



## **Terms and Conditions of Use of Digitised Theses from Trinity College Library Dublin**

### **Copyright statement**

All material supplied by Trinity College Library is protected by copyright (under the Copyright and Related Rights Act, 2000 as amended) and other relevant Intellectual Property Rights. By accessing and using a Digitised Thesis from Trinity College Library you acknowledge that all Intellectual Property Rights in any Works supplied are the sole and exclusive property of the copyright and/or other IPR holder. Specific copyright holders may not be explicitly identified. Use of materials from other sources within a thesis should not be construed as a claim over them.

A non-exclusive, non-transferable licence is hereby granted to those using or reproducing, in whole or in part, the material for valid purposes, providing the copyright owners are acknowledged using the normal conventions. Where specific permission to use material is required, this is identified and such permission must be sought from the copyright holder or agency cited.

### **Liability statement**

By using a Digitised Thesis, I accept that Trinity College Dublin bears no legal responsibility for the accuracy, legality or comprehensiveness of materials contained within the thesis, and that Trinity College Dublin accepts no liability for indirect, consequential, or incidental, damages or losses arising from use of the thesis for whatever reason. Information located in a thesis may be subject to specific use constraints, details of which may not be explicitly described. It is the responsibility of potential and actual users to be aware of such constraints and to abide by them. By making use of material from a digitised thesis, you accept these copyright and disclaimer provisions. Where it is brought to the attention of Trinity College Library that there may be a breach of copyright or other restraint, it is the policy to withdraw or take down access to a thesis while the issue is being resolved.

### **Access Agreement**

By using a Digitised Thesis from Trinity College Library you are bound by the following Terms & Conditions. Please read them carefully.

I have read and I understand the following statement: All material supplied via a Digitised Thesis from Trinity College Library is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of a thesis is not permitted, except that material may be duplicated by you for your research use or for educational purposes in electronic or print form providing the copyright owners are acknowledged using the normal conventions. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone. This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

# **Procedural Modelling of Urban Environments**

**Brian Cullen**

June 21, 2012

A thesis submitted to the University of Dublin, Trinity College in candidacy  
for the degree of Doctor of Philosophy in Computer Science.



## Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work. I agree that Trinity College Library may lend or copy this thesis upon request.



thesis 9869

# **Procedural Modelling of Urban Environments**

Publication No. \_\_\_\_\_

Brian Cullen, Ph.D.

University of Dublin, Trinity College , 2012

Supervisor: Carol O'Sullivan

# Abstract

Procedural modelling of urban environments has become an important topic in computer graphics. With the ever increasing demand for larger and more realistic content in games and movies, the time and cost to model urban content by hand is becoming unfeasible. Apart from the entertainment industry, large urban models are also desired for urban planning applications and emergency response training.

Despite efforts to simplify control over procedural systems, writing production rules still remains a difficult task. This is due to the trial and error approach a user must endure, which can be frustrating and unintuitive. Entire production systems must be written before results can be visible, while intended local changes could result in unintended global alterations in the model.

This research aims to support users with the task of writing production systems for procedural generation of building models. We first explore methods to simplify the manual authoring of production grammars, while addressing the lack of realistic variation that can result from using such systems. We demonstrate how real-time procedural generation can be used in conjunction with existing urban models to drastically increase the area of visible content. We then explore the inverse problem of automatically inferring production grammars from photographs. This amounts to detecting structural relationships between building parts by utilising image segmentation and symmetry detection algorithms. We explore the use of hierarchical symmetries to generate hundreds of varying building hybrids from a small set of example images.

## Relevant Publications

1. Cullen B., O'Sullivan C.: Symmetry Hybrids. In *Proceedings of the International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging(CAe'11)*. ACM, New York, USA, pp.33–38, 2011
2. Cullen B., O'Sullivan C.: A Caching Approach to Real-Time Procedural Generation of Cities from GIS Data. *Journal of WSCG*, 19(1–3), pp. 119–126, 2011
3. Cullen B., O'Sullivan C: Procedural Modelling from GIS Data. *IRLOGI GIS Conference Presentations*, 2009

# Acknowledgments

I wish to express my sincerest thanks and gratitude to my supervisor, Carol O'Sullivan. I am hugely grateful for her invaluable input and advice that made this work possible.

I would also like to thank my colleagues, Yann Morvan, Eric Risser, Anton Gerdelan and Fintan McGee, for their numerous suggestions over the past three years.

Finally, I owe the most gratitude to my family, Jackie, Frank and Ciara, for all of their support and encouragement.

BRIAN CULLEN

*University of Dublin, Trinity College*

*June 2012*



# Contents

|                                                          |             |
|----------------------------------------------------------|-------------|
| <b>Abstract</b>                                          | <b>iii</b>  |
| <b>Acknowledgments</b>                                   | <b>vi</b>   |
| <b>List of Figures</b>                                   | <b>x</b>    |
| <b>Acronyms</b>                                          | <b>xvii</b> |
| <b>Chapter 1 Introduction</b>                            | <b>1</b>    |
| 1.1 Motivation . . . . .                                 | 3           |
| 1.2 Research Question . . . . .                          | 5           |
| 1.3 Scope . . . . .                                      | 7           |
| 1.4 Contributions . . . . .                              | 9           |
| 1.5 Implementation . . . . .                             | 10          |
| 1.6 Summary Of Chapters . . . . .                        | 11          |
| <b>Chapter 2 Background</b>                              | <b>13</b>   |
| 2.1 Foundation Grammars . . . . .                        | 14          |
| 2.2 Procedural Building Generation . . . . .             | 15          |
| 2.2.1 Rule Based Generation . . . . .                    | 15          |
| 2.2.2 Real-Time Generation . . . . .                     | 19          |
| 2.2.3 Modelling Roofs . . . . .                          | 20          |
| 2.2.4 Modelling Interiors . . . . .                      | 22          |
| 2.3 Procedural City Layouts . . . . .                    | 24          |
| 2.3.1 Interactive Generation of Urban Layouts . . . . .  | 25          |
| 2.3.2 Simulation Based Urban Layout . . . . .            | 27          |
| 2.4 Inverse Procedural Modelling . . . . .               | 28          |
| 2.4.1 Understanding Images of Building Facades . . . . . | 29          |
| 2.4.2 Detecting Structure . . . . .                      | 32          |
| 2.4.3 Generating Grammar . . . . .                       | 35          |

|                                               |                                          |           |
|-----------------------------------------------|------------------------------------------|-----------|
| 2.4.4                                         | Towards Structured Hybrids . . . . .     | 37        |
| 2.5                                           | Discussion . . . . .                     | 38        |
| <b>Chapter 3 Procedural Modelling</b>         |                                          | <b>40</b> |
| 3.1                                           | Grammars . . . . .                       | 41        |
| 3.1.1                                         | L-System . . . . .                       | 41        |
| 3.1.2                                         | Shape Grammars . . . . .                 | 42        |
| 3.1.3                                         | Split Grammar . . . . .                  | 43        |
| 3.1.4                                         | CGA Grammars . . . . .                   | 44        |
| 3.2                                           | Building Generation . . . . .            | 46        |
| 3.2.1                                         | Object Oriented Buildings . . . . .      | 47        |
| 3.2.2                                         | Texture Variations . . . . .             | 50        |
| 3.2.3                                         | Roof Generation . . . . .                | 53        |
| 3.3                                           | Data Extraction from GIS . . . . .       | 53        |
| 3.3.1                                         | Extracting Building Footprints . . . . . | 54        |
| 3.3.2                                         | Extracting Roads . . . . .               | 55        |
| 3.4                                           | Real-Time Generation . . . . .           | 58        |
| 3.4.1                                         | Parallel Geometry Cache . . . . .        | 59        |
| 3.4.2                                         | Parallel Building Generation . . . . .   | 59        |
| 3.4.3                                         | Level of Detail . . . . .                | 64        |
| 3.4.4                                         | Other Optimisations . . . . .            | 65        |
| 3.5                                           | Discussion . . . . .                     | 65        |
| <b>Chapter 4 Inverse Procedural Modelling</b> |                                          | <b>68</b> |
| 4.1                                           | Rectifying . . . . .                     | 69        |
| 4.2                                           | Subdividing Facades . . . . .            | 69        |
| 4.2.1                                         | Edge Based . . . . .                     | 71        |
| 4.2.2                                         | Region Based . . . . .                   | 73        |
| 4.3                                           | Symmetry Matching . . . . .              | 80        |
| 4.4                                           | Extracting Elements . . . . .            | 81        |
| 4.4.1                                         | Detailed Segmentation . . . . .          | 83        |
| 4.5                                           | Rule Derivation . . . . .                | 88        |
| 4.6                                           | Summary . . . . .                        | 89        |
| 4.7                                           | Discussion . . . . .                     | 92        |
| <b>Chapter 5 Symmetry Hybrids</b>             |                                          | <b>93</b> |
| 5.1                                           | Symmetry Detection . . . . .             | 94        |
| 5.1.1                                         | Transformation Voting . . . . .          | 94        |

|                                             |                                        |            |
|---------------------------------------------|----------------------------------------|------------|
| 5.1.2                                       | Clustering . . . . .                   | 95         |
| 5.1.3                                       | Regularly repeating elements . . . . . | 97         |
| 5.1.4                                       | Hierarchical Symmetries . . . . .      | 100        |
| 5.1.5                                       | Grouping . . . . .                     | 102        |
| 5.2                                         | Generating Hybrids . . . . .           | 104        |
| 5.2.1                                       | Hybridising Symmetry Tree . . . . .    | 105        |
| 5.2.2                                       | Production System . . . . .            | 107        |
| 5.2.3                                       | Conditional Dependence . . . . .       | 109        |
| 5.3                                         | Results . . . . .                      | 111        |
| 5.3.1                                       | Abstract Patterns . . . . .            | 111        |
| 5.3.2                                       | Indexing Schemes . . . . .             | 113        |
| 5.3.3                                       | Building Facades . . . . .             | 116        |
| 5.4                                         | Discussion . . . . .                   | 116        |
| <b>Chapter 6 Conclusion and Future Work</b> |                                        | <b>119</b> |
| 6.1                                         | Contributions . . . . .                | 119        |
| 6.1.1                                       | Building Generation . . . . .          | 119        |
| 6.1.2                                       | Inverse Procedural Modelling . . . . . | 120        |
| 6.1.3                                       | Symmetry Hybrids . . . . .             | 120        |
| 6.2                                         | Conclusion . . . . .                   | 122        |
| 6.3                                         | Future Work . . . . .                  | 122        |
| <b>Bibliography</b>                         |                                        | <b>126</b> |

# List of Figures

|     |                                                                                                                                                                                                                                                                                                      |    |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1.1 | Metropolis: (a) Crowds of virtual humans have realistic animations and behave in a natural way. (b) The simulation incorporates a traffic simulation of the areas surrounding the Trinity College Dublin campus. . . . .                                                                             | 6  |
| 1.2 | System Implementation . . . . .                                                                                                                                                                                                                                                                      | 12 |
| 2.1 | Procedural reconstruction of Pompeii, showcasing the power of the CGA <sup>1</sup> shape grammar (Müller et al. [MWH*06]). . . . .                                                                                                                                                                   | 17 |
| 2.2 | From top, left: buttress, dormer windows, flying buttress, bay windows, curved plan, eight faces meeting on a symmetrical footprint with a chimney, hipped roof, curved roof, a horizontal overhang, an overhanging gable, standard gable and interior dormer windows (Kelly et al. [KW11]). . . . . | 21 |
| 2.3 | Interior models produced by the method of Merrel et al. [MSK10]. . . . .                                                                                                                                                                                                                             | 23 |
| 2.4 | Terminology used in the literature to describe urban layouts. . . . .                                                                                                                                                                                                                                | 24 |
| 2.5 | Complex road geometry generated by Galin et al. [GPGB11] . . . . .                                                                                                                                                                                                                                   | 27 |
| 2.6 | Regularly repeating symmetries can be detected in complex architectural models. (Pauly et al. [PMW*08]) . . . . .                                                                                                                                                                                    | 34 |
| 2.7 | (a) Input model. (b) Symmetric regions marked in red. (c) A set of symmetric curves that cuts the model into two pieces yields a docking site that corresponds to (d) a replacement operation. (e) A similar construction yields insertions and deletions. Bokeloh et al. [BWS10] . . . . .          | 37 |
| 3.1 | Example road network generated using a simple L-System production set. . . .                                                                                                                                                                                                                         | 43 |
| 3.2 | Top Row: A simple shape grammar rule. Middle Row: A possible derivation sequence. Bottom Row: Possible end results. . . . .                                                                                                                                                                          | 44 |
| 3.3 | (a) An example set of split grammar rules. (b) The result of derivation using the rules in (a). Image from [WWSR03] . . . . .                                                                                                                                                                        | 45 |

