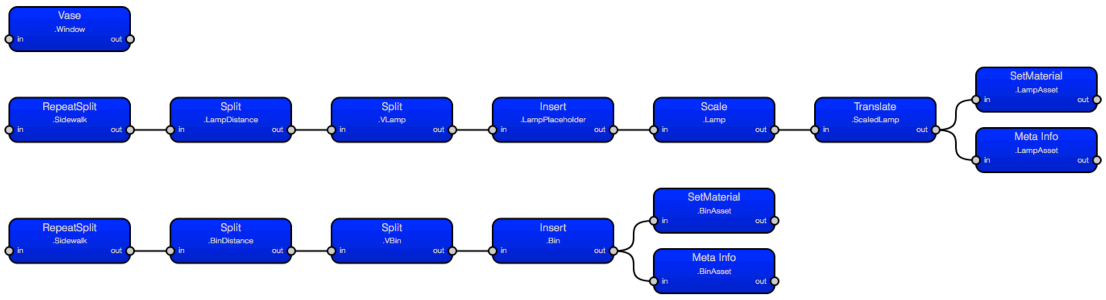
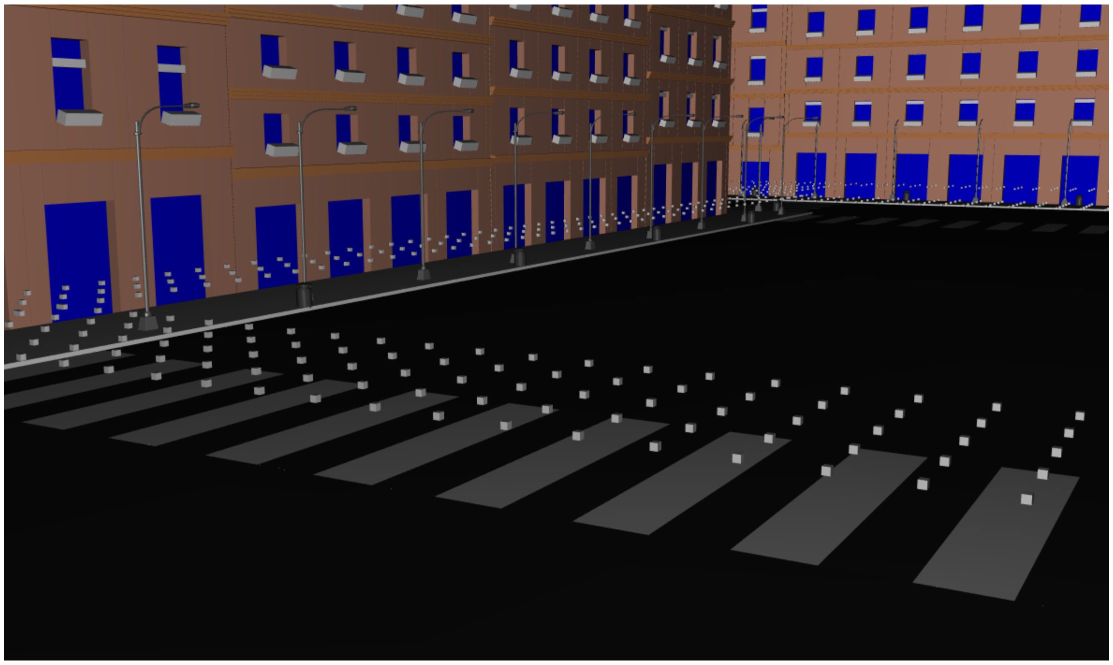


Figure 5.19: Scene layers responsible for creating the variation in Figure 5.18b.

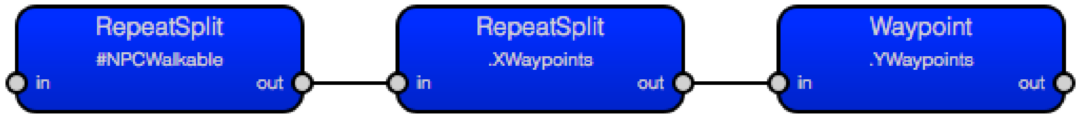
5.6.2 Generation of level mechanics

Game mechanics in an urban environment are largely related to the surrounding architecture. Furthermore, limiting the use of procedural modelling methodologies to only generate the geometry seems to be wasting the potential of such methods. A methodology such as the one presented in this chapter allows users to define some level mechanics, especially those related with the environment. Moreover, the runtime generation of levels requires some form of level mechanics to be generated accordingly.

As an example, consider an urban level populated with Non-Playable Characters (NPCs). To present the player with a plausible environment, the NPCs should be able to roam the city according to rules guided by common sense: they should walk in side-walks and cross-walks. Thus, some waypoints must be generated within these spaces to guide the NPCs. To achieve this with this system, the level designer could apply a *scene level* layer within which all shapes where NPCs could walk would be selected (e.g., #*NPCWalkable*) and distribute waypoints uniformly. Figure 5.20 illustrates waypoints distributed uniformly along the sidewalks and cross-walks (Figure 5.20a and the selection action layer that created such waypoints (Figure 5.20b).



(a) Waypoints generated with this methodology.



(b) Selection action tree responsible for creating the waypoints.

Figure 5.20: NPC Waypoints evenly distributed in sidewalks and cross-walks (top) and the layer that created them (bottom).

Creating NPCs spawning areas is just as simple. For example, the level designer can use a semantic query to select all doors (e.g., *.Door*) and replace the selected shapes with an animated game object from which characters can enter or exit the building.

Far more examples could be thought up such as placing special objects (e.g., roadblocks, barrels of explosives near destructible objects) or zones triggering events that play a role in the player progression through the level.

This system can also be used to generate or modify geometry in runtime. For instance, it is possible to hook the selection action system to a game events module and modify the environment in multiple ways. For example, when a rocket hits a building façade the system can select all shapes within the explosion radius, generate debris geometry and create dynamic game objects managed by the physics engine. In this process, different types shapes (e.g., walls, windows or traffic signs) can suffer different transformation processes.

Figure 5.21 shows an urban level pre- and post-destruction. Note how the façades of several buildings were affected, creating debris which are, then, managed by a physics engine to compute their final position. Similarly, lamp posts and bins throughout the scene were affected by the (hypothetical) explosion. Note that the realistic building destruction methodology is beyond this chapter's scope and the results merely demonstrate that the application of destruction is possible.

5.6.3 City Evolution

Any urban environment evolves over time with the activities of its population or by external factors, which can range from natural phenomena (e.g., tornadoes or earthquakes)

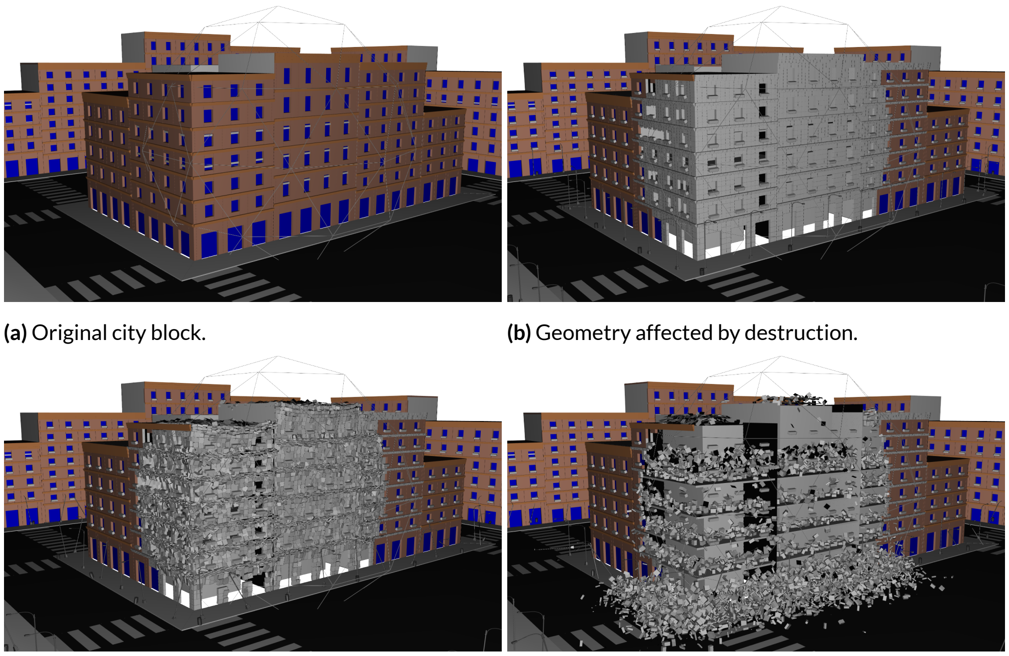

(a) Original city block.
(b) Geometry affected by destruction.
(c) Mid simulation geometry.
(d) After destruction.

Figure 5.21: A city block before (5.21a) destruction, the geometry affected within the destruction radius (5.21b) and two points during the simulation (5.21c and 5.21d).

to human-related activities (e.g., closing of an important company, war, a sudden population expansion). Quite often, modelling such evolution, for instance for a strategy game, requires manual modification of the existing assets to achieve the new look.

The selection-action paradigm would enable this kind of large changes at the urban level, and even at the landscape level, thus considerably reducing development costs and enabling artists to focus only on those landmarks that would require a special treatment due to storytelling reasons. For this, a new destruction set of rules that are hooked to all existing, original models, could be added as a sort of destruction layer on top of existing geometry. Basically, users would apply a selection of buildings according to their location in the map, and then would apply a destruction set of rules that would affect each building differently, from partial destruction (e.g., the roof) up to a whole structural demolition.

5.6.4 Weathering

Weathering at an urban level is an open topic of research [Muñ+16] that represents multiple challenges at different simulation levels, ranging from wind simulation up to pollution diffusion and interaction with the materials in the city. Such realistic simulations often need highly specialized algorithms to achieve the intended results.

However, for more simplistic weathering simulation needs (e.g., video-game) the requirements are different from the ones driven by more realistic simulations. In the case of video-games its main concern is the *credibility* of the game environment instead of an ultra-detailed simulation of reality. Thus, weathering in video-games can be considered as a purely art-related task. From this point of view, the techniques described in this chapter could serve as practical tools for the generation of different weathering effects, from simple stains resulting from urban fluxes, up to more complex deep weathering effects where the walls of a building are broken because of both the effect of the interaction with natural phenomena and the lack of maintenance.

Here, this framework would be used to select the surfaces (e.g., windows, doors or cornices) to apply the weathering effects to. Once these surfaces are selected, the applied action consists of a variety of weathering effects, including texture changes [DPH96; Bos+11], replacement of 3D models by other weathered ones, and even degradation of whole façade areas.

5.6.5 Application to non-procedural buildings

It is possible to apply this technique even for non-procedural models. Considering that a user starts with an initial, manually modelled building hull, such as the one in Figure 5.22a. It is, then, possible to apply strictly geometric queries to select shapes of interest and use a labelling operation to assign a symbol or tags to each one. From that moment on, the user is able to use regular semantic queries to detail each of the components.

This is specially interesting if interchangeable layers of procedural operations are considered. A first layer would label all interesting shapes for a given input building hull, while subsequent layers would create the rest of the geometry. As a result, it is possible to create variations (Figures 5.22b and 5.22c) from the same initial hull, simply by exchanging one or more layers.

5.6.6 Limitations

Currently the user selection inference does not account for filters with arguments as it would transform the search space from a discrete into a continuous one. Moreover, there could be some inconsistency in the queries generated for the same user selection through-

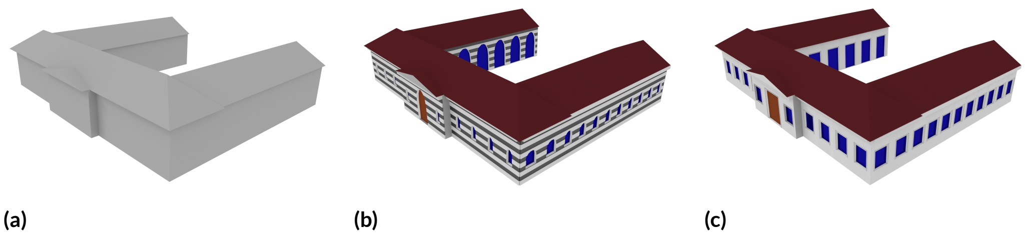


Figure 5.22: It is possible to use this system to non procedural building hull (a) and create variations (b and c) using layers of procedural operations.

out different runs. This could force the user to, sometimes, retry selections in order to obtain an acceptable candidate query set.

The geometric queries presented in this chapter are absolute in the sense that they are defined for a particular model instance and do not generalize well for model variations. However, note that making filters that accept queries as parameters is a direction for further development. In this case, it would be possible to create queries that depend on already existing shapes such as *#Window:facing(#Road):within(#Door, 5)* selecting all windows that are facing a road and are within 5 meters of a door. This would obviously create synchronization issues as a required shape might have not been created by the time a query that requires it is executed. Further research must be put towards this end.

Note that the user selection inference goal is to aid users in using the selection action paradigm and its current implementation (as genetic algorithms) can be swapped by another one without any loss in expressiveness in this methodology. Further research must be conducted as to solve both these problems.

Although semantic queries embody a great amount of expressiveness, the fact that they are defined in a text based way is a drawback, also requiring users to know what symbols, tags and filters are available to use. Nonetheless, it is mostly a matter of user interface as it is easy to imagine a system with a user interface able to offer aids to construct such queries easily. In the simplest case, it is possible to have an auto-complete system that would take into account the partially written query to suggest new query components. It is also possible to present an interface to create queries from scratch without using the text based approach. Nonetheless, for experienced users, writing the full query might be faster than using an interface and so both approaches should be available.

5.7 Summary

This chapter presented a selection action paradigm for procedural modelling of buildings. The selection is achieved through the use of semantic queries that exploit semantic information and hierarchical relations present in shape trees in order to select shapes. Once this process is complete, actions, which are procedural operations, are applied to sets of selected shapes. These semantic queries and procedural operations are paired in selection action nodes and are organized in a tree structure, constraining the scope of the selections. Buildings are generated by traversing this tree in a depth first order while applying the actions to the shapes selected by the queries.

To alleviate the user from the burden of manually constructing semantic queries, the system infers, ranks and presents a set of candidate queries based on a few user selected shapes. The user selection inference is achieved with the use of genetic algorithms, but it is possible to use other machine learning algorithms without loss of expressiveness in this paradigm.

As future work, these functionalities should be implemented and new usability studies should be performed. As a new research avenue it could be desirable to investigate further other algorithms to infer semantic queries from user selections, leading to even better results. Machine learning algorithms could prove to be useful to solve this issue.

This chapter also presented some possible applications of the selection-action paradigm for procedural modelling of urban environments within the context of video game development. Selection-action trees naturally support the concept of layers which can be used, for example, to separate structure from aspect, to produce variations on a base model or environment or to create level mechanics in a video game.

The presented paradigm is suited, not only for the offline generation of urban levels, but can also be used in runtime to model the evolution of such space or even as a consequence of specific dynamic events (e.g., apply destruction in a video-game when a rocket hits a façade). This chapter only scratched the surface of a few selected applications and there is still ample space for research into each one.

6
Sketch Based Procedural Modelling

The main purpose of this thesis is to devise, implement and evaluate a sketch based interface for the procedural modelling of buildings. The previous chapters have described some necessary steps that had to be taken back in order to provide a tighter bridge between both ends in terms of interactiveness and expressiveness. This chapter however, details how the link between sketching and procedural modelling has been achieved, building upon the contributions previously described.

This connection has not received much focus. However, this is a research avenue that has the potential to improve the content creation times for applications which require urban environments (e.g., games, cinema and urban planning). On the one hand, procedural modelling can generate vast amounts of models in a fraction of time as it would be required if the content was to be manually modelled. Nonetheless, it comes with its own problems of lack of direct control and expressiveness as previously described. On the other hand, sketch based interfaces provide a highly expressive tool to create 3D models but the sketch recognition and interpretation process is extremely ambiguous. For modelling man-made objects such recognition often relies on some sort of rules to translate the user sketch into a 3D object. The system presented in this chapter relies on heuristics retrieved from procedural operations themselves to infer the intent behind a set of user provide sketched lines and a gradient descent algorithm to estimate the parameters of such procedural operation.

In this field the main contribution has been the work by Nishida et al. [Nis+16]. This contribution relies on convolutional neural networks to both recognise a user sketch and translate it into a CGA rule and estimate its parameters. Therefore, as neural networks require training beforehand to be able to correctly process a given input, the presented system only works for a given set of pre-trained family of sketches. In contrast, the solution presented here tries to be as generic as possible and allows to any procedural operation to be recognised in any situation.

6.1 Overview

To overcome the gap between the sketch based interface and procedural modelling of buildings, this chapter presents a set of methods that create a pipeline that converts a user sketch into procedural rules that match the user's intention as closely as possible. Each of these methods has the purpose of either filtering and segmenting the sketch, help the user through the creation of more precise lines or translate the sketches into procedural rules that can, posteriorly, be modified to produce better results.

The methodology presented here lays its grounds on the contributions presented in previous chapters, with a higher incidence in the selection action paradigm described in Chapter 5. By using the semantic queries and selection inference mechanisms, the user is able to precisely select and define which shapes a given sketch should modify. This effectively tackles the user selection ambiguity in an interactive application for procedural modelling while, at the same time, allowing well-defined points where the procedural rule base can be extended.

The first stage in the sketch recognition pipeline is to filter and segment the user stroke. Firstly, this compensates for minor acquisition errors (e.g., user hand jitters) and, secondly, allows the user to create multiple line segments in a single stroke. This is achieved by analysing stroke speed and curvature.

So far, the user provided sketch is only placed in a bi-dimensional domain (i.e., the screen space). Therefore, once the individual line segments are obtained, the system converts them into line segments in the world space. This is achieved by sampling the parameter space of each type of procedural operation (e.g., extrusion or splits) and applying such operation to the selected shape which results in a new set of geometries from which the edges are extracted and projected to screen space. Each user provided 2D line segment is, then, compared with each projected edge according to a fitness function. The edge whose projection has the best score is inserted into the world space and stored in a configuration of line segments. The user is free to continue sketching which adds more line segments to this configuration.

Although, at this point, the user sketched lines have already been converted into 3D line segments, there still is the need to infer a procedural rule from a configuration of lines which should afterwards be inserted into the rule base. Whenever the system adds a set of new lines to the configuration the procedural inference system is started. This analyses the configuration and proposes a set of parametrised procedural rules which are ranked and presented to the user who is the final responsible for selecting the one that best fits the underlying intention.

The system has a set of heuristics that analyse and try to infer a procedural operation and estimate its parameters from the line configuration. Posteriorly, the system adjusts the parameters of these operations with a gradient descent method and a fitness function so that the geometry generated by the operation match the user provided lines as closely as possible.

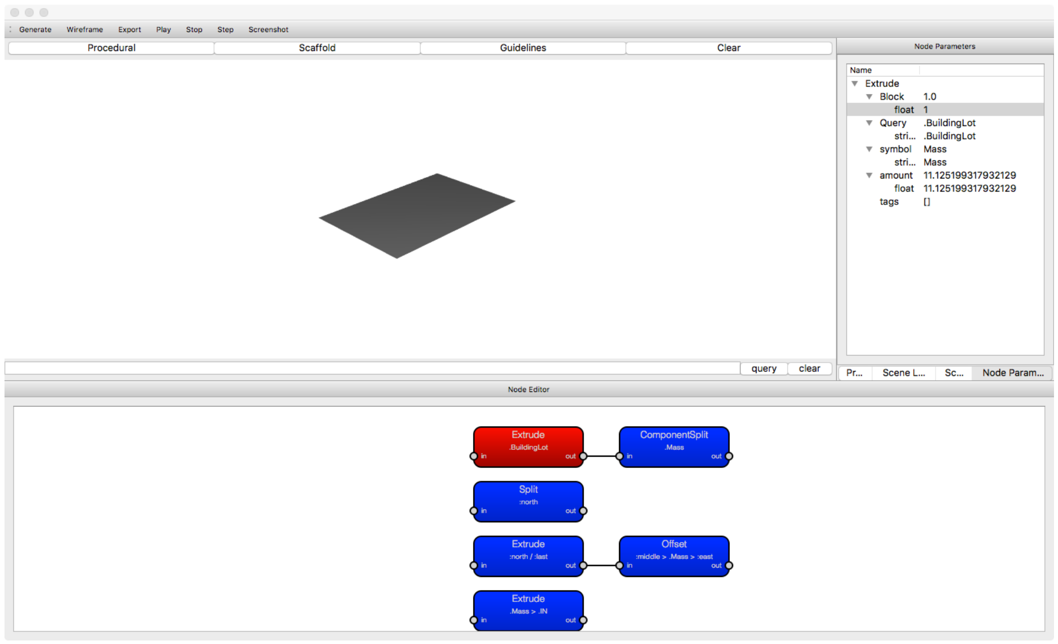
It is possible to, yet again, sample the parameter space of each type of procedural operation and perform the parameter adjustment with the gradient descent method starting with the operation that produced the best results. However, the heuristics provide a more efficient way to achieve the same results as there are fewer candidates to process.

After the procedural operation has been inferred, it is inserted into the rule base as a selection action node such that all the shapes selected by the semantic query are affected by the operation.

6.2 User Interface and Workflow

To better understand the terms and concepts explained throughout the remainder of this chapter, let us first describe the user interface and workflow.

The user is presented with an interface similar to the one in the previous chapter (Figure 6.1). However, the sketching capabilities are enabled within the interactive scene viewer and there are some sketching specific options the user is allowed to select. There is also an expectation list interface exposing the procedural candidates inferred by the system. The user must choose one of these in order to insert it into the rule base.



Node Parameters

Name	
▼ Extrude	
▼ Block	1.0
float	1
▼ Query	.BuildingLot
stri...	.BuildingLot
▼ symbol	Mass
stri...	Mass
▼ amount	11.125199317932129
float	11.125199317932129
tags	[]

Figure 6.1: The user interface presented to the user.

The user is, at first, presented with an initial configuration of shapes which can represent, e.g., a building lot as in Figure 6.1. As with the selection action paradigm, the user then provides the semantic query (either directly or by using the selection inference system) that select the shapes to be modified.

Once the selection has been made, the user is free to sketch over one of the selected shapes. For every stroke the user inserts through the sketching interface, the system runs all the different stages in the procedural operation inference pipeline. This yields a set of procedural operation candidates which are displayed in an expectation list by decreasing order of its fitness value. In other words, procedural operations whose result is more similar to the user sketch appear first in the list.

The user can explore the various suggestions presented to preview a wireframe representation of the resulting geometry if a given option was to be selected. Once the user selects the desired candidate, the system creates a selection action node with the user provided semantic query and the selected procedural operation and its parameters. This node is inserted in the rule base guaranteeing that all selected shapes are affected by the execution of the selection action node. At this point, executing the inserted node generates the desired geometry which is then inserted into the shape tree and the scene view is updated. The user can now repeat the process with the newly generated shapes to further add detail to the model.

Figure 6.2 illustrates some steps of a modelling session. In the first image, the user, starting with the building lot, sketches a vertical line (in blue) starting in one of the vertices. The system generates a candidate extrusion procedural operation which is inserted into the rule base and generates the building volume in the second image.

The user, in the third image, then repositions the viewpoint to have a better perspective with which to draw on the building's roof and sketches a rectangle inside the roof which is recognised as an offset operation. Then, the user selects the interior rectangle and draws a vertical line which is recognised as an extrusion operation.

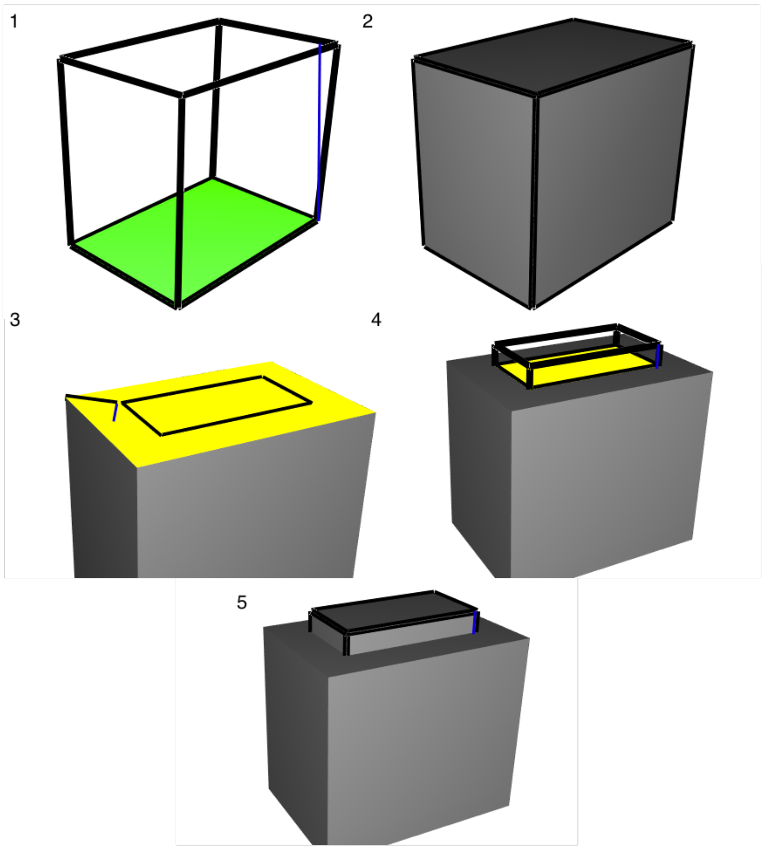


Figure 6.2: A few steps in a short modelling session.

6.3 Stroke Filtering and Segmentation

The first step in the sketch recognition and interpretation pipeline is the filtering and segmentation of the user input strokes. This is a process that is present in many sketch based interfaces for modelling and is a crucial one. It serves the purpose of minimising stroke acquisition issues and facilitate the recognition process. In this methodology, the segmentation algorithm proposed by Sezgin et al. [SSD06] was used. However, curves were not considered as the current methodology does not yet support curves.

A stroke in this system is a collection of screen space points annotated with a timestamp of the instant in which they were captured. Therefore, a point p is a tuple as defined in Equation 6.1.

$$p = \langle x, y, t \rangle \tag{6.1}$$

A stroke is merely a list of points sorted by the timestamp as defined in Equation 6.2 where t_i is the timestamp of point p_i. By this definition a stroke is, in fact, approximating the user original stroke with several line segments.

$$s = [p_i, \cdots, p_n], t_i < t_{i+1} \tag{6.2}$$

While the user is performing a stroke, the system acquires many points, potentially with a pixel resolution, depending on the acquisition rate and the velocity at which the user is sketching. Not all of these points convey required information for the recognition process (e.g., when the user tries to create a straight line) and some may even be result of some sort of error (e.g., hand jitters). Considering all acquired points may unnecessarily induce the procedural inference system into wrongfully producing undesired results. As such, some points can be safely discarded while the intention behind the user stroke is kept.

Consider the stroke performed by the user in Figure 6.3 which will be used throughout the section as an example. Most likely, the user wanted to draw a rectangle and, as such, many of the points can be discarded.

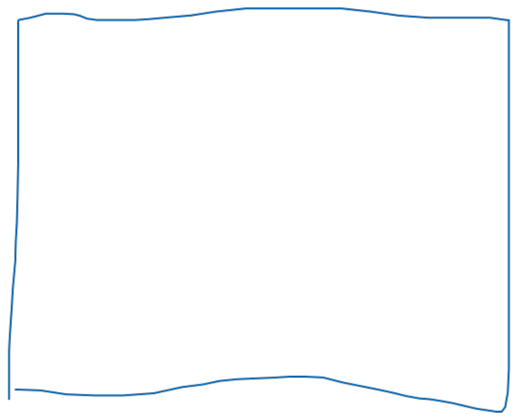


Figure 6.3: A stroke provided by a user where the intention was to draw a rectangle.

To achieve this, the system uses a combination of the velocity and curvature at each point in the stroke to segment the user stroke. This idea is typically used in several sketch based interfaces for modelling ([SSD06],) and operates under the assumption that the user sketches with a lower velocity around corner points and that, by definition, the curvature in these points is higher.

The speed metric at each point is derived from the position and timestamps of itself and neighbouring points while the curvature metric is derived from the position of a point and neighbouring points. In an initial step the system must, therefore, calculate the speed and curvature metrics for each point.

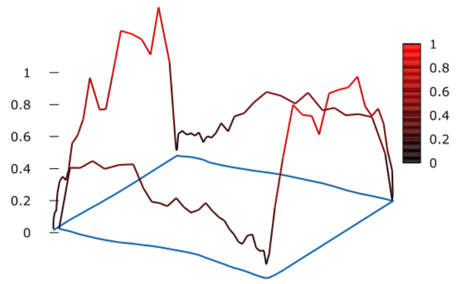
The speed of a given stroke point ($speed(p_i)$) is, thus, given by dividing the distance between two consecutive points by the time difference between the acquisition of each point and is properly defined in Equation 6.3.

$$speed(p_i) = \frac{|(x_{i+1}, y_{i+1}) - (x_i, y_i)|}{t_{i+1} - t_i} \tag{6.3}$$

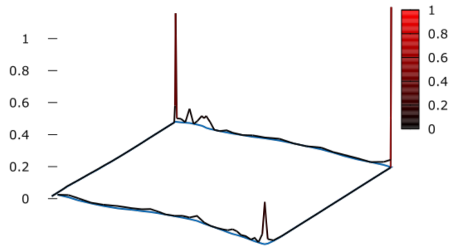
The curvature at each point in a stroke is given by the *Menger* curvature which is the reciprocal of the radius of the circle that passes through the given point p_i and its neighbouring points p_{i-1} and p_{i+1}. The curvature is then defined as given by Equation 6.4 where the denominator is the distance between p_{i-1} and p_{i+1} and the numerator is twice the sine of the angle between the two consecutive line segments and that intersect at p_i.

$$curv(p_i) = \begin{cases} \frac{2sin(\angle(x_{i-1},y_{i-1})(x_i,y_y)(x_{i+1},y_{y+1}))}{|(x_{i-1},y_{i-1})-(x_{i+1},y_{i+1})|} & \text{if } |(x_{i-1}, y_{i-1}) - (x_{i+1}, y_{i+1})| > 0 \\ 0 & \text{if } \textit{otherwise} \end{cases} \tag{6.4}$$

Considering the stroke given as input in Figure 6.3, the images in Figure 6.4a and 6.4b



(a) The speed metric for each point.



(b) The curvature metric for each point.

Figure 6.4: The speed and curvature metrics for each point in the stroke show in Figure 6.3. The values were scaled to the interval between 0 and 1 for illustration purposes.

show the stroke speed and curvature for each point. As the images demonstrate, the user sketching speed decreases while the curvature increases around the rectangle's vertices. As closed lines are not considered, the first and last vertices (bottom left in both images) don't have a curvature value.

The next step is to calculate the average of the speed and curvature metrics for a given stroke s. This provides baseline values for the user stroke which can be used to detect places where the stroke should be segmented (i.e., high curvature and low speed points). Therefore, the average speed and curvature are given in Equation 6.5.

$$\begin{aligned}\overline{speed(s)} &= \frac{1}{|s|}\sum_{i=0}^{n} speed(p_i)\\ \overline{curv(s)} &= \frac{1}{|s|}\sum_{i=0}^{n} curv(p_i)\end{aligned} \tag{6.5}$$

Posteriorly, the system collects all ranges of points whose speed is below a given threshold and whose curvature is above a given threshold defined as $1.1 * \overline{speed(s)}$ and $1.1 * \overline{curv(s)}$ metrics respectively. These values were obtained empirically as they provided good results during preliminary tests. A range is defined as a section of consecutive points in a given stroke.

At this point, the system possibly has several ranges within the user stroke which are highly likely to contain a point where the user finished a line segment and started a new one. To find such point, the system calculates, for each point within the range, a ranking metric which is a combination of a speed metric and a curvature metric.

The speed metric is based on the speed of a given point $speed(p_i)$ and the maximum speed throughout the entire stroke. The purpose of this metric is to favour points which have a lower speed compared to other points. This metric is given in Equation 6.6.

$$speed\_metric(p_i) = 1 - \frac{speed(p_i)}{max\_speed} \tag{6.6}$$

The curvature metric favours points within a range that maximise the curvature in relation to a set neighbouring points. Therefore, the curvature metric is defined as in Equation 6.7 where $curve\_length(p_a, p_b)$ is the sum of the length of all consecutive line segments defined between points p_a and p_b.

$$curvature\_metric(p_i, n) = \frac{||curv(p_{i-n}) - curv(p_{i+n})||}{curve\_length(p_{i-n}, p_{i+n})} \tag{6.7}$$

The final ranking for each point in the range is given by multiplying the speed and curvature metrics (Equation 6.8) which effectively favours points with low speed and high curvature. A neighbouring region of three points was chosen as it produced good results in preliminary experiments.

$$rank(p_i) = speed\_metric(p_i) * curvature\_metric(p_i, 3) \tag{6.8}$$

For each range, the highest ranked point is considered a candidate point for segmenting the user stroke. However, using all candidate points for this purpose could lead to bad results as the stroke could easily be over segmented. Likewise, using too few of these points could lead to a segmentation that does not closely fit the original user stroke. Therefore, each candidate segmentation combination is evaluated regarding its fitness to the original user stroke using a distance measure between the segmentation candidate and the original stroke.

Effectively, the fitness of a segmentation is an error function ε given by the average of the squared orthogonal distance of every point in the original stroke to the corresponding line segment in the segmentation. The orthogonal distance between points in the original user stroke and the segmentation lines is shown in Figure 6.5. This implies that segmentations with lower values of this fitness function match more closely the original user stroke. Equation 6.9, where $dist(p, l)$ is the orthogonal squared distance between point p and its corresponding line segment l, defines this error function in terms of the candidate segmentation C and the original stroke S.

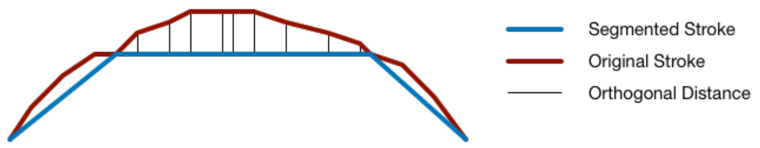


Figure 6.5: Orthogonal distance (black) between points in the original user stroke (red) and the respective segmentation line (blue).

$$\varepsilon(C, S) = \frac{1}{|S|} \sum_{i=0}^{n} dist(p_i, l_i) \tag{6.9}$$

As the segmentation uses points taken from the original stroke, applying only this function to the candidates would, invariably, lead to the candidates that contained more points. Therefore, the candidate with fewer points which has the error value below a given threshold is considered the best segmentation which is used as the final one to be passed along the sketch recognition pipeline.

6.4 Converting Strokes to World-Space Lines

Upon completing the segmentation of the original user stroke, the system now has a set of bidimensional line segments on screen space. It is now necessary to convert them into lines within the world space and correctly insert them into the scene. This allows the user to perform a set of sketches from one viewpoint and, then, continue sketching from another viewpoint, while the existing lines are kept in the correct place.

Moreover, it allows the procedural inference subsystem (described in the next section) to infer which operations best match the user intention. As this subsystem performs the matching in world space, converting the lines from screen space to world space is, indeed, required.

As there is an infinite number of 3D lines that can be projected onto the same screen space line due to depth ambiguity, finding the line in world space that best matches one of the lines produced by the user stroke is a highly ambiguous problem. The naive approach of projecting the line onto existing geometry while trying to find the best match in world space, although partially solving the depth ambiguity, comes with several issues of its own.