<sup>1</sup>Computer Generated Architecture



|      |                                                                                                                                                                                                                                                                                                                                                    |    |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.4  | Building2 inherits from Building1, specifying how windowsills should be added. Building3 also inherits from Building1, adding a ledge to each floor. Example code to generate these buildings is provided in Listings 3.5 - 3.7. . . . .                                                                                                           | 48 |
| 3.5  | Listing for elements: Multiple meshes can be defined to add variation, and different levels of details can be specified for each mesh. . . . .                                                                                                                                                                                                     | 48 |
| 3.6  | Listing for Building1 of Figure 3.4: Production rules are grouped into semantic blocks that correspond to different subparts of the building. . . . .                                                                                                                                                                                              | 49 |
| 3.7  | Listing for Building2 of Figure 3.4: Building2 inherits everything from Building1 but specifies how windowsills should be added. . . . .                                                                                                                                                                                                           | 50 |
| 3.8  | (a) No variation between windows of the same type. (b) Windows vary with some open and some closed, adding more realism to the facade. (c) Adding weathering effects such as stains under the windows further increases realism. . . . .                                                                                                           | 50 |
| 3.9  | Algorithm for stain generation. Bosch et al. [BLR*11] . . . . .                                                                                                                                                                                                                                                                                    | 52 |
| 3.10 | (a) Roof generation from building footprint. (b) Straight skeleton of the building footprint. (c) Points along skeleton are equidistant from the shape's boundaries. Internal vertices can be raised by distance $d$ . (d) Determine roof faces. . . . .                                                                                           | 53 |
| 3.11 | Different roof types can be created from the straight skeleton: (a) Hipped roof. (b) Gable roof can be created by translating the red vertices to the boundary of the building's footprint. . . . .                                                                                                                                                | 54 |
| 3.12 | Extracting building footprints from GIS data (left). Layer containing buildings is first chosen by the user (middle), while buildings are then extracted by finding loops in the data (right). . . . .                                                                                                                                             | 55 |
| 3.13 | (a) Road data is represented in the GIS <sup>2</sup> data as a set of lines connected at vertices. (b) A more useful representation where each vertex has a set of directional edges leaving itself. . . . .                                                                                                                                       | 55 |
| 3.14 | (a) Roadside edges $s_1$ and $s_2$ are created a distance $w$ (the road width) to the left of their respective edges. The intersection point $p$ between these roadside edges determine the terrain polygon vertices. (b) Roadside edges may intersect the page boundary. (c) The result is a set of polygons describing the page terrain. . . . . | 56 |
| 3.15 | (a) Traversing anti-clockwise along the road edges gives a set of vertices that define the terrain polygons. (b) Similarly, road polygons can be created by simply traversing in the opposite direction. In both cases, polygons can be triangulated using existing methods. . . . .                                                               | 57 |
| 3.16 | Resulting road network: (a) Road meshes are placed lower than the terrain geometry to create curbs. (b) By utilising GIS data, road meshes can be created for the entire city. . . . .                                                                                                                                                             | 57 |

<sup>2</sup>Geographical Information System

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                          |    |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.17 | System Overview. GIS data along with building descriptors are stored on an external data server. Production rule sets are stored on the clients system and procedurally generated by the <i>Shape System</i> . The <i>Geometry Cache</i> determines what geometry is to be generated and at what level of detail. Building and road meshes are then generated as a final step before being sent to the <i>Rendering System</i> . . . . . | 58 |
| 3.18 | Algorithm for procedurally generating buildings in parallel. . . . .                                                                                                                                                                                                                                                                                                                                                                     | 59 |
| 3.19 | As the camera moves towards the geometry, new pages must be loaded. The pages are organised into a queue and processed in order of their distance to the camera. The buildings within each page should be shared among parallel executing threads. . . . .                                                                                                                                                                               | 60 |
| 3.20 | A comparison of results with and without the cache described in Section 3.4.1. The system with the cache has a much higher frame rate and less jerky movements of the camera. There was an average of 1,922 buildings in the scene at any time with 2,038 buildings created and destroyed over the distance. . . . .                                                                                                                     | 62 |
| 3.21 | Time in seconds to generate buildings with different levels of multithreading. On the dual core machine two threads yielded the best performance. . . . .                                                                                                                                                                                                                                                                                | 63 |
| 3.22 | (a) Lowest level of detail. (b) Mid level of detail. (c) Highest level of detail. . .                                                                                                                                                                                                                                                                                                                                                    | 64 |
| 3.23 | Time taken in seconds to generate 10,000 buildings at three different levels of detail. . . . .                                                                                                                                                                                                                                                                                                                                          | 64 |
| 3.24 | City generated in real-time. Procedural building models are integrated with user created 3D CAD <sup>3</sup> models of the Trinity College Campus. By utilising GIS data, buildings may be placed on top of satellite imagery. . . . .                                                                                                                                                                                                   | 66 |
| 4.1  | Procedural rule sets can be learnt from photographs of building facades. This involves using image processing techniques to segment architectural elements and assigning depth to image regions, thereby creating a 3D model. . . . .                                                                                                                                                                                                    | 68 |
| 4.2  | (a) Input facade image has perspective distortion. (b) Rectified facade image. .                                                                                                                                                                                                                                                                                                                                                         | 70 |
| 4.3  | Facades are hierarchically subdivided (Müller et al. [MZWVG07]). First the facade is divided into a set of tiles that can be used to find repetitions. Repetitions in these tiles can later be used to further subdivide the facade into its composing elements. . . . .                                                                                                                                                                 | 70 |

---

<sup>3</sup>Computer Aided Design



|      |                                                                                                                                                                                                                                                                                                                                                                                              |    |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 4.4  | Facades can be divided into floors and columns using edge detection. Blue lines correspond to the accumulation functions $Vert(x)$ and $Hor(y)$ respectively. Green lines correspond to a smoothed version of these functions, removing high frequency components. The red lines are a linear fit through these functions that can be used to subdivide the functions for splitting. . . . . | 72 |
| 4.5  | (a) Input image to be segmented. (b-c) Horizontal and vertical edge responses are considered separately when subdividing the facade in the horizontal and vertical directions respectively. . . . .                                                                                                                                                                                          | 73 |
| 4.6  | Facades are divided into floors and columns using edge detection. Suitable thresholds must be chosen to accurately divide the facade into floors and columns. (a) A bad choice of threshold gives many edge response due to textures in the image. (b) A better choice can remove these edges at the expense of degrading the outline of architectural elements. . . . .                     | 74 |
| 4.7  | The result of binary thresholding the input facade in Figure 4.7a. Black and white correspond to the foreground and background regions respectively. (a) Much of the wall texture is segmented with the windows. (b) Thresholding on the red image channel alone gives better results. . . . .                                                                                               | 75 |
| 4.8  | Segmentation using k-means clustering for a choice of two, three and four clusters respectively. The greater the number of clusters the more detail that can be segmented from the background walls (depicted in white). . . . .                                                                                                                                                             | 77 |
| 4.9  | Binary quantisation algorithm. (a) Initial cluster contains all points. The cluster $C_n$ is split through its mean $\mu_n$ , perpendicular to its eigenvector $e_n$ with greatest eigenvalue as indicated by the purple line. (b) A new cluster $C_n$ is chosen with the greatest eigenvalue. (c) The process repeats until $k$ clusters have been created. . . . .                         | 79 |
| 4.10 | (a) Probability that pixels belong to the wall texture. White values indicated high probabilities. (b) Binary threshold of the probability image with threshold $T = 0.1$ . . . . .                                                                                                                                                                                                          | 80 |
| 4.11 | (a) Input facade image. (b) Matching tiles can be averaged to remove lighting variation and noise. . . . .                                                                                                                                                                                                                                                                                   | 81 |
| 4.12 | Facade tiles are iteratively subdivided into <i>regions</i> according to their gradient image. These <i>regions</i> are then recursively subdivided. Müller et al. [MZWVG07]                                                                                                                                                                                                                 | 81 |
| 4.13 | Split candidate edges (cyan) are chosen according to their length and proximity with the image border. Some candidate edges cross other edges (white) and cannot be chosen. Facade regions are then iteratively subdivided by choosing the best split candidates (blue). Müller et al. [MZWVG07] . . . . .                                                                                   | 82 |



|      |                                                                                                                                                                                                                                                                                                                                                                                                                                              |    |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 4.14 | Binary segmentation using the Gaussian mixture model approach from Section 4.2.2 fails to segment facade elements with similar colour distribution to the wall. An approach that takes gradient information into account is required for an accurate segmentation. . . . .                                                                                                                                                                   | 83 |
| 4.15 | (a) Graph is constructed from image. N-links are defined between neighbouring pixels (black edges) and T-links connect the pixel nodes to a background (source) and foreground (sink) nodes. Weights are assigned for both N-links and T-links, specifying an energy function for the segmentation. (b) An optimal segmentation can be found as a minimum cut $C$ through the graph, separating the background and foreground nodes. . . . . | 84 |
| 4.16 | (a) Initial graph with source node $T$ and sink node $S$ . Edge weights $w_e$ are given adjacent to each edge. (b) After calculation of max-flow, the min-cut is the set of edges with $w_e = 0$ . . . . .                                                                                                                                                                                                                                   | 86 |
| 4.17 | Facade elements segmented using GrabCut. Left: Input facade images. Right: Can achieve very accurate segmentations even when facade elements resemble the wall texture. . . . .                                                                                                                                                                                                                                                              | 87 |
| 4.18 | (a) Segmentation results in the window and windowsill being joined together. (b) User can select a split point to separate the two facade elements. . . . .                                                                                                                                                                                                                                                                                  | 88 |
| 4.19 | (a) Input facade image. (b) Extracted contour image can be used with the splitting technique of Müller et al. [MZWVG07]. . . . .                                                                                                                                                                                                                                                                                                             | 89 |
| 4.20 | Summary of segmentation method: (1) Rectify input photograph. (2) Compute background GMM <sup>4</sup> and subdivide the facade into tiles. (3) Perform the GrabCut segmentation technique on each facade tile. (4) Iteratively split along foreground contours using the method of Müller et al. [MZWVG07]. . . . .                                                                                                                          | 90 |
| 4.21 | Results of applying our segmentation method to three different facades. Left: Input rectified photographs. Middle: Segmented facade elements. Right: Resulting 3D facade models after manual depth adjustment. Shaders were added to the windows to provide reflection effects. . . . .                                                                                                                                                      | 91 |
| 5.1  | (a) The flower has a 5-fold rotational symmetry. (b) The butterfly has reflectional symmetry. (c) Bricks have translational symmetry. . . . .                                                                                                                                                                                                                                                                                                | 94 |
| 5.2  | (a) Shapes are represented with polar coordinates $(\alpha, d)$ and orientation $\beta$ . (b) Pairwise transformations between shapes are represented with a 3-tuple $t_{ab} = \langle \alpha, d, \beta \rangle$ . (c) Each transformation corresponds to one point in transformation space. . . . .                                                                                                                                         | 95 |