First, geometry is not always available. When a user starts creating a 3D model of a building the only geometry that is present is, usually, the building lot. This geometry is, by nature, contained within a plane. While it is possible to envision some operations to be recognised by sketching on a planar surface (e.g., split and repeat split), there is a recurrent need to add volumetric geometry to the building being modelled. It is trivial to imagine that, if a user tried to sketch a line orthogonal to a given planar shape (e.g., indicating that an extrusion should be applied), the line projected onto the scene could potentially intersect neighbouring geometry or no geometry at all as seen in Figure 6.6.

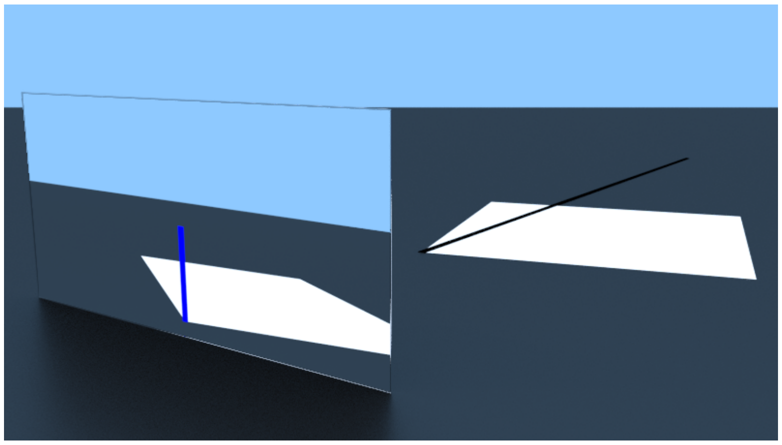


Figure 6.6: A line orthogonal to a planar shape may lose such property when projected onto the scene.

This leads to a second problem where lines projected onto the scene don't always correspond to the user intended topology and shape. Given the same example as before, a 2D line whose screen projection appears to be orthogonal to a planar shape may not retain such property after being projected onto the world geometry. As the 2D line could be projected onto several shapes, it is also possible that the projection results in several, non collinear lines. These situations are illustrated in Figure 6.7.

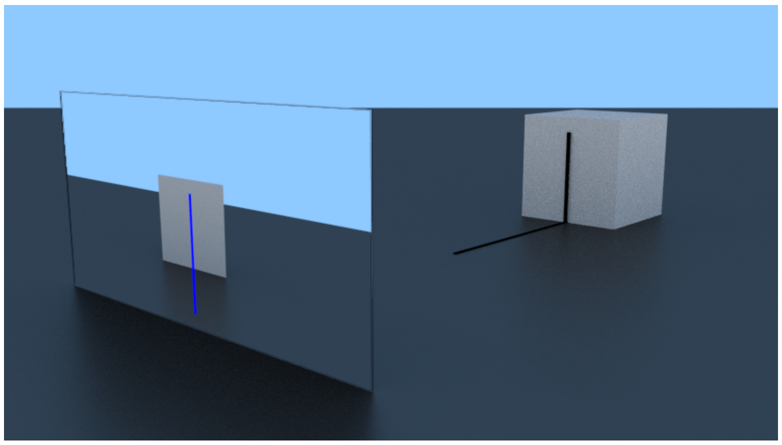


Figure 6.7: A vertical line on the screen when projected onto existing geometry may lose such property.

6.4.1 Line Candidate Generation

As seen, searching in world space for an ideal candidate 3D line that matches a given user stroke is not feasible. However, reversing the approach can be a more successful solution. In other words, generating candidate lines in 3D space which are, then, projected and compared to the original line in screen space reduces both the search space and the problem complexity.

For one, the number of world lines whose projection results in the same screen space line is reduced to the number of candidate lines that are coplanar within themselves and the user's viewpoint. In practice this number is rather low as for such situations to happen the user would have to place the system's viewpoint in very specific configurations.

Secondly, since the main purpose of the sketch based system described in this chapter is to create procedural models, it is possible to exploit the available procedural operations to inform the generation of candidate lines to be matched to the screen space line. This also reduces the search space for this problem as it is known beforehand the families of lines that can be conceived with the system.

A family of lines, in this case, is a set of lines that were extracted from the geometry generated by applying a procedural operation to a given shape while varying its parameters. For example, applying the extrusion operation to a given shape while varying its extrusion value results in the family of lines in Figure 6.8.

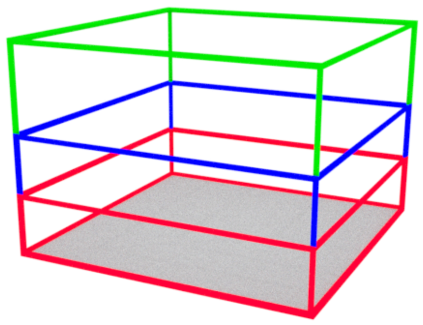


Figure 6.8: An example of a family of lines for an extrusion operation where the red, blue and green lines correspond to extrusion values of 1, 2 and 4.

Therefore, the system operates by applying all available procedural operations, varying its parameters, to the shape on which the user is currently sketching. The generated geometries are, obviously, not inserted into the scene but are analysed and their edges are extracted and inserted into a set of candidate lines.

With the exception of the procedural operation parameters, we have all the information necessary to generate candidate lines. Noting that a procedural operation has a set of parameters $\langle p_0, \cdots, p_n \rangle$ where each parameter has its own domain defined by its type (e.g., numeric parameters have a domain in $\mathbb{R}$), a solution is to sample from an interval within each parameter's domain.

Non numerical parameters, which usually represent procedural operation behaviour modifiers (e.g., on which axis should a shape be split), have a finite and discrete domain (e.g., x, y or z for split operations) and the system simply considers all possible combinations. For numeric parameters, as they usually represent continuous values such as extrusion amounts and split sizes, the sampling interval is defined based on the distance d between the shape and the system's current viewpoint position. This is to allow the sampling to properly scale based on the potential size of candidate line. For example, if the user's intention is to extrude a building lot in order to create the building volume, the amount to extrude would possibly be in the range of several meters. On the other hand, if the user is adding finer detail, such as creating the window frames, the range would be in the range of centimetres.

In the first scenario the system's viewpoint would have to be farther away from the shape (the building lot) in such a way that the user would be able to sketch an entire extrusion line corresponding to the desired extrusion amount. In the latter case, however, the viewpoint would need to be closer to the shape (the window). In both cases the user should be allowed to sketch small lines and, therefore, the system should be able to generate sample values near zero and, as such, the sampling interval is defined as $\delta = [0, \sigma]$.

To calculate the interval's maximum value let us first consider a right circular cone whose vertex is placed at the system's viewpoint position, the axis is oriented towards the visible part of the scene and the base is at the same distance as the shape being considered. The angle Θ between its axis and the generatrix is given by the system's maximum field of view (either the vertical or horizontal) which is a value that is commonly available in any rendering system and takes part in the projection matrix. The maximum value σ of the interval is thus defined as the radius of the cone's base circle. Therefore, σ is calculated as in Equation 6.10.

$$\sigma = tan(\Theta) \cdot d \tag{6.10}$$

Within the σ interval for a given parameter, the system uniformly samples N values, where N is calculated based on the screen space length of the user sketched line and is given by Equation 6.11 where $|s|$ is the screen space length, in pixels, of the sketched line and α is a configurable parameter which, intuitively, specifies the distance in pixels between samples.

In the implementation this value was set to 10.

$$N(s) = \frac{|s|}{\alpha} \tag{6.11}$$

Therefore, given the shape on which the user is sketching, the set of generated candidate lines is given with the algorithm in Listing 6.1.

```
function LineCandidates($S$):
    let candidates ← {}
    for op in set of procedural operations:
        for sp in sample(op.parameters):
            let geo ← apply_operation($S$, op, sp)
            candidates ← candidates + extract_edges(geo)
    return candidates
```

Listing 6.1: Algorithm to generate candidate lines for a given shape.

Each parameter p_i in a given procedural operation is then sampled based on whether it is a discrete or continuous parameter. In Listing 6.1 the *sample* function returns a set of tuples where each element corresponds to a different combination of parameter values as given by the sampling criteria explained above. Function *apply_operation* returns the geometry generated by applying the current operation *op* and sampled parameters *sp* to the input shape *S*. The geometry edges are, afterwards, extracted and inserted into the set of candidate lines.

6.4.2 Candidate Line Evaluation

Having generated the full set of candidate lines, the system then has to find which is the best match to the current screen space line. As this is done in screen space, the first step is to project each candidate line to screen. Since every property of a line segment (e.g., length and direction) is implicitly defined by its endpoints the candidate line evaluation is done by comparing the endpoints of the projected candidate line and the original screen space line. Both candidate line points are then projected onto the screen.

For each candidate line l_i the system projects its endpoints to screen space resulting in the projected points p_{l0} and p_{l1}. These points are then compared against the endpoints P_0 and P_1 of the original screen space line to find a fitness value between both lines which is computed based on the distance between each pair of endpoints belonging to different lines.

Let us first define a good evaluation function between a pair of points. Such function should favour points that are closer and, ideally, eliminate situations where the distance between them is far too great to be considered a viable candidate. The chosen function, given that its properties exactly match the aforementioned requirements, was the *Gaussian* function. Therefore, the fitness value between two endpoints p_i and p_j is given by δ Equation 6.12, where $length(p_i, p_j)$ represents the euclidean distance between both points. The r parameter controls the distance value after which this function yields a value of zero. Intuitively, one can think of this value as defining a circle of radius *r* around one of the points wherein the other point should lie in order to be considered a viable candidate.

$$\delta(p_i, p_j) = e^{-\frac{length(p_i - p_j)^2}{r^2}} \tag{6.12}$$

We are now in place to properly define the fitness function that compares the projected candidate line and the original screen space line, which is defined in Equation 6.13.

$$line\_fitness(P_0, P_1, p'_0, p'_1) = max(\delta(P_0, p'_0) * \delta(P_1, p'_1), \delta(P_0, p'_1) * \delta(P_1, p'_0)) \tag{6.13}$$

It is not desirable to force the user to sketch in a given direction for the stroke to be correctly converted into a world space line. The user should be free to produce a line in any of the two possible ways. For example, in case of a vertical line the user should be able to sketch downwards or upwards. Therefore, the system must take into account such situations and this is reflected in the evaluation function by considering both combinations of endpoints and taking the higher value as the effective fitness value.

The candidate line whose projected points present the best fitness to the original screen space line is then inserted into a configuration of world space lines. Every line in the configuration is rendered and shown to the user even if there is a change in the viewpoint, allowing the user to freely navigate the scene and arrange the best point of view that would allow the user a better drawing perspective.

6.5 Heuristics for the Inference of Procedural Operations

At this point, the system has knowledge of a configuration of world space lines that have been extracted from the segmented strokes using the method presented in the previous section. It is now possible to start the process of inferring the intended procedural operation from such lines which is achieved in two phases. The first stage analyses the configuration of shapes with a set of heuristics which try to infer specific uses of procedural operations (e.g., extrusion or splits) and roughly estimate the parameter values of such operations. The second stage is responsible to adjust the parameters of such procedural operations, using a gradient descent method, such that the generated geometry matches as closely as possible the current configuration of world space lines. This second stage will be discussed in the next section (Section 6.6).

Each inferred candidate procedural operation is sorted by its fitness value and presented to the user in an expectation list. The user is then responsible to choose which of the presented procedural operation is to be inserted into the rule base.

As previously mentioned, it is possible to produce (or reuse) sampled parameter values for each procedural operation and compare the generated geometry to the configuration of lines and, posteriorly, adjust the parameter values on the operation that results on the best fit. By using heuristics, however, the system can considerably reduce the number of candidate operations to test.

When the heuristics don't produce a good fit for the current configuration of lines, the system resorts to using the sampling approach as a fall-back, guaranteeing that a set of inferred procedural operations is available for the user to choose from.

As heuristics are intended to analyse configurations of lines and infer what procedural operation the user intended to apply to a given set of shapes while reducing the number of candidates to generate and compare, it is first necessary to explain what constitutes a good heuristic.

As this system employs heuristics to reduce the set of candidates to analyse further down the pipeline, it is important that a heuristic does not produce candidates that are not feasible, given a current configuration of lines. For example, if a configuration of lines has a line orthogonal to a planar shape, then it is highly unlikely that the user intended operation was a split or repeat split operation.

A heuristic should also produce a candidate procedural operation whose parameters are already close to the final solution. This accelerates the process of adjusting the parameters to better match the current configuration of lines as the gradient descent method requires fewer iterations to converge to a solution. Furthermore, it should also try to find relations, whenever possible, between the parameters that might be implicit in the configuration of lines. This is, for example, especially important in the split operation where often several of the split sizes must be equal. The heuristic, in this case, must inform the gradient descent method that such parameters must be kept equal throughout the process.

In order to take the maximum advantage of using heuristics for this purpose, the system must possess a heuristic for, at least, the most common procedural operations used. Empirically, these were considered to be the *extrusion*, *split*, *repeat split* and the *component split* operations. With this set of operations, it is already possible to create a large variety of buildings. Moreover, these were the original operations defined in the work of Müller et al. [Mül+06]. Therefore, the set of heuristics defined in the current section include heuristics for the mentioned operations and a heuristic for the offset operation. More heuristics can be implemented for other types of procedural operations and this is left for future work.

6.5.1 Extrusion Heuristic

The extrusion heuristic is a rather simple one. It analyses a configuration of lines searching for lines that start at one of the vertices of the current shape's geometry and are orthogonal to the shape or lines that are parallel to the edges of the current geometry. All lines in the configuration must follow this criterion as otherwise means that an extrusion is an infeasible operation to perform given the configuration of lines.

If, otherwise, all lines present such property this heuristic constructs an extrusion procedural operation whose extrusion value is set to the average of maximum distance between the shape and the given line. In other words, the maximum distance is either the distance between the shape and the line, in case it is parallel to one of the edges, or it is the opposite endpoint in case it is a line orthogonal to the shape.

6.5.2 Offset Heuristic

The offset heuristic searches in a configuration of lines for lines that are either collinear with the bissectrix between two adjacent edges in the geometry or lines parallel to a geometry edge whose endpoints are placed in the bissectrix between such edge and both of its adjacent edges. As previously, if a line is found not possessing such properties the offset operation is considered to be infeasible and the offset heuristic doesn't generate any candidate operation.

On the other hand, if all lines match the criteria this heuristic generates an offset procedural operation. The offset parameter value is, similarly to the extrusion heuristic, set to the average of the maximum distance between the lines and the corresponding geometry edge.

6.5.3 Split and Repeat Split Heuristic

The split and repeat split operations, due to their similarities, have their heuristics merged into a single one. As in the previous cases, this heuristic searches for lines that match given criteria and check for the feasibility of the line configuration to represent a split or repeat split operation.

The heuristic starts by checking along which axis the split is performed. It, therefore, checks that all lines are orthogonal to the same split axis, which by definition of the split operations must be one of the scope axis. Whenever there is at least one line that does not match this criterion this heuristic does not produce any candidate split operation.

As the split and repeat split operations are able to divide volumetric shapes, some lines may be in different faces of the shape. Each line in the configuration is, therefore, considered as being part of a split plane orthogonal to the split axis. Lines are clustered into split planes by finding lines that are within a threshold distance of one another. A split plane is created based on the average distance to the scope position of each clustered line. A split size is here defined as the distance between two consecutive split planes. Moreover, there are two implicit split planes which are placed at a distance of zero from the scope position and at a distance equal to the scope size along the respective axis.

This heuristic also has the responsibility of detecting split sizes that the user might consider as being equal. This is achieved by finding groups of split sizes that have similar values.

A split size group is constructed by collecting all split sizes whose split values do not differ by more than a given threshold from one another.

This threshold is given in order to accommodate for imprecisions while sketching and is relative to the scope size along the split axis which implicitly takes into account the distance between the system's viewpoint and the current shape. In other words, there is an implicit assumption that, for the user to have been able to sketch all split lines across the shape the system viewpoint must have been at a sufficient distance such that the entire shape is visible. Therefore, the larger a scope is, the farther away the viewpoint is from the shape and the sketching precision decreases. This threshold was empirically determined as five percent of the scope size along the axis.

At this moment, the heuristic already has sufficient information to produce a candidate split operation, setting its scope axis and split sizes as previously described. However, an extra step is required to produce a repeat split operation. The heuristic further analyses the split sizes and if the standard deviation between the split size values is less than 0.5 the heuristic generates a repeat split operation with the split size value equal to the average of the split sizes. This value was also determined empirically during preliminary tests.

6.5.4 Component Split Heuristic

The component split heuristic is, yet again, a rather simple one. As the output geometry of a component split operation consists of the individual faces of the input geometry, it is visually similar to the input geometry. Therefore, the extracted edges are the equal to the input edges.

Based on this, the component split heuristic analyses all lines in the current configuration of lines and if at least one of the lines is not coincident with one of the extracted edges it does not create a candidate component split operation. Otherwise, it creates a single component split candidate. However, the heuristic is not able to differentiate the faces with different selectors (e.g., top or side faces) as this depends on future operation that might operate on the faces differently. Therefore, the heuristic sets a singe selector on the created operation to select all faces.

The user is responsible, at this point, to modify the operation and set different selectors if required. Note that, by using the semantic queries, it is still possible to detail different faces using filters (e.g., *:up* for up faces or *:north* for north faces).

6.5.5 Sample Based Generation of Candidate Procedural Operations

To guarantee that the system generates procedural operation candidates to the subsequent stage in the pipeline, the system falls-back into sampling the parameter space of all available procedural operations. The generated candidates are then passed along to the next stage.

This approach is only used whenever the heuristics fail to generate valid candidates and is particularly useful for compound procedural operations (i.e., groups of user-defined and reusable procedural operations). It is impossible to conceive heuristics for such cases as the combinations of operations is unknown. Furthermore, the use of sampled based generation of candidate procedural operations serves the purpose of completeness in the system.

In practice, and to increase efficiency, the procedural operations generated in the line candidate generation phase (Section 6.4.1) are reused.

6.6 Parameter Adjustment

Once the candidate procedural operations have been obtained, either through the use of heuristics or by sampling the parameter space of procedural operations, the system must now find the set of parameters that match as closely as possible the configuration of lines. This is achieved by using a gradient descent method that converges the candidate operations to minimise an error function. Such error function is, in itself, a measure of distance between the user created configuration of lines and the edges extracted from the geometry obtained by applying the procedural operation and the sampled parameters to the shape the user is currently sketching on.

The candidate operations that have the smaller error value when compared to the configuration of lines are presented to the user in a decreasing order of its error value. The user must then select which candidate operation best fits the user intention.

6.6.1 Error Function

In the scenario of procedural operation parameter adjustment the system must try to find the combination of procedural operation parameter values that result in the set of lines that best match the user created configuration of lines. It is, therefore, a similarity metric in the sense that a lower error metric indicates that the two sets of lines are more similar.

Let us first define what are the goals of this error function while comparing two single line segments. It is important to note that the error function should be monotonic in all its domain to guarantee that the gradient descent method converges to the optimal solution and is not trapped in local optima. Therefore, it would be impossible to use functions such as the Gaussian as described in the candidate line evaluation (Section 6.4.1).

As a line segment is fully defined by both of its endpoints it is reasonable that a distance metric over line segments relies on the coordinates of such points. Therefore, to compute the distance value between two lines the system first encodes the coordinates of both lines in a vector in $\mathbb{R}^6$, here called line descriptor, such that $desc_{ab} = (x_a, y_a, z_a, x_b, y_b, z_b)$ which describes a line segment $\overline{ab}$ where $a = (x_a, y_a, z_a)$ and $b = (x_b, y_b, z_b)$ are its endpoints.

The error metric ε between two lines l_1 and l_2 can now be defined as the squared distance between its descriptors $desc_1$ and $desc_2$. This definition is given in Equation 6.14.

$$\varepsilon(l_1, l_2) = \sum_{i=0}^{5}(desc_{l1}[i] - desc_{l2}[i])^2 \tag{6.14}$$

The error metric between the user created configuration of lines and the lines extracted from the generated geometry is defined as the sum of the ε value between each line $l \in L$ in the configuration of lines and the extracted line $e \in E$ that minimises the ε value. In other words, the system finds the set of tuples $\langle l, e \rangle$ such that $\varepsilon(l, e)$ contributes the least to the overall error metric. The error metric between this two sets of lines is then defined as in Equation 6.15.

$$\varepsilon(L, E) = \sum_{i=0} \min_{e \in E}(\varepsilon(l_i, e)) \tag{6.15}$$

This error metric is the used to guide the gradient descent method such that it converges towards the procedural operation parameters that result in the geometry that best matches the user sketch.

6.6.2 Gradient Descent Method for Parameter Adjustment

To adjust the parameter values of a procedural operation in a way that the output geometry is similar to the user sketched lines the system uses a gradient descent method which is explained in this subsection. The gradient descent method is an iterative optimization algorithm whose aim is to find the minimum of a function. Therefore, it is important to formulate the current problem as a function.

As the goal is to adjust the parameter values such that it minimises the error function described before, this function should have the parameter as its independent variables and the value of the error function for such parameter values should be output. As such, the function to optimize is the one given in Equation 6.16. In this function, *op* represents the procedural operation, *S* is the shape on which the user is sketching and *L* is the current configuration of lines.

$$F_{op,S,L}(p_0, \cdots, p_n) = \varepsilon(L, edges(op(p_0, \cdots, p_n, S))) \tag{6.16}$$

This function first generates the output geometry for the given procedural operation, parameter values and shape (by treating *op* as a function) whose edges are posteriorly extracted with the *edges* function. The error function between the extracted edges and the configuration of lines is then calculated.

Starting with a given combination of parameter values, the system must find the direction which provides a faster path to minimise the error function. Preferably, the values produced by the heuristics described above should be used as these are already a good estimate of the final value. However, given that the error function as described has a single global minimum the process can start with any combination and is guaranteed that the result will converge to the minimum.

To find the direction (which can be thought of as a vector in $\mathbb{R}^n$ space) with the steepest descent towards the minimum, knowing the function gradient $\nabla F(p_o, \cdots, p_n)$ is required. This is computed by finding the function value at the immediate neighbourhood of the point $(p_0, \cdots, p_n)$. Therefore, the system varies each parameter $p_i, i \in [0..n[$ by a small amount δ such that $p'_i = p_i + \delta$ and $p''_i = p_i - \delta$ and calculates the function with both values. Similarly, F'_{p_i} and F''_{p_i} represents the value $F_{op,S,L}$ returns when the i^{th} parameter takes the value of p'_i and p''_i respectively while the remaining parameters are kept unmodified.

The system approximates the partial derivative of each parameter p_i by finding the variation observed in the function F when the parameter is perturbed by $+\delta$ and $-\delta$ respectively and averaging the signed difference between both values by 2δ. For small values of δ this represents a good approximation to the rate by which the function varies according to p_i. The gradient $\nabla F_{op,S,L}$ is then computed as in Equation 6.17 which effectively defines a vector in $\mathbb{R}^n$.

$$\nabla F_{op,S,L}(p_0, \cdots, p_n) = \left(\cdots, \frac{F'_{p_i} - F''_{p_i}}{2\delta}, \cdots \right) \tag{6.17}$$

Having defined the function and its gradient, it is now possible to explain how the gradient descent method operates in order to obtain the parameters that can generate the geometry that best matches the user sketch. As mentioned, the gradient descent method is an iterative approach that, starting at a given point in the parameter space, seeks to find the minimum of a function.

On each iteration the method finds the function value $v_i = F_{op,S,L}(P_i)$ and gradient at point P_i and moves to a new point P_{i+1} in the parameter space by adding a fraction of the gradient such that $P_{i+1} = P_i + \alpha \nabla F$, where α is the step size. It then repeats the process until a predetermined number of iterations have passed or the variation in the function value is neglegible (i.e., when $|v_i - v_{i+1}| \approx 0$). To avoid excessive oscillations when the gradient descent is close to the minimum the step size is reduced by a fraction of its current value. The algorithm is depicted in Listing 6.2.

```
function GradientDescent(P_0, op, S, L, numIterations, stepSize,
   factor):
   let P_i ← P_0
   let v ← []
   for i in [0..numIterations]:
      let v[i] ← $F_{op,S,L}$(P_i)
      P_i ← P_i + stepSize * $\nabla$ F(P_i)
      stepSize ← factor*stepSize
      if |v[i]-v[i-1]| ≈ 0:
         break
  return P_i
```

Listing 6.2: The gradient descent method.

In the previous listing P_0 is the initial combination of parameter values, *op*, S and L represent the candidate procedural operation, the shape on which the user is sketching and L is the current configuration of lines respectively. The number of iteration the gradient descent method should execute is given by *numIterations*, the step size is given by *stepSize* and the fraction by which the step size should be reduced after each iteration is given by *factor*.

Note that each time $F_{op,S,L}$ is computed for any combination of parameter values (including for $p_i + \partial$ and $p_i - \partial$), a procedural operation is performed and the edges of its geometry are extracted. This is a potentially expensive operation and is why the heuristics described before play a fundamental role in accelerating the process.

Once the system has produced a set of candidate procedural operations whose parameters have been adjusted to fit the user configuration of lines as closely as possible, they are sorted by increasing value of their error function ε and presented to the user. At this moment, the user is free to choose the candidate that best matches the intentions.

The system will then create a procedural rule with the query the user specified to indicate to which shapes the operation should be applied and the inferred procedural operation and parameters. Finally, this rule is inserted as a free node in the selection action tree that generates the current model.

6.7 Tools for Precise Sketching of Architecture

Although the presented contributions towards sketch based procedural modelling already allow creating procedural operations based on user sketches, the techniques presented do not accommodate well for more precise ways of drawing architecture. For example, it is extremely difficult to split a given shape in two shapes with the exact same size without modifying the parameters in the resulting procedural operation.

Therefore, a set of techniques to allow the user to create architectural models more precisely is required. Even though traditional WIMP based interfaces provide the utmost accurate way of defining measures as the user can specify the desired measure with the widgets provided (e.g., a text box), it forces the user to switch the attention to other parts of the interface, possibly disrupting the creative flow. It is desirable, then, to provide tools that do not steer the user away from the sketch based interface.

6.7.1 Analytical Drawing

Inspired in the drawing techniques that exist in the field of analytical drawing, Schmidt et al. [Sch+09] proposed a system that allows users to define support lines that help the user to precisely create lines in 3D space. As the authors noted, this set of tools are ideally suited to architectural drawing which is the ultimate goal of this thesis.

The logic behind these techniques and, as such, in this system is to allow the users to sketch temporary lines which take advantage of known geometrical properties which, in turn, will aid the user in the creation of the final geometry. Analytical drawing consists of drawing sets of regulating lines which express 3D relationships in the drawing. These lines will provide context and constraints with which the user can, more accurately, specify the lines that make the final geometry with enhanced precision.

In geometric perspective drawing, artists rely on points and directions with well-known characteristics to define new points and lines incrementally. Starting with an empty sheet or canvas, the artist usually starts by defining the horizon line, often referred to as eye level. Objects below this line are below the eye level and, otherwise, are above the eye level. The artist can then define vanishing points on the horizon line to which parallel lines in the scene being drawn converge. From the vanishing points, perspective lines can be drawn which can serve as a basis on which the artist can start sketching the scene.

To further enhance the user's capability of creating precise geometry only using the sketching interface, a subset of the techniques defined in the work of Schmidt et al. [Sch+09] were implemented in the prototype. Namely, the guidelines that allow users to create a line extending to a vanishing point starting from a given vertex in the geometry, and support lines which allow users to create lines that will not be taken into account in the procedural operation inference but can be useful to create the final procedural lines that will trigger such system. The inference of these type of lines is achieved in the same manner as described in Schmidt et al. [Sch+09].

Although the authors kept the bi-dimensional representation of a sketch, in this implementation this is discarded as the lines are automatically converted into 3D space. Moreover, as the present system generates new geometry keeping both representations would easily become desynchronised and could lead to image clutter and confuse the user.

6.7.2 Guidelines

While drawing, artists often rely on vanishing points to correctly create the illusion of perspective. In a drawing under geometric perspective, a vanishing line represents a 3D direction through a given point which intersects the horizon at a vanishing point. Parallel 3D directions always converge into the same vanishing point. In this system, vanishing lines are referred to as guidelines.

Vanishing lines are, then, instrumental in architectural drawings as these subjects, being man-made, often possess large quantities of parallel and orthogonal lines. Given a set of points in a drawing, the artist can create lines that resemble parallelism under geometric perspective simply by connecting each given point to the same vanishing point. Likewise, to create the illusion of intersecting orthogonal lines, the artist connects the intersection point to the vanishing points associated with orthogonal directions.

Guidelines are defined by a point P_0 and a direction $\vec{d}$ such that $P_0 \in \{\textit{scene vertices}\}$ and $\vec{d} \in \{\textit{scene edge directions}\}$. Therefore, a guideline can only be constructed from already existing geometry as it requires a scene vertex and the direction of a scene edge. Scene vertices can be defined as any vertex that belongs to the existing model geometry, or any sort of intersection between guidelines, support lines or geometry edges.

To create a guideline the user, after selecting the guideline sketching mode (by clicking in the *Guidelines* button in the top bar shown in Figure 6.1), starts sketching from an existing vertex towards the direction of a vanishing point. Once the user starts sketching a line in such mode, the system presents visual cues that show the potential guidelines that can be created from the chosen point. The guideline is created as soon as the user finishes the current stroke and if the endpoint is sufficiently close to one of the given potential guidelines.

Guidelines can then be used to construct other guidelines or support lines as they are well-defined geometric constructs that specifically represent the user's intention. For example, it is possible to take the intersection of two guidelines to create a third guideline in the direction of another vanishing point.

6.7.3 Support Lines

While guidelines are lines that extend infinitely, starting a specific vertex, to both directions of a vanishing point, support lines are line segments that rely on existing geometric constructs to be defined. Therefore, a support line is defined as the line segment between points P_i and P_j as long as these points belong to a precisely defined geometric construct. In other words, they must be coincident with a scene vertex, guideline or support line.

To create a support line the user first selects the support line sketching mode (by clicking in the *Support Line* button in the top bar shown in Figure 6.1) and sketching a line from an existing geometric construct (e.g., a vertex or guideline) to another one. Once the user finishes the stroke the system creates the support line that best matches the line provided by the user.

The support lines alone will not trigger the procedural operation inference mechanism as they have, as the name indicates, merely a support function. It is possible to promote a support line to a procedural line by double sketching over an existing support line. Once a given line is promoted, the inference system starts and a new set of procedural operations are presented as described in Section 6.6.2.

With the use of guidelines and support lines, the user is capable of defining procedural operations that have more precise parameters that match the user intention. For example, it becomes trivial to create split operations that exactly divide a face in half by first finding the centre of the face using two diagonal support lines (Figure 6.9a) and then define a guideline starting from such point and extends to the horizontal vanishing point (Figure 6.9b). This guideline will intersect the face edges at (at least) two other points which can be used to create another support line (Figure 6.9c). By promoting this support line to a procedural line, the system will infer the split operation with two exactly equal split values (Figure 6.9d).

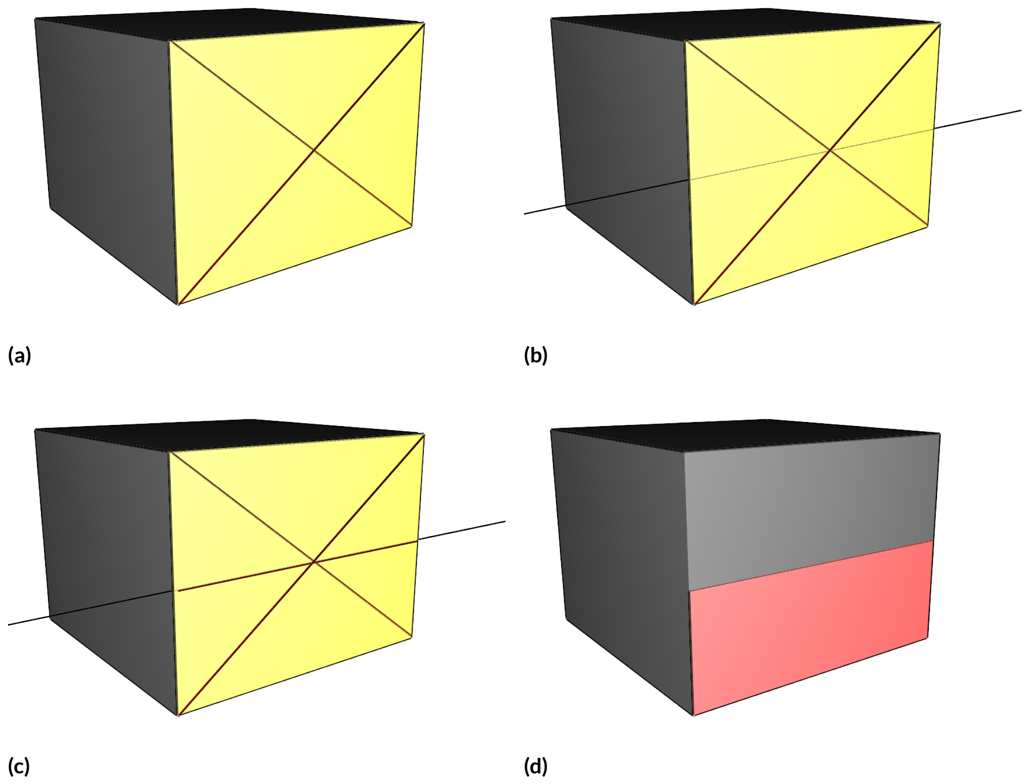


Figure 6.9: Using guidelines and support lines to precisely split a face in half.

6.8 Summary

This chapter presented the contributions towards a sketch based procedural modelling system oriented at creating architectural models. Building on top of the selection action paradigm (Chapter 5), which allows a precise and robust way of specifying which shapes in a procedural model will be affected by a given user sketch, this chapter demonstrates how the end goal of this thesis was achieved.

The field of Sketch Based Interfaces for Modelling has been extensively studied throughout the last decades and this chapter demonstrates how it is possible to reuse some of the previous contributions in the field to develop a sketch based procedural modelling system. This is the case of the stroke filtering and segmentation algorithm proposed by Sezgin et al. [SSD06] and the system based on analytical drawing proposed by Schmidt et al. [Sch+09] from which this work drew inspiration for the candidate line evaluation sub-system.

Given the user provided stroke, the system first proceeds to its filtering and segmentation which yields a clean set of screen-space lines. Posteriorly, and given the shape on which the user is sketching, the system generates a set of candidate world-space lines based on the available procedural operations by sampling their parameter space. These lines are projected onto screen-space and compared to the original, user-provided lines with a Gaussian-based evaluation function. The best match is inserted into a configuration of world-space lines.

Afterwards, the system runs a set of heuristics which propose candidate procedural operations given the current configuration of world-space lines. These heuristics fill the operation parameters with estimated values based on the current set of lines. The purpose is to have a pre-evaluation of the user intention and provide a reduced set of candidates to the following stage in the pipeline which adjusts the parameter values to create geometry that better matches the user-provided lines. This is achieved with a gradient descent method that, for each iteration compares the edges of the candidate geometry to the configuration of lines with an error function based on the distance between pairs of lines. The candidate procedural operations are presented, ranked by their evaluation score, to the user who must then choose the one that best fits the present intention. The operation is inserted in the rule base as a selection-action node with the user-defined query and in a place within the selection-action tree that will affect all the desired shapes.

Although the system allows the user to completely create buildings with a sketch interface with minor deviations to other parts of the user interface, there are some desirable features that are yet to be explored. The system currently only supports creation of rules and the user must resort to the selection-action tree and node parameters to perform modifications to the rule base. Changes to parameter values could be overcome by using oversketching on the geometry to symbolise an alteration to a given parameter. Other modifications would, potentially, require more complex sketching gestures that would symbolise replacement operations in the rule base.

Also, there is currently no integration with layers and vectorial shapes (as presented in Chapter 4) which would further increase the expressiveness of this system. Furthermore, these are features akin to ones found in many diagram and image editing packages which are familiar to many users of artistic background and, more specifically, 3D modellers which are the main intended audience of this system. Both these developments are left for future work.

Finally, the creation and insertion of procedural rules is done in a simple way by inserting the created rule as a free node in the end of the selection action tree. Although this approach works since the query is effectively applied to all existing geometry, this creates a list of nodes with no hierarchy whatsoever and can easily become unmanageable. Further work should put towards solving this issue.

Nonetheless, this approach seems to be a solid starting point from which many more usages could be conceived.

7 Results and Discussion

Evaluation is a crucial part of any research as it allows conclusions regarding the work undertaken to be drawn and understanding what is the impact of the contributions. Therefore, this chapter shows the evaluation results, alongside the corresponding discussion, of each of the contributions presented in this thesis.

It is also in this chapter that the research questions are explicitly answered and the hypothesis confirmed.

7.1 Layered Shape Grammars

To evaluate layered shape grammars, a set of examples exposing the capabilities of such paradigm were conceived and described. Such examples showcase the decomposition of planar shapes into layers and the vectorial definition of shapes and its advantages in the procedural modelling of buildings.

This section also compares the layered shape grammars to other shape grammars (e.g., the CGA grammar [Mül+06]) demonstrating some limitations that can be solved by using layered shape grammars.

Since the greatest contributions of layered shape grammars are related to planar shapes, most of the examples are focused around façades and the increase in expressiveness this extension provides.

7.1.1 Simple Façade Example

The layered shape grammar in Listing 7.1 shows a simple façade containing several layers as well as shapes created with a vectorial definition. Note that the model is completely generated with layered shape grammars and does not include any pre-modelled asset. This is a limitation that often shows in regular shape grammars and, therefore, this represents an improvement in shape grammar expressiveness.

The *Facade* rule creates two layers. As implicit, the *Floors* layer will contain the ground and top floors definition, while the *Entrance* layer will contain the arched door. The separation into two layers, allows the door to span several floors while the normalisation process guarantees that there are no geometry conflicts with the windows. Also, the door placement is completely independent of the floors and windows, enabling more variations by simply adjusting a few parameters.

The *GroundFloor* rule also creates two layers: one for the horizontal stripes and one for the arched windows. Note the seamless integration of the arched shapes with the stripes, resulting in more complex shapes.

```
Facade → layers("Floors","Entrance")
  { FloorsSplit | Entrance2 };
FloorsSplit → segment(y,2,~1)
  { GroundFloor | TopFloors };
TopFloors → segment(y,2)
  { Floor }*;
Floor → segment(x,1.5)
  { TileSegment }*;
TileSegment → segment(x,'0.1,~1,'0.1)
  { ε | TileV | ε };
TileV → segment(y,'0.1,~1,'0.1)
  { ε | WindowTile | ε };
GroundFloor → layers("stripes","windows")
  { Stripes | GroundWindows };
GroundWindows → segment(x,0.9,1,0.9)
  { ε | GroundWindowTile | ε }*;
GroundWindowTile → segment(y,0.35,0.1,~1,0.1)
  { ε | extrude(0.35) | ArchedWindow| ε };
ArchedWindow →
  vectorial("M0,0 L0,0.5 C0,1 1,1 1,0.5 L1,0Z")
  extrude(-0.3);
Entrance → segment(x,~1,3,~1)
  { ε | EntranceTile | ε };
EntranceTile → segment(y,3.5,~1)
  { ArchedDoor | ε };
ArchedDoor →
  vectorial("M0,0 L0,0.5 C0,1 1,1 1,0.5 L1,0Z")
  extrude(-0.5);
Stripes → segment(y,'0.2,'0.2')
  { extrude(0.1) | extrude(0.2) }*;
```

Listing 7.1: Layered shape grammar example.

Some rules were ommitted for brevity. The resulting façade can be seen in Fig 7.1a and its decomposition into layers can be seen in Fig 7.1b. Note that, although the vectorial definition for the door and the windows is the same, the generated shapes are of different size since the polygon is interpolated inside the predecessor's procedural item bounding rectangle.

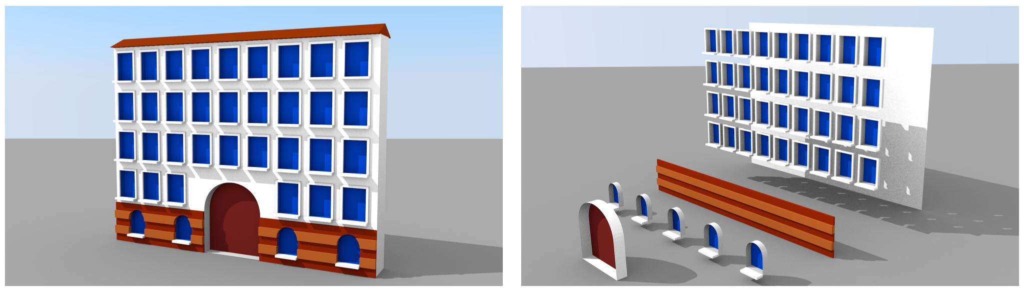

(a) (b)

Figure 7.1: The example façade (Subsection 7.1.1), on the left, and its decomposition into layers, on the right.

As a side note, the use of interactive layers (as described in Section 8.2.1) would allow the user to specify both the position, size and alignments of each layer interactively and completely independent of each other.

7.1.2 Expressiveness

The use of layers and vectorially defined shapes within planar shapes grants a new set of possibilities allowing for a greater range of procedural models to be created. Such improvement in expressiveness is evaluated in this subsection by demonstrating some examples that can be quite difficult to achieve using current procedural modelling techniques.

Complex Windows

Layers and the planar shape normalisation algorithm allow the creation of more complex configuration of shapes by allowing several shapes to be overlapping. In Listing 7.2 it is possible to see a layered shape grammar that creates a window made up of several similar structures.

```
MultiComponentWindow → layers("Bottom", "Middle", "Top")
  { WindowBottom | WindowMiddle | WindowTop }
WindowBottom → segment(x,~1,5,~1)
  { ε | WindowBottomVSegment | ε }
WindowBottomVSegment → segment(y,~1,5,~1)
  { ε | WindowComponent | ε }
WindowMiddle → segment(x,~5,3,~1)
  { ε | WindowMiddleVSegment | ε }
WindowMiddleVSegment → segment(y,~10,2,~1)
  { ε | WindowComponent | ε }
WindowTop → segment(x,~1,3,~1)
  { ε | WindowTopVSegment | ε }
```

```
WindowTopVSegment → segment(y,~1,2~10)
  { ε | WindowComponent | ε }
WindowComponent → ...
```

Listing 7.2: Layered shape grammar that uses layers to create complex windows.

This layered shape grammar creates three layers on a planar shape with the *MultiComponentWindow* rule. Each layer, in turn, positions the window component within the layer by using vertical and horizontal segments with different dimensions. The bottom-most layer, defined by *WindowBottom* creates the largest central window component, the middle layer (*WindowMiddle* rule) creates the top right component and, finally, the *WindowTop* rule creates the bottom window component. The *WindowComponent* rule is responsible for creating each component of the window, including the window frames, by using more segment operations and extrusions. The fixed geometry property of the window frames was set to false causing the frames of a deeper window component to adapt to the ones above it. The results of this layered shape grammar can be seen in Figure 7.2.

The layered approach also has the advantage of easily toggling on or off the execution of a layer allowing for a greater variability of generated models. Figure 7.3 shows the results obtained by switching off the execution of one or more layers in Listing 7.2. As such, Figure 7.3a shows the case where layer *MultiComponentWindow.Middle* has been switched off, Figure 7.3b represents the execution without the generation of the *MultiComponentWindow.Top* results and, finally, 7.3c does not have the bottom-most window component

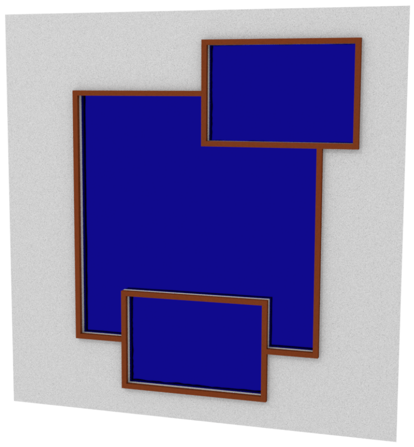


Figure 7.2: A complex window created with layers.

obtained by turning the execution of *MultiComponentWindow.Bottom* layer off.

As a side note, it is possible to see that each layer is defined with a rule structure similar to that defined in Section 8.2.1. Therefore, each layer could be turned into an interactive layer where users could perform the interactive operations defined in Section 8.2.1.

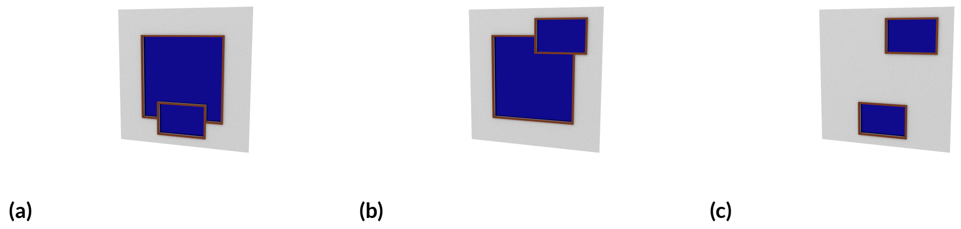


Figure 7.3: Variations obtained by toggling on or off some layers in Listing 7.2.

Modelling the Mikimoto Ginza Building

To further demonstrate the improvements in expressiveness layered shape grammars provide over split based methods, this section aims at showcasing an approximation to the modelling of the Mikimoto Ginza building (Figure 4.1). This building was selected for its irregular façade shapes and structure which gives it a random look.

As the main features to be highlighted in this model are within its façades, this section will describe in great detail the approach taken in its generation focusing on the façades. The combination of rules to achieve the building coarse volume from its axiom and its further separation into façades is a simple extrusion followed by a component split as can be seen in Listing 7.3.

```
Mikimoto → extrude(50) Mass
Mass → comp(face, side){ Facade }
```

Listing 7.3: Rules to generate the mass volume and each façade of the Mikimoto Building

The main idea this example aims to demonstrate is that placement of shapes can occur outside regular grid-like structures. For this, windows were placed such that there is no split line (either vertical or horizontal) across the facade that can effectively be able to partition it. Although this is not the case in the original Mikimoto building, it serves to illustrate the improvements over split based methods.

In this example this is achieved by conceiving two overlapping layers that subdivide the façades into regular grids using the segment operation, creating cells of similar sizes. However by offsetting one of the layers slightly upwards and to the right allows the placement of shapes (in this case, the windows) over the other layer's segment lines which can be seen in the Figure 7.4. There is an additional layer that generates the horizontal stripe in the top area of each façade. Listing 7.4 shows the creation of layers and the rule that generates the horizontal stripe.

```
Facade → layers("Windows1", "Windows2", "Stripe")
  {Windows1, Windows2, Stripe}
Stripe → extrude(0.5)
```

Listing 7.4: Rule that generate the layers for each façade.

The bottom-most layer (Figure 7.4a) is generated with the layered shape grammar extract in Listing 7.5 where it is possible to see the creation of a segment for the entrance (rule *Windows1*) and the generation of a regular grid with rules *TopFloorWindows1*, *Floors1* and *Floors2*.

```
Windows1 → segment(y, 2, ~1)
  { Entrance, TopFloorsWindows1 }
TopFloorsWindows1 → segment(y, 5, 5)
  { Floors1, Floors2 }*
Floors1 → segment(x,5,5)
  { Window1, ε }*
```

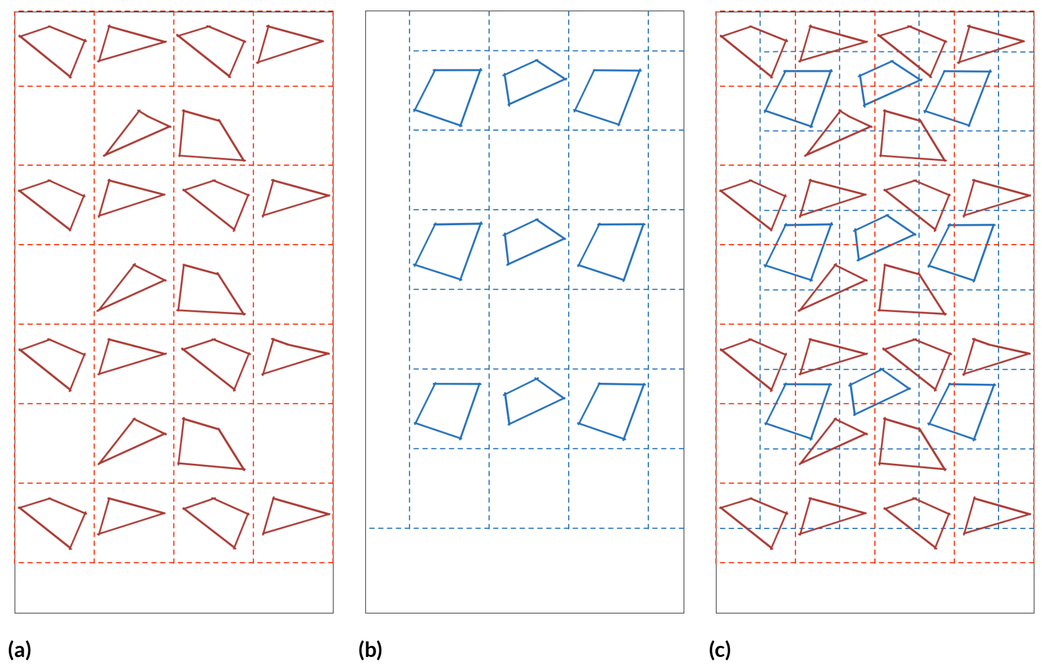


Figure 7.4: Layers used to place the windows in the Mikimoto Ginza building.

```
Floors2 -> segment(x, 5, 5)
  { ε, Window2 }*
```

Listing 7.5: Layered shape grammar to generate the bottom layer for the Mikimoto Building

Similarly, the second layer of windows is achieved by partitioning the façade into a regular grid with cells, albeit slightly offset from the previous layer. This can be seen in Listing 7.6. The empty shape in rules *Windows2* and *Windows2HPadding* represents the vertical and horizontal offset this regular grid has in relation to the previous layer.

```
Windows2 → segment(y, 3.125, ~1)
  { ε, Windows2HPadding }
```

```
Windows2HPadding → segment(x, 2.5, ~1)
    { ε, TopFloorsWindows2 }
TopFloorsWindows2 → segment(y, 5, 5)
  { Floors3, Floors4 }*
Floors3 → segment(x, 5, 5)
  { ε, Window4 }*
Floors4 → segment(x, 5, 5)
  { Window3, ε }
```

Listing 7.6: Layered shape grammar to generate the second layer of windows.

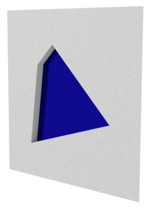
Each of the cells is then further detailed with an irregularly shaped window by using the vectorial operation. In this example, four different windows were used and can be seen in Figure 7.5.

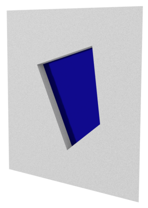
```
VecWindow1 → vectorial("M0,0 L1,0.3 L0.3,1 L0.1,0.8 Z", 30)
    WindowExtr;
VecWindow2 → vectorial("M0.5,0 L0.9,0.3 L0.8,1 L0.1,0.8 Z", 30)
    WindowExtr;
VecWindow3 → vectorial("M0,0.1 L0.5,0 L1,0.1 L0.8,1Z", 30)
    WindowExtr;
VecWindow4 → vectorial("M0,0 L0.8,0.1 L1,0.5 L0.9,1 L0.1,1Z", 30)
    WindowExtr;
WindowExtr → extrude(-0.5);
```

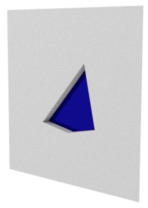
Listing 7.7: Vectorial definition of each of the windows in the Mikimoto Ginza building.

The planar shape normalisation process guarantees that the correct and intended geometry is generated, resulting in the model presented in Figure 7.6.

Note how the horizontal stripe in the higher part of the building is overlapping and oc-


(a)


(b)


(c)

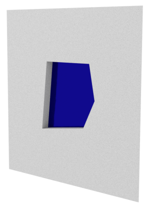

(d)

Figure 7.5: Windows used for the Mikimoto building (Figure 7.6).

cluding some windows. This feature is also present in the original Mikimoto Ginza building and cannot be modelled solely with the use of split based methods hence proving a greater level of expressiveness.

Apart from being possible to conceive a line that partitions the Mikimoto building façades, another difference with respect to the presented model is that the latter only has four distinct shapes for windows whereas each window of the original building has a different shape. This is due to the nature of procedural modelling being based on patterns and repetitions which is, in fact, one of the greatest drawbacks in procedural modelling. Moreover, the current layered shape grammar implementation does not implement stochastic,

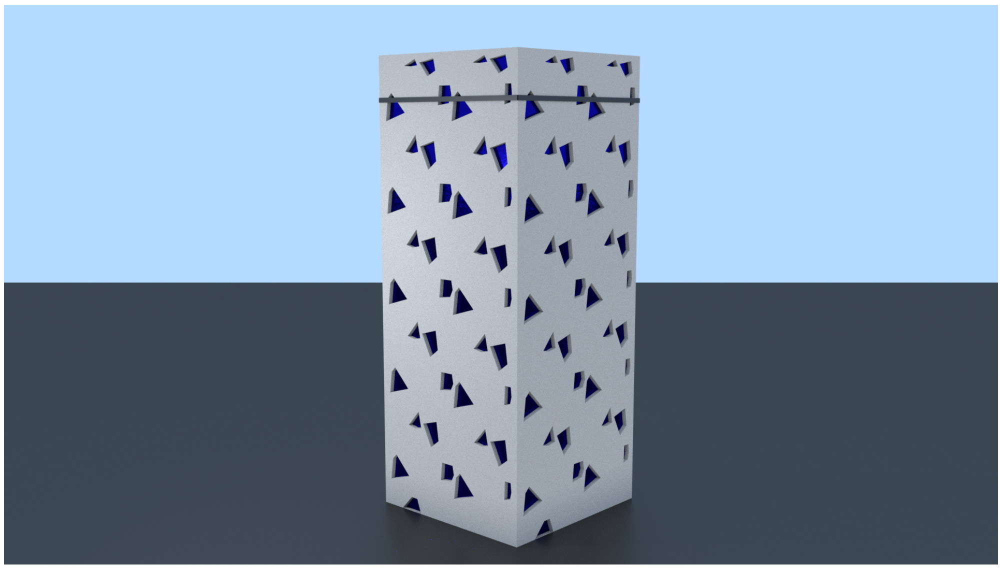


Figure 7.6: An approximation of the Mikimoto Ginza (Figure 4.1) created with layered shape grammars.

conditional or random rules and operations, making the generation of a façade with an apparent randomness impossible.

7.1.3 Comparison to Split Based Methods

To generate the simple façade example (Section 7.1.1) using split based methods only, several modifications were introduced and are present in Listing 7.8.

```
Facade → split(y,4,~1)
  { GroundFloor | TopFloors }
TopFloors → split(y, 2)
  { Floor }*
Floor → split(x,1.5)
  { 1.5: Tile }*
Tile → split(x,'0.1,~1,'0.1)
```

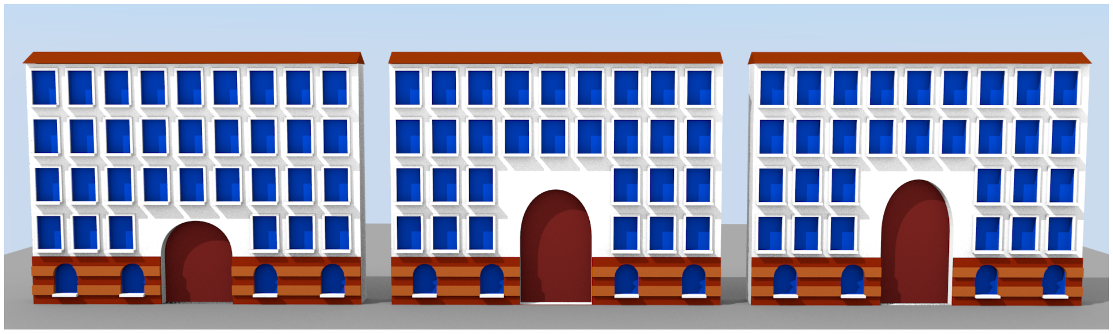
```
  { ε | TileV | ε }
TileV → split(y,'0.1,~1,'0.1)
  { ε | WindowTile | ε }
GroundFloor → split(x,4.5,4.5,4.5)
  { MFSide | EntranceTile | MFSide }
MFSide → split(y,2,2)
  { GroundFloorSide | TopFloors }
GroundFloorSide → split(x,~1,~1,~1)
  { Stripes | GroundWindowTile | Stripes }*
GroundWindowTile → segment(y,0.35,~1,0.1)
  { Stripe1 | WindowAsset | Stripe1 }
EntranceTile → split(x,~1,3,~1)
  { EStripes | DoorAsset | EStripes }
EStripes → split(y,2,~1)
  { Stripes | ε }
```

Listing 7.8: Comparison to regular shape grammars.

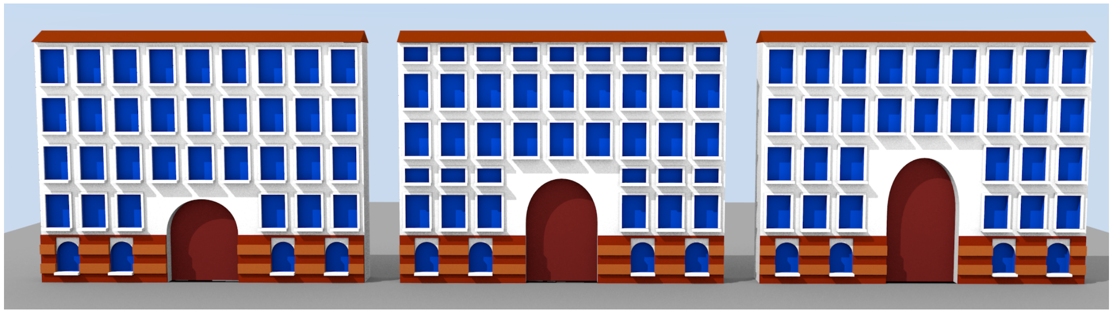
Split operations now have to, simultaneously, account for the entrance, floors and horizontal stripes. Layers, on the other hand, treat these elements separately, which provides a greater amount of variability with fewer modifications to the rule base, while still providing correct results.

For example in the layered shape grammars system, modifying the entrance height in the façade can be achieved by tuning the segment operation parameters in the *EntranceTile* rule in the previous example. In the split based method, this height is, already, implicitly defined in the *Facade* rule. Modifying the split parameters leads to unwanted repercussions in other parts of the façade. This is illustrated in Figure 7.7, where the images show an increasing entrance height and, while this methodology (Figure 7.7a) keeps the windows with a constant size, the split-based method deforms two of the façade floors (Figure 7.7b).

Furthermore, interactive operations as hypothesised in Section 8.2.1 in regular shape grammars are rather complex to implement as procedural operations and the generated procedural items are not sufficiently isolated from one another. As a consequence, modifying one parameter would have consequences on other parts of the procedural model.



(a)



(b)

Figure 7.7: Changing the entrance size using the layered approach, 7.7a, leads to better results than with the split based approach, 7.7b.

Another consequence of the layered approach, is that shape placement is not constrained by split lines and, as such, intentional misalignments of shapes are easily achieved. In Figure 7.8 we see the same façade further detailed with a two-storey balcony in an additional layer which is not aligned with the regular grid of windows. This example also shows that complex modifications can be introduced with new layers. On the other hand, with traditional shape grammars several procedural rules would have to be rewritten.

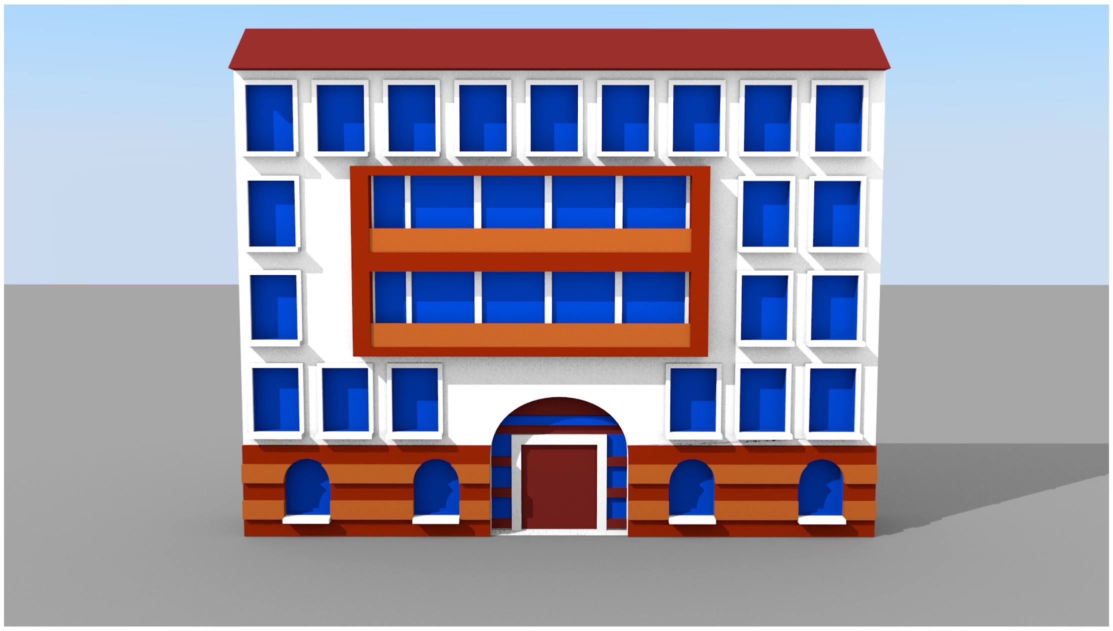


Figure 7.8: Intentionally misaligned balconies in the example façade.

Moreover, in split-based methods, the arched elements have to be imported as pre-modelled assets and, as seen in the ground floor windows (Figure 7.9), can be difficult to integrate. By using a vectorial definition, the planar shape normalisation process allows the seamless composition of these windows and the horizontal stripes.

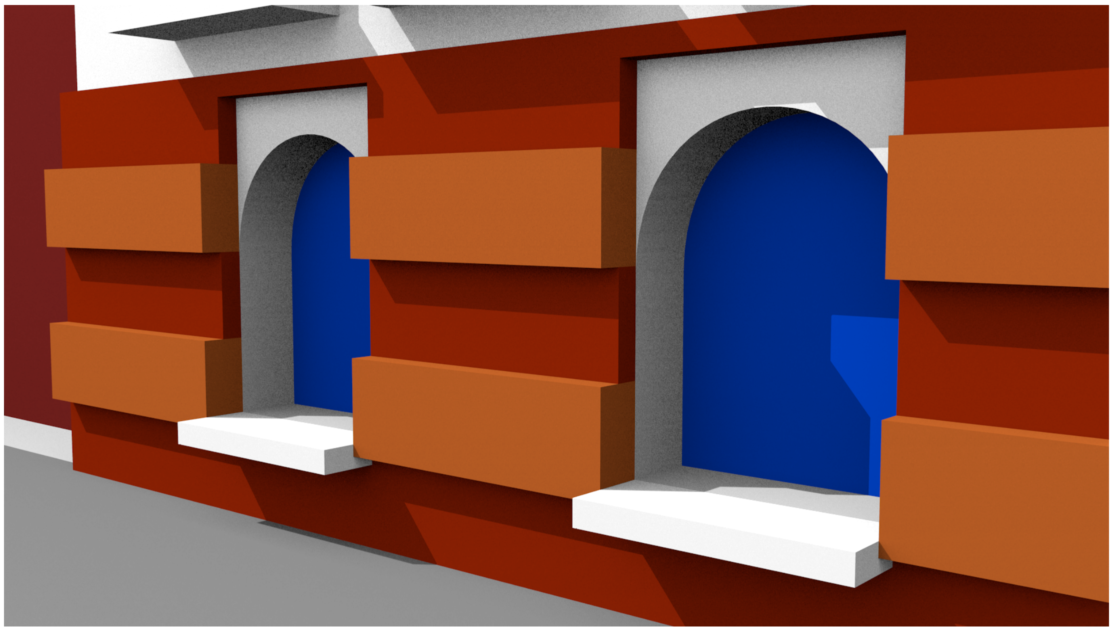


Figure 7.9: Detail of the ground windows in the split-based approach.

7.2 Generalised Selections for Improved Direct Control in Procedural Modelling

To evaluate the proposed selection action paradigm for procedural modelling of urban environments, along with the user interaction and user selection inference, this section presents a set of examples exposing the capabilities of such paradigm. This section presents several different evaluations that were performed with this evaluation.

Firstly, a short modelling session explaining how the user would interact with the paradigm is presented which is, then, followed by an evaluation of the system usability. This is followed by some examples of semantic queries over a building model showcasing the expressiveness of semantic queries and a comparison between the selection action paradigm and other procedural modelling methodologies. Finally, the user selection inference is evaluated with an experiment using synthetic data.

7.2.1 Modelling Session Example

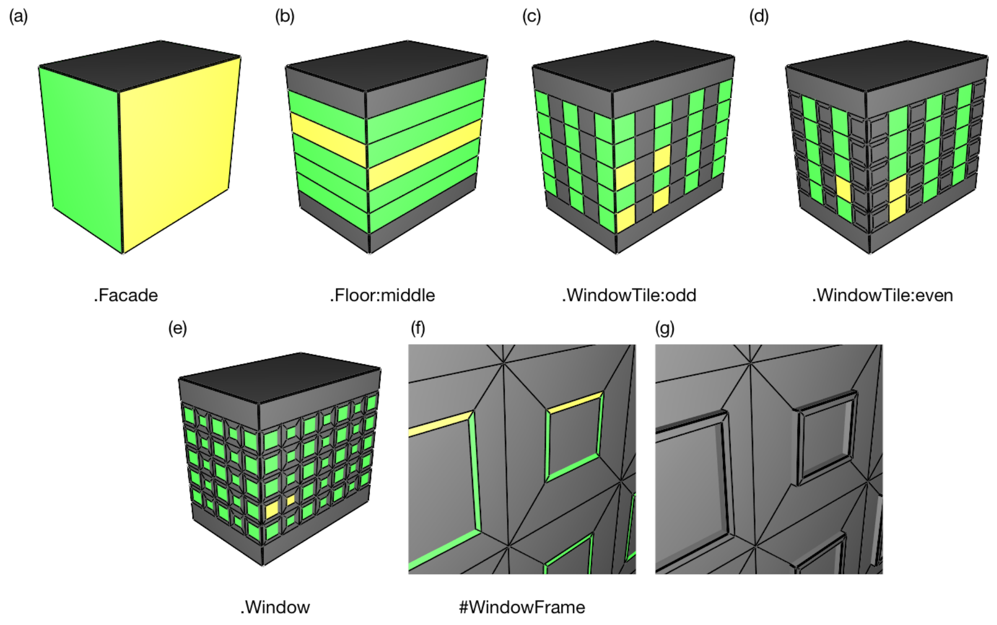


Figure 7.10: A modelling session with this system showing several selections (top row) and the operations applied (bottom row).

To illustrate this system, consider the example modelling session on Figure 7.10.

The selection-action tree used in the modelling session is presented in Figure 7.11 where each number corresponds to a step in the modelling session.

The user starts with a building volume already decomposed into its several façades. The

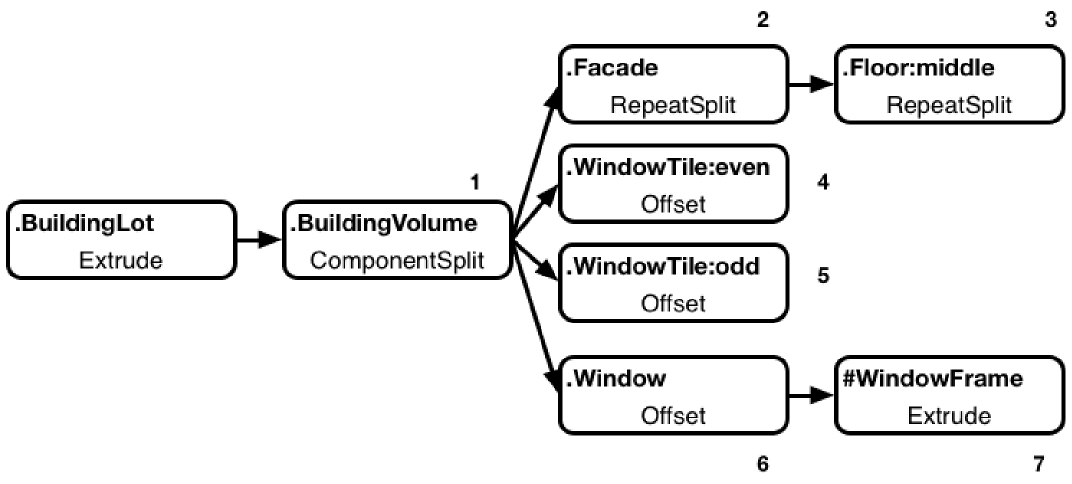


Figure 7.11: The selection-action tree for the example.

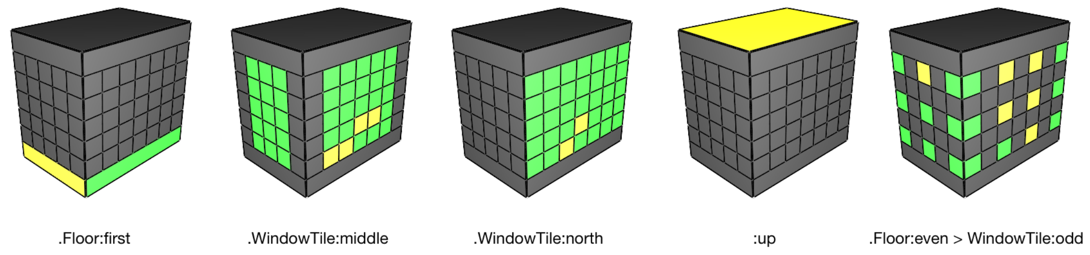


Figure 7.12: Possible user selections in the example in Figure 7.10. User selections are in yellow, results of applying one of the suggested queries.

user selects a single façade (illustrated in Figure 7.10a in yellow) to which the system suggests the *.Facade* query, selecting all façades (show in green). With this query the user instantiates a repeat split node dividing all façades into floors. Upon selecting two of the floors the system proposes the *.Floor:middle* query which the user selects (Figure 7.10b). Yet again, the user instantiates a repeat split node to divide all middle floors into tiles.

Next, the user selects three of the odd indexed tiles (Figure 7.10c) to which the system proposes the *.WindowTile:odd* query. After selecting such query, the user creates an offset node creating a shape for the window. Similarly, the user selects two of the even indexed tiles, selects the *.WindowTile:even* proposed query and creates another offset node to create a smaller window (Figure 7.10d). After selecting a few windows, the system suggests the *.Window* query which is selected by the user (Figure 7.10e). A new offset node creates the glass and window frames. After selecting two of the window frames the system proposes the *#WindowFrame* which the user selects to, posteriorly, create an extrusion node (Figure 7.10f). A rendering of the building modelled up to this point is presented in Figure 7.13.

Some other examples of selections that the user could have performed during the modelling session are presented in Figure 7.12.

Note that the user is always able to select and operate on shapes regardless of what operation was responsible for its generation, decoupling the geometry from the procedural rules.