---

<sup>4</sup>Gaussian mixture model

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |     |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 5.3  | In a regular structure, transformations form an accumulative pattern. Image from Pauly et al. [PMW*08] . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                           | 97  |
| 5.4  | (a) The fish approximate a grid pattern. (b) Cluster layout reflects this pattern in transformation space. Each point represents a transformation between two fish (translation in Cartesian coordinates). (c) The optimal grid pattern with basis vectors $g_1$ and $g_2$ that describes the layout of the fish, i.e., how transformation clusters would be organised if the fish had perfect symmetry. . . . .                                                                                                                                                           | 98  |
| 5.5  | (a) Patterns can be composed of hierarchical symmetry relations between shapes. The blue shapes form cyclic groups of $45^\circ$ which in turn form a cyclic group of $30^\circ$ . (b) Grid pattern appears in transformation space (projected onto rotation subspace) with the x-axis giving the rotation component $\beta$ (see Figure 5.2) of transformations in degrees. However, symmetry grids from different hierarchical levels are overlaid on top of each other, giving the appearance of a $15^\circ$ dominant rotational symmetry, which is incorrect. . . . . | 100 |
| 5.6  | Larger transformations are composed of chains of smaller transformations within the same symmetry group. Here, transformation $t_2$ is equivalent to performing $t_1$ three times. . . . .                                                                                                                                                                                                                                                                                                                                                                                 | 101 |
| 5.7  | (a) Lowest level symmetry is detected first, creating a node in the symmetry tree. (b) Shapes are grouped into patches according to the detected symmetry. The next symmetry level is detected and the tree grows from the bottom up. (c) The process keeps iterating until only one patch is left, defining the root node of the tree. . . . .                                                                                                                                                                                                                            | 102 |
| 5.8  | Two cases for grouping shapes: (a) Shapes must be grouped into their individual symmetry groups before being grouped together. (b) Shapes must be grouped together before being grouped into the same symmetry group. . . . .                                                                                                                                                                                                                                                                                                                                              | 103 |
| 5.9  | Precedence rules for grouping by symmetry. $A_i$ 's are grouped before being grouped with $B$ . Wang et al. [WXL*11] . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                             | 104 |
| 5.10 | Iterative grouping of facade representation. (a) Precedence rules result in overlapping groups at stage 4. (b) By including the overlap exception constraint, overlapping no longer occurs, resulting in a more suitable grouping for split grammars. . . . .                                                                                                                                                                                                                                                                                                              | 105 |
| 5.11 | Input patterns are both composed of a dominant rotational symmetry. Their corresponding symmetry trees are very similar. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                           | 105 |
| 5.12 | Algorithm to control the amount of variation a hybrid can exhibit. The subroutine <i>find</i> searches for a subtree within a bigger <i>tree</i> , returning the matching nodes and matching score. . . . .                                                                                                                                                                                                                                                                                                                                                                | 107 |



|      |                                                                                                                                                                                                                                                                            |     |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 5.13 | (a–b) Graphs before hybridisation. (c) Subtree matching with a matching depth of 2. (d) Subtree matching with a matching depth of 1. Broken red line indicates where nodes have been connected as siblings. . . . .                                                        | 107 |
| 5.14 | (a) Input patterns. (b) Subtree matching depth set to 2. Some variation is possible while preserving the input structure. (c) Subtree matching depth set to 1. Much more variation is possible as less of the original structure is preserved. .                           | 108 |
| 5.15 | Object Dependencies . . . . .                                                                                                                                                                                                                                              | 110 |
| 5.16 | Groups of elements are labelled from the bottom up. Prefixing a <i>H</i> represents a horizontal grouping and <i>V</i> represents a vertical grouping. . . . .                                                                                                             | 111 |
| 5.17 | Left: Input flowers to be hybridised. Outputs reflect the rotational symmetry of the petals. . . . .                                                                                                                                                                       | 112 |
| 5.18 | Left: Input ornamental designs to be hybridised. Many new output designs can be achieved (the background disk was added manually). . . . .                                                                                                                                 | 112 |
| 5.19 | Fractal patterns can result if symmetry nodes are matched between different hierarchy levels. . . . .                                                                                                                                                                      | 112 |
| 5.20 | Input sketches used to test the preservation of contextual relations in generated hybrids. . . . .                                                                                                                                                                         | 113 |
| 5.21 | Unique Index: Contextual relations between elements of the same group are preserved (e.g., a brown windowsill is only observed under a green cornice). .                                                                                                                   | 114 |
| 5.22 | According to Symmetry: Does not preserve relations between elements (e.g., a red windowsill can appear under a different window type than that which was observed in the inputs). Building elements are chosen based on their symmetry layout in the input images. . . . . | 114 |
| 5.23 | According to Symmetry and Unique Index: Preserve relationships between element types. . . . .                                                                                                                                                                              | 115 |
| 5.24 | Input facade images to be hybridised. Facades were segmented using the techniques presented in Chapter 4. Resulting hybrid building facades are illustrated in Figure 5.25. . . . .                                                                                        | 116 |
| 5.25 | Results for generating hybrid building facades from segmented photographs. The input photographs can be seen in Figure 5.24. Many new building facades can be created from just a few input images. . . . .                                                                | 117 |
| 5.26 | Contextual relations can be between elements on different branches of the symmetry tree. Balconies (blue) only appear between columns (green). . . . .                                                                                                                     | 118 |

# Acronyms

**CAD** Computer Aided Design

**CGI** Computer Generated Imagery

**L-Systems** Lindenmayer Systems

**CGA** Computer Generated Architecture

**FFD** Free Form Deformation

**GIS** Geographical Information System

**LOD** Level Of Detail

**pdf** probability density function

**GMM** Gaussian mixture model

**RJCMC** Reverse Jump Markov Chain Monte Carlo

**MCMC** Markov Chain Monte Carlo

**MRF** Markov Random Field

**SIFT** Scale Invariant Feature Transform

**ISM** Implicit Shape Model

**IK** Inverse Kinematics

**CPU** Central Processing Unit

**GPU** Graphics Processing Unit

**LIDAR** Light Detection And Ranging

**PCA** Principle Component Analysis

**NCC** Normalised Cross Correlation

# Chapter 1

## Introduction

**V**IRTUAL RECONSTRUCTION of real-world cities has become an important topic in Computer Science. There are numerous applications of virtual cities spanning multiple industries. Urban planners require detailed visualisations of the effects that potential developments will have on their surrounding environment. Such representations are important to see how a new development will look from different viewpoints in the city, how it will occlude the view of existing buildings as well how it will affect traffic congestion, pedestrian flow and the local ecosystem. These visualisations often incorporate detailed simulations of the urban populace including its socio-economic, transportation, land use and environmental policies. They are important for researchers and government agencies to view the effect such policies and infrastructure will have on the community, such as accessibility, property value and affordability. Health and safety organisations are concerned with how accessible a property is for emergency services in the event of a fire or other emergency, and how to prevent fires from spreading to neighbouring buildings. Often behavioural simulations are required to plan evacuation and containment procedures in the event of a natural disaster, virus outbreak or other catastrophe. All of these important applications require virtual models of a city's infrastructure.

Apart from these important real-world considerations, virtual environments are of great interest for the entertainment industry. Movies increasingly rely on CGI<sup>1</sup> for special effects that would otherwise be impossible to achieve. In many movies, virtually reconstructed cities are used to realistically render scenes of destruction from alien invasions to natural disasters, and often depict fully animated fantasy worlds. As advances in computer hardware bring more and more processing power at affordable costs for use in interactive home entertainment, larger and more detailed virtual environments are increasingly being demanded in computer games. Popular games such as *Grand Theft Auto*, *Fallout*, *Final Fantasy* and many more have huge virtual cities for players to explore. Online services such as Google Maps now provide 3D views

---

<sup>1</sup>Computer Generated Imagery



of reconstructed building models, thus helping users navigate through real world cities.

Modelling virtual cities for the above mentioned applications can be very costly and time consuming. Modelling applications like 3D Studio Max are often used to create them, and it can take days to realistically model a single building. LIDAR<sup>2</sup> scans can be utilised for automatic aerial and ground based reconstruction of buildings, but are very expensive and require scans from different viewpoints to be manually stitched together. Data from these scans is typically too dense for real-time rendering and contains noise and imperfections that need to be manually edited (Recent work on scan consolidation addresses this problem [ZSW\*10, NSZ\*10]). Advances in structure from motion techniques allow reconstruction from image sequences. This is a far cheaper method for reconstruction but the results are often noisy and inaccurate. Combinations of these approaches are often used together to reconstruct urban scenes and require significant manual effort to produce realistic, low resolution meshes for use with rendering systems.

Procedural generation addresses many problems associated with manually modelling large-scale realistic city models. Thousands of buildings can be generated in seconds from production grammars. The potential of procedural city modelling was first demonstrated commercially by Weta Digital in King Kong [Whio6]. New York City was procedurally reconstructed as it looked in 1933, only requiring landmark buildings to be manually created. Procedural city generation has therefore become a popular alternative for large scale city modelling. With the development of commercial tools like CityEngine [pro], procedural city generation is now integrated in the development pipeline of multiple movie and game studios, recently being used in the production of Pixar's Cars 2 to reconstruct London city [Fre11].

Successful procedural systems for city generation have been inspired by previous work for tree generation. Like trees, road networks are complex hierarchical structures containing many branches. Hence, street generation can be achieved using parametric production grammars similar to those used for tree generation. Typically, a user will write a production grammar capable of generating a random road network and fine tune the results by adjusting input parameters. Buildings are generated in a similar way, but require different production grammars that take into account the stricter spatial restrictions between their many composing parts. These production grammars must take into account contextual relations between a derivation and its environment so that the correct placement of windows and doors takes place, e.g., a window should not be occluded by a neighbouring building.

Despite efforts to simplify control over procedural systems, writing production rules still remains a difficult task. This is due to the trial and error approach a user must endure, which can be both frustrating and unintuitive. Entire production systems must be written before results can be visible, while intended local changes could result in unintended global alterations in the

---

<sup>2</sup>Light Detection And Ranging

model. Our research goal is to support users with the task of writing production systems for procedural generation of building models. We first explore methods to simplify the manual authoring of production grammars, while addressing the lack of realistic variation that can result from using such systems. We demonstrate how real-time procedural generation can be used in conjunction with existing urban models to drastically increase the area of visible content. We then explore the inverse problem of automatically inferring production grammars from photographs. This amounts to detecting structural relationships between building parts by utilising image segmentation and symmetry detection algorithms. We explore the use of hierarchical symmetries to generate hundreds of varying building hybrids from a small set of example images.

## 1.1 Motivation

One particular motivation for this research stems from *Metropolis*, a research project with the goal of simulating a populated Dublin City. *Metropolis* is inspired by an earlier project, *Virtual Dublin* [HO03], which was a 3D simulation of Dublin city, populated with virtual humans. *Metropolis* built on this system, introducing realistic, high resolution building models and plausible appearance and behaviour of virtual agents. The project covers a wide scope of research areas:

**Rendering** The system is required to render large crowds of people interacting with a large virtual environment in real-time. The environment is made up of high-resolution buildings, trees, lights and other detailed props, thereby providing a big challenge for real-time rendering. The system implements popular LOD<sup>3</sup> strategies and the use of imposters to maintain interactive rendering times. Much progress has been made towards improving rendering performance, specifically with rendering large crowds of varied animated agents [MLD\*08, MLH\*09, LPDO10].

**Animation** Since the environment is populated with large crowds of virtual humans, correct animation is crucial to the system. This involves many calculations to display the correct animation sequence for the trajectories of moving agents. It is important to avoid artefacts such as foot-sliding, sharp movements, unexpected animation transitions and other phenomena that are perceived as abnormal. These calculations are computationally expensive for large crowds of virtual agents, and there is ongoing research towards reducing this cost without introducing perceivable artifacts [PKM\*10, PHO11].

**Behaviour** Realistic behaviour among virtual agents is essential for any convincing simulation. This includes the path planning and interactions of pedestrians and traffic. Pedestrians

---

<sup>3</sup>Level Of Detail



should avoid walking into each other or wandering around in a random fashion. Groups of people must interact with each other in a believable way, such as, striking up conversations with each other and answering telephone calls. Motor vehicles should stick to the roads, obey traffic signals and calculate appropriate stopping distances to avoid collisions. Traffic simulation has been implemented on top of a physics engine to evaluate driving dynamics. Large vehicles may overturn if they turn too sharply or lose traction with the road if they suddenly stop. Significant research has been conducted towards believable crowd behaviour [EMO10, EPO11, EGO10], traffic simulation [GO10], and collision avoidance [PGO09].

**Audio** Apart from visual realism, audio is crucial to any simulation of an urban environment. This involves typical urban noises from traffic and pedestrians. Background noise must transition correctly between congested traffic areas and pedestrian only zones. Directional noise should complement the user's position and orientation to give a strong sense of presence in the environment. Surround sound systems are used in conjunction with noise maps to achieve this. Creating a realistic 3d sound environment is a difficult task and a large effort has been made to improve on existing methods [MRPS08, MKG\*10, SRMG10].

**Neuroscience** Large scale urban simulations are of interest to neuroscientists trying to understand how humans perceive their environment. Metropolis has provided a good platform for numerous psychophysical experiments that help understand the human mind. Such experiments have helped understand depth perception [KGBR10], how we evaluate emotions [MMON09], and how human behaviour can influence basic perceptions such as the sex of a person [MJH\*09, MO10].