7.2.2 Usability

To evaluate the usability of this method, seven users (three of which expert users in procedural modelling) were introduced to the prototype. These users, all with a computer science background, have used procedural modelling in industry and research settings. In a first step, the users watched a video introducing the methodology and explaining the interaction with the prototype, followed by a short discussion where they were allowed to clarify

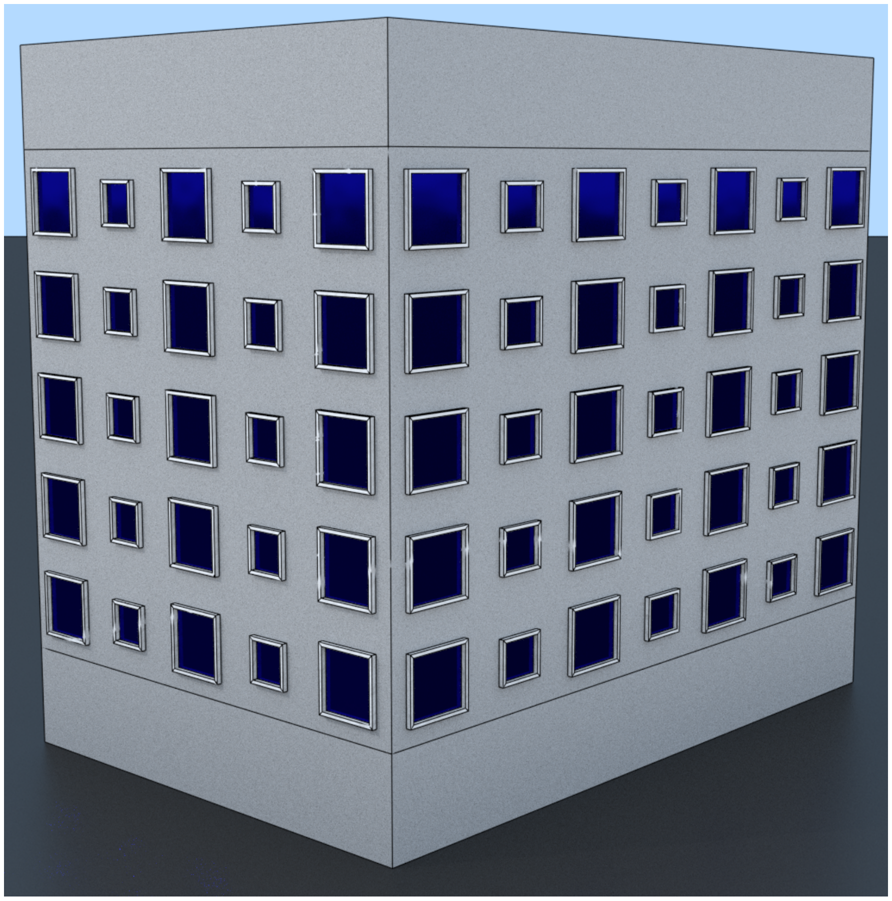


Figure 7.13: Example of a building created with the selection-action paradigm.

any doubts. They were then asked to use the prototype. For this, the users had to follow a small script detailing the operations they had to take and, as a final step, were allowed to experiment freely with the prototype.

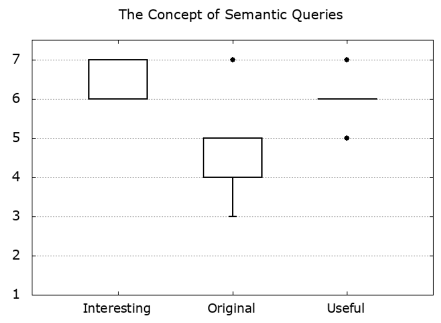
Posteriorly, they filled a short questionnaire (Appendix A) on their impressions regarding the methodology and the prototype, followed by a short discussion. This consisted in a series of questions in a *Likert* scale ranging from 1 (a mostly negative opinion) to 7 (mostly positive opinion). In Figure 7.14a we show the results for the questions aimed at evaluating the concept of semantic queries, the queries generated by the inference system, the impact such system has in the procedural modelling process and how the methodology impacted the whole creation process.

Figure 7.14a shows how the users evaluated the concept of semantic query regarding its interest, usefulness and whether they think it allows original models to be created within a procedural modelling framework.

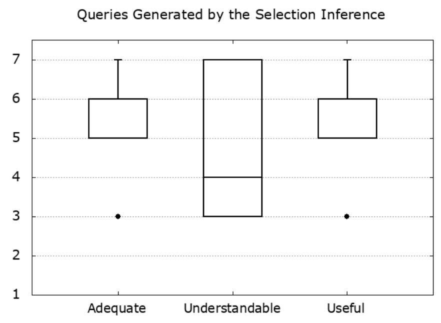
The users were also asked to rank the generated queries in terms of their adequacy within the modelling context (i.e., if they matched the user's intention) and usefulness (i.e., if they were useful during the specific task). The questionnaire also measured if the generated queries were easy to understand. This is shown in Figure 7.14b.

Regarding the value of having a selection inference system within a procedural modelling framework, the users were asked to provide their opinion on whether they think such system allowed a faster, simpler and more intuitive creation of models. These results can be seen in Figure 7.14c.

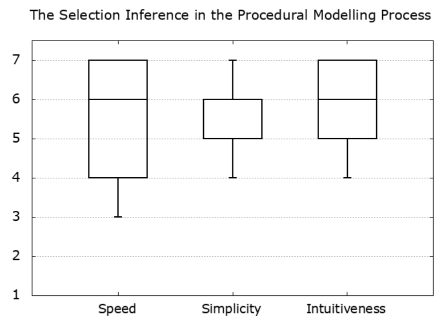
Finally, the users were also asked about how they felt about the proposed methodology for procedural modelling in general and if it allows the creation of models more expressively and more creatively. Knowing if the users felt that this methodology required a greater level of mental effort was also considered an interesting question. These results are illustrated in



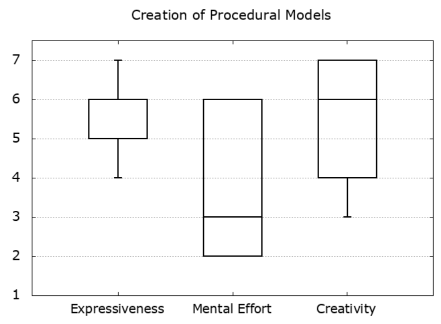
(a)



(b)



(c)



(d)

Figure 7.14: Summary of some answers to the user evaluation questionnaire. Values are in a *Likert* scale where higher values mean a more positive user opinion for a parameter.

Figure 7.14d.

As can be seen, the users evaluated the methodology with a neutral or positive value (i.e., having an average value equal or above 4) in almost all the parameters. The single exception is the mental effort parameter which has an average value of 3.57. This means that the users were required a higher mental effort to use the methodology. Nonetheless, it is also possible for the mental effort to be reduced after being exposed to this methodology for a longer time. More research into understanding whether this is the case and improving this issue is required.

During the informal discussion most of the users mentioned that the queries were very powerful and allowed to express selections (and, therefore, perform actions upon them) that were extremely difficult using current procedural modelling methodologies. Moreover, some users stated that they felt the methodology made procedural modelling more accessible and easier, especially for users that are not experienced with this type of modelling.

On the other hand, users commented on the lack of visual cues to help on the creation of the queries. Therefore, the users tended to express that they had to make a great mental effort in order to understand the structure of the shape tree and know which symbols, tags and filters were available. Some users noted and suggested that an auto-complete system could help the user refine the queries by issuing partial queries and limiting the next query components. Another suggestion was to implement a shape inspector, which would help visualize the symbols and tags of a given set of selected shapes.

One of the users expressed concerns over the volatility of hierarchical relations when new rules are created which can, potentially, break queries in existing rules. It is possible, however, to argue that this is an issue that is innate to many types of systems (not only in computer graphics but in other fields) and that some systems are more robust than others. Shape grammars, for example, present a similar drawback whereas node-based systems, due to the explicit links between nodes, are less volatile to changes. This is, indeed, a point that requires further research.

7.2.3 Selection Examples

To showcase the expressiveness of the semantic selections Figure 7.15 shows some examples on a simple building (top left).

The second image on the top row illustrates how even (or odd) floors can be selected within this building with a simple query (*#Floor:even*), combining the *Floor* tag and *even* filter. The use of tags allows an extra layer of semantic information as different façades may have different symbols for the floors.

The following image shows the same principles being applied to columns of windows in all façades by using the *.WindowTile:odd* query. Note that the middle column of windows is not selected as it doesn't have the *WindowTile* symbol. The fourth image chains these two queries using a *children* hierarchical relation, resulting in the selection of all odd windows in all even floors.

The first image in the second row illustrates the selection of all windows in the façade facing east with the *.Facade:east* > *#Window* query. In this case, the *descendants* hierarchical relation was used. In contrast to a previous example, the *Window* tag was used to select all types of windows (ground floor, middle column and remaining windows).

Next it is shown how all frames are selected using the *Frame* tag and the final two images select the frames surrounding all windows (*#Window* $\sim$ *#Frame*) and the door (*#Door* $\sim$ *#Frame*) respectively. To achieve this the *siblings* hierarchical relation was used.

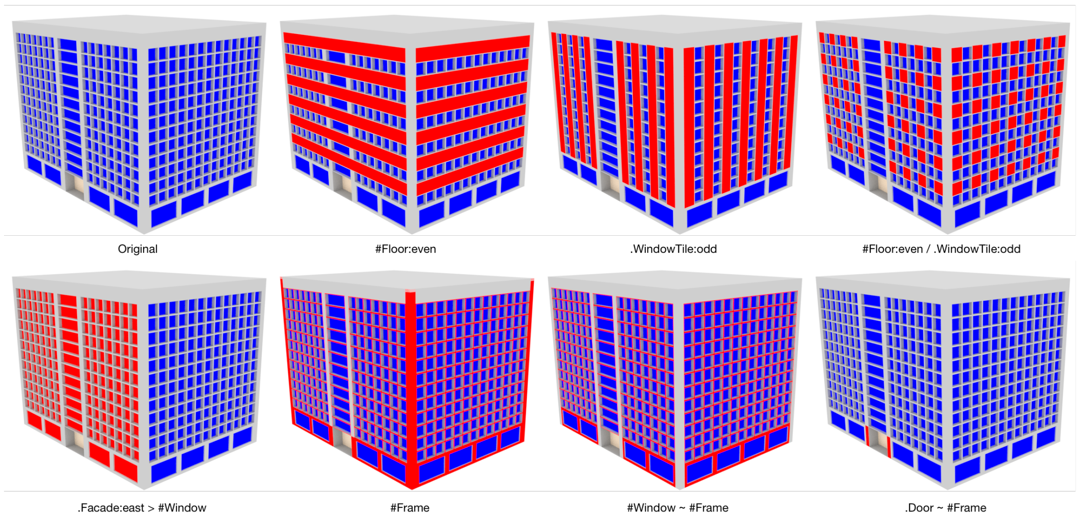


Figure 7.15: Examples of semantic selections over a simple building.

7.2.4 Comparison to existing methods

To evaluate the selection-action paradigm it was also compared with other procedural modelling methodologies, more specifically with the one used by CityEngine [Esr14] and also presented by Patow [Pat12]. Therefore, to achieve the building in Figure 7.16 both methodologies were used and the total number of atomic operations and the time an experienced user took to obtain such result were retrieved.

By using this methodology the user created the selection-action tree represented in Figure 7.17a, accounting for a total of 20 atomic operations. Using the presented system the user took 18 minutes. Using the visual shape-grammar language the user created the graph represented in Figure 7.17b, which has 25 atomic operations. This approach took roughly 25 minutes of user modelling time.

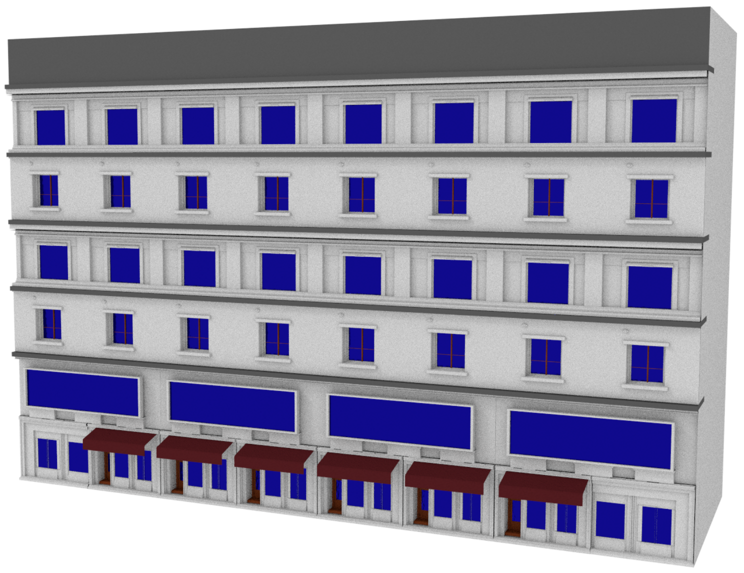


Figure 7.16: Building used to compare traditional methodologies with the one presented here.

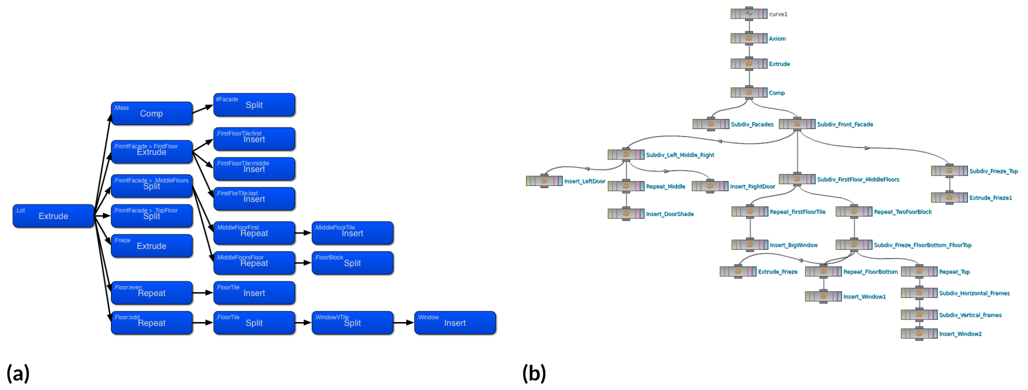


Figure 7.17: Comparison between this methodology (a) and the one presented in [Pat12](b) to generate the model in Figure 7.16.

In the graph representation (as well as in shape grammars) the user is sometimes forced to create different rules that create similar structures to account for different derivation paths. For example, both *Subdiv_Facades* and *Subdiv_Front_Facade* rules in Figure 7.17b produce the division of a façade into ground floor, middle floors and top floor. However, since the user wants to model the front façade differently, only the latter rule is further detailed.

On the other hand, the selection-action paradigm only requires a single operation to create this division. In this case, the *Split* operation with the *#Facade* query. Actually, it is possible to detail the front façade separately with the queries starting with *.FrontFacade*. This difference in operation is the main responsible for the reduction in number of atomic operations.

Moreover, shape grammars do not easily accommodate for changes introduced based on a query. Although it is possible to run a selection inference algorithm in any shape tree, grammars do not possess enough flexibility to directly allow the creation of rules with the result of the inference. Indeed, it would be necessary to visit each rule that generates any of the selected shapes and modify it to reflect the user intention.

Consider, for example a shape grammar rule that creates floors in a façade as shown in the following listing.

```
Facade → split(y) { 1 : Floor }*
```

At this point, if the user selected even floors to detail differently, the inference system might generate the *.Floor:even* query. To reflect the results of such selection in the grammar, the *Facade* rule would have to be changed as shown in the following extract.

```
Facade → split(y) {
   1 : EvenFloor |
   1 : OddFloor
}*
```

Considering hierarchical queries implies more complex changes in the rule base. For example, the *.Facade:north* > *.Floor:even* would have to modify, not only the rules responsible for generating the *Floor* shape, but also those that generate the *Facade* shapes.

```
BuildingVolume → comp(faces){
   north : NorthFacade |
   all : Facade
}
```

These are simple examples but it is trivial to imagine that, as the model evolves and more queries are applied, the complexity of the rules can become unmanageable, creating unnecessary difficulties in modifying existing rules. The solution is the selection-action tree, where operations are directly applied to the selected shapes.

This way, the number of rules is kept to manageable level and are executed in a specific order that the user understands and can inspect visually.

7.2.5 User Selection Inference

To evaluate the inference system synthetic user selections were obtained from sampling some shapes selected by pre-defined queries. It is assumed that these queries represent the user's final intention and that the sampled shapes represent a possible user selection. The queries generated by the system were compared to the original query, retrieving statistics related to the scores and a difference metric.

The difference metric $\delta(q_1, q_2)$ is given by the Levenshtein distance [Lev66] between the original query and a query generated by the inference system. The purpose of this metric is to evaluate how close to the original query the candidates become by calculating the string editing distance between two queries. Although the system may consider queries different from the original it is still possible that it generates a query that selects the same set of shapes as the original. This metric allows to detect these situations.

Therefore, the evaluation starts by issuing a query, representing the user intention, over exemplar shape trees. Fifty percent of the resulting selected shapes are sampled (representing the user selection) which are passed onto the selection inference process. For each iteration of the genetic algorithm, we collect a set of the 20 queries that have the best score.

Six metrics are calculated based on this set: the minimum, maximum and average score and the minimum, maximum and average difference towards the original query. For each example this process was repeated three times averaging the metrics.

Results are demonstrated with two common scenarios: the first is the case of selecting even façade tiles in each even floor in a very simplistic model (given by the *.Floor:even* > *.WindowTile:even* query), while the second selects all windows in the first floor in a more complex model (given by the *.FriezedFloor:first* > *.Window* query).

In the first example, Figure 7.18a shows the synthetic user selection in yellow and the resulting selection of the best query generated by the system (in yellow and green). It is possible to see that from four selected shapes, the system was capable of generating a query that matched the intention.

Figure 7.18b shows the evolution of the minimum, maximum and average scores of the 20 best candidates as the genetic algorithm progresses through 100 iterations. Similarly, Figure 7.18c shows how the minimum, maximum and average difference value (∂) for the same candidates evolve. Note that, from the first iteration there are already candidate queries that select some user shapes.

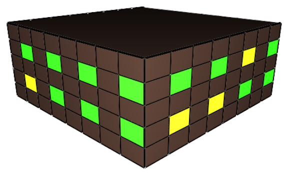
In this case, the best query at the first iteration was the *.WindowTile* with a score of 0.41. There is also a decrease in the difference metric throughout the iterations, showing that the selection inference tends to produce candidates closer to the simulated user intention.

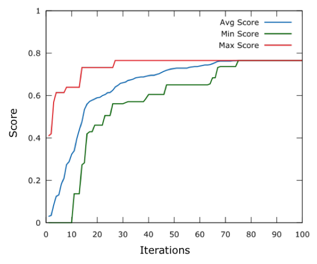
In fact, the difference metric reaches a value of zero (meaning that the *.Floor:even* > *.WindowTile:even* was found) at the 32nd iteration, although the system reports the best query to be *:first* > *.Floor:even* > *:even* with a score of 0.77. This is explained by the fact that different queries might result in the same selected shapes. The third row in Table 7.1 shows that the resulting best candidate for this example was *.Floor:even* > *.Tile:even:middle*.

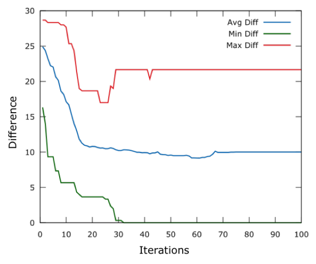
Having a large discrepancy between the maximum and minimum difference values while the maximum score value is high is a good indicator that the system was able to find several queries that closely match the user selection.

Likewise, the bottom row of Figure 7.18 shows the score and difference metrics of the second example. Although it is a more complex model which has a larger shape tree, it is possible to see that the selection inference was capable of generating a query that corresponds exactly to the user intention (as seen on the sixth row in Table 7.1). Figure 7.18d shows the user selection in yellow and the results of applying the best query in yellow and green.

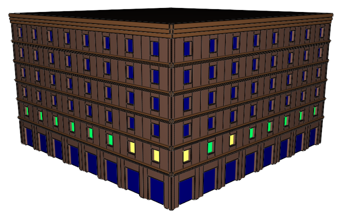
Table 7.1 summarizes some results of the user selection inference system. The *Original* row refers to the query used to simulate the user intention, *Best* refers to the best candidate found through all iterations, *Avg. Time* refers to the average time per iteration, the *Max Score* and *Min Difference* correspond to the best score and minimum difference between a candidate query and the original query. The *Avg. Time*, *Max Score* and *Min Difference* are averaged by the number of times the process was run (i.e., three).

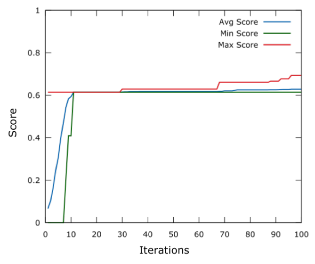


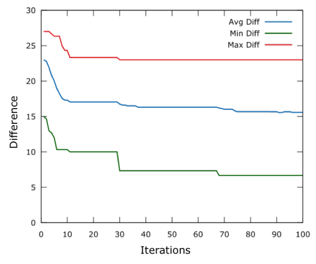




(a) **(b)** **(c)**







(d) **(e)** **(f)**

Figure 7.18: In the left column, for a given user selection (shown in yellow) the system generates a query that selects the shapes shown in yellow and green. For each of the examples, the centre column shows the evolution of the best score and the right column shows the evolution of the difference metric (∂) between the synthetic and the best candidate queries.

Although the selection inference system takes some time to run through each iteration (up to 6 seconds), making it less suitable for an interactive application, note that in most cases the system produces good results in a fraction of these iterations. Reasons for such a long delay include the fact that finding the set of shapes that match a query requires traversing (possibly several times) most of the shape tree. In general, the processing time of queries varies in proportion to the number of selectors as it requires more traversals of the shape tree. As the inference system generates queries that are more adapted to the user selection, they tend to rely more on hierarchical relations and, therefore, the time to run an iteration tends to increase during the inference run. Nonetheless, caching hierarchical results should

		Avg. Time (s/iter)	Max score	Min Difference
Original Best	*.WindowTile* *.WindowTile*	0.35	0.77	0
Original Best	*.Floor:even > .WindowTile:even* *.Floor:even / :even*	0.64	0.774	0
Original Best	*.WindowTile:even* *.WindowTile:even*	2.4	0.74	0
Original Best	*.WindowTile:middle:north* *.Facade / :north > .WindowTile:middle*	0.22	0.66	9.00
Original Best	*.Floor:even > .WindowTile:north:odd* *:north > .Floor:even > .WindowTile*	0.32	0.63	14.33
Original Best	*.FriezedFloor:first > #Glass* *.MiddleFloors > :first > #Glass*	5.61	0.69	6.67

Table 7.1: Results of the user selection inference mechanism for some queries.

largely optimize this process.

It is important to mention that genetic algorithms always have the possibility of generating better candidates given enough time. The implemented solution is to run the inference process on the background and incrementally present new results until the user either selects a candidate or cancels the process, thus reducing the need of an early termination criterion because this process does not block the user from continuing the creation once a suitable candidate is found.

The selection inference score function (Equation 5.19) is also sensitive to the number of shapes the user has selected for a given intention. If the user selects more shapes that are matched by a particular query, the system produces good candidates in a shorter amount of time.

The score for the same query also varies as a function of the number of user selected shapes as the score function is designed to penalize queries that select shapes other than those selected by the user. This is a side effect of trying to solve a problem with incomplete information.

Figure 7.19 shows the evolution of the maximum score for different sampling factors of the shapes selected by the *.Floor:even > .WindowTile:north:odd* query. Although all the best queries generated select exactly the same set of shapes, they are given different scores. The best query generated when only ten percent of the shapes were sampled was *:even:north / .WindowTile:odd* with a score of 0.45. When fifty percent of shapes were selected the system generated the *:north / .Floor:even > :odd* query with a score of 0.68. Finally, when all the shapes are selected, the system presented the *.BuildingVolume > :even:north / .WindowTile:odd* as the best query with a score of 1.

These results demonstrate the flexibility of the selection inference mechanism and present an avenue of research to improve the interactiveness of procedural modelling systems by allowing more direct control.

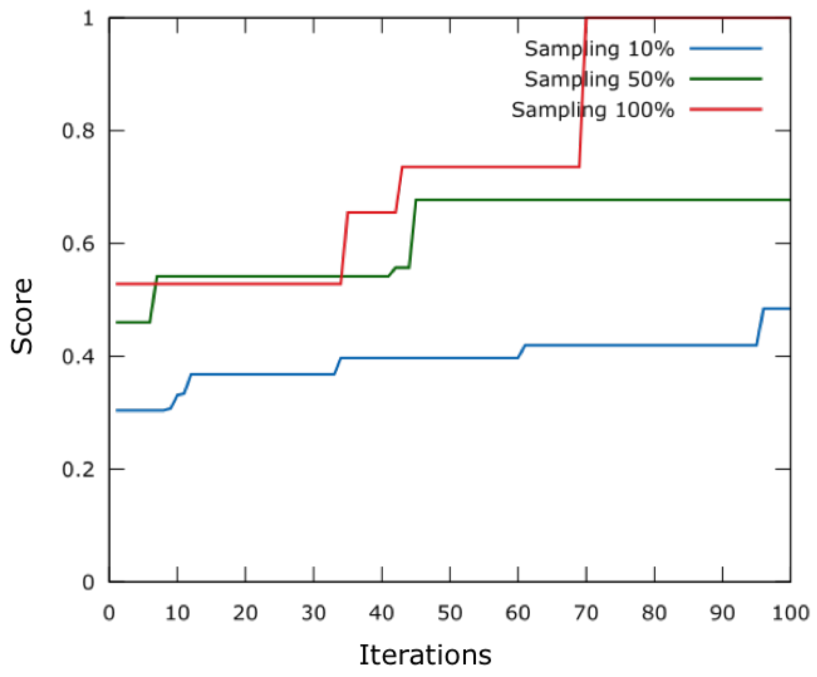


Figure 7.19: Score variation when 10%, 50% and 100% of the shapes are sampled.

7.3 Sketch Based Procedural Modelling

It is utterly important to evaluate the overall contribution of this thesis in an overall way. This is reflected on the sketch based procedural modelling methodology as it relies on the previous contribution of generalized selections and user selection inference.

Being an interactive system, its evaluation calls for an user evaluation as it was done in the previous section. Such evaluation aims at understanding the advantages or disadvantages of integrating procedural modelling with sketch based interfaces for modelling and unfold new avenues for future work.

7.3.1 Methodology

To perform this user evaluation, a set of 8 users were asked to participate in a user evaluation study. This study consisted of three stages: an initial questionnaire for user characterisation; a video describing the methodology and associated prototype with a second questionnaire to collect the users' initial opinions on the methodology; finally, users experimented with the prototype followed by a third questionnaire on the final opinions on the methodology.

In the first stage, users were asked to fill a short questionnaire which aimed to characterise each user regarding their professional background and their experience with traditional modelling and procedural modelling. The users were also asked to provide their opinions on the procedural modelling tools they had previously used. Furthermore, this questionnaire also aimed at understanding what was typically the workflow during the early stages of modelling, i.e., if the users usually created some sort of concept art that guided them through the rest of the modelling process and what sort of tools were used in this process.

After watching a brief video explaining the sketch based methodology, the selection-action paradigm and the selection inference in some detail, the users were asked to fill a second questionnaire. This time, the goal of the questionnaire was to collect the user's opinion about the presented methodology in terms of its perceived usefulness and perceived comparison with other procedural modelling tools. Because some users didn't have previous experience with procedural modelling methodologies, some questions were optional as it would be impossible for the users to provide any sort of accurate answer.

The third stage in the user evaluation started by having the users perform three modelling tasks with the prototype. Users were provided a script which guided them through to the completion of the first two tasks, whereas in the third task they were asked to freely use the prototype to model an example building model.

During the tasks execution some evaluation metrics were retrieved which can be categorised whether as *user-dependant* or *system-dependant*. The former groups metrics such as the number of times the user didn't understand a given concept or made some sort of mistake, while the latter contains metrics relating to the number of times the system did not perform as intended.

Users were, then, asked to fill a third and final questionnaire that aimed at collecting the user's opinions about the prototype and the implemented methodology. This questionnaire contained a subset of the previous one which allows to compare the perceived versus effective usefulness of this methodology. Moreover, by asking the same questions before and after the user had a chance to experiment with the prototype also helped minimise the impact of possible implementation issues in the results. Similar to the previous questionnaire, some questions were optional for users with no procedural modelling background.

7.3.2 User Characterisation

As previously mentioned, eight users participated in the user evaluation for this prototype. In this set of users, five had a software engineering background while the remaining 3 were designers. The questionnaire used can be found in Appendix B. It is also important to mention the fact that three of the users reported not having any sort of previous procedural modelling experience. Figure 7.20 illustrates the answers given by users with regards to their traditional and procedural modelling experience. Therefore, in this analysis the users were divided into two groups. Group *PM* for users with procedural modelling experience and group *NPM* for users with no procedural modelling experience.

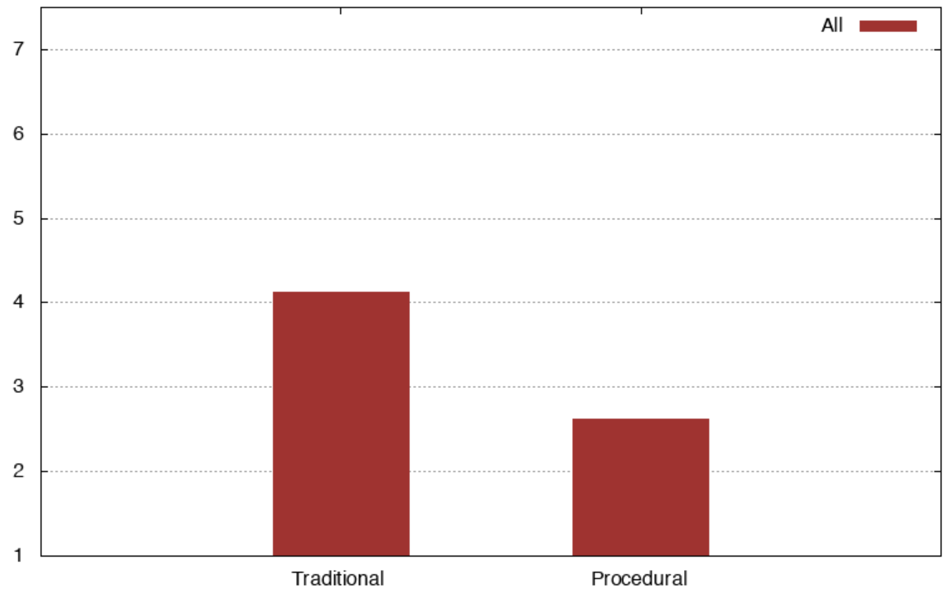


Figure 7.20: User experience with traditional and procedural modelling tools.

To obtain a better understanding of how the users felt regarding procedural modelling tools, the ones with previous experience in such methods were asked about their opinion about the features, interaction and expressiveness of such tools. These results are shown in Figure 7.21 in a scale ranging between a value of 1 (*Horrible*) and 7 (*Great*). Regarding the provided features and the general opinion, the users tended to answer either 4 or 5 denoting a slightly positive opinion. The users' opinion was more divided with respect to the means of interaction and the expressiveness current procedural modelling tools offer.

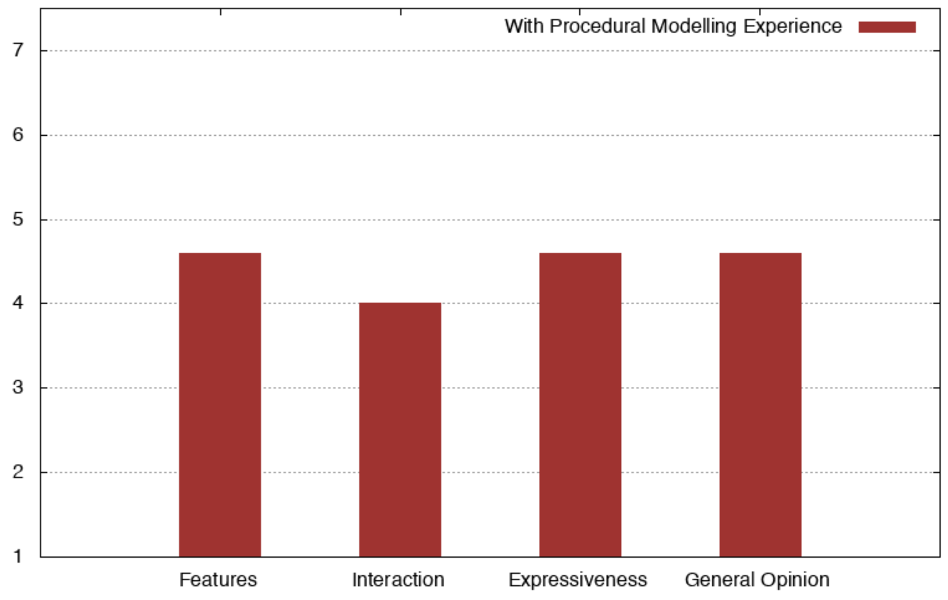
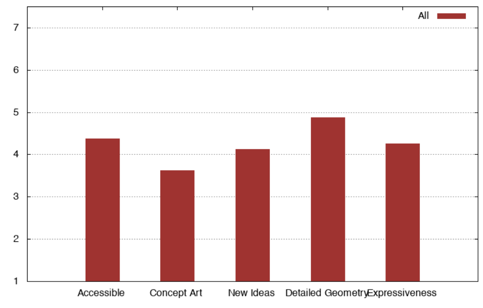
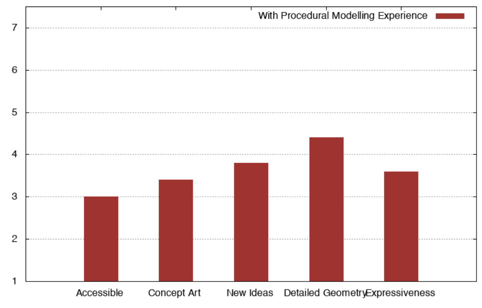


Figure 7.21: Users' opinion about procedural modelling tools.

A single user answered that such tools offered a high level of expressiveness (answering with a value of 6). To be noted, this user reported a high level of experience in procedural



(a)



(b)

Figure 7.22: User's opinions regarding traditional (left) and procedural modelling tools (right)

modelling (with a value of 6). On the other hand, the single user that replied that such tools offer a low level of expressiveness also reported a general lack of experience with procedural modelling (answering with a value of 2). This can be an indication that procedural modelling tools have a steep learning curve.

The users were also asked about their opinion regarding the capabilities of both traditional and procedural modelling methods in terms of whether such techniques are accessible to the user (i.e., easy to use), allow users to easily create concept art, rapidly explore new ideas, whether they are able to create detailed geometry and if they are expressive enough to convey the user's ideas. Figure 7.22a shows the results for traditional tools for all the users.

Regarding the users opinions about procedural modelling tools the results in Figure 7.22b show the answers provided by users with experience in such methods as these were the only ones capable of responding.

From the results, it is possible to observe that users reported that procedural modelling tools are less accessible to the end user than traditional tools. This is expected as current procedural methodologies often require more knowledge of the underlying paradigm and require that users think about a sequence of rules to apply to the geometry. Regarding creating concept art, users reported that the capabilities of both traditional and procedural tools were similar with a value of 3.6 for the former and 3.4 for the latter.

On the capability of exploring new ideas users reported that traditional tools were better equipped (with an average value of 4.1) than procedural tools (with an average value of 3.8). This comes as unexpected since procedural modelling can create more variations by simply changing the parameters. This should present as an advantage over traditional tools as creating variations of a model using such tools requires more complex manipulations over the model.

Unsurprisingly, users reported that traditional tools allowed to create more detailed geometry than procedural tools and, on the other hand, that the former allowed for a greater level of expressiveness when comparing with the latter. This demonstrates one of the well-known drawbacks of procedural modelling which tends to create more repetitive models.

It was also deemed important to understand what tools users tend to employ in the early stages of modelling. Therefore, the initial questionnaire presented a question to clarify whether the users employed the modelling program itself, pencil and paper or any sort of drawing program to obtain digital versions of drawn concept art. These results are presented in Figure 7.23 which shows the averages for all users, for users of a software engineering background and those reported as designers.

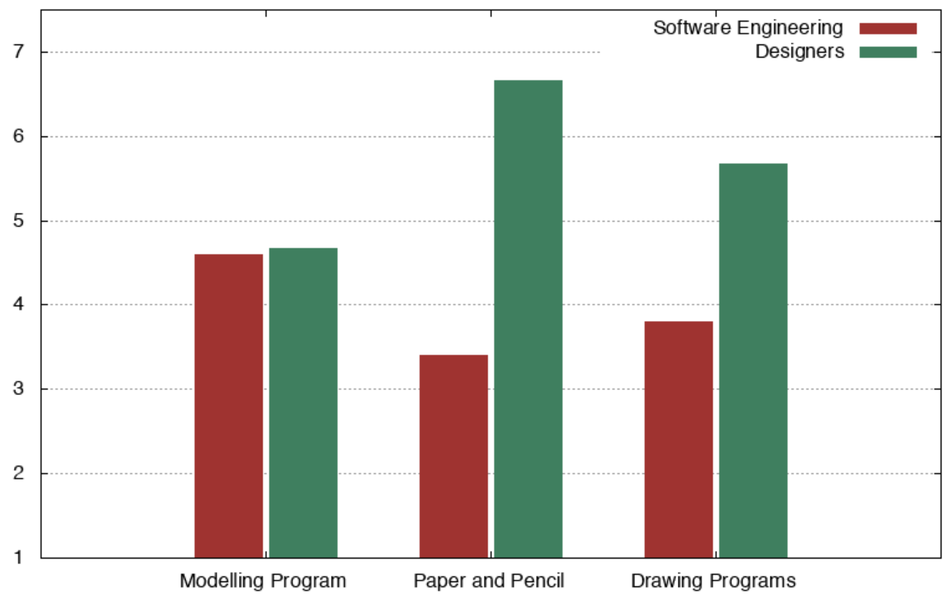


Figure 7.23: Tools used in early stages of modelling.

The results clearly show a discrepancy between the usage of paper and pencil and drawing programs between software engineers and designers where the former tended to use such tools more rarely and the latter more often.

Users were also asked about their opinion on the purpose of concept art in the early stages of modelling with regards on whether they used concept art to gain a better understanding of the scene being modelled and whether it allowed users to rapidly explore new ideas. Regarding the conversion from concept art to final models, users were asked if they believed that this was a straightforward process and whether it represented a bottleneck in the modelling workflow. Figure 7.24 shows the average results for these questions for users with and without procedural modelling experience.

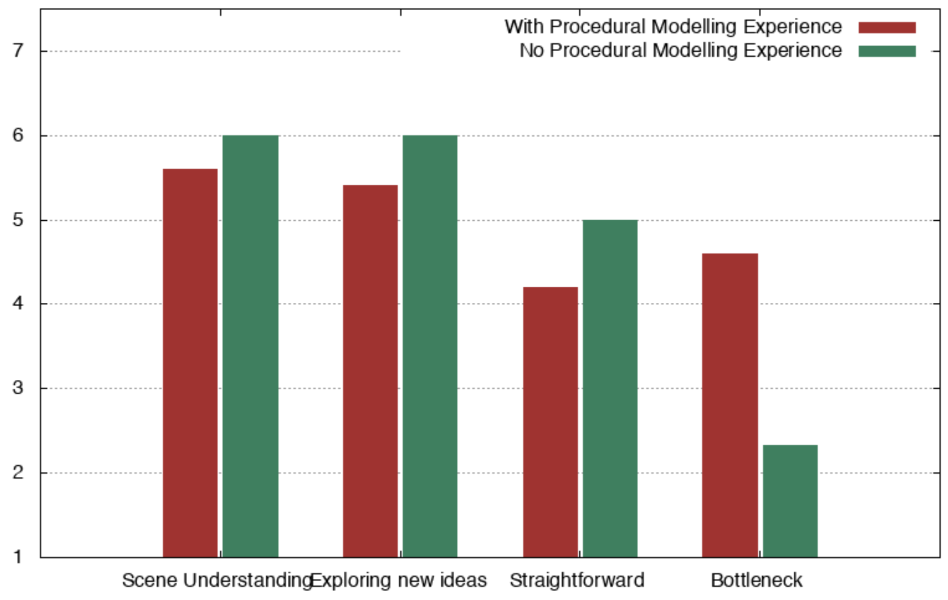


Figure 7.24: Concept art in the modelling process.

Both types of users generally agreed that concept art in the early stages of modelling is rather useful for understanding the scene to be modelled and that it allowed a quick exploration of new ideas. The opinions differed with respect to the process of converting concept art into final 3D models. Users with no procedural modelling experience generally felt that it was a more straightforward process with a low bottleneck in the modelling workflow than users with procedural modelling experience. This is rather surprising as the conversion between concept drawings and final 3D models has been described as a bottleneck, for which sketch based interfaces for modelling presented a solution [Ols+09].

7.3.3 Initial Opinions Regarding the Methodology and Prototype

After being presented with a short video describing the entire methodology and the prototype interaction, users were asked to fill a second questionnaire (found in Appendix C) that aimed at collecting the perceived interest and usefulness regarding the methodology as well as an early comparison with traditional and procedural modelling tools.

The second questionnaire asked the participants their opinion regarding the presented methodology interest, originality and usefulness towards procedural modelling of buildings. The results for these three questions are presented in Figure 7.25 for both users with and without procedural modelling experience.

Although the results were positive in this regard, there is clear distinction between both types of users where the ones with procedural modelling experience seem to favour the presented approach. More specifically, users with no prior experience in procedural modelling felt that the methodology was not as useful as users with experience in such techniques. A possible explanation for this discrepancy is the fact that users in the *PM* group have a more clear understanding of the drawbacks in procedural modelling and could more easily accept the approach presented in this thesis as an improvement. On the other hand, users in the *NPM* group are used to modelling capabilities in traditional modelling tools which offer

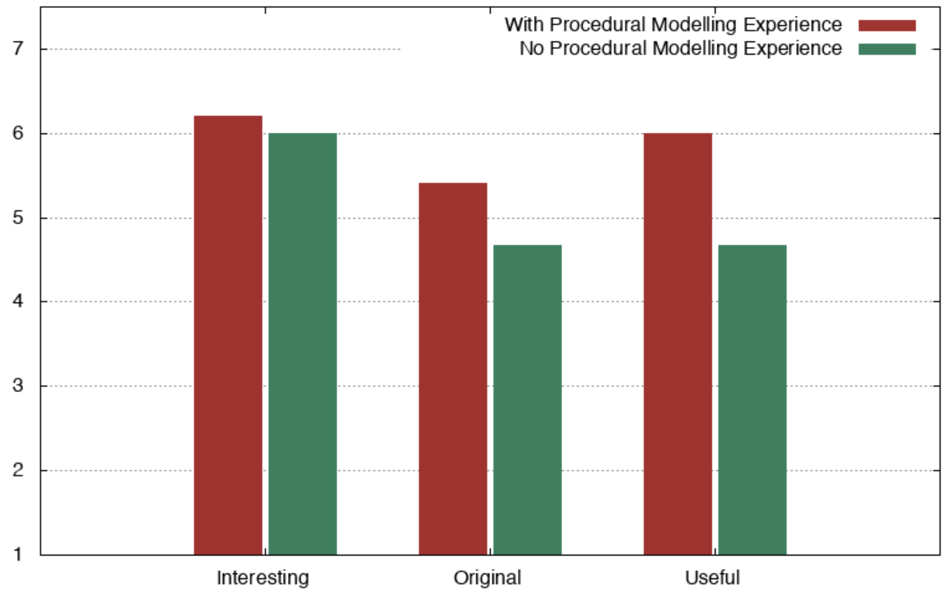


Figure 7.25: Early opinions about the presented methodology.

an enormous amount of expressiveness which procedural modelling tools cannot offer and regarded this as a drawback in the methodology.

To understand how the users felt about the prototype adequacy to building modelling, the users were asked their opinion about whether the presented methodology was familiar, adequate and sufficient to effectively create 3D models of buildings. The results for these questions are presented in Firgure 7.26.

Yet again, there is a clear discrepancy between the *PM* and *NPM* groups, especially in how the users felt regarding the familiarity of the methodology. Whereas users in the *PM* group reported a high score (an average value of 5.8) in this parameter, users in the *NPM*

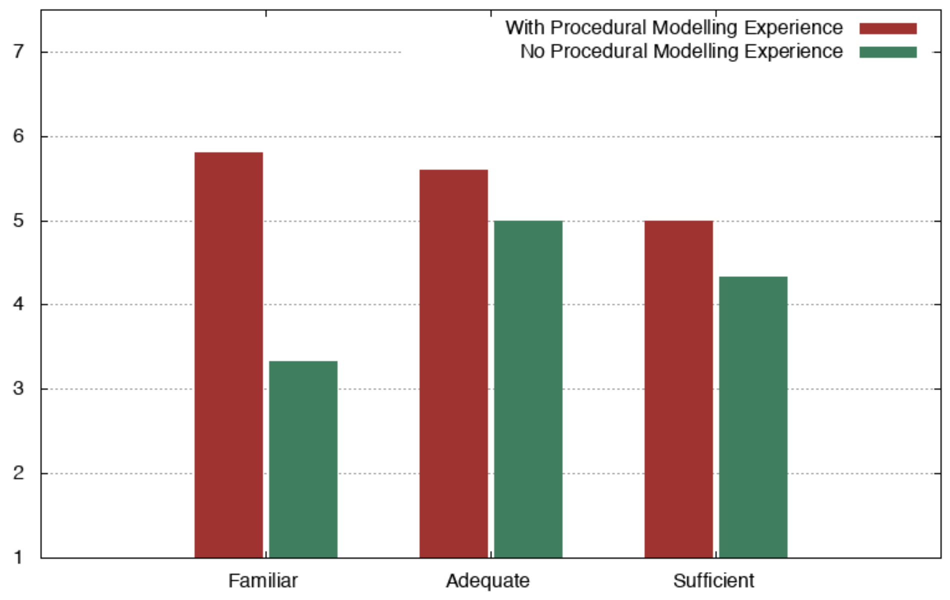


Figure 7.26: Adequacy of the methodology to building modelling

group reported an average negative value of 3.3. This can, trivially, be explained by the fact users in the *NPM* group do not possess enough knowledge about procedural modelling methodologies and prefer to use traditional modelling tools.

Before being allowed to experiment with the prototype, the users were asked their opinion about the presented sketching interaction regarding their perception about its ease of use, intuitiveness and completion. These results are summarised in Figure 7.27.

To this end, it is possible to observe that users in the *PM* group believed that the sketching interface was easier to used and more complete than users in the *NPM* group. The inverse applies to the intuitiveness of the sketching interface. In general, the users reported

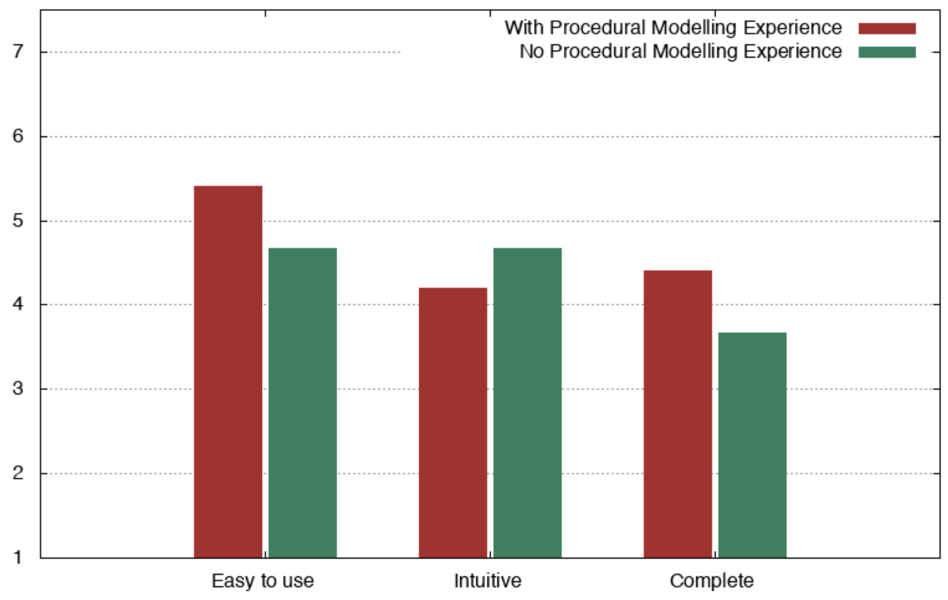


Figure 7.27: The prototype's sketching interaction.

a neutral or negative score for the intuitiveness and completion of the sketching interface. This indicates that more research should be put towards improving these aspects.

The participants in the *PM* group were asked to compare (based on the video) several aspects of the presented methodology with current procedural modelling tools. The first set of questions aimed at gathering the general opinion about the capabilities of both methodologies. Users were asked if they felt the presented methodology was easier to understand, more adequate, more intuitive than current procedural modelling methods. They were also asked if they believe the presented methodology improves the general modelling workflow and if such workflow is more pleasant. These results are shown in Figure 7.28.

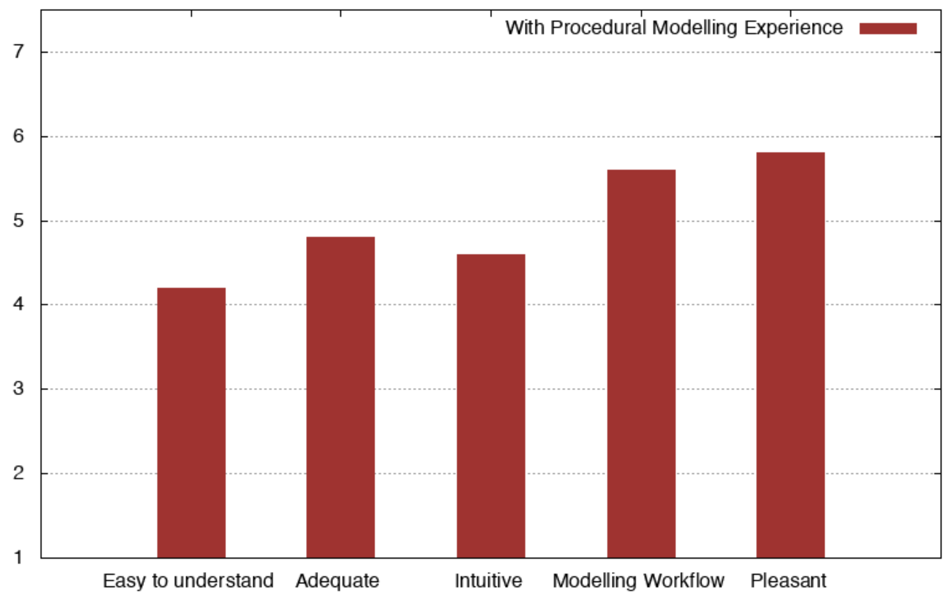


Figure 7.28: Comparison with current procedural modelling tools regarding their capabilities.

Results show slightly positive scores for the first three questions with average values of 4.2, 4.8 and 4.6 respectively. On the other hand, results for the last two questions had a better average score (5.6 and 5.8 respectively) showing that users felt that the presented methodology improves the modelling workflow and provides a more pleasant modelling experience than other procedural modelling tools.

The second set of questions aimed at gathering data on how the users felt regarding the capabilities of the prototype when it comes to the expressiveness, mental effort and creativity of the prototype compared with other procedural modelling tools. The results for the users in the *PM* group are shown in Figure 7.29.

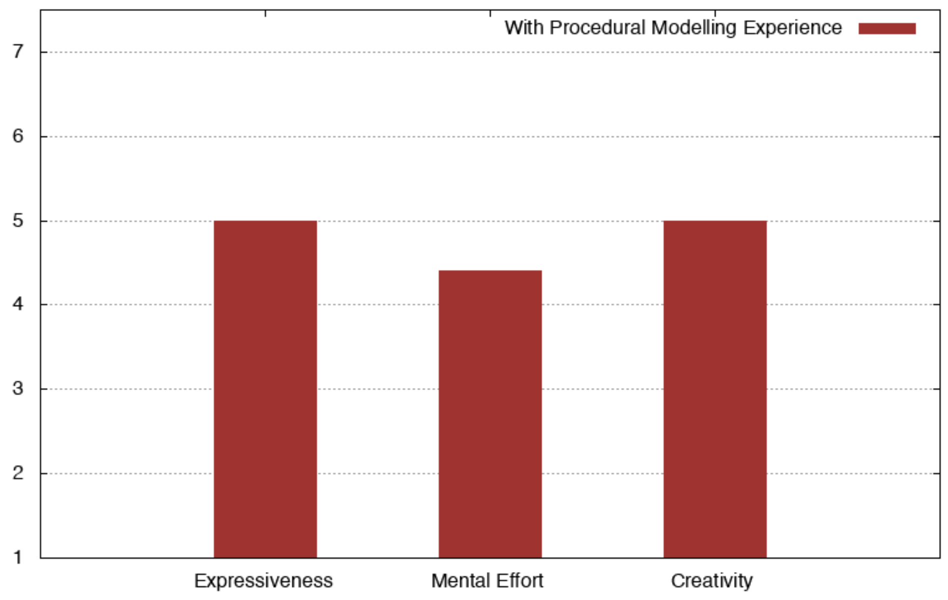


Figure 7.29: User opinion about model creation capabilities of the prototype when compared to other procedural modelling tools.

Results show that these users perceived a slight improvement on the expressiveness and creativity over other procedural modelling tools with average scores of 5.0 in both cases. With an average score of 4.4, the users also reported that the mental effort was slightly less when using the prototype when compared to other tools.

The participants with procedural modelling experience were also asked about their perception on what advantages the sketching interface would bring to the procedural modelling workflow in terms of whether it allowed users to be more creative, required less knowledge of procedural modelling methodologies and if it allowed a greater focus on the creative workflow instead of having to think about the set of rules that would generate a given building. The results for this set of answers are presented in Figure 7.30.

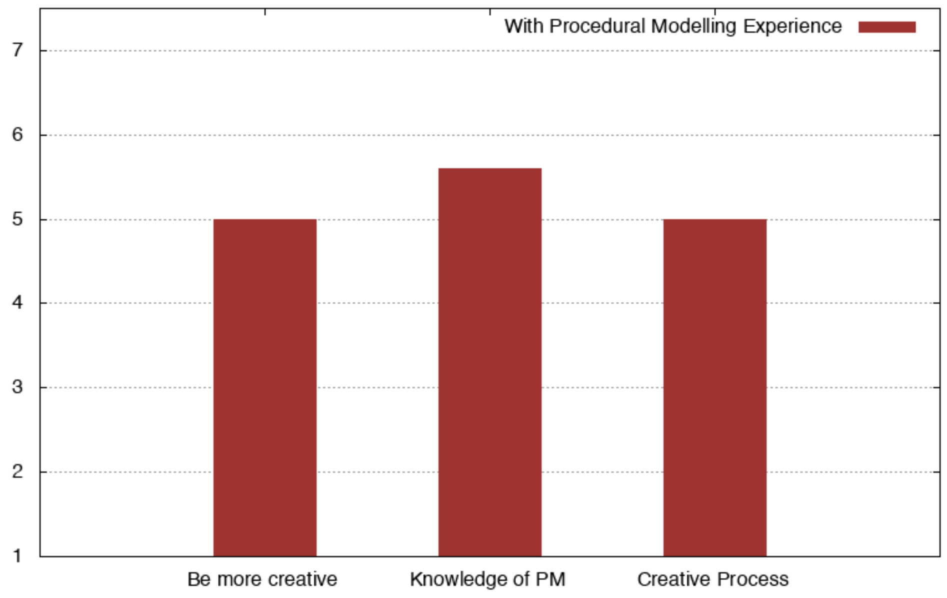


Figure 7.30: Opinions regarding the benefits the sketching interface brings for procedural modelling.

With average values of 5.0, 5.6 and 5.0 for each of these questions respectively, the users reported an increase in all aspects, especially when it comes to reducing the knowledge about procedural methodologies required from users. This represents a greatly desired advantage as it could help the broader acceptance of procedural methodologies by any kind of users.

Finally, all participating users were asked about their general opinion regarding the presented prototype based solely on the video. These questions aimed at understanding if the users perceived the prototype as easy to use, if it was stimulating or frustrating to use and if it was flexible enough for the task of procedural modelling of buildings. These results are shown in Figure 7.31.

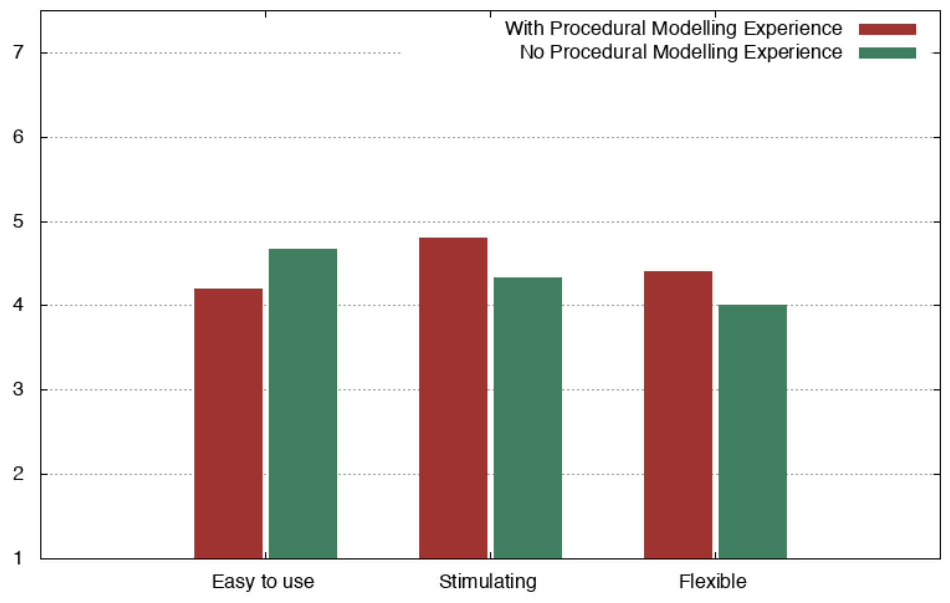


Figure 7.31: Early opinions about the prototype.

The perceived opinion was only slightly positive with average scores of 4.2, 4.8 and 4.4 for users in the *PM* group and 4.6, 4.3 and 4.0 for users in the *NPM* group. Again, users in the *PM* group seem to have a better acceptance towards the presented methodology.

7.3.4 First task: Sketching Interface for Procedural Modelling

After filling the second questionnaire the users were asked to perform several tasks with the prototype that implemented this methodology. In the first task users had a script (Appendix D) that guided them through the various steps that lead from the initial state as seen in Figure 7.32 to the desired outcome as can be seen in Figure 7.33.

The purpose of this task was to assess the sketching interface and the suggested procedural operations and inferred parameters.

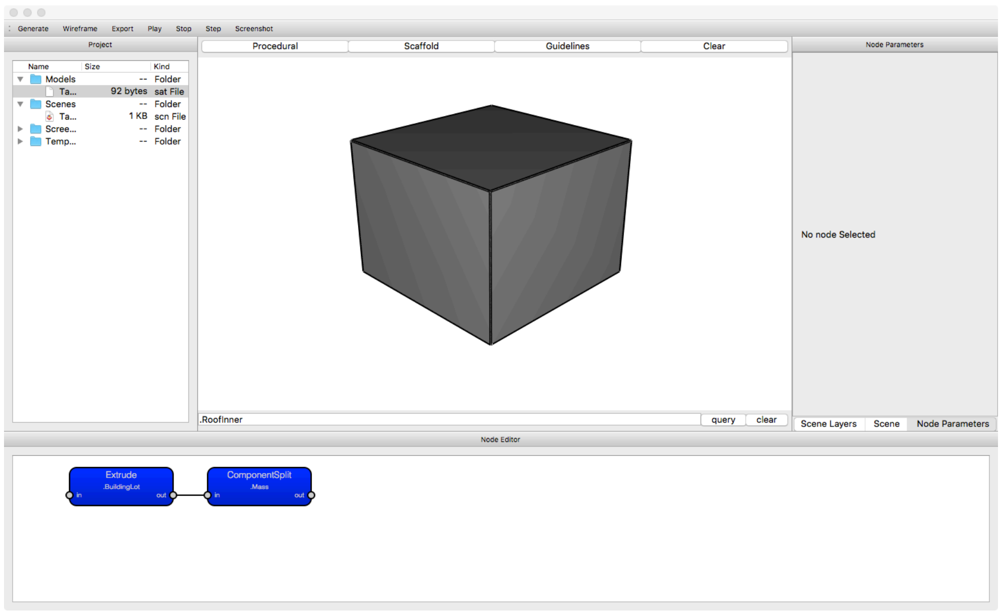


Figure 7.32: Initial state of the first task presented to the users.

All users were able to successfully complete this task with an average time of 12 minutes and 23 seconds. There were no major differences in the average time taken by users in the *PM* and *NPM* groups.

Within the user-dependant category, the metric where users performed worse was the number of times users sketched with an incorrect selection with an average value of 1.0. This would happen whenever the user started a sketch without any shape selected or with

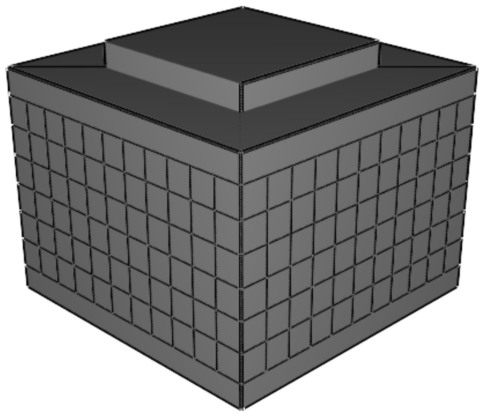


Figure 7.33: Desired outcome of the first task presented to the users.

more than one shape selected for drawing. As it stands, the methodology requires the user to specify a single shape to which sketches will be associated with. This value clearly represents a drawback of the current methodology. Ideally, the system should be able to infer to which shape a given sketch is related.

The second most frequent user dependant metric was related with not understanding or misunderstanding the presented semantic queries with an average value of 0.875. This metric records how many times the user expressly commented that he or she wasn't able to understand the queries or selected a query that wasn't fit for the present intention.

On the system-dependant metrics, the system didn't infer the procedural operation parameters with an average of 1.0 across all participants. This would mostly happen for the offset operation prior to creating the extrusion in the roof where the system would either create an operation with an offset value that clearly was too big or too small when compared to the user sketch.

7.3.5 Second task: Usage of Guidelines and Support Lines

In this second task, users were also provided with a script (Appendix E) that led them through to the successful completion of this task, starting with the state shown in Figure 7.34 and resulting in the desired model as shown in Figure 7.35. This task was targeted at evaluating the support lines and guidelines as an aid in the precise sketching of procedural lines that would trigger the procedural inference.

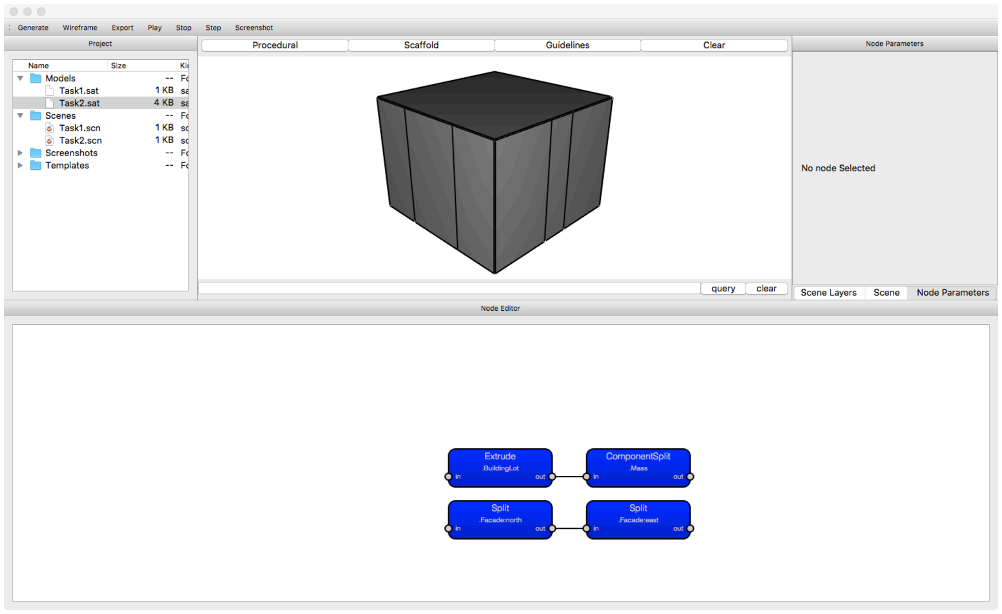


Figure 7.34: Initial state of the second task presented to the users.

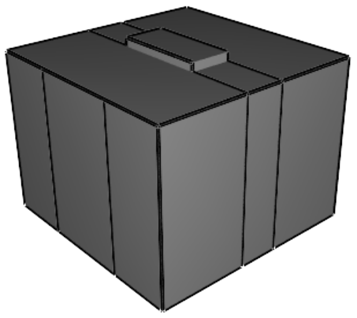


Figure 7.35: Desired outcome of the second task presented to the users.

Once again, all users were able to successfully complete the task with an average time of 12 minutes and 17 seconds without any major differences between both groups of participants.

In this scenario, the user-dependant metric with the worse performance was the fact that users had a tendency to double sketch not over a support line (with an average of 0.625 across all users) but over one of the guidelines. This points to the fact that requiring the user to create a support line from a guideline and posteriorly requiring a double sketch to promote it to a procedural line may be too cumbersome and repetitive. A solution would be to allow the double sketch to be performed the guideline directly which would promote a part of it (e.g., the line segment between two points) to a procedural line to trigger the procedural inference.

Users also tried to sketch with the wrong sketching mode with an average of 0.375. This would happen in situations when, for example, the user was supposed to sketch a guideline but the support line mode was active. The lack of visual cues indicating which was the active mode may have contributed to this. Ideally, however, the system should be able to infer these modes from the user sketch itself without requiring any mode switch. Moreover, this would be more akin to the paper and pencil metaphor.

Finally, the system failed to recognise double sketches (with an average value of 4.375) and guidelines (with an average value of 3.0). These situations would seem to happen at random points which seems to point to implementation issues. Apart from being, obviously, undesirable these situations may have increased the user frustration.

7.3.6 Third task: Modelling an Example Building

The final task was conceived as a stress test in order to have the users freely use the prototype and all available tools to reach a final, desired model illustrated in Figure 7.36 starting from a simple initial building volume. For this task, users did not have access to a script but merely a text explaining what they were supposed to do (Appendix F).

In this task, the users were also able to accomplish the intended results within an acceptable range of variation. The average time to complete across all users was 10 minutes and 58 seconds and, once again, no major differences between both types of users.

Within this task the user-dependant metric that showed the worst performance and by a large margin was, as in the first task, the sketching with the wrong selection with an average of 2.75. This increase is justified by the lack of a script that would remind the users to

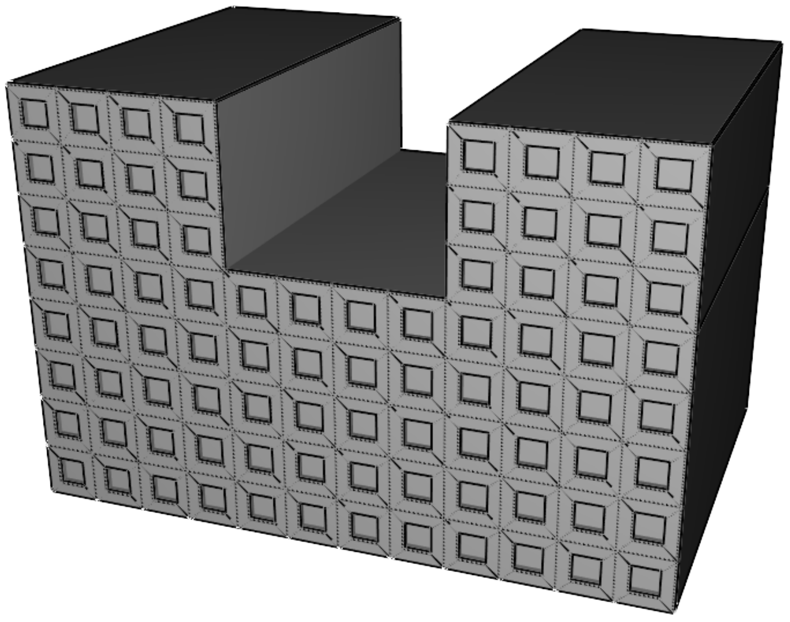


Figure 7.36: Desired outcome of the third task presented to the users.

perform the correct selection.

The second worst performant user-dependant metric was not understanding the presented queries or selecting a wrong query for the current intention with an average of 0.875 across all users. Again, this points to the fact that the semantic queries, albeit enhancing the procedural modelling expressiveness, seem to be too cumbersome and impose a high cognitive effort in users.

Also related to the semantic queries, the system failed to provide a valid query for the current user intention with an average value of 1.125. This would cause the user to repeat the selection multiple times until a valid query was shown.

The second most frequent cause of error in the system-dependant metrics was failing to provide a valid operation for the given sketch with an average value of 0.875. Again this meant the user would have to repeat the sketch in hopes of obtaining a valid operation.

7.3.7 User Opinion After Task Completion

After completing all three tasks, the participants were asked to fill a third and final questionnaire (Appendix G) that aimed at gathering the participants final opinions on the methodology. This questionnaire also had the purpose of understanding the variation in opinions after the users had contact with the prototype, minimising potential implementation issues.

The first set of questions in the third questionnaire aimed at providing data regarding the methodology's adequacy to building modelling and is in all similar to the one presented in Figure 7.26. The users were once again asked about the familiarity, adequacy and sufficiency of the methodology for the building modelling tasks. These results are shown in Figure 7.37.