**Modelling** A realistic urban simulation requires highly detailed building models offering a challenge for artists. Aerial LIDAR scans can assist professional artists in creating accurate urban models, sparking research to improve the useability and reduce the cost of capturing this data [MHC\*08, HCL09]. However, creating detailed urban models is still time-consuming and very expensive. Other methods for visualising areas of the city that are not yet modelled are being explored. Research has been conducted on the use of panoramic photographs, and how to smoothly transition between 3d models and photographic representations [MO09]. Our research explores the use of procedural modelling to fill in areas that are not yet modelled.

An illustration of the Metropolis simulation is given in Figure 1.1. Metropolis has detailed CAD models of the buildings within the Trinity College Dublin campus and surrounding areas. Creating new buildings to expand the simulation area is both time-consuming and expensive.

Our motivation lies in extending the simulation to cover a much larger area that would be uneconomical to model manually. Many buildings in surrounding areas have the same style of architecture, making them ideal candidates to be filled with procedural building models. Procedural buildings also offer a suitable backdrop for the city, giving the illusion of a much larger simulation. We believe procedural modelling could be used in conjunction with manually created CAD models to create a template city. The city could be incrementally refined by adjusting parameters or adding external models, until a plausible reconstruction is acquired.

## 1.2 Research Question

This research focuses on using procedural modelling to aid in the reconstruction of an existing city. The most successful approaches and industry standard for procedural city generation utilise production grammars, thus prompting us to specifically focus on them. Writing production rules to generate buildings can be very difficult. The resulting models cannot be seen until the entire production rule set is written, therefore requiring a trial and error process. This problem is particularly frustrating in the case of writing stochastic production rules, in which multiple different rules could be chosen for a single derivation step. It is therefore simpler to write multiple highly deterministic production rule sets to generate different styles of buildings. However, this approach results in highly redundant code and a severe lack of variation among output models.

This research aims to support users with the task of writing production systems for procedural generation of building models. Inverse procedural modelling is the goal of inferring production grammars from input examples. This still unsolved problem has become a topic of interest in recent work. We too will tackle this problem with particular regard to inferring production rules from photographs. Previous work has made progress towards this goal, but can only infer a deterministic production grammar for a given input (since no variation in structure can be observed in a single instance). Hence, there is still a difficulty in writing production grammars capable of generating output buildings with variation between them. Our research question can then be defined as follows:

*“How can the writing of procedural systems that model variation between architectural types be simplified, with particular regard to inferring stochastic production grammars from images?”*

Research towards answering this question needs to tackle several problems. Simplifying the writing of procedural systems covers both manual writing and inverse procedural modelling. Since photographs are by far the cheapest and most abundant source of urban data, we specifically focus on using them as exemplars for inverse procedural modelling. This calls for an





(a)



(b)

Figure 1.1: Metropolis: (a) Crowds of virtual humans have realistic animations and behave in a natural way. (b) The simulation incorporates a traffic simulation of the areas surrounding the Trinity College Dublin campus.

exploration of computer vision techniques to accurately segment architectural elements from photographs. We are interested in inferring stochastic production grammars from several input images, such that one production grammar can reproduce all input examples by exploiting structural similarities between them. We will show that by preserving these structural similarities it is possible to generate plausible hybrid variations that still resemble the inputs. Hence, a further breakdown of the questions we must answer is as follows:

1. How can we simplify the manual writing of production grammars that model different building styles?
2. How can we detect spatial relationships between building elements in images in order to infer a production grammar capable of generating a similar building?
3. Can a high level understanding of the contextual relationships between building elements be utilised to generate new plausible building styles?

We use *contextual relationships* in this sense to mean modelling the semantic identity of important architectural elements and the relative order, distances and symmetries between them, while preserving that context when altering rules.

### 1.3 Scope

In this section, we discuss the scope of the research presented in this thesis. We will identify specific areas of concern and outline directions of work that fall outside the scope of our research. For completeness, a full review of previous research related to procedural generation of cities will be given in Chapter 2. This will help identify how our research question fits into the body of previous work. We start by identifying two separate categories of work for procedural generation of cities:

1. Procedural generation of street layouts.
2. Procedural generation of building models.

The research described in this thesis is concerned with using procedural modelling as a tool to aid in the reconstruction of cities. Therefore, we are not concerned with generating new street layouts and will focus solely on generating building and road models from input data. We obtain building footprints and street topology from GIS data of Dublin City, and utilise this data for procedural generation. We are also not concerned with generating building interiors but we will review research towards this goal. The specific aspects of building generation that we are concerned with are as follows:



**Exterior Generation** This includes the generation of textured wall meshes, and placement of architectural elements, such as windows and doors. Following previous approaches, we do not try to procedurally generate complex architectural elements, but instead utilise externally created models. These can be high resolution meshes created in modelling applications such as 3d Studio Max, or simple textures extracted from photographs. We base our procedural generation on the CGA shape grammar as proposed by Müller et al. [MWH\*06]. This choice is based on the success of commercial applications, namely CityEngine [pro], which also utilise the CGA shape grammar.

**Roof Modelling** We are interested in modelling roof styles common to Dublin city. Since production grammars are unsuitable for generating complex roof models, we implement popular roof generation algorithms from the literature.

**Real-Time Generation** We are interested in real-time generation of procedural cities. We will consider techniques that can improve the generation times for buildings. Recent work has shown that huge speed increases can be achieved by evaluating simple rules on the GPU<sup>4</sup>. However, these approaches limit the complexity of generated buildings and limit further post processing. Therefore, we are only concerned with generating building geometry on the CPU<sup>5</sup> for processing with the standard rendering pipeline. Similarly, we do not focus on improving rendering speeds of generated buildings as rendering optimisation is a separate research topic.

We will explore inverse procedural modelling as a means to simplify the writing of production systems to generate buildings. Inverse procedural modelling is a broad term that covers inferring procedural rules from any type of input exemplar. Many data types could be used, including LIDAR scans, aerial photographs, geometric models and depth information captured from stereoscopic systems. Since street level photographs are the most common source of urban data, we limit ourselves to inferring production grammars from facade images. We will consider the following:

**Image Segmentation** In order to utilise facade images as a source of data for inverse procedural modelling, it is first necessary to detect architectural elements. This involves segmenting windows, doors and other building elements. We are not concerned with automatically labelling extracted elements and therefore implement a manual labelling approach. We will explore specialised techniques for segmenting facade images.

**Symmetry Detection** We will explore the use of symmetry detection to understand structural relationships between segmented architectural components. Since buildings are rigid

---

<sup>4</sup>Graphics Processing Unit

<sup>5</sup>Central Processing Unit

bodies, we are not concerned with recent progress on intrinsic symmetry detection. We explore the use of a standardised representation to describe the symmetry hierarchy of an arbitrary input pattern and consider building facades as a specialised case of this.

**Inferring Grammar** We will review previous approaches towards inverse procedural modelling. We will consider methods for inferring production systems from general patterns, as well as those specific to building facades. Previous approaches have focussed on inferring deterministic production rules from input examples. We are interested in inferring stochastic production systems that can produce many varying outputs from a few input exemplars. There has been recent interest in using texture synthesis to generate hybrid images from input exemplars [RHDG10]. However, these approaches do not model an understanding of the structural relationships between input components. As such they are unsuitable for a procedural representation, and do not fit in existing frameworks for procedural building modelling, hence, we do not consider them further.

## 1.4 Contributions

This thesis makes a number of contributions to the domain of procedural building generation. The major contributions are as follows:

1. We introduce the idea of *object-oriented building style sheets*, a new representation to simplify the manual writing of production rules for generating buildings. Object oriented building styles allow new building styles to inherit from previously written scripts. This reduces code redundancy by allowing the user to simply specify subsets of a building that need to be changed, and also improves the readability of production grammars.
2. We combine the use of *weathering simulation* and procedural generation to add more variation and improve the realism of generated buildings. The procedural system places semantic labels on building elements to tag them as stain generators.
3. We propose a *multi-state cache for real-time generation of buildings*. The cache exploits temporal coherency by distributing the load of building generation over multiple frames. Through a gradual generation process that balances memory and speed, we can achieve smoother loading of geometry pages compared to simple lazy generation.
4. We present a *new approach to segment facade images* that is compatible with previous approaches for inverse procedural modelling of building facades. Previous approaches are limited to segmenting rectangular facade elements and require fine tuning of thresholds on a per image basis. Our approach utilises graph cut methods to overcome these problems and thus obtains a detailed segmentation of building facade elements.



5. We tackle the problem of hierarchical symmetry detection and introduce a new algorithm to uncover a tree based representation of an input pattern's structure. Through the use of precedence rules we can ensure that similar patterns have very similar symmetry trees.
6. We present the *new concept of symmetry hybrids*. By preserving the similarities and exploiting the variation between tree based representations of input patterns, we can infer stochastic production grammars capable of generating hundreds of new varying hybrid patterns. We present building facades as a specialised case of this process.
7. We introduce preliminary work towards developing a *context-aware production grammar* capable of modelling dependencies between different branches on a derivation tree. By propagating variables between production rules, we can use conditional probabilities to model different contexts that need to be preserved.

## 1.5 Implementation

This section will discuss the implementation details and the technologies that are used. Figure 1.2 gives an overview of how the following systems fit together:

**Procedural System** The procedural system is written in the C++ language and controls all procedural generation. A parser was written so that CGA shape grammar rules may be inputted.

**Facade Segmenter** Facade images are segmented using computer vision algorithms. The system is written in C++. OpenCv (Open Source Computer Vision Library) is used whenever possible for image processing. Segmented building elements are manually labelled for further processing.

**Inverse Procedural System** The inverse procedural system detects symmetry relations between input shapes. Stochastic production grammars are inferred by detecting structural similarities between input exemplars. The system is written in MATLAB (Matrix Laboratory) to make use of its numerical libraries.

**Rendering Engine** The rendering engine is written in C++ and utilises Ogre (Object-Oriented Graphics Rendering Engine) for real-time rendering. Ogre allows abstraction from specific system rendering libraries (such as OpenGL or DirectX) through the use of high level classes. This simplifies the writing of cross-platform code. Ogre plugin libraries are used to control geometry paging and environment rendering.

**GIS Database** City layout information is extracted from GIS data. Building footprint polylines and the road network topology are stored in an external MySQL database.

## 1.6 Summary Of Chapters

This thesis will present our ongoing research related to procedural modelling of urban environments. Its structure is as follows:

**Chapter 2** reviews previous work for procedural generation of urban environments. We will explore state of the art methods for procedural building and street generation. We will discuss recent inverse procedural modelling techniques that could help with the writing of procedural systems for building generation. This will involve a detailed review of computer vision approaches for segmenting facade images, and symmetry detection algorithms that could help understand the underlying structure of a building.

**Chapter 3** will present our work towards real-time procedural reconstruction of Dublin city, and how procedural buildings may be combined with existing CAD models. We will present a new approach to reduce the overhead and redundancies of writing production grammars for building generation, and address issues regarding the realism of output models.

**Chapter 4** will present our work for quick and accurate facade segmentation, with minimal user interaction. Photographs are an abundant source of data that could be used to automatically infer production grammars. This greatly simplifies the writing of production grammars to generate buildings.

**Chapter 5** will present our work towards generating new content with structured variation from input examples. We will utilise symmetry detection algorithms to understand relations between facade elements. We will present an approach to utilise context in production grammars to aid in the generation of structured hybrids.