As it is possible to observe, there has been a significant increase in the reported opinions in the familiarity and adequacy within the *NPM* group as the average score for these questions increased from 3.3 and 5.0 to 4.6 and 5.6 respectively. On the other hand, users in the *PM* group reported a reduction in these same parameters with a decrease in these average scores of 5.8 and 5.6 to 4.8 and 4.8 respectively. Both groups reported a reduction in the

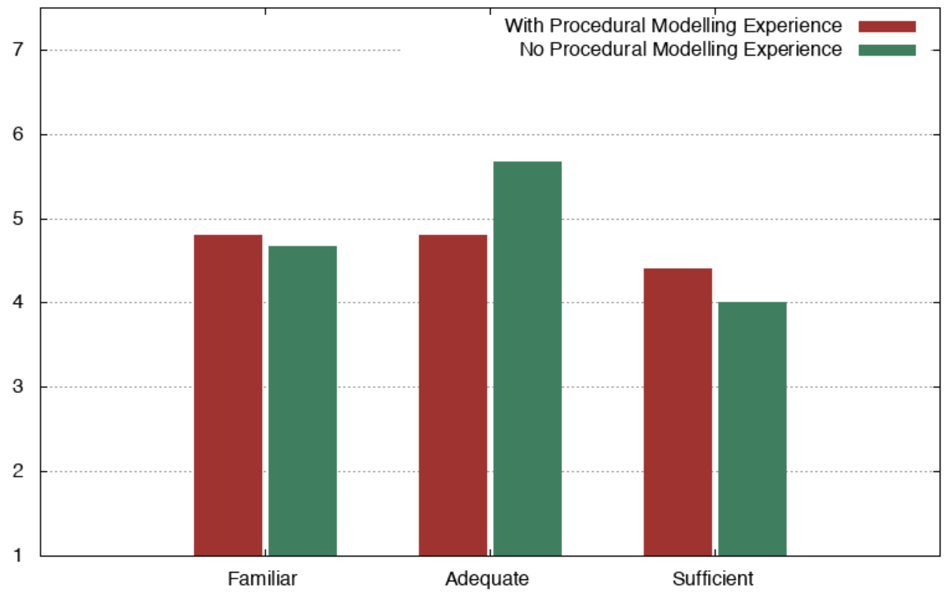


Figure 7.37: Participants' opinions about the methodology after task completion.

sufficient parameter having its average score reduced from 5.0 and 4.3 to 4.4 and 4.0 for the users in the *PM* and *NPM* groups respectively.

The second set of questions was aimed at comparing the users' opinions regarding the sketching interaction after having contact with the prototype. As in the previous set, these questions are identical to the ones presented in Figure 7.27 and asked the users about how easy, intuitive and complete the sketching interface was in their opinion. The results can be seen in Figure 7.38.

Users in the *NPM* group reported an increase in all three parameters with the average score being increased from 4.6, 4.6 and 3.6 to 5.0, 5.3 and 4.0 in each question respectively.

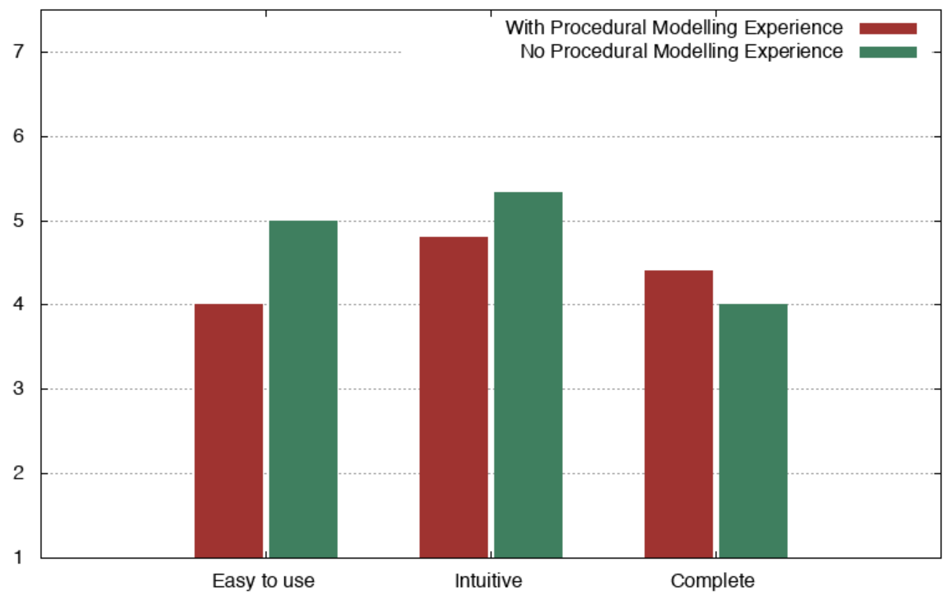


Figure 7.38: Participants' opinions about the sketching interaction after task completion.

Users in the *PM* group reported a significant reduction in the *Easy to Use* parameter with the average score decreasing from 5.4 to 4.0. On the other hand, these users reported only a slight increase in the *Intuitive* parameter with the average score seeing an increase from 4.2 to 4.8 while the *Complete* parameter saw no variation (average score of 4.4).

The results from the two previous sets of questions show that the contact with the prototype has surpassed the expectations of users without prior contact with procedural modelling tools but, on the other hand, users with such experience might still prefer the current procedural modelling methodologies.

As in the previous questionnaire, users in the *PM* group were also asked to compare the presented methodology with current procedural modelling tools within the same set of parameters. Similarly, the users were asked about the general opinion about the capabilities of both methodologies. More specifically users were asked about the opinions regarding if the presented methodology was easier to understand, more adequate and intuitive, whether it improved the modelling workflow and if the modelling process was more pleasant. As in previous cases, these questions were only answered by users with procedural modelling experience. The results of these questions are presented in Figure 7.39.

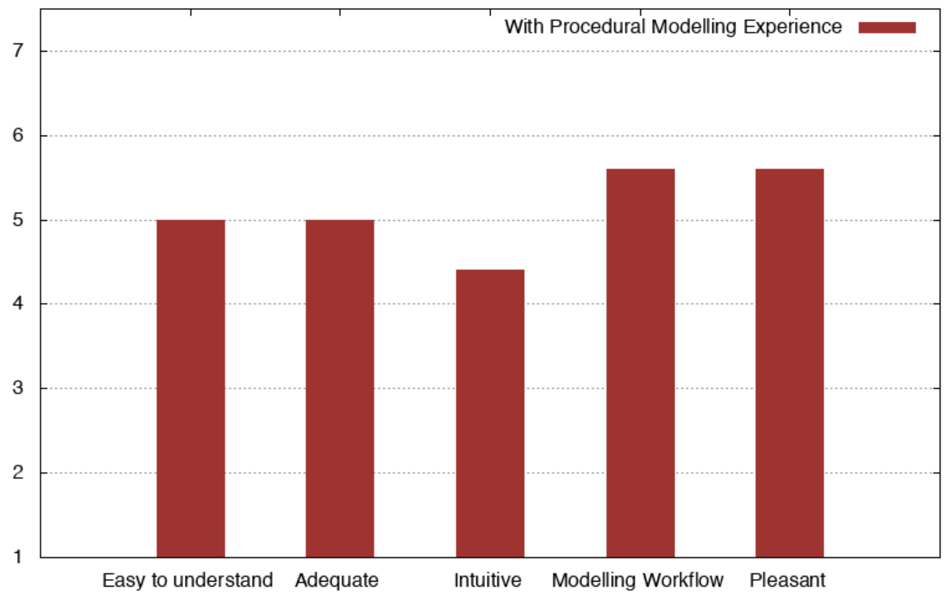


Figure 7.39: Participants' general opinions about the methodology when compared with other procedural modelling methods.

As seen in the results, the participants reported an increase in the first two parameters (*Easy to Understand* and *Adequate*) with an increase in average score from 4.2 and 4.8 respectively to 5.0 in both cases. The *Intuitive* and *Pleasant* parameters saw a decrease from 4.6 and 5.8 to 4.4 and 5.6 respectively. Finally, the *Modelling Workflow* parameter was left unchanged with an average score of 5.6.

The participants' opinion on the model creation capabilities of the methodology compared with other procedural modelling methods was also evaluated again in respect to its expressiveness, mental effort and creativity. These results are shown in Figure 7.40.

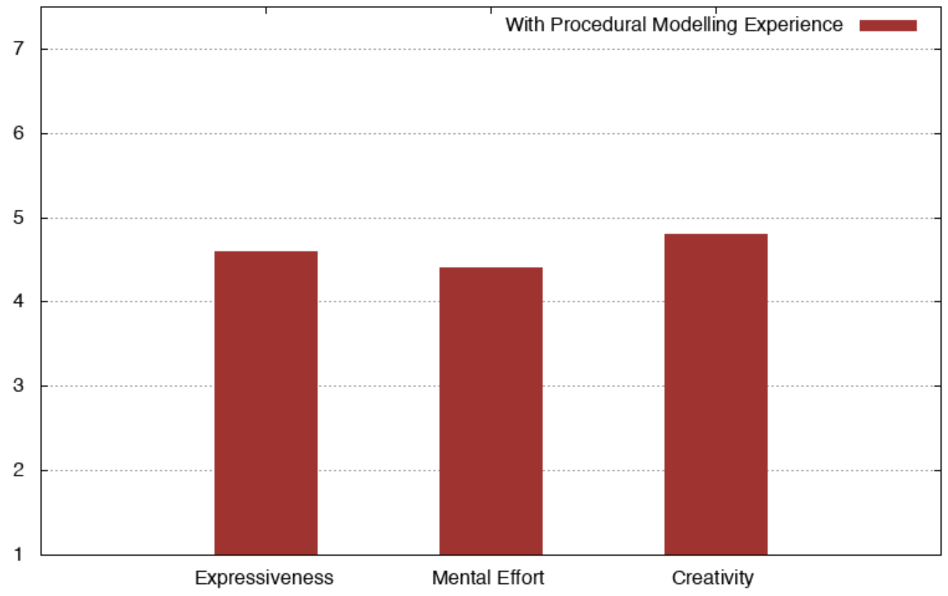


Figure 7.40: Participants' opinions about the model creation capabilities after task completion.

The results demonstrate a small decrease in *Expressiveness* and *Creativity* parameters with the respective average scores decreasing from 5.0 in both cases to 4.6 and 4.8. The *Mental Effort* parameter saw no variation having an average score of 4.4.

Regarding the sketching interface and the advantage they bring to procedural modelling methodologies when compared with other techniques, participants were, again, asked whether they felt it allowed to be more creative, required less knowledge of the underlying methodology and if it allowed users to have a greater focus on the creative workflow. The results for this set of answers are presented in Figure 7.41.

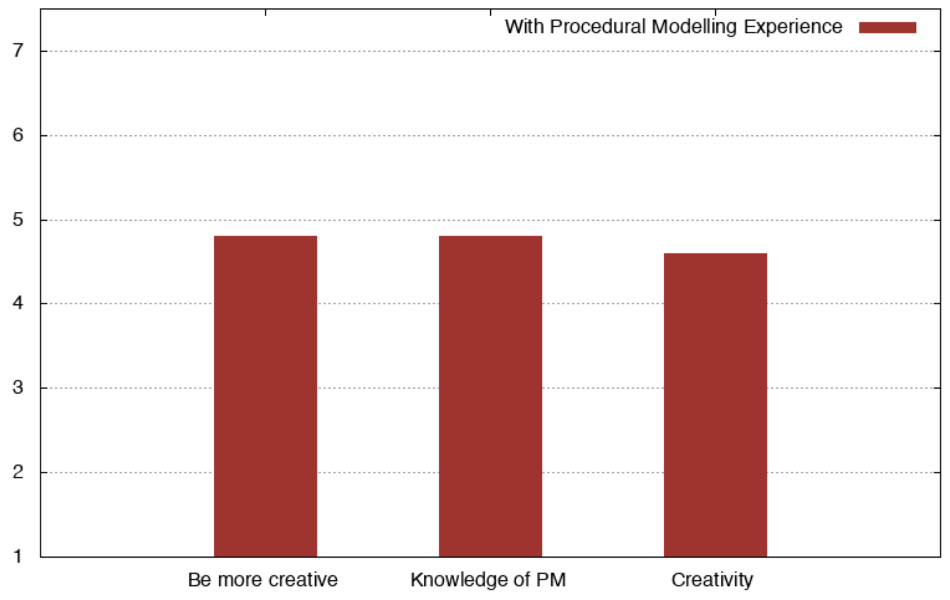


Figure 7.41: Participants' opinions about the sketching interface after task completion.

In this instance, the participants reported a reduction in all parameters as the results show a decrease in average scores from 5.0, 5.6 and 5.0 to 4.8, 4.8 and 4.6 for each parameter respectively.

The questionnaire asked the participants to express their opinions regarding the adequacy, usefulness and correctness of both the procedural operations the system suggested after the completion of each user stroke. The same parameters were evaluated for the inferred parameters after the user selected one of the suggested procedural operations. These results are shown in Figure 7.42.

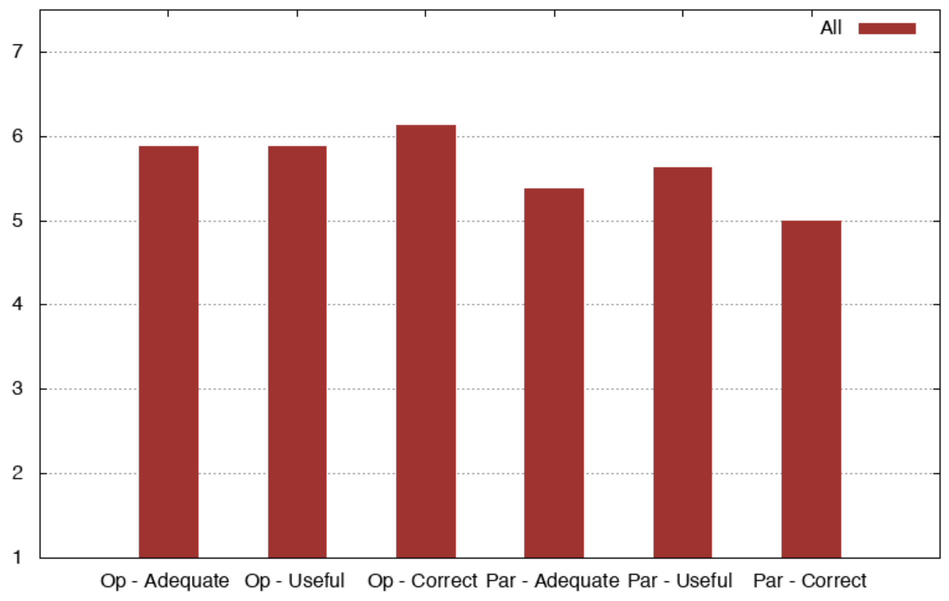


Figure 7.42: Participants' opinions about the correctness of the suggested procedural operations and parameters.

These results show overall positive responses in each parameter without any average score value being below 5.0. The average score values for the suggested procedural operations are 5.9, 5.9 and 6.1 for the *Op - Adequate*, *Op - Useful* and *Op - Correct* parameters. Likewise, for the *Par - Adequate*, *Par - Useful* and *Par - Correct* parameters the participants reported an average score of 5.4, 5.6 and 5.0 respectively.

The participants' opinion regarding the helpfulness, adequacy and ease of use of both support lines and guidelines has been asked in the third questionnaire. The results are shown for both the *PM* and *NPM* group in Figure 7.43.

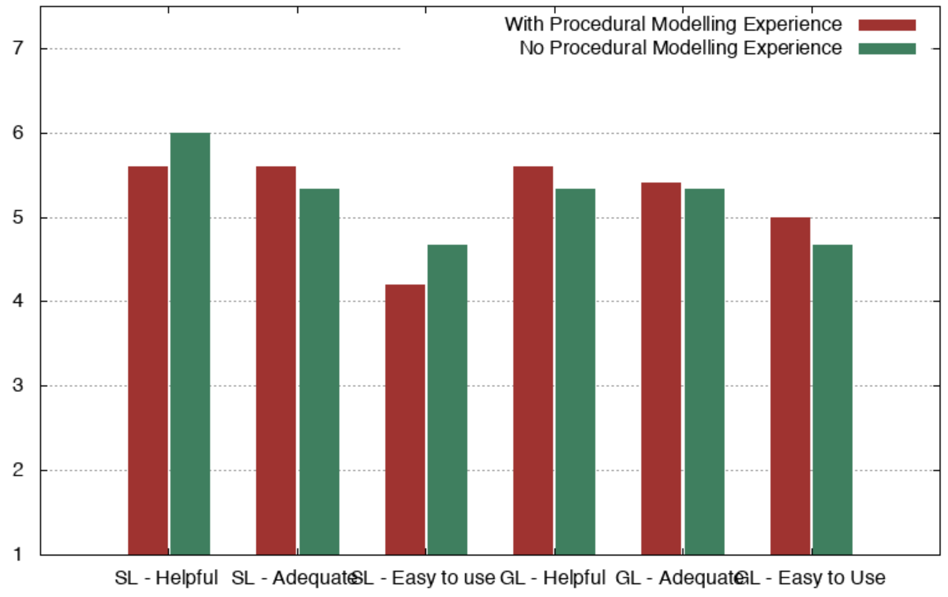


Figure 7.43: Participants' opinions about the usage of support lines and guidelines.

As it is possible to observe, the results indicate a generally positive opinion regarding these two interaction aids with the exception of the *SL - Easy to Use* parameter which had an average score of 4.2 among the *PM* group and 4.6 among the *NPM* group. It may be possible to explain the lower average value for this parameter with the fact that many users didn't like the double sketching interaction to promote a support line to a procedural line.

Similar to the second questionnaire (Figure 7.31), users were again asked their general opinion regarding the prototype's ease of use, whether it was stimulating or frustrating to use and if it had a desirable level of flexibility. The results for these parameters are illustrated in Figure 7.44.

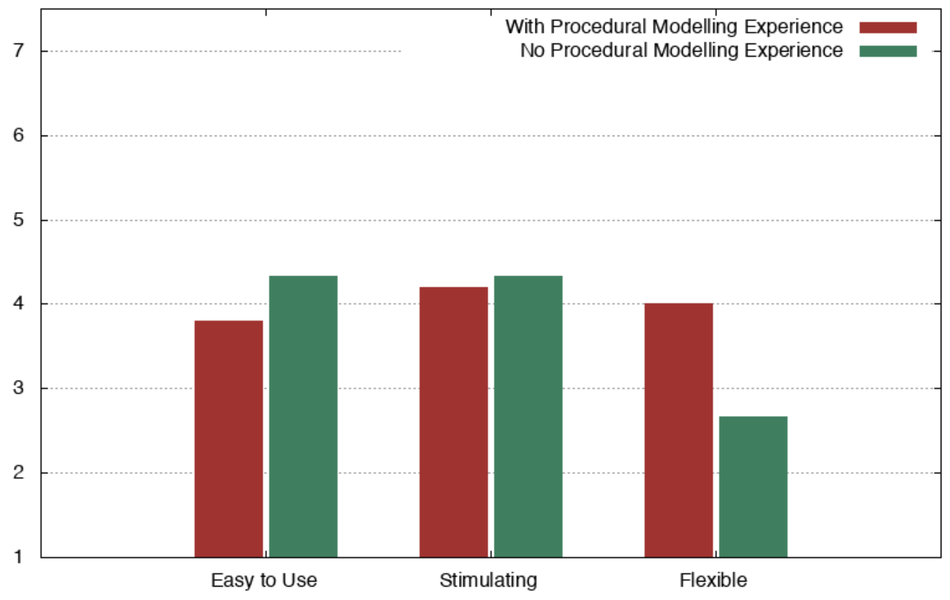


Figure 7.44: Participants' opinions about the prototype after task completion.

The results show a decrease in all of these parameters with the exception of the *Stimulating* for the *NPM* group which remained at an average score of 4.3. The *Easy to use*, *Stimulating* and *Flexible* parameters for the *PM* group showed a decrease from an average score of 4.2, 4.8 and 4.4 to 3.8, 4.2 and 4.0 respectively. Likewise, for the *NPM* group, the *Easy to use* and the *Flexible* parameters saw a reduction from 4.6 and 4.0 to 4.3 and 2.6 respectively.

Finally, the last set of questions in the third questionnaire aimed at gathering data regarding the prototype features with respect to its completion, suitability for the procedural modelling of buildings, whether the proposed interaction was intuitive and if it was easy to understand overall. These results are summarised in Figure 7.45.

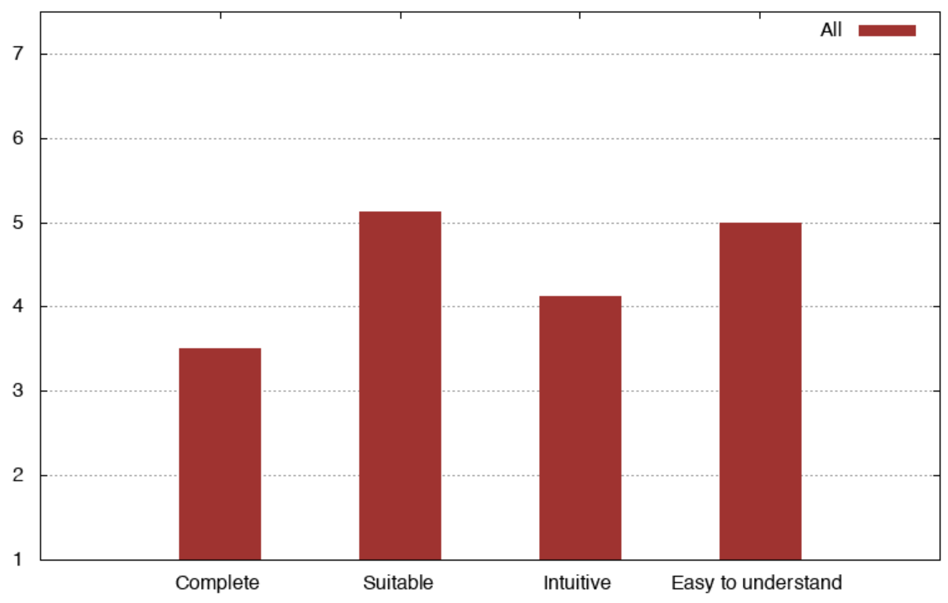


Figure 7.45: Participants' opinions about the prototype features after task completion.

As it is possible to observe, the participants found the prototype to be suitable for the task at hand (with an average score of 5.1) and easy to understand (with an average score of 5.0). On the other hand, the participants reported that the prototype did not feel complete (with an average value of 3.5) and evaluated its intuitiveness with an overall neutral average score of 4.1.

7.3.8 Discussion

After an analysis of all three questionnaires and the tasks participants were asked to perform, it is now possible to provide a discussion on the results obtained and draw some conclusions.

For all three tasks the average time to complete was similar throughout both the group of users with procedural modelling experience (*PM* group) and those without such experience (*NPM* group). This is an obvious sign that the presented methodology has extended the acceptance of procedural modelling methods to users without prior knowledge of such systems. Although it is possible to argue that the tasks had a script to guide the users throughout the process, this is not true for the last task where the users were free to use the prototype's functionalities as they saw fit. Thus, it is possible to conclude that a sketch based procedural modelling tool can indeed reduce the (somewhat steep) learning curve of procedural modelling by shielding away some underlying constructs and techniques.

Users in the *PM* group also reported that the methodology improves the procedural modelling workflow, making it more pleasant than with other procedural modelling tools. The third questionnaire results show that these users believe that the system is also easier to understand than other procedural systems.

Nonetheless, there are a few drawbacks in the methodology that require more research and improvement. The over-reliance on semantic queries seems to be posing a burden to users who had difficulties in understanding their meaning and usefulness. The textual representation of queries seems to be overly complex for users to successfully make use of it.

Moreover, the selection inference system did not always provide a good enough set of candidate queries forcing users to retry several selections until the results were satisfactory. This is consequence of using a non-deterministic algorithm (i.e., genetic algorithms) to create candidate queries.

The sketching interface also had some drawbacks that may have resulted in increased user frustration. Requiring the user to perform multiple selections to obtain a query and forcing the user to select a single shape from the query-selected shapes caused some confusion. The methodology should be able to infer what shape a given sketch is related to.

In some occasions the procedural operation and parameter inference system failed to provide adequate suggestions. This is clearly a point to improve in the system, although users reported a high level of adequacy, usefulness and correctness in the related questions in the questionnaires.

In architectural settings, guidelines and support lines are useful and adequate as reported by the participants in the user study. However, there are still improvements to be made in order to make these tools easier to use, especially the support lines. The main reason for this is the fact that users tended to double sketch directly over a guideline whereas the methodology only accepts such interaction on support lines.

Despite all the mentioned drawbacks, the presented methodology seems to be a good starting point for further research that will be left for future work.

7.4 Summary

This chapter was dedicated to providing an evaluation and discussion of the results of all contribution in presented in this thesis.

Following the same outline as the rest of this document, this chapter first presented the evaluation of layered shape grammars. As seen, this contribution improves the expressiveness of shape grammars by allowing layers to be defined within planar surfaces which, after being merged, can create more complex geometry. Furthermore, layers provide a non-linear workflow by rearranging or conditionally executing only a subset of the layers. This allows more variations to be produced with less effort. Vectorially defined shapes also allow the user to create complex geometry that could only be modelled externally and imported into the modelling workflow as a static asset. Combining both of these contributions allows to procedurally model buildings with complex layouts such as the Mikimoto Ginza building.

The chapter then presented the evaluation of the selection-action and selection inference contributions. The usability of such system was evaluated with a user study where the participants were presented with a video describing the methodology and were asked to perform a modelling task by following a script. Finally, they had to answer a short questionnaire with their opinions about the methodology. The results showed that this contribution enhanced the procedural modelling workflow and expressiveness but, on the other hand, queries were not easy to understand and the methodology required a higher level of mental effort.

The devised user selection inference system was also evaluated. In this case, synthetic queries simulating possible user intentions were issued over exemplar models and the generated candidates throughout several iterations were evaluated based on the score and difference towards the original query. Results have shown that the inference mechanism is able to infer complex queries based on a few user-selected shapes. In an interactive application where several candidate queries are presented to the user, the ideal situation is to have a high average score and a large variation between the minimum and maximum difference in queries. This means that the system presents a large range of different queries where most queries presented will be suitable for the given user intention. Nonetheless, the system currently takes too long to be useful in an interactive application.

Finally, this chapter presented the user evaluation of the sketch based interface for procedural modelling which is the main contribution in this thesis. This evaluation was performed as a user study where participants were required to perform three tasks and fill several questionnaires. Users were divided into two groups depending on their experience with procedural modelling. After analysing the results it was observed that participants in both groups were able to finish all tasks within a similar average time despite their prior experience with procedural modelling methods. Therefore, it is possible to say that sketching interfaces can increase the acceptance of procedural tools by users with no knowledge of these techniques since it seems to have reduced the learning curve. Results have also shown that the sketching interface improves the procedural modelling workflow and makes it more pleasant.

Nonetheless, the sketching interface still requires more research to be conducted as there were a number of drawbacks. Usage of queries seems to be a culprit in the user frustration with the system as several users reported not understanding or choosing the wrong query for the given intention. The procedural operation and parameter inference system sometimes failed to provide good candidates for a given sketch requiring the user to repeat the sketch entirely. Finally, some improvements in the sketching workflow should be explored to make the system more intuitive and easier to use. This is the case of requiring the user to perform multiple types of selection before starting to sketch and requiring the existence of support lines before performing a double sketch to trigger the procedural operation inference system.

8 Conclusions and Future Work

This chapter presents, in a first instance, future avenues of research as well as ideas considered as worth of being given more exploration in the area of sketch based procedural modelling of buildings. These have as basis the contributions presented in this thesis. Such is the case of, among others, using layers to enable an interactive procedural modelling of buildings, the integration of layers within the selection action paradigm presented in Chapter 5 and using user drawn lines to define vectorial shapes within layers to enhance the expressiveness of procedural modelling methods.

The chapter ends with the conclusions of the document, noting the most important contributions and results obtained during this work alongside some final remarks and notes.

8.1 Research Questions

This section aims at answering the research questions posed in the beginning of this document in Section 1.2 based on the results demonstrated in this chapter. It is a fundamental step as it allows drawing conclusions about the proposed contributions in any research work.

Each subsection aims at answering each research question separately.

8.1.1 How can procedural modelling methods be improved to provide enough expressiveness to be better paired with sketch interfaces?

As seen in Section 1.5.2, one of the problems in the currently most accepted procedural modelling methods is that they often only provide geometric operations that are based on split lines and planes. This causes the resulting geometry to be mainly rectangular or cuboid and shape configurations that resemble a grid pattern. On the other hand, sketch based interfaces provide an interface that allows users to create geometry that is almost free-form. This, evidently, is a mismatch in terms of expressiveness between procedural modelling and sketch based interfaces for modelling.

To elevate the expressiveness of procedural modelling methods to be more comparable to that of sketching interfaces, this thesis proposed extending shape grammars with layers and shapes that are vectorially defined. One of the advantages being that it allows the creation of more complex shapes that weren't created via a split operation directly within the procedural modelling tool, whereas current methods would require such shapes to be externally modelled and imported.

The other advantage present in layered shape grammars is that layers follow a representation that is more akin to how humans perceive façades [Zha+13]. In the context of sketch based procedural modelling of buildings this can be considered an advantage as users will typically start with the broad building structure and patterns and, posteriorly, add more detail on top of the geometry that is already present.

This is reflected on the fact that users can define patterns for, e.g., a façade such as the layout for its windows and then create new structures that break such pattern such as a balcony or ornament that spans multiple floors. The rules that define these two sets of geometry can be inserted into different independent layers which are merged to create the final geometry. In turn, the final merged shapes can have complex geometries as they are produced with boolean operations between multiple shapes. Layers also provide a non-linear workflow allowing the users to explore more variations by toggling the generation of the geometry for a given layer on or off, or setting its parameters independently.

Although the vectorially defined operations haven't been implemented in the proposed sketching interface, it is trivial to consider it as a possibility as there are methods that can convert an arbitrary user sketch to a vectorial form. This can then be inserted into the procedural modelling rule to allow the user to create more complex shapes. Layers can also be implemented in the sketch based procedural modelling system by reusing the layer paradigm usually found in many 2D editing software. The user would start by creating a layer within a planar shape similarly to how layers are created in canvases in image editing programs. Procedural rules would be created on the layer that is currently selected. These two contributions have been left for future work, but it is assumed that devising a methodology for their integration is rather straightforward.

8.1.2 What is required from a procedural modelling method to improve the amount of direct control a user can have over the generated models?

One of the issues with current procedural modelling methods is the lack of direct control as identified in Subsection 1.5.1. Often, the only way users have to define the final look of a building is to modify the rule base, adding, removing or parametrising rules which is achieved in an area in the user interface that is distinct from where the model is displayed. Hence, lacking any sort of direct control as there is a visual disconnection between what the user can manipulate and the resulting geometry.

An issue that contributes to this problem is the user selection ambiguity as described in 1.5.3. One of the main components that allow a greater direct control is the ability to select exactly the geometries that are to be modified. In procedural modelling this is not a trivial problem to solve since, on the one hand, the same set of user selected shapes can have multiple meanings and, on the other hand, it is required that the operations performed on such shapes are persisted even if some procedural rules and parameters or its parameters change.

The work presented in this thesis tries to condense most of the interaction necessary to model a building into the user interface element where the generated building is indeed visualized and, therefore, users must be able to select what shapes are to be modified. To this end, semantic queries were presented in Chapter 5 which allow for a user selection inference system to be created. The user must simply select a subset of shapes that express the intention and the system will infer the query that provides the best match to the intention. The only interaction that is disconnected from the building model is in choosing the query that best fits the user intention.

A solution to increase the amount of direct control a user can have over generated models is, thus, to provide a query based approach to procedural modelling. This can be combined with a more interactive way to specify procedural rules which, in this case, are sketch based interfaces for modelling.

Such interfaces also have the added benefit of increasing the direct control in procedural modelling methods in two distinct ways. First, they allow specifying the operation to be performed directly in the building model by sketching the intended final result and, secondly, they also implicitly provide the parameters for such operation. For example, the extrusion operation can be specified by drawing orthogonal lines from a shape's vertex and the extrusion amount is implicitly defined as the average length of such lines. Once again, the only interaction that is not performed directly on the building model is in selecting one of the procedural operations suggested by the system.

8.1.3 How can semantics present in procedural modelling rules be used in a sketch based procedural modelling system?

This research question is deeply tied to the previous as the answer to both relies on the semantic queries presented in Chapter 5.

One of the by-products of procedural modelling, especially in shape grammars, is the concept of a shape tree where shapes that are output by a procedural operation are inserted as children of the shape that was given as input to such operation. Moreover, shapes in shape trees can carry semantic information explicitly under the form of symbols and tags and implicitly as hierarchical relations. Such semantic information has barely been explored in the context of procedural modelling.

The contributions presented in Chapter 5 are fundamental to allow the sketch based procedural modelling system as detailed in Chapter 6. In a sketch based system, as in many other interactive modelling approaches, specifying the geometry that is to be modified is crucial. However, while in traditional modelling tools this is easily achieved by means of selecting geometries (e.g., via point and click) the same is not true for procedural modelling.

The data amplification in procedural modelling methodologies allows the generation of vast amounts of geometries from a small set of rules. This, however, also implies that a set of selected shapes may not be valid if any of the parameters in the procedural rules change and, therefore, defining the geometries to be modified becomes highly ambiguous.

As seen, the semantic queries presented in Chapter 5 can be used to overcome this problem which is a necessary step to provide the direct control required in a sketch based procedural modelling system.

Other usages can be envisioned such as using the implicitly and explicitly defined semantics in a procedural model to predict what sort of elements (e.g., windows, ornaments) a user may be intending to draw given the currently selected shapes. This is, however, left for future work.

8.1.4 How can a sketch based interface for modelling be used in the procedural modelling of buildings?

The answer to this research question lies heavily in Chapter 6 where this thesis presented the contributions towards a sketch based procedural modelling system for buildings.

As seen, and as in every other sketch based modelling approaches it is extremely important to allow the user to directly select the geometries or shapes which are to be manipulated. This is usually the first step in an interactive system which, however, does not have a trivial solution in the context of procedural modelling. To this end, the chosen solution was to rely on the contributions of Chapter 5 to allow the user to specify the shapes to be modified. Nonetheless, any other approach that is able to unambiguously be used to the same end could be employed instead.

In a procedural modelling system, once the user is capable of defining unambiguously the set of shapes to be modified, it is then possible for the user to specify the operations to perform, which in a sketch based interface is achieved by sketching. As seen in Chapter 3 some sketch based modelling systems provide iconic sketches to be performed which are posteriorly translated to actions.

While it would be possible to define a similar paradigm for procedural modelling systems since there is only a well-defined set of atomic operations that can be performed at any given moment, it would require the user to memorise the iconic sketch representation for every operation. It was deemed as a better approach to provide a means for the user to sketch the intended final geometry for a given procedural operation on the geometry. It is the system's responsibility to take the user's sketches and transform them into procedural operations and parameters that closely match the user's intention and insert such rules into the rule base.

Moreover, using formal drawing techniques such as analytical drawing has as great impact in the way sketch based interfaces can be used in the procedural modelling of buildings. Being tailored for man-made objects these techniques allow a precise definition of rules and parameters that exactly match the intention behind the user sketch, with the benefit of avoiding the need of resorting to other parts of the user interface. The user is, thus, able to define all the required rules within the sketching interface.

8.1.5 How can procedural modelling rules and parameters be defined through sketch based interfaces for modelling?

As seen in Chapter 6, it is possible to use sketch based interfaces to define rules and parameters for the procedural modelling of buildings. The approach presented in this thesis relies on a multi-stage pipeline that processes the user's sketch from the initial screen space stroke to the generation of a parametrised procedural modelling rule.

The first stage in such pipeline is to segment a screen space stroke into multiple line segments by using speed and curvature metrics. Each line segment is individually processed to find its final position and characteristics within the 3D scene. For a given selected shape, the system generates a family of lines for each type of available procedural operation and compares each line projected to screen coordinates with the user line segment. The most similar line becomes the final 3D space line and is inserted into the scene.

Once all user line segments are converted to 3D lines, the procedural operation inference system is triggered. The system resorts to a set of heuristics that filter off some impossible procedural operations for the current configuration of lines and produces a set of candidate operations with estimated parameter values. Each candidate is then passed onto the gradient descent method which will adjust the parameters to closely match the user provided lines. Finally, the candidates are presented to the user who will select the best match for the current needs.

8.1.6 Can a sketch based interface for modelling improve the workflow of methods for procedural modelling of buildings?

As the results have shown, the participants in the sketch based procedural modelling user evaluation (Section 7.3) that had experience with procedural modelling tools reported an increase in the modelling workflow when compared with other similar tools (Figure 7.28 and Figure 7.39). The same set of users also reported an increase of creativity, expressiveness and focus on the creative process when compared with other procedural modelling tools (Figure 7.40 and Figure 7.41). On the other hand, the participants in the user study reported that the prototype was not easy to use, not flexible enough and not stimulating enough (Figure 7.44).

It is then possible to conclude that sketch based interfaces for modelling can improve the workflow of procedural modelling methods but that the presented approach is still not enough to achieve this. One of the identified main reasons is that the semantic queries to achieve the actionable shape selection is too cumbersome for some users, requiring too much mental effort to keep in mind the shape tree configuration. Moreover, the repetitive cycle of selecting shapes, choosing a useful candidate query from the presented ones and selecting a single shape on which to sketch seems to be a source of confusion and frustration when using the methodology. Implementation issues that caused sketches to be wrongly interpreted (either for procedural operations or guidelines and support lines) may have biased these results negatively.

Therefore, there is still a great amount of work to be done prior to considering the presented methodology completely useful as a full sketch based procedural modelling system. Nonetheless, it seems to be a solid starting point.

8.1.7 How can a sketch based interface for modelling enhance the expressiveness and direct control of procedural methods?

In the presented methodology, users are able to directly select the shapes that they wish to modify. The system is responsible for extrapolating the selection into a semantic definition that is adaptable to variations of the base model. This has been an open problem in procedural modelling of buildings that has prevented more interactive methods to be devised to which this thesis has presented a possible solution.

Moreover, by being able to partially sketch the aspect of the intended final geometry directly within existing geometry, the amount of times the user has to divert the attention to other parts of the user interface is reduced. Both these contributions have increased the amount of direct control the user has over the procedural modelling process.

On the other hand, the selection-action approach as described in Chapter 5 (and necessary in the sketch based methodology) also has the side effect of increasing the expressiveness of procedural methods by allowing the user to specify complex selections that are not only based on symbols and geometry flow but through implicit and explicit semantics. It becomes possible to easily detail different parts of the geometry given their hierarchical, geometrical and semantic attributes in a way that is not coupled with the execution of previous procedural rules, which is something that is observable in shape grammars and node-based approaches.

Another factor that increases the expressiveness of this type of systems is the (hypothesised) integration of the sketch based interfaces with a layered and vectorial based approach to procedural modelling of buildings such as presented in Chapter 4. Layers allow a greater number of variations to be created with a small effort while vectorially defined shapes within the layers can be used to produce more complex shapes that are not easy to achieve with current procedural methods alone. This brings the expressiveness level of procedural methods closer to the ones observable in sketch based interfaces for modelling.

8.2 Future Work

It is often desirable that any research work raises new questions which, in turn, open new lines of exploration within or outside its own field. During the development of this thesis several new avenues were identified and this section intends to briefly explain some of the possible future contributions.

Some of the items described here represent future work that directly stem from the work that was undertaken, while others are merely integrations between the presented contributions that have the potential to further enhance the flexibility of a sketch based methodology for procedural modelling of buildings.

8.2.1 Interactive Procedural Modelling with Layers

A consequence of extending shape grammars to include layers and vectorially defined shapes is that it is possible to construct shape grammars in a way that allows some level of interactive procedural modelling through *drag-and-drop* operations. This section presents a theoretical framework that exploits layered shape grammars for the purpose of increasing the interactivity of procedural modelling of buildings.

The main idea lies in the usage of a given pattern of rules and the manipulation of their parameters to simulate an architectural element which is able to being dragged within a planar shape. Therefore, this pattern makes use of the *layers* and *segment* operations as seen in Listing 8.1.

```
Planar → layers("Layer1", "Layer2", ..., "LayerN")
  {Layer1, Layer2, ..., LayerN}
LayerN → segment("x", left, width, ~1)
  {ε, VTileN, ε}
VTileN → segment("y", bottom, height, ~1)
  {ε, PayloadN, ε}
```

Listing 8.1: Layered shape grammar for interactive procedural modelling

This listing defines a *Planar* rule which represents the planar shape to which interactive architectural elements are added. As seen, this procedural item is decomposed in layers (*Layer1* through *LayerN*) which are further segmented horizontally and then vertically as given by the rules *LayerN* and *VTileN*. The *PayloadN* represents the architectural element to be inserted into the planar shape.

This pattern acts as a wrapper for a set of rules that generate architectural elements which can be placed and sized within the planar shape interactively, taking into account the remaining elements present. This is similar to the use of the layers in image editing tools.

Consider, for example, the façade in Figure 8.1, where a door is partially occluding a regular grid of windows. Using layered shape grammars, such façade could easily be decomposed into three distinct layers. The bottom-most layer would include the background shape and lateral frames, the middle layer would define the regular grid of windows and,

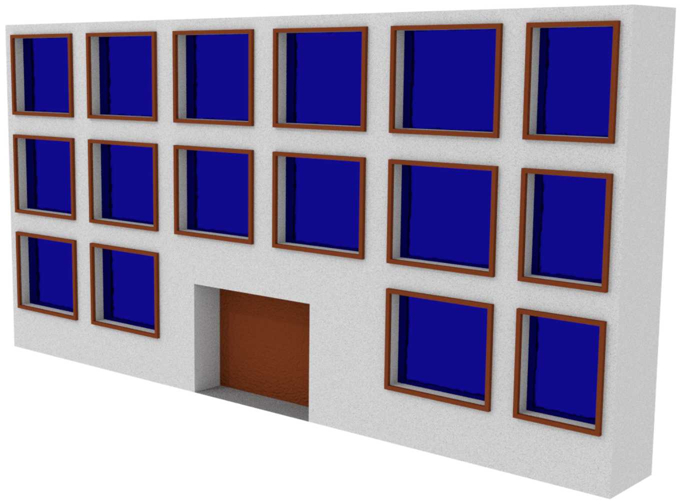


Figure 8.1: An example façade where the door is partially occluding a grid of windows.

finally, the top-most layer would generate the door. The derivation algorithm would correctly handle the occlusions. The layered shape grammar that would generate such façade is given in Listing 8.2.1.

```
Facade → layers("Background", "Windows", "Door")
  {Background, Windows, Door}
Background → ...
Windows → segment("x", windows_left, windows_width, ~1)
  {ε, WindowsVTile, ε}
WindowsVTile → segment("y", windows_bottom, windows_height, ~1)
  {ε, WindowsGrid, ε}
Door → segment("x", door_left, door_width, ~1)
  {ε, DoorVTileN, ε}
DoorVTileN → segment("y", door_bottom, door_height, ~1)
  {ε, DoorElement, ε}
```

It is easily visible that the *Windows* and *Door* sub-grammars follow the same pattern as the one in Listing 8.1 and, therefore, it is possible to define the *InteractiveLayer* template that is expanded into the pattern in Listing 8.2.

```
InteractiveLayer(Name, Payload, bottom, left, width, height):
  Name → segment("x", left, width, ~1)
    {ε, Name+"VTile", ε}
  Name+"VTile" → segment("y", bottom, height, ~1)
    {ε, Payload, ε}
```

Listing 8.2: Template replacement

The shape grammar in Listing 8.2.1 can then be defined as in Listing 8.3 and its decomposition into layers can be seen in Figure 8.2. It is possible to observe that the placement of the *payload* architectural elements within a 2D scene is given by four parameters: *bottom*, *left*, *width* and *height*, all present in the *InteractiveLayer* template. In this image, the *Windows Position* is defined as $(windows_{left}, windows_{bottom})$ and *Door Position* as $(door_{left}, door_{bottom})$.

```
Facade → layers("Background", "Windows", "Door")
  {Background, Windows, Door}
InteractiveLayer(Windows, WindowsGrid, windows_bottom, windows_left,
    windows_width, windows_height)
InteractiveLayer(Door, DoorElement, door_bottom, door_left, door_width, door_height)
Background → ...
```

Listing 8.3: Decomposition into layers using the *InteractiveLayer* template.

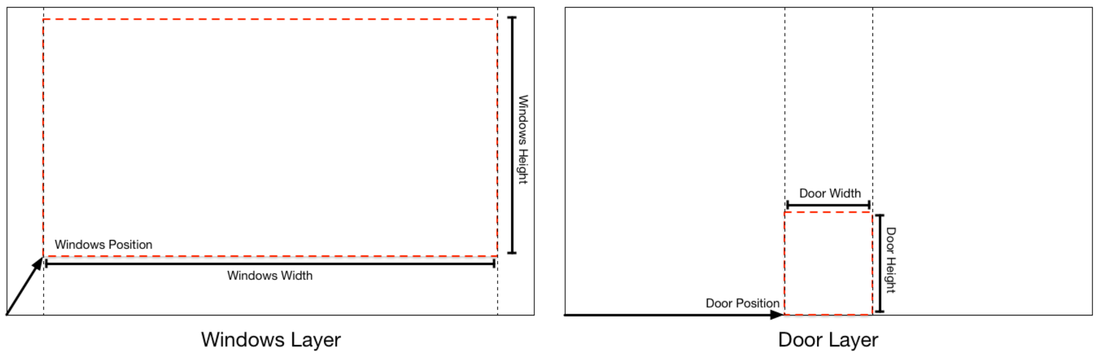


Figure 8.2: Schematic decomposition into layers of the shape grammar in Listing 8.3.

As such, it becomes simple to define interactive operations that make use of the template to modify the layered shape grammars to reflect the user intention by simply manipulating these four parameters. Such operations can be defined as a function that takes an InteractiveLayer$l$ and output a modified InteractiveLayerl' as defined in Equation 8.1.

$$\begin{aligned} InteractiveOperation : InteractiveLayer \rightarrow InteractiveLayer \\ l' = InteractiveOperation(l) \end{aligned} \tag{8.1}$$

For example, it is possible to define the *drag* operation to translate a given *payload* by a given amount as defined in Equation 8.2.

$$drag(l, \delta b, \delta l) = InteractiveLayer(l.Name, l.Payload, l.bottom{+}\delta b, l.left{+}\delta l, l.width, l.height) \tag{8.2}$$

Similarly, the *resize* operation could be defined as in Equation 8.3.

$$drag(l, \delta w, \delta h) = InteractiveLayer(l.Name, l.Payload, l.bottom, l.left, l.width{+}\delta w, l.height{+}\delta h) \tag{8.3}$$

Several more operations could be defined that would perform more complex operations with interactive layers such as those in the following list.

- same_size: Given two InteractiveLayer*l1* and *l2* set the size of *l2* to be equal to *l1*'s size.
- align: Given two InteractiveLayer*l1*, *l2* and an edge parameter (top, bottom, left, right) align the respective edge of *l2* to *l1*'s edge.
- center: Given two InteractiveLayer*l1*, *l2* and a axis parameter (vertical, horizontal) center *l2* vertically or horizontally to *l1*.

These operations can be found in most diagram and image editing software and are, therefore, proven to be helpful in interactive scenarios. It could also further enhance the user creativity within a sketch based procedural modelling system as it would allow achieving more variations with less effort.

A hypothetical interactive system could then expose these operations to the user directly within the resulting 3D model in such a way that the user could have direct control over the positioning and size of such InteractiveLayerand their *payloads*. A similar approach has been presented by Jesus et al. [JCS15] but to a smaller extent. An example of the drag operation has already been seen in Figure 1.2.

Since this is currently a theoretical framework, it is not possible to extrapolate the usefulness of such operations into the domain of procedural modelling of buildings. Future work should be put into developing such methodology and further evaluate whether interactive procedural modelling would benefit from it.

8.2.2 Integration of Layers in the Selection Action Paradigm

As it currently stands, there is no integration between the sketching interface with the layered shape grammars approach as described in Chapter 4. The main issue that prevented such integration is that it is not clearly evident when the normalisation step would take place in the selection action paradigm and how the system would infer it from user sketches. Ideally, the normalisation should happen automatically instead of requiring the user to create a normalisation procedural operation, relieving the user from such burden.

Given that the scheduling policy of procedural operations in the selection-action paradigm is a depth first traversal of the tree which contrasts with the procedural derivation mechanism of layered shape grammars which is based on a priority queue, some sort of hybrid mechanism could be the solution to this issue. This is, therefore, a problem closely related with the scheduling of procedural operations.

Once such integration is achieved, layers (and potentially the drag-and-drop mechanism described in Section 8.2.1) would be available to the sketch interface to make use of. This, however, raises the problem of how layers could be handled by the sketching interface alone, without resorting to use other parts of the interface. Nonetheless, it is a concept that most users are familiar with from image editing software.

The sketching interaction would, additionally, benefit from such integration since it would allow the user to sketch complex shapes that would be defined vectorially and seamlessly merged with other geometries, further increasing the expressiveness of such system.

8.3 Conclusions

Procedural modelling has acquired a place of great importance in Computer Graphics applications. It allows the creation of vast amounts of content with only a fraction of effort that would, otherwise, be required with traditional modelling tools where every element requires individual hand modelling. The impacts are, thus, extremely significative when there is a need to create a large number of 3D models.

This is the case of urban environments which typically have a lot of distinct elements (e.g., buildings, roads, urban furniture). In turn, each of such elements has its own individual characteristics packed with fine detail that further increase its uniqueness. Even though two buildings are designed by an architect to be exactly alike, in practice there will always be subtle differences, stemming from differences in construction, weathering or even improper use, that will give each building a unique look.

Modelling every single different urban element manually involves a great amount of effort and, thus, content creators have shifted, whenever possible, to tools that allow the rapid creation of models. Here, enters procedural modelling, which generates models from a set of parametrised rules. Varying such parameters yields slightly different models and, if we consider randomness, it is possible to create a (possibly) infinite number of distinct models. Modelling the slight differences between the two buildings described above has become somewhat simpler.

However, the content creator still has to create the rules to generate the models which, unfortunately, is not always simple. The currently most accepted forms of urban procedural modelling are the shape grammars and node based approaches which require an in-depth knowledge of procedural modelling methods. In the case of shape grammars this is further aggravated by the fact that the user must define the rules in a text based form that resembles a programming language.

Both shape grammars and node based approaches have the issue of incurring in a define-generate-analyse cycle which breaks the creative process, reduces the direct control over the models and gives the user only a small amount of visual feedback of what has changed between generations. Moreover, the definition of rules is not coupled with the visual representation of the models themselves. The user has to create the rules using auxiliary interfaces and visualize the results in the model viewer.

On the other hand, the content creator is, likely, of an artistic background (e.g., designer or architect) and this type of users are, most likely, proficient in drawing. The natural process in these situations is, usually, to first create concept drawings to obtain a clearer idea of how the final result should look and, then, to actually create the model (or procedural rules) in a modelling software. This clearly introduces a bottleneck. Bridging the gap between concept works and the final 3D model has always been one of the main purposes of sketch based interfaces for modelling. This thesis aimed at achieving the same results within the realm of procedural modelling of buildings.

Nonetheless, it was evident that the current, most accepted techniques for this purpose were not expressive enough. On the one hand, sketching allows to conceive all possible (and even impossible) shapes. On the other hand, procedural modelling which is currently over-reliant on split operations tend to restrict the generated shapes to rectangles or cuboids. More complex shapes need to be created elsewhere and imported as an asset into the modelling workflow.

Therefore, the first contribution within this thesis tried to enhance the expressiveness of shape grammars by allowing vectorially defined, arbitrary 2D shapes to be defined within faces of procedural models. This can be used to create arched windows, doors or more elaborate ornaments within façades for example. Such shapes are defined in layers within the faces which are posteriorly merged to create the final 3D geometry, taking into consideration and solving possible geometry conflicts between the vectorial shapes.

A sketching based interface is a deeply interactive metaphor where the user is constantly modifying the present model. To be able to precisely specify to which architectural elements a given change is to be applied, the user must somehow select parts of the model. While on traditional and sketch based modelling software the selection is exact and unambiguous (i.e., the selection is precisely what the user specified), in a procedural modelling methodology a selection can have multiple meanings. Due to the data amplification nature of these methods, a few selected floors can reflect the user intention of selecting the exact set of floors, all floors or some other superset of the selection. Moreover, the selection must remain valid throughout changes to the rule base.

Therefore, the second contribution in this thesis introduces a selection mechanism for procedural modelling of buildings that relies on semantic queries. This creates an intermediary level between the user intention and the final selection that adapts to rule changes (as long as the semantics remain the same) and allow the user to describe to what shapes modifications should be applied. However, manually creating the queries can easily become a burden to which this thesis presents a selection inference system based on genetic algorithms. The user must simply select a subset of shapes he or she wants to select and the system presents several candidate queries that try to match the intention.

With the capability of specifying which shapes will be affected by an operation, the user can then sketch on the selected shapes and have the procedural rules created by the system. This brings us to the third contribution in this thesis. Laying its foundations on the previous contributions and the already extensive and highly studied field of Sketch Based Interfaces for Modelling, this thesis presents a procedural modelling sketching system.

Here, the user sketches are compared in screen space to candidate lines obtained directly from existing procedural operations. The candidate line that best matches the user sketch is introduced in the 3D scene, allowing the user to continue sketching from another view point. On each new line, the system runs a set of heuristics that produce candidate procedural operations, estimating its parameters based on the current set of 3D lines. Each candidate is then fed to a parameter adjustment step which, with a gradient descent algorithm, will minimise the difference between the edges of the candidate generated geometry and the current set of 3D lines. The candidates with the lower difference are presented from which the user can choose.

Upon evaluating all the above contributions, results have shown that the possibility of defining layers in faces of procedural models which may contain vectorially defined 2D shapes increases the expressiveness of procedural modelling methods that rely on split operations. This has been demonstrated by constructing buildings with arched elements, and windows that overlap split lines showing the capability of creating irregular patterns that do not follow grid pattern. Comparing with regular shape grammars it's evident that this is an improvement. This, however also comes with the limitation that elements can too easily become unaligned, something that is much harder to happen using split methods alone. On the other hand, if the designer intends to create misalignments in the architecture this can be achieved easily.

The selection action paradigm has also demonstrated improvements over current architectural procedural modelling techniques when describing to which elements the procedural operations should be applied. The queries provide a mechanism with which users can precisely specify combinations of shapes that would be extremely difficult with, e.g., shape grammars. The selection inference subsystem has also proven to correctly infer the user intention given a small number of user selected shapes. Nonetheless, given its implementation under a genetic algorithm, it can produce different sets of candidate queries between different runs and requires some time to produce reasonable candidates. More effort should be put towards researching alternative implementations for the same goal. Machine learning and artificial intelligence methods should prove to be useful.

The sketching interface for procedural modelling of buildings has been evaluated with a set of eight users. The participants were grouped into two distinct groups based on whether they had previous contact with procedural modelling methodologies. They were asked to complete three tasks, two of which had a script guiding them to the completion while in the last they were allowed to freely use the prototype to model an exemplar building, and fill three questionnaires reporting their opinion about the methodology. Results show that participants in both groups were able to complete the tasks with a similar average time, which leads to the conclusion that the sketch based interface is able to reduce the learning curve of procedural modelling of buildings.

Participants with prior procedural modelling experience also reported an improvement in the modelling workflow when compared with regular procedural modelling techniques and that the sketching interface made the modelling process more pleasant. Nonetheless, there are a few drawbacks in the sketching interface which are tied with the over-reliance on the semantic queries as users reported that these were too cumbersome and complex to be easily usable. Moreover, users were required to perform repetitive actions that contributed to the user frustration. Examples of these are the multiple selections to obtain a valid semantic query and, posteriorly, selecting a single shape on which to sketch.

The integration of procedural modelling methods for urban environments with a sketch based interface appears to enhance the intuitiveness of the former while aiming to minimise the disruption to the creative process. Nonetheless, procedural methods require an improvement over its expressive power in order to be better paired with sketching interfaces. This thesis aimed to create such a bridge and, although more work is required in order to have a complete sketch based procedural modelling system, the results seem to confirm that this can be a solid starting point.

A

Semantic Selections in Procedural Modelling

1. Expertise with 3D Modelling

	1	2	3	4	5	6	7	
None	☐	☐	☐	☐	☐	☐	☐	Expert

2. Which 3D modelling tools have you used?

3. Expertise with Procedural Modelling

	1	2	3	4	5	6	7	
None	☐	☐	☐	☐	☐	☐	☐	Expert

4. Which Procedural Modelling tool have you used?

Semantic Queries

5. I found the ideas behind the semantic queries

6.

	1	2	3	4	5	6	7	
Uninteresting	☐	☐	☐	☐	☐	☐	☐	Very Interesting

7.

	1	2	3	4	5	6	7	
Common	☐	☐	☐	☐	☐	☐	☐	Original

8.

	1	2	3	4	5	6	7	
Not Useful	☐	☐	☐	☐	☐	☐	☐	Very Useful

9. I found the node tree

10.

	1	2	3	4	5	6	7	
Difficult to understand	☐	☐	☐	☐	☐	☐	☐	Easy to understand

11.

	1	2	3	4	5	6	7	
Not intuitive	☐	☐	☐	☐	☐	☐	☐	Intuitive

12.

	1	2	3	4	5	6	7	
Not Useful	☐	☐	☐	☐	☐	☐	☐	Very Useful

13. I found the procedural rule application order

14.

	1	2	3	4	5	6	7	
Not Intuitive	☐	☐	☐	☐	☐	☐	☐	Intuitive

15.

	1	2	3	4	5	6	7	
Not Useful	☐	☐	☐	☐	☐	☐	☐	Useful

Comparison to other procedural modelling tools

16. I found the prototype to be

17.

	1	2	3	4	5	6	7	
More difficult to understand	☐	☐	☐	☐	☐	☐	☐	Easier to understand

18.

	1	2	3	4	5	6	7	
Less adequate	☐	☐	☐	☐	☐	☐	☐	More adequate

19. I found that the prototype allowed me to create models

20.

	1	2	3	4	5	6	7	
Less expressively	☐	☐	☐	☐	☐	☐	☐	More expressively

21.

	1	2	3	4	5	6	7	
With less thought	☐	☐	☐	☐	☐	☐	☐	Had to think more

22.

	1	2	3	4	5	6	7	
Less creatively	☐	☐	☐	☐	☐	☐	☐	More creatively

The Semantic Query Inference

23. The generated queries were

24.

	1	2	3	4	5	6	7	
Inadequate	☐	☐	☐	☐	☐	☐	☐	Adequate

25.

	1	2	3	4	5	6	7	
Difficult to under-stand	☐	☐	☐	☐	☐	☐	☐	Easy to under-stand

26.

	1	2	3	4	5	6	7	
Not useful	☐	☐	☐	☐	☐	☐	☐	Useful

27. The ordering in which the queries were presented was

28.

	1	2	3	4	5	6	7	
Not useful	☐	☐	☐	☐	☐	☐	☐	Useful

29.

	1	2	3	4	5	6	7	
Not as expected	☐	☐	☐	☐	☐	☐	☐	As expected

30.

	1	2	3	4	5	6	7	
Not helpful	☐	☐	☐	☐	☐	☐	☐	Helpful

31. The selection inference, in general, made the procedural modelling process

32.

	1	2	3	4	5	6	7	
Slower	☐	☐	☐	☐	☐	☐	☐	Faster

33.

	1	2	3	4	5	6	7	
More complex	☐	☐	☐	☐	☐	☐	☐	Simpler

34.

	1	2	3	4	5	6	7	
Less intuitive	☐	☐	☐	☐	☐	☐	☐	More intuitive

The Prototype

35. The prototype in general was

36.

	1	2	3	4	5	6	7	
Not easy to use	☐	☐	☐	☐	☐	☐	☐	Easy to use

37.

	1	2	3	4	5	6	7	
Frustrating	☐	☐	☐	☐	☐	☐	☐	Stimulating

38.

	1	2	3	4	5	6	7	
Rigid	☐	☐	☐	☐	☐	☐	☐	Flexible

39. Regarding the prototype features

40.

	1	2	3	4	5	6	7	
Incomplete	☐	☐	☐	☐	☐	☐	☐	Complete

41.

	1	2	3	4	5	6	7	
Not suitable	☐	☐	☐	☐	☐	☐	☐	Suitable

42.

	1	2	3	4	5	6	7	
Not intuitive	☐	☐	☐	☐	☐	☐	☐	Intuitive

43.

	1	2	3	4	5	6	7	
Hard to Understand	☐	☐	☐	☐	☐	☐	☐	Easy to Understand

Suggestions and Critics

44. Please leave any suggestions or critics here

B

Sketch Based Interaction for Procedural Modelling: Initial Questionnaire

User Characterisation

1. User ID

___

2. Professional Area

____

3. Expertise with 3D Modelling

	1	2	3	4	5	6	7	
None	☐	☐	☐	☐	☐	☐	☐	Expert

4. Which 3D modelling tools have you used?

____

____

____

____

____

5. Expertise with Procedural Modelling

	1	2	3	4	5	6	7	
None	☐	☐	☐	☐	☐	☐	☐	Expert

6. Which Procedural Modelling tools have you used?

____

____

____

____

____

What is your opinion regarding the procedural modelling tools you have used

7. Features

	1	2	3	4	5	6	7	
Horrible	☐	☐	☐	☐	☐	☐	☐	Great

8. Interaction

	1	2	3	4	5	6	7	
Horrible	☐	☐	☐	☐	☐	☐	☐	Great

9. Expressiveness

	1	2	3	4	5	6	7	
Horrible	☐	☐	☐	☐	☐	☐	☐	Great

10. General Opinion

	1	2	3	4	5	6	7	
Horrible	☐	☐	☐	☐	☐	☐	☐	Great

Early Stages of Modelling

In the early stages of modelling I use

11. The modelling program itself

	1	2	3	4	5	6	7	
Never	☐	☐	☐	☐	☐	☐	☐	Always

12. Paper and Pencil

	1	2	3	4	5	6	7	
Never	☐	☐	☐	☐	☐	☐	☐	Always

13. Drawing Programs

	1	2	3	4	5	6	7	
Never	☐	☐	☐	☐	☐	☐	☐	Always

I believe concept drawings are useful for

14. Having a better understanding of the final scene

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

15. Rapidly exploring new ideas

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

I believe creating a 3D model from concept drawings

16. Is very straight forward

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

17. Is a bottleneck on the modelling process

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

I believe traditional modelling tools

18. Are, in general

	1	2	3	4	5	6	7	
Extremely hard to use	☐	☐	☐	☐	☐	☐	☐	Extremely easy to use

19. Are easy to use in the creation of concept art

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

20. Allow me to easily explore new ideas

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

21. Allow me to create detailed geometry

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

22. Are expressive enough to convey my ideas

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

I believe procedural modelling tools

23. Are, in general

	1	2	3	4	5	6	7	
Extremely hard to use	☐	☐	☐	☐	☐	☐	☐	Extremely easy to use

24. Are easy to use in the creation of concept art

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

25. Allow me to easily explore new ideas

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

26. Allow me to create detailed geometry

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

27. Are expressive enough to convey my ideas

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

C

Sketch Based Interaction for Procedural Modelling: Questionnaire After Prototype Presentation

1. User ID

Regarding the methodology

I found the ideas behind the methodology

2.

	1	2	3	4	5	6	7	
Uninteresting	☐	☐	☐	☐	☐	☐	☐	Very Interesting

3.

	1	2	3	4	5	6	7	
Ordinary	☐	☐	☐	☐	☐	☐	☐	Original

4.

	1	2	3	4	5	6	7	
Useless	☐	☐	☐	☐	☐	☐	☐	Useful

With respect to building modelling, I found the methodology to be

5.

	1	2	3	4	5	6	7	
Unfamiliar	☐	☐	☐	☐	☐	☐	☐	Familiar

6.

	1	2	3	4	5	6	7	
Inadequate	☐	☐	☐	☐	☐	☐	☐	Adequate

7.

	1	2	3	4	5	6	7	
Insufficient	☐	☐	☐	☐	☐	☐	☐	Sufficient

I found that the sketching interaction seemed

8.

	1	2	3	4	5	6	7	
Hard to use	☐	☐	☐	☐	☐	☐	☐	Easy to use

9.

	1	2	3	4	5	6	7	
Unintuitive	☐	☐	☐	☐	☐	☐	☐	Intuitive

10.

	1	2	3	4	5	6	7	
Incomplete	☐	☐	☐	☐	☐	☐	☐	Complete

Comparing with regular procedural modelling methods

I believe the prototype to be

11.

	1	2	3	4	5	6	7	
More difficult to understand	☐	☐	☐	☐	☐	☐	☐	Easier to understand

12.

	1	2	3	4	5	6	7	
Less adequate	☐	☐	☐	☐	☐	☐	☐	More adequate

13.

	1	2	3	4	5	6	7	
Less intuitive	☐	☐	☐	☐	☐	☐	☐	More intuitive

I believe that the prototype allows to create models

14.

	1	2	3	4	5	6	7	
Less Expressively	☐	☐	☐	☐	☐	☐	☐	More Expressively

15.

	1	2	3	4	5	6	7	
With more thought	☐	☐	☐	☐	☐	☐	☐	With less thought

16.

	1	2	3	4	5	6	7	
Less Creatively	☐	☐	☐	☐	☐	☐	☐	More Creatively

I believe the sketching interface allows to

17. Be more creative

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

18. Think less about the procedural modelling rules

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

19. Focus on the creative process

	1	2	3	4	5	6	7	
Strongly Agree	☐	☐	☐	☐	☐	☐	☐	Strongly Disagree

In general, I believe the methodology

20. Improves the procedural modelling workflow.

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

21. Makes the modelling process more pleasant.

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

The prototype

The prototype in general seems

22.

	1	2	3	4	5	6	7	
Hard to use	☐	☐	☐	☐	☐	☐	☐	Easy to use

23.

	1	2	3	4	5	6	7	
Frustrating	☐	☐	☐	☐	☐	☐	☐	Stimulating

24.

	1	2	3	4	5	6	7	
Rigid	☐	☐	☐	☐	☐	☐	☐	Flexible

D

Sketch Based Interaction for Procedural Modelling: Task 1

In this first task you will model a simple building with a grid façade similar to the one in Figure D.1. To do this you will use the selection inference the sketching interface as described in the video. This script will guide you through the steps required. You are free to "think aloud" through the process and ask questions as you go.

Steps

When the task starts you will see the prototype in a state similar to the one in Figure E.2.

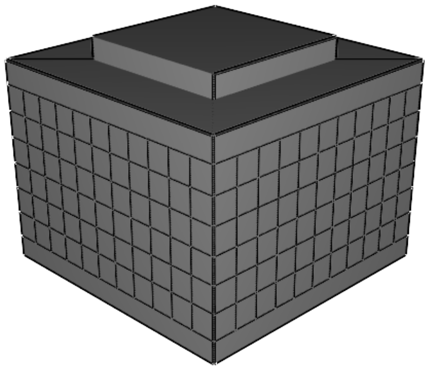
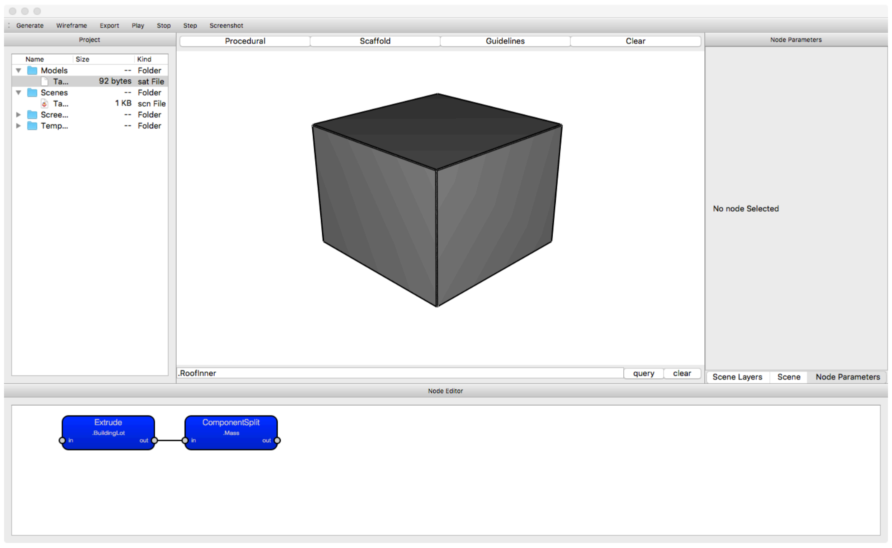


Figure D.1



Name	Size	Kind
Models	--	Folder
Ta...	92 bytes	sat File
Scenes	--	Folder
Ta...	1 KB	scn File
Scree...	--	Folder
Temp...	--	Folder

.RoofInner

Figure D.2

1.Select both visible façades.

2.In the menu that appears, select the .*Facade* or a similar query and press enter.

3.Select one of the façades.

4.In the selected façade draw two lines for the first and last floors.

5.In the menu that appears, select the *Split along Y* option and press enter.

6.In the node parameters set the symbol for the middle split to *CenterFloors*.

7.Press generate.

8.Select the shape corresponding to the middle floors.

9.In the menu that appears select the *.CenterFloors* query.

10.Draw a consecutive series of equally spaced horizontal lines to represent the middle floors.

11.In the menu that appears select the *Repeat split along Y*.

12.In the node parameters set the floors symbol to *Floor*.

13.Press generate.

14.Select a few floors.

15.In the menu that appears select the *.Floor* or similar query.

16.In one of the selected floors draw a consecutive series of equally spaced vertical lines to represent blocks of windows.

17.In the menu that appears select the *Repeat split along X*.

18.In the node parameters set the window block symbol to *WindowBlock*.

19.Press generate.

20.Select the roof shape.

21.In the menu that appears select the *.Roof* query.

22.Draw another concentric rectangle in the roof shape.

23.In the menu that appears select the *Offset* option and press enter.

24.In the node parameters set the inner shape symbol to *RoofInner*.

25.Select the roof inner shape.

26.In the menu that appears select the *.RoofInner* or similar query and press enter.

27.Draw a vertical line upwards from one of the selected shape's inner vertices.

28.In the menu that appears select the *Extrude by* option and press enter.

E

Sketch Based Interaction for Procedural Modelling: Task2

In the second task you will use the guidelines and scaffolding lines to help define more precise sizes. The final result should be similar to the one in Figure E.1. You are free to "think aloud" through the process and ask questions as you go.

Steps

When the task starts you will see the prototype in a state similar to the one in Figure E.2.

1.Select the roof shape.

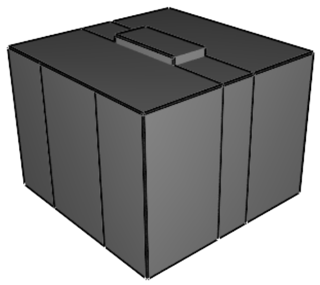


Figure E.1

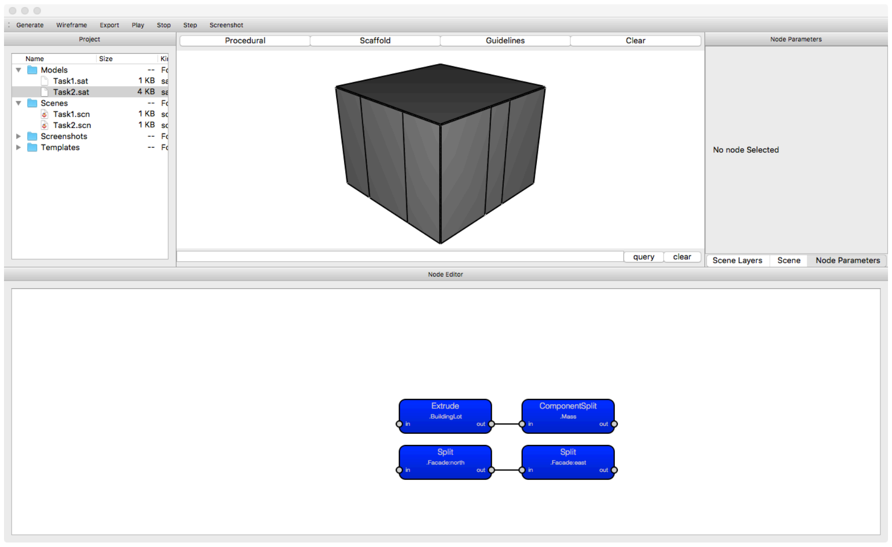


Figure E.2

2.In the menu that appears, select the *.Roof* or a similar query and press enter.