**Chapter 6** summarises our contributions and also suggests directions for future work.

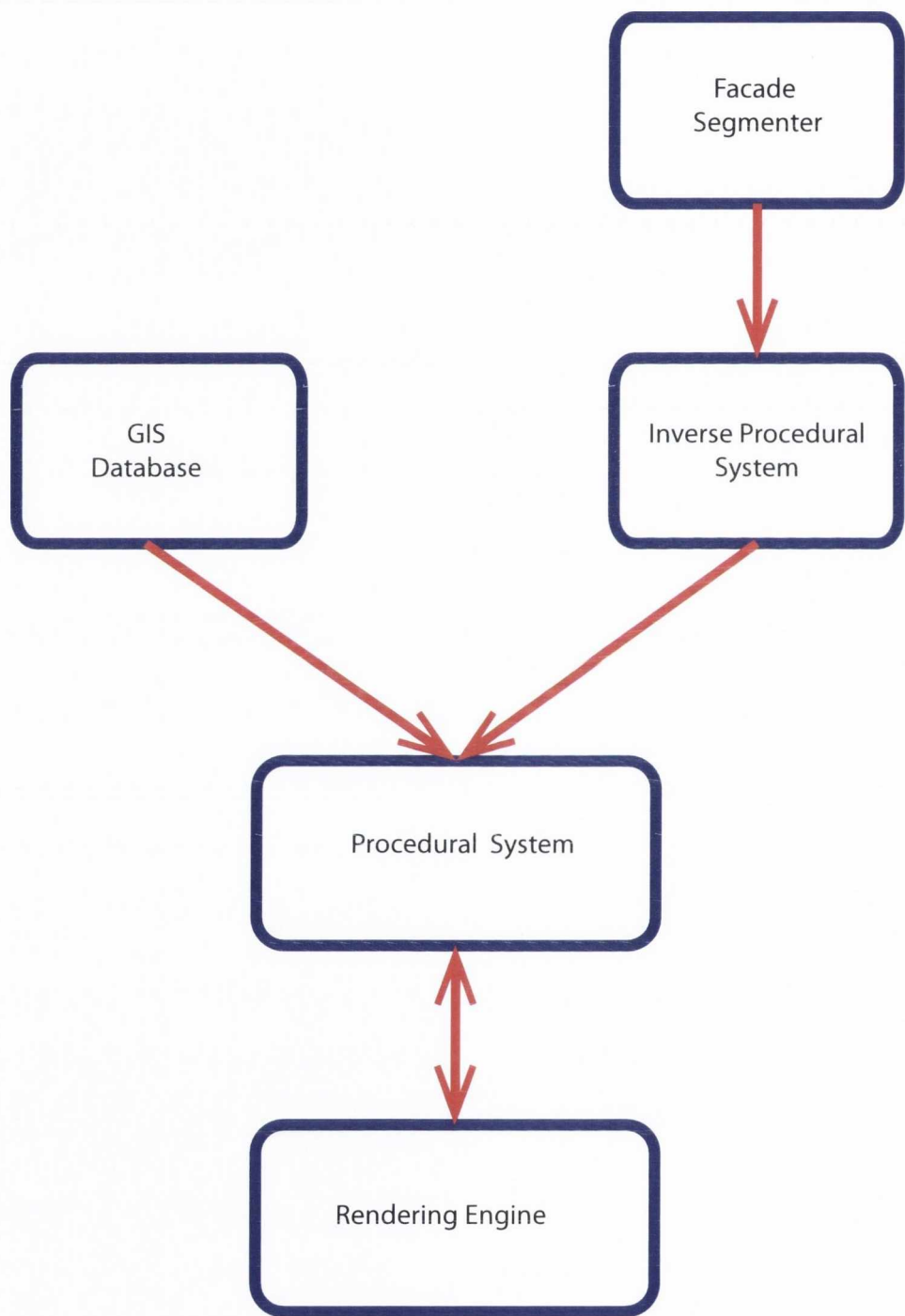


Figure 1.2: System Implementation



## Chapter 2

# Background

**P**ROCEDURAL GENERATION is a broad term referring to techniques regarding the creation of 3d-models and other content from rules or parameters governing some algorithm. It has been successfully applied to many different domains, including fractal visualisation [Man83], sound synthesis [Miro1], texture generation [Ebeo3], terrain generation [SKG\*09] and the modelling of the growth and morphology of organisms [PL90]. In recent years, there has been growing interest in the applicability of procedural techniques to generate urban content. Procedural generation of entire cities spans a broad set of algorithms, including the modelling of architecture, road networks, ecosystems and street layouts. Procedural generation can be differentiated from statistical approaches such as texture synthesis, in that procedural systems attempt to model an understanding of the underlying structure of a class of inputs, thereby giving parameterised control over generated outputs. However, this distinction is sometimes hard to make as some hybrid approaches use methods common to both procedural and example based data driven techniques. For simplicity, we will discuss these hybrid approaches as well as procedural modelling techniques and identify the differences.

This chapter will review previous work relevant to the procedural generation of urban environments. Generating a city can be broken down into two main tasks:

**Generation of street layouts** This process involves algorithms for generating new city layouts, including road networks and building footprints. These algorithms may be based on production systems, example layouts or a simulated growth of the city taking into account high level constraints. Section 2.3 will review work in this area.

**Generation of building models** Due to the different nature of the problem, generating building meshes usually requires a different approach to that of generating street layouts. Many successful approaches are based on production systems, with different methods for user control. We will review work in this area in Section 2.2.

A review of work in these areas will demonstrate that production grammars can be successfully utilised for the purpose of generating street layouts and building models. However, writing production grammars is still a difficult trial and error process that can be quite unintuitive. Inverse procedural modelling aims to simplify this process through the automatic inference of production grammars from exemplars. Photographs of urban scenes are by far the most readily available source of examples, therefore we will place a particular emphasis on inferring production grammars from images. This can be broken down into three steps:

1. The use of computer vision techniques to segment architectural elements from images of building facades.
2. Understanding the structure of a facade by detecting symmetry relations between the architectural elements of which it is composed.
3. Inferring a production grammar from this understanding of the structure.

In recent work, reviewed in Section 2.4, progress has been made towards solving these problems. Furthermore, we will review relevant work aimed towards the goal of inferring production grammars capable of generating hybrid building styles, a direction that has not yet been directly explored in the literature. We will start by reviewing two classes of production grammars that form the foundation of much of the reviewed work.

## 2.1 Foundation Grammars

Many procedural approaches to generating urban content have been inspired by previous work for generating trees and early work on formal grammars to describe architecture. Since these approaches form the foundation for many ideas common to both building and street layout generation, we will discuss them in detail, and will further elaborate upon them on in Chapter 3.1.

**L-Systems** L-Systems<sup>1</sup> are a context free grammar, first proposed by Lindenmayer for the modelling of multicellular organisms [Lin68]. They consist of a parallel string rewriting system, capable of modelling the growth of plants and trees [PL90]. They have become popular in the area of computer graphics after the introduction of graphical commands for visualisation of geometric structures [Pru86]. Due to their success in modelling complex structures like plants and trees, that are otherwise difficult to create manually, many extensions have been proposed to improve their modelling capabilities. One notable extension was the introduction of Open L-Systems by Měch and Prusinkiewicz [MP96],

---

<sup>1</sup>Lindenmayer Systems



allowing models to query their environment and previously generated models to affect subsequent derivation steps. Further control over the derivation of L-Systems can be accomplished by passing parameters to production rules [Han92]. Variation between derived models is achieved by introducing stochastic production rules, whereby multiple productions are specified for a non-terminal symbol, each with a given probability of being chosen.

**Shape Grammars** Stiny pioneered the idea of shape grammars [SG72, Sti80, Sti82] which can be used for generating complex shapes within a given spatial area. Shape grammars are generative grammars that operate on sets of lines, with simple rules specified for the replacement of shapes (an arrangement of lines under any Euclidean transformation) within a model. They can produce many variations for a simple set of rules and have been used for the construction and analysis of architectural designs [Duao2, Fle87, SM78, KE81]. Derivation can be controlled by introducing labels to specify where replacements are permitted to occur. However, shape grammars are difficult to implement on a machine because of the number of transformations that must be searched before a rule can be selected and applied.

## 2.2 Procedural Building Generation

In this section we will review popular techniques for the procedural generation of 3d building models. We will mainly review systems that employ production systems as they have been the most successful for generating realistic content.

### 2.2.1 Rule Based Generation

Detailed architectural models can be created using production systems, in which a modeller is required to write production rules describing the modelling steps to generate a desired architectural style. The strength of production systems lies in their ability to provide detailed descriptions and yet generate randomness in a structured way.

Parish et al. [PM01] introduced the idea of using L-Systems [PL90] to model architectural content. In this cornerstone paper, buildings are generated using a bounding volume as an axiom to the L-System and, through iterative scaling and branching operations, a number of building types can be defined. A unique aspect of this approach is that subsequent derivations produce more detailed buildings, allowing simple generation of different LOD. Facade details are created using a procedural texture, where hierarchical layers can be specified to control the scaling and regularity of windows. Facade style images (to control the grid like layout of facade elements) need to be supplied by the user and no depth is possible on facade details. The use of



L-Systems for generating buildings has many advantages such as automatic generation of LOD and easy generation of many variations.

However, due to their growth-like behaviour, L-Systems are not suitable for modelling spatially constrained structures such as building facade details. Shape grammars are more suitable to model such spatially constrained structures but are difficult to implement, thus motivating the creation of the *split grammar*, which is a modified version of shape grammars. Split grammars, as introduced by Wonka et al. [WWSR03], is a specialised set of shape grammars, which define a shape to be a primitive building block such as a cuboid or prism. A set of shape manipulation rules is defined, notably split and replace operations, which can iteratively subdivide a facade into its composing architectural elements. In addition, each shape is assigned a set of attributes that specify textual, positional, depth and other parameters that may be used to control the derivation. Results show building facades with detailed facade elements exhibiting depth.

Liu et al. [LWH\*05] take the opposite approach to split grammars and introduce a *constructive grammar* for generation of ancient Chinese architecture. Conversely to split grammars that start with a bounding box and iteratively subdivide building blocks to add more detail, a constructive grammar starts with an unbounded area and a set of building blocks that are progressively combined following a set of formalised rules. Arrangement of pillars and other architectural components follow a handbook of formalised constraints for the construction of Chinese architecture. Their approach works well but is restricted to a very specific type of architecture and is not applicable to western architectural styles.

Müller et al. [MWH\*06] expand on the idea of split grammars by developing the CGA shape grammar. The CGA shape grammar addresses some shortcomings of split grammars, returning to an L-System style string replacement grammar to easily specify production rules. Rather than controlling a LOGO style turtle, the CGA shape grammar works with parameterised symbols representing shape operations such as those in split grammars. One major contribution of the CGA shape grammar is the unification of commands that work on different dimensions. For instance, a volume can be decomposed into multiple 2D faces or 1D edges, thereby easily detailing subparts of the shape. Each shape is given a *scope* that represents its bounding box and a shape may query the environment to see if it is occluded by something else in the city, thus aiding the placement of windows and doors. Terminal shapes can be replaced with complex 3d-models, which produces highly realistic building models. The CGA shape grammar has been implemented in CityEngine [pro], and used to showcase virtual replicas of multiple cities around the world, and also to visualise reconstructions of ancient cities from archeological sites [MVW\*06], see Figure 2.1.

The CGA shape grammar is restricted to the generation of rectangular facade elements, due to the bounding-box representation of a shape's scope. Krecklau et al. [KPK10] get around this



Figure 2.1: Procedural reconstruction of Pompeii, showcasing the power of the CGA shape grammar (Müller et al. [MWH\*06]).

problem with the introduction of an FFD<sup>2</sup> [SP86] non-terminal object to allow the manipulation of arbitrary shapes. They adapt concepts from general purpose programming languages to modify the CGA shape grammar, creating  $G^2$  (Generalised Grammar).  $G^2$  allows the creation of non-terminal classes to encapsulate different modelling strategies for working with different shape types, i.e., boxes or FFD objects. Non-terminal symbols may be passed as parameters to production rules to allow the creation of abstract structure templates. Their generalised modelling language is demonstrated through the generation of detailed architecture and plant models.

Krecklau et al. [KK11a] introduce a way to procedurally model interconnected structures. They extend the CGA shape grammar to allow the specification of attaching points on a model's surface, that can be used as candidate end points for interconnections. The system can choose two candidate end points belonging to separate models, if they satisfy some relation. Using IK<sup>3</sup>, a chain of rigid objects is created that connects the two points. Using this technique, it is possible to generate bridges and catenaries that connect buildings together.

Another approach that can be used to procedurally model buildings that are structurally sound is described in Whiting et al. [WOD09]. Constraints that restrict the shape of a generated building are applied by calculating an infeasibility measure. They demonstrate how wall widths and supports are changed as the building's shape is altered. External forces can also be applied, thereby simulating the effects of earthquakes and other phenomena.

<sup>2</sup>Free Form Deformation

<sup>3</sup>Inverse Kinematics



## Controlling Derivations

Modifying buildings by editing production rules is still very difficult and requires an in-depth knowledge of how the subdivision rules are applied. This makes it difficult for modellers who are unfamiliar with programming to use production systems to create architecture. To tackle this problem, Lipp et al. [LWWo8] provide an interface for the visual editing of production rules. They allow the creation of complete rule sets without editing a single line of code, thereby combining the power of production systems with a traditional modelling interface.