3.Click on the *Guidelines* button on the top.

4.From one of the vertices of a split line on one of the façades draw a line through the roof shape to create a guideline.

5.Do the same for the other split line in the same façade.

6.Click on the *Scaffold* button on the top.

7.Use the guidelines to create two split lines in the roof.

8.Double sketch over each of the split lines.

9.In the menu that appears select the *Split along ...* option and press enter.

10.In the node parameters set the symbol for the middle split to *Split1*.

11.Press generate.

12.Select the *Split1* shape.

13.In the menu that appears select the *.Split1* query.

14.Click on the *Guidelines* button on the top.

15.Similar to the first façade, create a guideline for each split line in the other façade.

16.Click on the *Scaffold* button on the top.

17.Use the guidelines to create two split lines in the roof.

18.Double sketch over each of the split lines.

19.In the menu that appears select the *Split along...* option and press enter.

20.In the node parameters set the symbol for the middle split to *Split2*.

21.Select the *Split2* shape.

22.In the menu that appears select the *Split2* query.

23.Draw a vertical line upwards from one of the vertices of the *Split2* shape.

24.In the menu that appears select the *Extrude by* option and press enter.

F

Sketch Based Interaction for Procedural Modelling: Task 3

In the third and task you will freely use the prototype in order to create a building similar to the one in F.1. Again, you are free to "think aloud" through the process and ask questions as you go. As a side note, the initial building volume with which you will be presented is a 36 units wide, 25 units deep and 15 units tall.

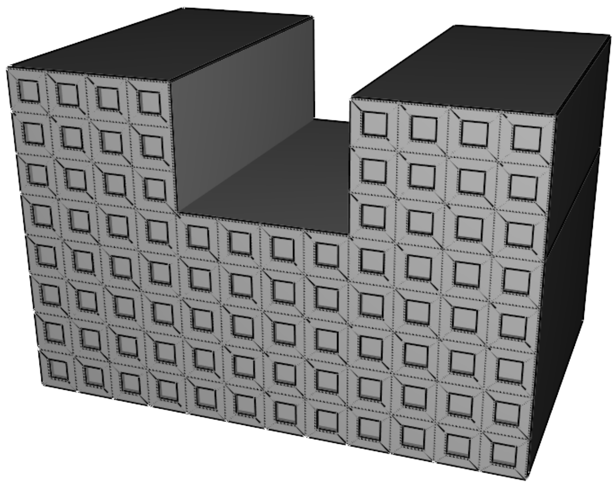


Figure F.1

G

Sketch Based Interaction for Procedural Modelling: Questionnaire After Experimenting with the Prototype

1. User ID

Regarding the methodology

With respect to building modelling, I found the methodology to be

2.

	1	2	3	4	5	6	7	
Unfamiliar	☐	☐	☐	☐	☐	☐	☐	Familiar

3.

	1	2	3	4	5	6	7	
Inadequate	☐	☐	☐	☐	☐	☐	☐	Adequate

4.

	1	2	3	4	5	6	7	
Insufficient	☐	☐	☐	☐	☐	☐	☐	Sufficient

The Sketching Interface

I found that the sketching interaction was

5.

	1	2	3	4	5	6	7	
Hard to use	☐	☐	☐	☐	☐	☐	☐	Easy to use

6.

	1	2	3	4	5	6	7	
Unintuitive	☐	☐	☐	☐	☐	☐	☐	Intuitive

7.

	1	2	3	4	5	6	7	
Incomplete	☐	☐	☐	☐	☐	☐	☐	Complete

I FOUND THAT THE SUPPORT LINES WERE

8.

	1	2	3	4	5	6	7	
Not helpful	☐	☐	☐	☐	☐	☐	☐	Helpful

9.

	1	2	3	4	5	6	7	
Inadequate	☐	☐	☐	☐	☐	☐	☐	Adequate

10.

	1	2	3	4	5	6	7	
Hard to use	☐	☐	☐	☐	☐	☐	☐	Easy to use

I FOUND THAT THE GUIDELINES WERE

11.

	1	2	3	4	5	6	7	
Not helpful	☐	☐	☐	☐	☐	☐	☐	Helpful

12.

	1	2	3	4	5	6	7	
Inadequate	☐	☐	☐	☐	☐	☐	☐	Adequate

13.

	1	2	3	4	5	6	7	
Hard to use	☐	☐	☐	☐	☐	☐	☐	Easy to use

I found that the procedural operations the system suggested were

14.

	1	2	3	4	5	6	7	
Not Adequate	☐	☐	☐	☐	☐	☐	☐	Adequate

15.

	1	2	3	4	5	6	7	
Not useful	☐	☐	☐	☐	☐	☐	☐	Useful

16.

	1	2	3	4	5	6	7	
Incorrect	☐	☐	☐	☐	☐	☐	☐	Correct

I found that the inferred parameters in the procedural operations were

17.

	1	2	3	4	5	6	7	
Not Adequate	☐	☐	☐	☐	☐	☐	☐	Adequate

18.

	1	2	3	4	5	6	7	
Not useful	☐	☐	☐	☐	☐	☐	☐	Useful

19.

	1	2	3	4	5	6	7	
Incorrect	☐	☐	☐	☐	☐	☐	☐	Correct

Comparing with regular procedural modelling methods

I found the prototype to be

20.

	1	2	3	4	5	6	7	
More difficult to understand	☐	☐	☐	☐	☐	☐	☐	Easier to under-stand

21.

	1	2	3	4	5	6	7	
Less adequate	☐	☐	☐	☐	☐	☐	☐	More adequate

22.

	1	2	3	4	5	6	7	
Less intuitive	☐	☐	☐	☐	☐	☐	☐	More intuitive

23.

	1	2	3	4	5	6	7	
More frustrating	☐	☐	☐	☐	☐	☐	☐	Less frustrating

I found that the prototype allowed me to create models

24.

	1	2	3	4	5	6	7	
Less Expressively	☐	☐	☐	☐	☐	☐	☐	More Expressively

25.

	1	2	3	4	5	6	7	
With more thought	☐	☐	☐	☐	☐	☐	☐	With less thought

26.

	1	2	3	4	5	6	7	
Less Creatively	☐	☐	☐	☐	☐	☐	☐	More Creatively

THE SKETCHING INTERFACE ALLOWED ME TO

27. Be more creative

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

28. Think less about the procedural modelling rules

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

29. Focus on the creative process

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

IN GENERAL, I BELIEVE THE METHODOLOGY

30. Improves the procedural modelling workflow.

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

31. Makes the modelling process more pleasant.

	1	2	3	4	5	6	7	
Strongly Disagree	☐	☐	☐	☐	☐	☐	☐	Strongly Agree

The prototype

The prototype in general was

32.

	1	2	3	4	5	6	7	
Hard to use	☐	☐	☐	☐	☐	☐	☐	Easy to use

33.

	1	2	3	4	5	6	7	
Frustrating	☐	☐	☐	☐	☐	☐	☐	Stimulating

34.

	1	2	3	4	5	6	7	
Rigid	☐	☐	☐	☐	☐	☐	☐	Flexible

I found the prototype features to be

35.

	1	2	3	4	5	6	7	
Incomplete	☐	☐	☐	☐	☐	☐	☐	Complete

36.

	1	2	3	4	5	6	7	
Not Suitable	☐	☐	☐	☐	☐	☐	☐	Suitable

37.

	1	2	3	4	5	6	7	
Not Intuitive	☐	☐	☐	☐	☐	☐	☐	Intuitive

38.

	1	2	3	4	5	6	7	
Hard to Understand	☐	☐	☐	☐	☐	☐	☐	Easy to Understand

Suggestions and Critics

39. Please leave any suggestions or critics here

____

____

____

____

____

____

____

____

Bibliography

[AGB04] Ileana Anca Alexe, Véronique Gaildrat, and Loïc Barthe. "Interactive modelling from sketches using spherical implicit functions". In: *Proceedings of the 3rd international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*. AFRIGRAPH '04. ACM, 2004, pp. 25–34. DOI: 10.1145/1029949.1029953.

[ASA07] Nguyen Hoang Anh, Alexei Sourin, and Parimal Aswani. "Physically based hydraulic erosion simulation on graphics processing unit". In: *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*. GRAPHITE '07. 2007, pp. 257–264. DOI: 10.1145/1321261.1321308.

[BBP13] Santiago Barroso, Gonzalo Besuievsky, and Gustavo Patow. "Visual copy & paste for procedurally modeled buildings by ruleset rewriting". In: *Computers & Graphics* 37.4 (2013), pp. 238–246. DOI: 10.1016/j.cag.2013.01.003.

[BC90] Michael J Banks and Elaine Cohen. "Real time spline curves from interactively sketched data". In: *SIGGRAPH Computer Graphics* 24.2 (1990), pp. 99–107. DOI: 10.1145/91394.91425.

[BF01] Bedrich Beneš and Rafael Forsbach. "Layered Data Representation for Visual Simulation of Terrain Erosion". In: *Proceedings of the 17th Spring Conference on Computer Graphics*. SCCG '01. 2001, pp. 80–86.

[BN08] Eric Bruneton and Fabrice Neyret. "Real-time rendering and editing of vector-based terrains". In: *Computer Graphics Forum* 27.2 (2008), pp. 311–320. DOI: 10.1111/j.1467-8659.2008.01128.x.

[Bos+11] Carles Bosch et al. "Image-guided Weathering: A New Approach Applied to Flow Phenomena". In: *ACM Transactions on Graphics* 30.3 (2011), 20:1–20:13. DOI: 10.1145/1966394.1966399.

[BP12] Santiago Barroso and Gustavo Patow. “Visual Language Generalization for Procedural Modeling of Buildings”. In: *Spanish Computer Graphics Conference*. 2012, pp. 57–66. DOI: 10.2312/LocalChapterEvents/CEIG/CEIG12/057-066.

[BP13] Gonzalo Besuievsky and Gustavo Patow. “Customizable Lod For Procedural Architecture”. In: *Computer Graphics Forum* 32.8 (2013), pp. 26–34. DOI: 10.1111/Cgf.12141.

[BWK14] Jan Beneš, Alexander Wilkie, and Jaroslav Křivánek. “Procedural Modelling of Urban Road Networks”. In: *Computer Graphics Forum* 33.6 (2014), pp. 132–142. DOI: 10.1111/cgf.12283.

[CB09] Giliam J. P. de Carpentier and Rafael Bidarra. “Interactive GPU-based procedural heightfield brushes”. In: *Proceedings of the 4th International Conference on Foundations of Digital Games*. FDG ’09. 2009, pp. 55–62. DOI: 10.1145/1536513.1536532.

[Che+05] Joseph Jacob Cherlin et al. “Sketch-based modeling with few strokes”. In: *Proceedings of the 21st Spring Conference on Computer graphics - SCCG ’05*. ACM, 2005, p. 137. DOI: 10.1145/1090122.1090145.

[Che+08a] Guoning Chen et al. “Interactive procedural street modeling”. In: *ACM Transaction on Graphics*. SIGGRAPH ’08 27.3 (2008), 103:1–103:10. DOI: 10.1145/1399504.1360702.

[Che+08b] Xuejin Chen et al. “Sketching reality: Realistic interpretation of architectural designs”. In: *ACM Transaction on Graphics* 27.2 (2008), 11:1–11:15. DOI: 10.1145/1356682.1356684.

[Coe+07] António Coelho et al. “Expeditious Modelling of Virtual Urban Environments with Geospatial L-systems”. In: *Computer Graphics Forum* 26.4 (2007), pp. 769–782.

[CS07] Frederic Cordier and Hyewon Seo. “Free-Form Sketching of Self-Occluding Objects”. In: *IEEE Computer Graphics and Applications* 27.1 (2007), pp. 50–59. DOI: 10.1109/MCG.2007.8.

[Dan07] Danc. *Lost Garden: Content is Bad*. http://lostgarden.com/2007/02/content-is-bad.html. 2007.

[DB05] Jürgen Döllner and Henrik Buchholz. "Continuous level-of-detail modeling of buildings in 3D city models". In: *Proceedings of the 13th Annual ACM International Workshop on Geographic Information Systems*. GIS '05. 2005, pp. 173–181. DOI: 10.1145/1097064.1097089.

[DE03] Geoffrey M. Draper and Parris K. Egbert. "A Gestural Interface to Free-Form Deformation". In: *Proceedings of the Graphics Interface 2003 Conference*. 2003, pp. 113–120. DOI: 10.20380/GI2003.14.

[Dis09] B Discoe. *Artificial Terrain Generation*. http://www.vterrain.org/elevation/artificial. 2009.

[DPH96] Julie Dorsey, Hans Køhling Pedersen, and Pat Hanrahan. "Flow and changes in appearance". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. 1996, pp. 411–420. DOI: 10.1145/237170.237280.

[Esr14] Esri. *CityEngine*. http://www.esri.com/software/cityengine. 2014.

[FFC82] Alain Fournier, Don Fussell, and Loren Carpenter. "Computer rendering of stochastic models". In: *Communications of the ACM* 25.6 (1982), pp. 371–384. DOI: 10.1145/358523.358553.

[Fin08] Dieter Finkenzeller. "Detailed Building Facades". In: *IEEE Computer Graphics and Applications* 28.3 (2008), pp. 58–66. DOI: 0.1109/MCG.2008.50.

[Fun+03] Thomas Funkhouser et al. "A Search Engine for 3D Models". In: *ACM Transactions on Graphics* 22.1 (2003), pp. 83–105. DOI: 10.1145/588272.588279.

[GK01] Manuel Gamito and F Kenton Musgrave. "Procedural Landscapes with Overhangs". In: *10th Portuguese Computer Graphics Meeting*. 2001, pp. 33–42.

[Gre+03] Stefan Greuter et al. “Real-time Procedural Generation of ‘Pseudo Infinite’ Cities”. In: *Proceedings of the 1st nternational Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*. GRAPHITE ’03. ACM, 2003, pp. 87–94. DOI: 10.1145/604471.604490.

[Gro94] Markus Gross. “The fat pencil, the cocktail napkin, and the slide library”. In: *In Proceedings of ACADIA94 National Conference*. 1994, pp. 103–113.

[Hae+10] Simon Haegler et al. “Grammar-based Encoding of Facades”. In: *Proceedings of the 21st Eurographics Conference on Rendering*. 2010, pp. 1479–1487. DOI: 10.1111/j.1467-8659.2010.01745.x.

[Hal08] L Halliwell. *Procedural content generation*. http://lukealiiwell.wordpress.com/2008/08/05/pro content-generation/. 2008.

[Iga+97] Takeo Igarashi et al. “Interactive beautification: a technique for rapid geometric design”. In: *Proceedings of the 10th annual ACM symposium on User interface software and technology*. ACM, 1997, pp. 105–114. DOI: 10.1145/263407.263525.

[IH01] Takeo Igarashi and John F Hughes. “A suggestive interface for 3D drawing”. In: *Proceedings of the 14th annual ACM symposium on User interface software and technology*. ACM, 2001, pp. 173–181. DOI: 10.1145/502348.502379.

[IMT99] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. “Teddy: a sketching interface for 3D freeform design”. In: *ACM SIGGRAPH 2007 courses*. SIGGRAPH ’07. 1999, pp. 409–416. DOI: 10.1145/1281500.1281532.

[JCS15] Diego Jesus, António Coelho, and António Augusto Sousa. “Towards Interactive Procedural Modelling of Buildings”. In: *Proceedings of the 31st Spring Conference on Computer Graphics*. 2015, pp. 109–112. DOI: 10.1145/2788539.2788554.

[JCS16] Diego Jesus, António Coelho, and António Augusto Sousa. “Layered shape grammars for procedural modelling of buildings”. In: *The Visual Computer* 32.6 (2016), pp. 933–943. DOI: 10.1007/s00371-016-1254-8.

[Jes+12] Diego Jesus et al. "Modeling Urban Environments from Geospatial Data A Pipeline for Procedural Modeling". In: *Proceedings of PCG 2012 - Workshop on Procedural Content Generation for Games, co-located with the Foundations of Digital Games 2012*. 2012, 5:1–5:8. DOI: 10.1145/2538528.2538533.

[Jes+18] Diego Jesus et al. "Generalized selections for direct control in procedural buildings". In: *Computers & Graphics* 72 (2018), pp. 106–121. DOI: 10.1016/j.cag.2018.02.003.

[JLA04] Roland Juchmes, Pierre Leclercq, and Sleiman Azar. "A Multi-Agent System for the Interpretation of Architectural Sketches". In: *Proceedings of the First Eurographics Conference on Sketch-Based Interfaces and Modeling*. Eurographics Association, 2004, pp. 53–61. DOI: 10.2312/SBM/SBM04/053-061.

[KBK13] Lars Krecklau, Janis Born, and Leif Kobbelt. "View-Dependent Realtime Rendering of Procedural Facades with High Geometric Detail". In: *Computer Graphics Forum* 32.2 (2013), pp. 479–488. DOI: 10.1111/cgf.12068.

[KH06] Olga A Karpenko and John F Hughes. "SmoothSketch: 3D free-form shapes from complex sketches". In: *ACM Transaction on Graphics* 25.3 (2006), pp. 589–598. DOI: 10.1145/1141911.1141928.

[KHR04] Olga Karpenko, John F Hughes, and Ramesh Raskar. "Epipolar methods for multi-view sketching". In: *Proceedings of the First Eurographics conference on Sketch-Based Interfaces and Modeling*. SBM'04. Eurographics Association, 2004, pp. 167–173. DOI: 10.2312/SBM/SBM04/167-173.

[KM07] George Kelly and Hugh McCabe. "Citygen: An Interactive System for Procedural City Generation". In: *Proceedings of GDTW 2007: The Fifth Annual International Conference in Computer Game Design and Technology*. 2007, pp. 8–16.

[KMN88] Alex D Kelley, Michael C Malin, and Gregory M Nielson. "Terrain simulation using a model of stream erosion". In: *SIGGRAPH Comput. Graph.* 22.4 (1988), pp. 263–268. DOI: 10.1145/378456.378519.

[KPK10] Lars Krecklau, Darko Pavic, and Leif Kobbelt. "Generalized Use of Non-Terminal Symbols for Procedural Modeling". In: *Computer Graphics Forum* 29.8 (2010), pp. 2291–2303. DOI: 10.1111/j.1467-8659.2010.01714.x.

[KW11] Tom Kelly and Peter Wonka. "Interactive architectural modeling with procedural extrusions". In: *ACM Transactions on Graphics* 30.2 (2011), 14:1–14:15. DOI: 10.1145/1944846.1944854.

[Lev66] Vladimir Iosifovich Levenshtein. "Binary codes capable of correcting deletions, insertions and reversals". In: *Soviet Physics Doklady* 10.8 (1966), pp. 707–710.

[LF08] Jeehyung Lee and Thomas Funkhouser. "Sketch-based Search and Composition of 3D Models". In: *Proceedings of the Fifth Eurographics Conference on Sketch-Based Interfaces and Modeling*. SBM'08. Eurographics Association, 2008, pp. 97–104. DOI: 10.2312/SBM/SBM08/097-104.

[Lin68] Aristid Lindenmayer. "Mathematical models for cellular interactions in development I. Filaments with one-sided inputs". In: *Journal of Theoretical Biology* 18.3 (1968), pp. 280–299. DOI: https://doi.org/10.1016/0022-5193(68)90079-9.

[Lip+11] M Lipp et al. "Interactive Modeling of City Layouts using Layers of Procedural Content". In: *Computer Graphics Forum* 30.2 (2011), pp. 345–354. DOI: 10.1111/j.1467-8659.2011.01865.x.

[Lon+12] Steven Longay et al. "TreeSketch: Interactive Procedural Modeling of Trees on a Tablet". In: *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*. SBIM '12. 2012, pp. 107–120. DOI: 10.2312/SBM/SBM12/107-120.

[LS07] H Lipson and M Shpitalni. "Optimization-based reconstruction of a 3D object from a single freehand line drawing". In: *ACM SIGGRAPH 2007 courses*. SIGGRAPH '07. 2007. DOI: 10.1145/1281500.1281556.

[LWW03] Thomas Lechner, Ben Watson, and Uri Wilensky. "Procedural city modeling". In: *1st Midwestern Graphics Conference*. 2003.

[LWW08] Markus Lipp, Peter Wonka, and Michael Wimmer. “Interactive visual editing of grammars for procedural architecture”. In: *ACM Transactions on Graphics* 27.3 (2008), 102:1–102:10. DOI: 10.1145/1360612.1360701.

[Man83] Benoit B Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freedman and Co., 1983. ISBN: 0-7167-1186-9.

[Mar+97] I Marák et al. “Terrain Erosion Model Based on Rewriting of Matrices”. In: *Proceedings of WSCG'97*. 1997, pp. 341–351.

[Mar96] Paul Martz. *Generating Random Fractal Terrain*. http://www.gameprogrammer.com/fractal.html. July 1996.

[Mer07] Paul Merrell. “Example-based model synthesis”. In: *Proceedings of the 2007 symposium on Interactive 3D graphics and games - I3D '07*. ACM Press, 2007, pp. 105–112. DOI: 10.1145/1230100.1230119.

[Mil86] Gavin S P Miller. “The definition and rendering of terrain maps”. In: *SIGGRAPH Computer Graphics* 20.4 (1986), pp. 39–48. DOI: 10.1145/15886.15890.

[MKM89] F. Kenton Musgrave, Craig E. Kolb, and Robert S. Mace. “The synthesis and rendering of eroded fractal terrains”. In: *SIGGRAPH Computer Graphics* 23.3 (1989), pp. 41–50. DOI: 10.1145/74334.74337.

[MM08] Paul Merrell and Dinesh Manocha. “Continuous model synthesis”. In: *ACM Transactions on Graphics* 27.5 (2008), 158:1–158:7. DOI: 10.1145/1409060.1409111.

[MM11] Paul Merrel and Dinesh Manocha. “Model Synthesis: A General Procedural Modeling Algorithm”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.6 (2011), pp. 715–728. DOI: 10.1109/TVCG.2010.112.

[MP96] Radomír Měch and Przemyslaw Prusinkiewicz. “Visual models of plants interacting with their environment”. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. SIGGRAPH '96. 1996, pp. 397–410. DOI: 10.1145/237170.237279.

[Mül+06] Pascal Müller et al. “Procedural modeling of buildings”. In: *ACM Transactions on Graphics* 25.3 (2006), pp. 614–623. DOI: 10.1145/1141911.1141931.

[Mül+07] Pascal Müller et al. “Image-based procedural modeling of facades”. In: *ACM Transactions on Graphics* 26.3 (2007), p. 85. DOI: 10.1145/1275808.1276484.

[Muñ+16] I. Muñoz-Pandiella et al. “Weathering of Urban Scenes: Challenges and Possible Solutions”. In: *Workshop on Material Appearance Modeling*. 2016. DOI: 10.2312/mam.20161248.

[Mus93] Forest Kenton Musgrave. “Methods for realistic landscape imaging”. PhD thesis. Yale University, 1993.

[MWW12] Przemyslaw Musialski, Michael Wimmer, and Peter Wonka. “Interactive Coherence-Based Facade Modeling”. In: *Computer Graphics Forum* 31.2pt3 (2012), pp. 661–670. DOI: 10.1111/j.1467-8659.2012.03045.x.

[Nea+05] Andrew Nealen et al. “A sketch-based interface for detail-preserving mesh editing”. In: *ACM Transactions on Graphics* 24.3 (2005), p. 1142. DOI: 10.1145/1073204.1073324.

[Nea+07] Andrew Nealen et al. “FiberMesh: Designing Freeform Surfaces with 3D Curves”. In: *ACM Transaction on Graphics* 26.3 (2007). DOI: 10.1145/1276377.1276429.

[Nis+16] Gen Nishida et al. “Interactive Sketching of Urban Procedural Models”. In: *ACM Transactions on Graphics* 35.4 (2016), 130:1–130:11. DOI: 10.1145/2897824.2925951.

[Ols+05] L. Olsen et al. “Sketch-based mesh augmentation”. In: *2nd Eurographics workshop on sketch-based interfaces and modeling (SBIM)*. The Eurographics Association, 2005. DOI: 10.2312/SBM/SBM05/043-052.

[Ols+09] Luke Olsen et al. “Sketch-based modeling: A survey”. In: *Computers & Graphics* 33.1 (2009), pp. 85–103. DOI: 10.1016/j.cag.2008.09.013.

[Ols+11] DJ Olsen et al. “Sketch-based building modelling”. In: *GRAPP 2011 - Proceedings of the International Conference on Computer Graphics Theory and Applications*. 2011, pp. 119–124.

[Olso4] Jacob Olsen. “Realtime procedural terrain generation”. In: *University of Southern Denmark* (2004), p. 20.

[Pat12] Gustavo Patow. “User-Friendly Graph Editing for Procedural Modeling of Buildings”. In: *IEEE Computer Graphics and Applications* 32 (2 2012), pp. 66–75. DOI: 10.1109/MCG.2010.104.

[Per+00] João Paulo Pereira et al. “Towards Calligraphic Interfaces: Sketching 3D Scenes with Gestures and Context Icons”. In: *The 8-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2000, WSCG 2000, University of West Bohemia, Campus Bory, Plzen-Bory, Czech Republic, February 7-11, 2000*. 2000.

[Per+03] João P. Pereira et al. “Calligraphic interfaces: Mixed metaphors for design”. In: *Interactive Systems. Design, Specification, and Verification*. Springer Berlin Heidelberg, 2003, pp. 154–170. ISBN: 978-3-540-39929-2.

[Pey+09] A. Peytavie et al. “Arches: a Framework for Modeling Complex Terrains”. In: *Computer Graphics Forum* 28.2 (2009), pp. 457–467. DOI: 10.1111/j.1467-8659.2009.01385.x.

[Pie87] L Piegl. “Interactive Data Interpolation by Rational Bezier Curves”. In: *IEEE Computer Graphics Applications* 7.4 (1987), pp. 45–58. DOI: 10.1109/MCG.1987.276871.

[PJ85] Ken Perlin and S A N Francisco July. “An image synthesizer”. In: *SIGGRAPH Computer Graphics* 19.3 (1985), pp. 287–296. DOI: 10.1145/325165.325247.

[PJM94] Przemyslaw Prusinkiewicz, Mark James, and Radomír Měch. “Synthetic topiary”. In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. SIGGRAPH '94. 1994, pp. 351–358. DOI: 10.1145/192161.192254.

[PL90] P Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., 1990. ISBN: 0-387-97297-8.

[PM01] Yoav I H Parish and Pascal Müller. “Procedural modeling of cities”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’01. 2001, pp. 301–308. DOI: 10.1145/383259.383292.

[PP13] Oriol Pueyo and Gustavo Patow. “Structuring urban data”. In: *The Visual Computer: International Journal of Computer Graphics* 30.2 (2013), pp. 159–172. DOI: 10.1007/s00371-013-0791-7.

[RCR09] Roberto Rodrigues, António Coelho, and Luís Reis. “Modelação expedita de edifícios monumentais a partir de descrições textuais”. In: *Encontro Português de Computação Gráfica*. 2009, pp. 35–43.

[Rem08] C Remo. *MIGS: Far Cry 2’s Guay On The Importance of Procedural Content*. http://www.gamasutra.com/php-bin/news%5Findex.php?story=21165. 2008.

[Rob11] María Dolores Robles-Ortega. “Navegacion e interaccion en entornos urbanos reales”. PhD thesis. Universidad de Jaén, 2011.

[ROF13] María Dolores Robles-Ortega, Lidia Ortega, and Francisco R Feito. “A new approach to create textured urban models through genetic algorithms”. In: *Information Sciences* 243 (2013), pp. 1–19. DOI: 10.1016/j.ins.2013.03.053.

[RP10] Remei Ridorsa and Gustavo Patow. “The skylineEngine System”. In: *XX Congresso Espanõl de Informática Gráfica*. 2010, pp. 207–216.

[San+14] Aitor Santamaría-Ibirika et al. “Procedural approach to volumetric terrain generation”. In: *The Visual Computer: International Journal of Computer Graphics* 30.9 (2014), pp. 997–1007. DOI: 10.1007/s00371-013-0909-y.

[SC04] Amit Shesh and Baoquan Chen. “SMARTPAPER: An Interactive and User Friendly Sketching System”. In: *Computer Graphics Forum* 23.3 (2004), pp. 301–310. DOI: 10.1111/j.1467-8659.2004.00761.x.

[Sch+06] R Schmidt et al. “ShapeShop: sketch-based solid modeling with BlobTrees”. In: *ACM SIGGRAPH 2006 Courses*. SIGGRAPH ’06. ACM, 2006. DOI: 10.1145/1185657.1185775.

[Sch+09] Ryan Schmidt et al. "Analytic Drawing of 3D Scaffolds". In: *ACM Transactions on Graphics* 28.5 (2009), 149:1–149:10. DOI: 10.1145/1661412.1618495.

[SI07] HyoJong Shin and Takeo Igarashi. "Magic canvas: interactive design of a 3-D scene prototype from freehand sketches". In: *Proceedings of Graphics Interface 2007*. GI '07. 2007, pp. 63–70. DOI: 10.1145/1268517.1268530.

[Sil+13] Pedro Brandão Silva et al. "Node-Based Shape Grammar Representation and Editing". In: *Proceedings of PCG 2013 - Workshop on Procedural Content Generation for Games, co-located with the Eighth International Conference on the Foundations of Digital Games*. 2013.

[Sil+15] Pedro Brandão Silva et al. "Procedural Content Graphs for Urban Modeling". In: *International Journal of Computer Games Technology* 2015 (2015), pp. 1–15. DOI: 10.1155/2015/808904.

[Sil10] Pedro Brandão Silva. "Modelação Procedimental para Desenvolvimento de Jogos de Computador". MA thesis. Faculdade de Engenharia da Universidade do Porto, 2010.

[SM15] Michael Schwarz and Pascal Müller. "Advanced procedural modeling of architecture". In: *ACM Transactions on Graphics* 34.4 (2015), 107:1–107:12. DOI: 10.1145/2766956.

[Sme+09] Ruben M. Smelik et al. "A Survey of Procedural Methods for Terrain Modelling". In: *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation*. 2009.

[Sme+11] Ruben M. Smelik et al. "A declarative approach to procedural modeling of virtual worlds". In: *Computers & Graphics* 35.2 (2011), pp. 352–363. DOI: 10.1016/j.cag.2010.11.011.

[Sof07] Introversion Software. *Procedural Content Generation*. http://www.gamecareerguide.com/features/336/
2007.

[SS05] S Stachniak and W Sturzlinger. "An Algorithm for Automated Fractal Terrain Deformation". In: *In Proceedings of Computer Graphics and Artificial Intelligence*. 2005, pp. 64–76.

[SS08] Ryan Schmidt and Karan Singh. “Sketch-Based Procedural Surface Modeling and Compositing Using Surface Trees”. In: *Computer Graphics Forum* 27.2 (2008), pp. 321–330. DOI: 10.1111/j.1467-8659.2008.01129.x.

[SSD06] Tevfik Metin Sezgin, Thomas Stahovich, and Randall Davis. “Sketch based interfaces: early processing for sketch understanding”. In: *ACM SIGGRAPH 2006 Courses*. 2006, pp. 1–8. DOI: 10.1145/971478.971487.

[ST14] Ruben M. Smelik and Tim Tutenel. “A Survey on Procedural Modelling for Virtual Worlds”. In: *Computer Graphics Forum* 33.6 (2014), pp. 31–50. DOI: 10.1111/cgf.12276.

[Šta+08] Ondrej Št’ava et al. “Interactive terrain modeling using hydraulic erosion”. In: *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’08. 2008, pp. 201–210.

[Šta+10] O. Št’ava et al. “Inverse Procedural Modeling by Automatic Generation of L-systems”. In: *Computer Graphics Forum* 29.2 (2010), pp. 665–674. DOI: 10.1111/j.1467-8659.2009.01636.x.

[Sun+02] Jing Sun et al. “Template-based generation of road networks for virtual city modeling”. In: *Proceedings of the ACM symposium on Virtual reality software and technology*. VRST ’02. 2002, pp. 33–40. DOI: 10.1145/585740.585747.

[the04] .theprodukkt. *Kkrieger*. http://www.theprodukkt.com/kkrieger. 2004.

[Tut+08] Tim Tutenel et al. “The role of semantics in games and simulations”. In: *Computers in Entertainment* 6.4 (2008), 57:1–57:35. DOI: 10.1145/1461999.1462009.

[Tut+09] Tim Tutenel et al. “Using Semantics to Improve the Design of Game Worlds”. In: *AIIDE - 5th Conference on Artificial Intelligence and Interactive Digital Entertainment*. 2009, pp. 100–105.

[Tut+11a] Tim Tutenel et al. “Generating Consistent Buildings: A Semantic Approach for Integrating Procedural Techniques”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011), pp. 274–288. DOI: 10.1109/TCIAIG.2011.2162842.

[Tut+11b] Tim Tutenel et al. “Procedural filters for customization of virtual worlds”. In: *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. PCGames ’11. 2011, 5:1–5:8. DOI: 10.1145/2000919.2000924.

[WH94] L R Williams and A R Hanson. “Perceptual completion of occluded surfaces”. In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, pp. 104–112. DOI: 10.1109/CVPR.1994.323803.

[Wik10] Procedural Content Generation Wiki. *What PCG Is*. http://www.pcg.wikidot.com/what-pcg-is. 2010.

[Won+03] Peter Wonka et al. “Instant architecture”. In: *ACM Transactions on Graphics*. SIGGRAPH ’03 22.3 (2003), pp. 669–677. DOI: 10.1145/882262.882324.

[You09] Shamus Young. *Fuel: Defining Procedural*. http://www.shamusyoung.com/twentysidedtale/?p=5134. 2009.

[YS12] Qizhi Yu and Anthony Steed. “Example-based Road Network Synthesis”. In: *Eurographics - Short Papers*. 2012, pp. 53–56. DOI: 10.2312/conf/EG2012/short/053-056.

[YSP05] Chen Yang, Dana Sharon, and Michiel van de Panne. “Sketch-based modeling of parameterized objects”. In: *ACM SIGGRAPH 2005 Sketches*. SIGGRAPH ’05. 2005, p. 89. DOI: 10.1145/1187112.1187219.

[Zha+13] Hao Zhang et al. “Layered analysis of irregular facades via symmetry maximization”. In: *ACM Transactions on Graphics* 32.4 (2013), p. 1. DOI: 10.1145/2461912.2461923.

[ZHH96] Robert C Zeleznik, Kenneth P Herndon, and John F Hughes. “SKETCH: an interface for sketching 3D scenes”. In: *ACM SIGGRAPH 2006 Courses*. SIGGRAPH ’06. 1996, pp. 163–170. DOI: 10.1145/1185657.1185770.

[Zho+07] Howard Zhou et al. “Terrain Synthesis from Digital Elevation Models”. In: *IEEE Transactions on Visualization and Computer Graphics* 13.4 (2007), pp. 834–848. DOI: 10.1109/TVCG.2007.1027.

[Zmu+13] Rene Zmugg et al. “Deformation-Aware Split Grammars for Architectural Models”. In: *2013 International Conference on Cyberworlds*. IEEE, 2013, pp. 4–11. DOI: 10.1109/CW.2013.11.