Procedural systems may be written to generate highly complex models with large variations between possible derivations. There is a notable tradeoff between how much variation is possible and how closely a derived model resembles a desired structure. For instance, it is possible to write a production set to generate buildings or trees that resemble a desired shape. However, the same production set would need to be altered in order to generate models that resemble a new shape. Another approach is to use a parametric grammar in which many varying models may be derived by simply altering the input parameters. For highly complex models, there may be hundreds of parameters, thus making it an extremely difficult task to fine tune the shape of the desired output. Recent work of Talton et al. [TLL\*11] addresses this problem by considering a production system as a probabilistic inference problem on the set of all possible derivations of a given rule set. Given a high level sketch or voxel model of a desired output shape, they can automatically choose production rules and parameters that maximise the likelihood of generating a similar model. This is done using a RJMCMC<sup>4</sup> approach (Green's method [Gre95]) to reduce the optimisation to a sampling problem. Their algorithm is demonstrated using many different grammar types, which shows its applicability to popular modelling problems such as tree and architecture generation.

Another approach to control the derivation of complex models is to break the production system into a set of smaller production systems that communicate with each other. Beneš et al. [BvMM11] introduce guided procedural modelling, whereby the user draws a set of planar polygon guides that are linked together. Each guide is associated with its own L-System that can control the shape of derived models by means of clipping. When a model grows beyond its guide, a message is passed along a link and triggers the production system belonging to a connected guide. Rather than adjusting the parameters of a single complex production system, the user can break the production system into mutually exclusive subsystems, thus reducing the complexity. The guides allow interactive editing of derived models.

---

<sup>4</sup>Reverse Jump Markov Chain Monte Carlo



### 2.2.2 Real-Time Generation

Many previous approaches are too computationally expensive to be applied for real-time generation of cities, which is an important topic for many interactive applications. In particular, computer games could utilise real-time generation to build ever-growing 3d environments, thereby drastically increasing the size of explorable virtual worlds. Furthermore, recent work has shown that real-time generation can reduce the memory bottleneck of sending vertex buffer objects to the GPU. By evaluating compressed procedural descriptions directly on the GPU, standard vertex processing steps can be skipped entirely and thus improve rendering times for urban scenes.

Greuter et al. [GPSLo3] use a lazy generation technique to construct a potentially infinitely large city. Buildings are constructed when they are visible in the view frustum. The system seeds a random number generator based on the buildings' coordinates, thus allowing each building to maintain a persistent style. The cities do not look photo realistic and the road network is a simple grid pattern. In general, grammar based approaches such as the CGA shape grammar are not fast enough to generate thousands of detailed buildings in real-time. Recent approaches explore the use of fragment shaders to evaluate simplified versions of the CGA shape grammar, in order to exploit the parallelism of modern GPU architectures.

Ali et al. [AYRW09] introduce a method to improve rendering speeds for building facades. They introduce compressed facade displacement maps, which can encode facade structure in a highly compressed form. Facades are decompressed on the GPU and a fragment shader performs a ray casting algorithm to compute displacements. While their approach does not generate these displacement maps in real-time, it does take the first step towards evaluating facade structure on the GPU. A more recent approach that can evaluate split grammars on the GPU is given in Haegler et al. [HWA\*10]. Production rules based on CGA shape grammar are stored as textures and evaluated per-pixel using CUDA. Texture atlas IDs and displacement maps are created from these rules and evaluated on the shader. The results exhibit a highly improved memory footprint as intermediate geometry is not created, thereby reducing the bottleneck for memory transfer to the GPU. However, their approach does not allow the use of stochastic or parametrised production rules, and therefore requires a set of deterministic rules to be chosen for each facade.

Krecklau and Kobbelt [KK11b] present a similar approach that does allow the use of stochastic production rules. They map a set of procedural rules based on the CGA shape grammar to RGB colours, so that a rule set and its parameters can be represented with a texture. The creation of multiple texture layers gives the effect of transparency of windows. Inspired by the interior mapping technique of Van Dongen [J0008], the illusion of building interiors can be rendered. While they can quickly generate facades on the GPU, the results are however limited to the generation of flat facades. Visualisation of detailed geometry for facade elements is not

possible using these methods.

### 2.2.3 Modelling Roofs

Subdivision based grammars are not suitable for generating roofs for buildings with footprints of arbitrary shape. Algorithms to generate roofs are commonly based on the *straight skeleton* of a building footprint [AAAG95, AA96]. The straight skeleton is a shrunk version of a building's footprint where all points are equidistant to the footprint's boundary. It is calculated using a sweep plane approach, whereby a plane is swept up from the building's footprint, and continuously shrunk until it approximates the medial axis with straight line segments. Laycock et al. [LDo3] demonstrate that a variety of different roof styles can be derived from the straight skeleton. They implement the straight skeleton algorithm proposed in [FO98] to generate the common hipped roof. They also demonstrate that it is possible to create roofs of different styles such as mansard, gabel, gambrel and dutch hip, by modifying the straight skeleton. A technique for dealing with building extensions by use of rectilinear polygons is introduced. Generating roofs from the straight skeleton restricts each roof face to have the same angle. Weighted straight skeletons for polygon decomposition were proposed in Aurenhammer [Auro8], to allow weights to be applied to boundary edges, thereby controlling the rate of shrinking. This allows more complex roof shapes to be derived, although a working model was proposed for convex polygons only.

Soon describes an algorithm capable of modelling complex roofs common to east Asian buildings, such as temples and pagodas [Soo09]. This procedural approach is based on a mathematical formulation for specific roof styles. While complex and parameterised roof styles are possible, this approach is very constrained by the shape of building and cannot be generalised for arbitrary building shapes.

Kelly et al. present an approach to generate an entire building including complex roofs [KW11]. They argue that subdivision based production rules cannot correctly handle complex building shapes and suggest combining roof modelling and building generation in the same step. Inspired by the sweep-line approach of straight skeleton algorithms, they allow the user to control the geometry of an extruded building by assigning a user drawn guideline to each edge of the footprint. A plane is swept up from the footprint, which follows these profile lines to produce an extruded volume that has differing angles for each face. Procedural detail and external meshes can then be attached to this complex volume to produce complex building shapes with overhanging roofs. Figure 2.2 illustrates an assortment of roof types that can be created using this method.



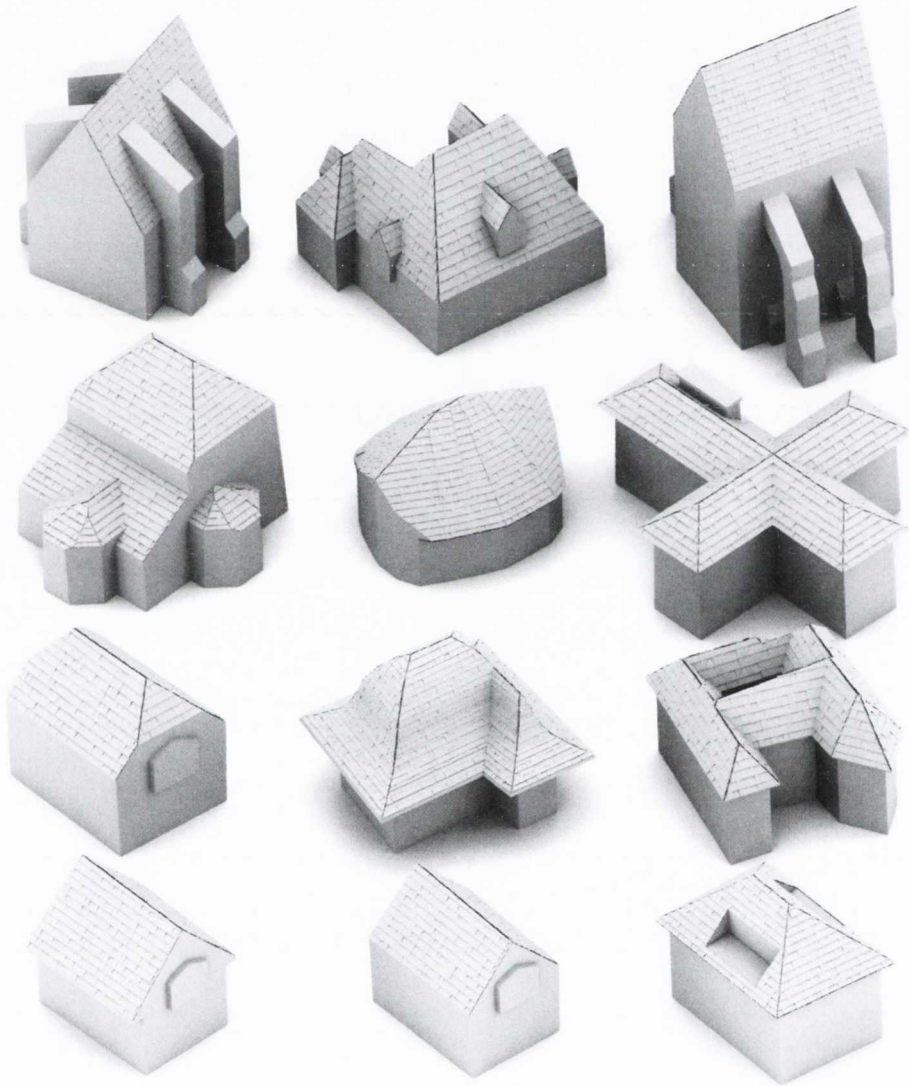


Figure 2.2: From top, left: buttress, dormer windows, flying buttress, bay windows, curved plan, eight faces meeting on a symmetrical footprint with a chimney, hipped roof, curved roof, a horizontal overhang, an overhanging gable, standard gable and interior dormer windows (Kelly et al. [KW11]).



### 2.2.4 Modelling Interiors

We have seen how procedural city models can be an effective method for generating large scale virtual cities. Building exteriors and street layouts with a realistic appearance can be generated from production grammars and other procedural techniques. However, little work has been done in recent years to generate realistic graphical models for the interiors of buildings. A working approach would have great potential for interactive applications such as computer games, as the level of exploration possible in a procedural city would be drastically increased.

Computer aided interior design has been a topic of interest in architectural design for many years. It is a process difficult to formalise due to the many different factors affecting human comfort. Many algorithms have been proposed with the goal of assisting architects to design concept layouts: see Shaviv [Sha74] for a review of early methods. A more recent approach is proposed by Michalek and Papalambros [MP02], where a genetic algorithm is used to optimise floor plan layout by searching over the space of possible connections between rooms. However, it is only possible to organise rectangular rooms using this method. Hahn et al. [HBW06a, HBW06b] present a system capable of modelling simple interiors in real-time. They divide the building into different floors and rooms with portals between them. However, none of these techniques take into account any user specifications or real world data. Due to the many different real-world factors that affect floor plan design, simple geometric algorithms are not sufficient to produce realistic results. Merrell et al. [MSK10] take a machine learning approach to the problem. Rather than trying to specify a set of criteria that must be satisfied for a valid floor plan, they take a data driven approach and train a Bayesian network on a corpus of real world architectural programs. The architectural programs directly describe room layouts, areas, aspect ratios and connectivity of rooms. From this corpus it is possible to learn semantic relationships between architectural plans and create a maximum likelihood formulation for generating new layouts from high level parameters. Examples of interior models created using their approach can be seen in Figure 2.3.

None of the above-mentioned floor plan generation techniques place furniture in rooms. Plausible arrangements of furniture are necessary to generate realistic rooms, and recently there has been growing interest in this goal. Fisher et al. [FSH11] present an approach to learn the structural relationships of a scene, specifically furniture layouts in rooms. Their algorithm then finds semantically similar scenes from a database. While the presented approach cannot synthesise new layouts, their representation is a step towards generating plausible scenes. A similar approach that can learn hierarchical relationships between furniture is described by Yu et al. [YYT\*11], where new furniture layouts can be synthesised by optimising a cost function. The cost function takes human factors, such as visibility and accessibility of different types of furniture, into account. Furniture is randomly placed in a room and a simulated annealing process minimises the cost function, thereby rearranging the furniture for a plausible layout. A percep-



Figure 2.3: Interior models produced by the method of Merrel et al. [MSK10].



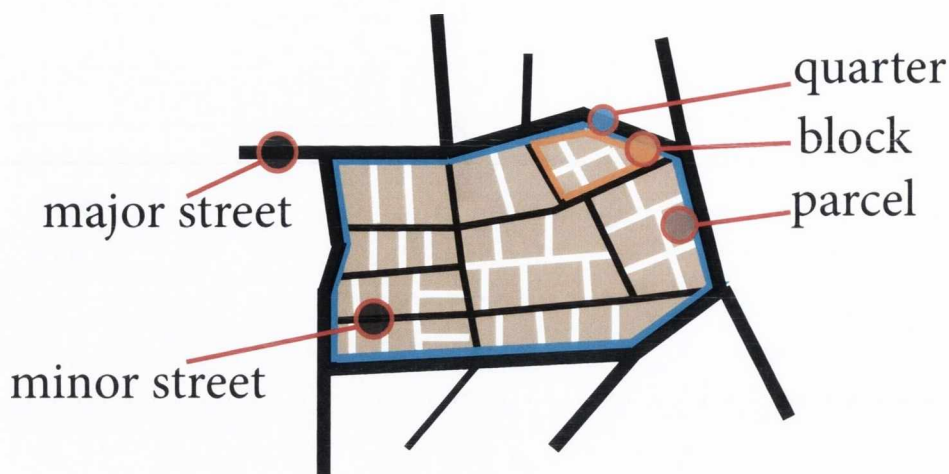


Figure 2.4: Terminology used in the literature to describe urban layouts.

tual experiment was conducted to validate the plausibility of the synthesised furniture layouts.

This section has reviewed procedural modelling techniques for the generation of building models. We have discussed previous work that can generate the exteriors and interiors of buildings. An outline of a building's footprint is required as input to most of the reviewed methods. Poly-lines that describe building footprints can be obtained from mapping data, or generated using procedural algorithms. In the next section, we will review previous work for generating new city layouts, including road networks and building footprints.

## 2.3 Procedural City Layouts

Layout generation involves the automatic layout of roads and placement of urban content that is crucial for generating an entire city. Urban planning applications require the ability to view changes to city layouts and to see the effect a proposed road network would have on traffic congestion. Using procedural techniques, such changes can be made interactively, which is a great improvement on manual systems. This section will discuss popular procedural techniques to generate new urban layouts and also to alter existing layouts. Procedural generation of urban layouts can be divided into two categories:

**Interactive Generation** involves techniques that offer detailed low level control over the generated urban layouts. These techniques are useful for reconstructing cities and often offer real-time editing capabilities. The goal of these techniques is to model urban layouts according to some preconceived artistic design of the city.



**Simulation Based Generation** involves high level approaches that simulate growing cities with socio-economic and ecological properties. These simulations model interactions between a city's growth and its population. High level parameters evolve over time, thereby modelling population distributions that affect industrial and residential usage of city areas and providing feedback to alter the geometry.

We will review previous work leading up to state of the art techniques from both categories. The terminology for city layouts used in the literature is illustrated in Figure 2.4. Typically, a city layout is generated by progressively subdividing the land to form a hierarchy. The terms commonly used to describe a city hierarchy are described as follows:

**Major and minor streets** divide the city into different regions of land. Major streets are much wider than minor streets.

**Parcels or lots** are small plots of land (belonging to some owner) on which one or more buildings may be built. Small roads may separate parcels.

**Blocks** are groups of parcels that are surrounded by minor streets.

**Quarters** are areas of the city surrounded by major roads.

**Highways** are roads that interconnect cities.

### 2.3.1 Interactive Generation of Urban Layouts

Parish et al. [PMo1] introduce the use of L-Systems to grow road networks in a similar way to branches on a tree. Modifying L-Systems, the concept of self-sensitivity is introduced so that road branches can determine if they intersect each other and create intersections if this is the case. Roads can be generated that follow patterns such as grid, radial, unconstrained or elevated layouts, according to observed networks in major cities. Street blocks are determined and divided into parcels of buildings, with rectangular footprints fitted in order to generate buildings. They demonstrate how procedural techniques are a powerful tool for the generation of entire cities by reconstructing the grid style layout of Manhattan. Parameters can be used to control the derivation but do not provide local control over specific segments of the city. Changing parameters amounts to regenerating the entire city which is time consuming and unpredictable. This is perhaps the most influential work in the area of procedural cities. However, it is difficult to fine tune the results because the variables do not give enough control over the road layout. Some of these issues were addressed by Kelly and McCabe [KM07] and Whelan et al. [WKMo8], who created the *Citygen* system that allows the real-time and interactive modification of roads and the tweaking of parameters. The user provides a height map

and layout of the roads and the system then automatically places buildings and other details. The system allows local regeneration for editing operations, thus removing the need to regenerate the entire city for each edit. Again, these methods are subject to the problem of multiple parameters that must be fine tuned, and offer limited editing capability over road and building generation.

Other approaches have used Voronoi diagrams [Aur91] and subdivision algorithms, to generate road templates that can be stitched together to form a network [SYBGo2]. Combinations of this approach with L-System generation was explored by Glass et al. [GMBo6] to duplicate the South African road networks, although not much control over generation was offered.

Chen et al. [CEW\*08] introduce the use of tensor fields to guide road network generation. They make the observation that in any part of a city, traffic flows in two major directions correspond to the major and minor eigenvectors of a symmetric second order tensor. The user controls road generation by editing the tensor fields using the interactive techniques discussed in [ZHT07]. Individual roads can be interactively edited by manipulating their vertices. However, the system does not provide the capability to move or rotate roads in a local way and does not create building parcels or footprints. This would require an alternative generation step described in other works, which would not allow any feedback between road editing and the placement of buildings.

Aliaga et al. [AVBo8] take a different approach to reconfiguring road networks. Using vector data of an existing road network, they form a graph to represent road intersections and parcels of land. Then using k-means clustering [Mac67], user-deformed parcels are replaced with similar parcels from elsewhere in the city. Land parcels can be classified by zones and the city is reconfigured whenever a region's zoning information is changed. Aerial images are operated on directly and no mechanism to synthesise new road networks is provided. In [ABVA08], the authors improve on this system to allow the synthesis of completely new areas of the city. Road intersections are given attributes such as the intersection angle, road importance and area of neighbouring parcels. The user can copy and paste intersection points into new parts of the city and the algorithm will generate roads in the same style as the original data. Cities with different road structures can then be blended together. However, the system does not take into account building footprint data and can warp buildings when stretching image areas to fit new parcels. These approaches take up to 3 hours to preprocess the data.

While much research has been conducted on procedural street generation, editing the layout of generated streets is still an open problem. Global changes to the road network destroys local edits. Local editing can unintentionally warp or destroy the structure of surrounding streets and buildings, thereby requiring local regeneration which is often difficult to control. Lipp et al. [LSWW11] present a new method to tackle this problem, where interactive editing of street networks allows the translation and rotation of individual roads and the merging of





Figure 2.5: Complex road geometry generated by Galin et al. [GPGB11]

different urban layouts. Their method ensures that edited layouts are always valid. They follow a layering approach inspired by image manipulation programs, where different urban layers are merged using a graph cut technique.

In general, procedural generation of street layouts does not take into account the terrain geometry when placing road networks. Roads that interconnect cities often traverse mountains and other harsh terrains. Galin et al. realistically generate roads, bridges and tunnels over complex terrain [GPMG10] by using an anisotropic shortest path algorithm to compute the path between two points that minimises the line integral of a cost function along the path. This cost function also takes into account the formation of procedural bridges and tunnels. The user can influence the chosen path by tuning cost functions for slope, curvature, vegetation and water. This process results in realistic road meshes that are procedurally created, incorporating tunnels and bridges to traverse complex terrain. An extension to model the hierarchical network of roads interconnecting cities was proposed in [GPGB11]. This is a fundamentally different problem to city road network generation. Hierarchies of roads are modelled, from highways to major then minor roads, using a graph based representation. As in [GPMG10] a cost based function of the line integral of the road is evaluated, but it now takes the traversal of towns and cities into account. Examples of the complex road geometry created using their approach can be seen in Figure 2.5.

### 2.3.2 Simulation Based Urban Layout

Behavioural modelling has long been of interest for the purpose of urban planning simulation. A complete review of simulation techniques for urban planning applications is outside the scope of this thesis. We refer the reader to Waddle and Ulfarsson [WU04] for a comprehensive review. Popular simulation tools for urban planning, such as *Urbansim* [Wado2, WBN\*03], do not link behavioural simulation with geometry creation. However, recent techniques that attempt to link behavioural simulation with city generation in order to generate realistic city layouts will now be reviewed in detail. These techniques have a wide variety of use from urban planning applications to interactive games.

Lechner et al. [LRW\*06] demonstrate a system that models urban land use over time,



thereby allowing the user to identify the usage of various regions such as commercial, residential, industrial and park/leisure. An agent based simulation is incorporated, where agents interact with their environment in order to alter its properties. The city's land use evolves over time to alter the road network and building density maps. The system does not generate geometry and simply models the growth of a grid based road network. A similar but less complex simulation that visualises the output using simple geometry generated with L-Systems is described in [HMFNo4]. Weber et al. present a system that not only models urban land use, but provides a fully interactive simulation with geometry [WMWGo9]. The system models socio-economic properties such as land price, wealth, transportation policies and takes into account the environment topological features such as water and forest maps. The main contribution of this paper is that it does not rely on land-use simulation on a regular grid, but rather takes a detailed geometric approach to create arbitrary road networks and building footprints. The simulation works at interactive speeds, while allowing the user to visualise the city growth over time and, to view the effects that changes on policies would have on the population. However, the simulation does not offer any feedback to the behavioural system to effect geometry changes. Vanegas et al. [VABWo9] go one step further, by providing continuous feedback between geometric modelling and behavioural simulation. If the user increases the population, the city evolves to create more jobs and residential areas. Streets and buildings are automatically altered to reflect these changes. If the user alters the geometry, this also has an effect on the population and jobs, e.g., increasing the height of office buildings will cause the simulation to allocate more jobs, which will increase the population in surrounding areas. A dynamical system is utilised to keep the system in equilibrium after each user edit.

Most efforts towards procedural generation of city layouts have focussed on the creation of street networks and building footprints. Little work has been done towards generating realistic placement of vegetation. Plants and trees are very important for the visual appeal and realism of a city, so Beneš et al. [BMJ\*11] propose a simulation for urban ecosystems. They simulate the seeding of plants both naturally and controlled by the cities inhabitants. They compute a distribution of vegetation, whereby plants are likely to appear next to residential buildings such as the back gardens of houses. Their simulation could be implemented with existing procedural cities along with procedural approaches to generate realistic trees and other vegetation.

## 2.4 Inverse Procedural Modelling

Shape grammars can generate realistic building models and can adequately create them in any style or shape. However, it is very cumbersome to manually write the rule sets, and is unintuitive since the output model is not seen until the end. Inverse procedural modelling attempts to solve this problem by inferring procedural rules from some example model. This involves a

detailed understanding of the structure of example models so that production grammars may be inferred. Since images are by far the most prevalent source of urban data, we will focus on using them for examples. This requires a detailed segmentation of building facade images, in order to understand the structure of the architectural elements of which they are composed. Uncovering symmetric relations among these elements is a necessary step towards inferring production grammars capable of reproducing input examples. Hence, we will review work on the topic of symmetry detection, and will start by discussing specialised techniques to segment architectural elements in building facade images.

### 2.4.1 Understanding Images of Building Facades

Photographs are the cheapest and most readily available form of data that can be used for urban modelling. With services like Flickr and Google Street View, anyone can have access to a wealth of urban images. This section is concerned with exploring computer vision approaches to understanding and segmenting urban scenes. Accurate segmentation of images is crucial to generating procedural descriptions from images. While there has been much work done on reconstructing architectural models from image sequences, aerial photographs or even single view reconstruction, a full review is outside the scope of this thesis. We are concerned with inferring a detailed understanding of the structure of building facades, and will thus focus our review on work directly relevant to this goal. For a review of popular work in the area of reconstruction from images, we refer the reader to Zach [Zac07].

#### Segmenting Architectural Elements

There has been a variety of approaches to segmenting building facades into semantic components. Early approaches use predefined templates and prior information learnt from training sets. Schindler and Bauer [SB03] use structure from motion techniques to reconstruct an image from multiple views, where predefined 3D templates are used to find windows in the reconstruction. Dick et al. [DTC04] detect windows in image sequences using wavelets to find areas with strong vertical and horizontal features. Buildings are reconstructed from parts (images of windows and doors) by varying parameters that describe their width, brightness, and other features, while comparing the reconstruction to the input image. Parameters are learnt using an RJMCMC approach using information about the differences between the reconstruction and the input images. Another approach by Brenner and Rippenda [BR06] also uses a Bayesian model to learn architectural priors about window shapes and sizes from a training set. Using RJMCMC they generate a semantic derivation tree to guide the facade segmentation (facades can only be segmented according to a manually input grammar with attributes learnt in the training set). They utilise LIDAR scans in addition to image data. Windows are then detected



based on colour and the fact that they lie behind the wall plane [DTCo4, WTT\*02, WZo2]. Mayer and Reznik extend the approach of [DTCo4], integrating ISM<sup>5</sup> [LS04] with MCMC<sup>6</sup> to classify windows [MR05, MR06, MR07a]. These approaches strongly rely on learnt priors and as such do not scale well to general facade segmentation. More useful approaches have since been proposed to derive an understanding of facade structures from a single image.

Lee and Nevatia [LNo4] determine locations of windows in a rectified image by projecting strong edges to form an accumulation profile (window edges should be stronger than other edge responses). Given a calibration profile, (user drawn wireframe model of the building) they can automatically determine the depth of windows by projecting back onto the original perspective image.

Later methods exploit the symmetrical nature of general architecture to make segmentation more robust. Korah and Rasmussen [KR07] use a Bayesian framework employing a MRF<sup>7</sup> to detect lattices of rectangles in a facade image. Müller et al. [MZWVG07] introduce a system that can semi-automatically infer a production rule set from a single orthographic image of a building. The system reduces a facade image by detecting repeated floors and tiles and removing them from the image, thereby producing what they call ‘the irreducible facade.’ This irreducible facade is then sequentially split based on its gradient image, until some threshold is reached, using the repeated image sections for error correction. Segmented shapes are then matched using a database of known architectural elements and classified. Van Gool et al. [VGZVDBM07] utilise the same approach but propose a camera calibration step for images with large perspective distortion. Intrinsic camera parameters are learnt from vanishing lines, thus permitting a fully automatic depth recovery of facade elements.

Richard et al. [RRAo8] take a different approach to facade understanding. Their solution involves merging histograms created using various image analysis algorithms including Hough line detection, edge detection and background detection. This solution provides a more robust framework for detecting facade features and is not as reliant on repeated features or preset thresholds. It should perform faster than the approach described in [MZWVG07] as it does not have to span a 3D search space while searching for symmetries. Teboul et al. [TSKP10] similarly use a splitting approach in conjunction with a context free grammar to guide the segmentation. A connection is made between grammar semantics and the facade image using probabilistic learning techniques.

Xiao et al. [XFT\*08] use a structure from motion method to capture depth and camera pose from multiple images taken along a street. This information is used for top down subdivision and bottom up merging with detection of architectural elements. In-painting is used to fill in occluded areas of the images and the system provides interactive tools for the user to refine the

---

<sup>5</sup>Implicit Shape Model

<sup>6</sup>Markov Chain Monte Carlo

<sup>7</sup>Markov Random Field



model. They later improve their system making it fully automatic, as described in [XFZ\*09].

### Detecting Regularities

Very often photographs of a building facade have occluding objects in the scene, which are very difficult to avoid as street lights, pedestrians, cables and trees are prevalent in street locations. Previously discussed techniques for facade segmentation rely on clean images without these occluding foreground objects. For correct segmentation it is necessary to remove occlusions in a preprocessing step before applying these algorithms. Urban images are full of redundant information in the form of repeated image patches. Detecting these regularities can allow better understanding of the image, while providing a mechanism to overwrite unwanted occluding objects captured in the scene. Several efforts have been made to detect repeated content in images for use in compression and restoration [KWR07, JFK03, WWOHo8]. These algorithms generate an epitome image, i.e., an image that contains the minimum information required to reconstruct the original image in full. Wang et al. [WWOHo8] provide the best compression ratio and allow random access reconstruction but use a brute force block matching technique. Results are good for images with high repeatability such as highrise buildings or textured scenes but miss repeated content where illumination greatly changes. More recently, Gao et al. [GHLY09] have presented a method to detect structural semantics in images. Using SIFT<sup>8</sup> descriptors [Lowe04], they generate a graph taking into account the relative spatial locations and orientations between features. They can detect repeated content and the relationships between repeated features over different illuminations and in images with low repetition. They demonstrate faster execution times than Wang et al. [WWOHo8].

Uncovering pixel similarities permits the detection of symmetries in the image. Regular repeating symmetries can be used to remove occlusions or aid in segmenting foreground objects that exhibit this symmetry. Hays et al. [HLELo6] present an iterative technique capable of detecting lattices in images. Their approach considers near regular textures, and can detect lattices that are warped. Liu et al. [LBHLo8] use the same approach to detect lattices in the image to remove mesh like objects. Pixels are determined to belong to the mesh if they have the same colour distribution and fit with the detected symmetry pattern. An algorithm designed with the purpose of removing occlusions from images of facades is given in Musialski et al. [MWR\*09]. This method detects repeated image patches using a Monte Carlo algorithm. Local neighbourhoods around randomly sampled feature points (Harris corners [HS88]) are compared using NCC<sup>9</sup>. Using a voting algorithm similar to that used by Mitra et al. [MGPo6], translational symmetries are detected. By using this detected symmetry, pixels with a high degree of symmetry in the image can be propagated in conjunction with edge based smooth-

---

<sup>8</sup>Scale Invariant Feature Transform

<sup>9</sup>Normalised Cross Correlation

ing, thereby overwriting patches that have been occluded. Understanding symmetries in a facade image could also lead to improved segmentation of facade elements. Musialski et al. demonstrate another application of their symmetry detection algorithm in [MRM\*10], where a dominant translational symmetry is detected and used to find repeating facade tiles in the image. The technique is largely concerned with detecting optimal offsets so that segmented facade tiles do not cut through facade elements. This technique could be used to replace the computationally expensive brute force search for initial symmetries in [MZWVG07].

### Semantic Segmentation

The previously discussed approaches rely heavily on restrictive assumptions about the structure of buildings and many require user input and rectified images with no perspective distortion. In this regard they are unsuitable for modelling large collections of images. Recent work on semantic segmentation of urban scenes may provide a mechanism to overcome some of these limitations. Hoiem et al. [HEH05, HSEH07] use multiple cues including location, colour, texture and vanishing points to classify regions of urban scenes. Relative spatial location and orientation of objects can also help reinforce segmentation [MK09]. Xiao and Quan [XBK09] present a similar framework but use 3D reconstruction from multiple images to obtain 3D features for use with AdaBoost classifiers [SY99]. A different approach to understanding street side images is presented by Wendel [WDB10]. This simple approach detects repeating features in the image. Making the assumption that these features are likely to belong to building facades, they perform graph cut segmentation [FHo4] to segment facades from the scene. Using these approaches it would be possible to automatically extract building facades and rectify them for use with facade segmentation algorithms. Occluders could also be classified and removed from consideration in any further processing.

#### 2.4.2 Detecting Structure

Symmetry can reveal a lot about the structure of an object and plays an important role in human cognition [Ley92, Koh92]. Typical man-made structures exhibit symmetries amongst their composing parts. Building facades are composed of many repeating architectural elements and have a defining structure according to their architectural style. Detecting the hierarchical symmetry relations between elements is a crucial step towards understanding their structure.

Early approaches to symmetry detection were based on 2D data sets and attempted to reduce the problem to a 1D pattern matching problem [Ata85, WVV85]. Later, a variety of techniques were proposed to tackle the problem in 3D. Some of the proposed approaches are based on octree traversal [MIK93], Gaussian images [SS97], singular value decomposition [SSo6] and generalised moments [MSHS06]. However, these methods can only detect perfect symmetries



and fail to detect the imperfect symmetries present in noisy data. Recently, a number of algorithms have been proposed to overcome this problem. Successful algorithms for the detection of partial extrinsic symmetries can be divided into two different classes: *transformation voting* and *graph matching*.

### Transformation Voting Based Symmetry Detection

Transformation voting algorithms have been very successful in detecting rotational, translational and reflectional symmetries in input data. They are currently the most widely used technique for symmetry detection. Podolak et al. [PSG\*06] compute a symmetry transform that captures a continuous measure of all possible planar reflections through an input model. Local maximums in the computed transform correspond to strong planar reflections in the image. Mitra et al. [MGPO6] take a similar approach to detect partial symmetries composed of general Euclidean transformations in a 3D model. They map the rotation and translation between pairs of points to transformation space, similar to the Hough transform [Hou59, Bal87]. When all point pairs have been mapped it is possible to use mean shift clustering [CM02] to find the relations of strongest symmetry. After symmetry has been detected it can be used to deform an object to exhibit more global symmetry [MGPO7]. To complement this approach, Pauly et al. [PMW\*08] present a framework to detect repeated elements within a structure that follow a pattern. They observe that transformations between regularly repeating objects present themselves as a regular grid in transformation space. They propose an energy minimisation algorithm to detect this grid among noisy data. Results are shown for the complex architectural models in Figure 2.6. This method has been extended to work with elements that are arranged in curved paths [YMO9]. This is done by clustering based on transformation distances only, and iteratively grouping transformation clusters of equal distance. Examples of symmetries and structural regularities in architectural models are given in [MP08].

### Graph Based

A recent technique for symmetry detection has become a popular alternative to transformation voting algorithms. Transformation voting techniques are subject to the curse of dimensionality [Belo3] as the number of degrees of freedom increases, thus limiting them to rigid transformations of a few dimensions. Graph based symmetry detection can better generalise to transformations with higher degrees of freedom, limited only by the detail of feature descriptors.

Graph based symmetry detection was first introduced by Berner et al. [BBW\*08]. Features are detected in the input geometry and a graph is constructed. Nodes in the graph represent features and edges represent transformations to k-nearest neighbouring features. Symmetries present themselves as repeating subgraphs within the graph of features. A subgraph match-



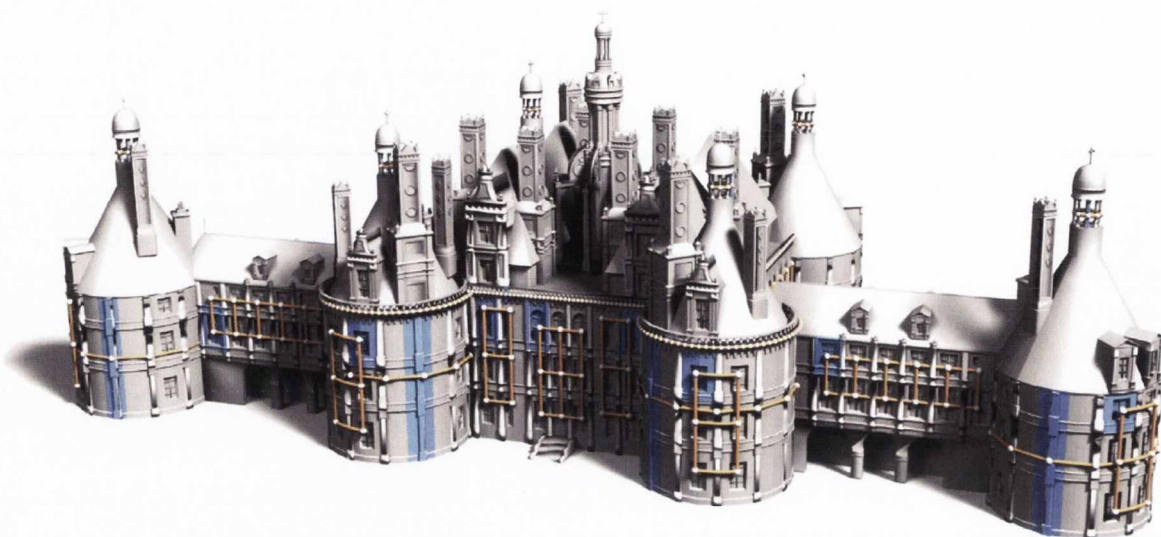


Figure 2.6: Regularly repeating symmetries can be detected in complex architectural models. (Pauly et al. [PMW\*08])

ing algorithm based on similarity distance is utilised to find all pair-wise matching subgraphs. Maximal symmetries are then detected as matching subgraphs with the most regular occurrence. Bokelah et al. [BBW\*09] present an improvement on the previous technique. Instead of using point pairs, they only consider feature lines in order to reduce the search space. They report improved performance over transformation voting algorithms and can detect repeated elements that are not restricted to a regular pattern.

The previous mentioned work is concerned with extrinsic symmetry detection. Intrinsic symmetries are those that are present under some deformation (e.g., a repeating pattern on a cloth that is draped over some object). Methods to detect such symmetries are useful for objects with changing poses, such as people and animals, and are useful in object registration applications. Since our focus is on architecture, for which extrinsic symmetry detection is more applicable, we will not give a detailed review here. However, it should be noted that solving the problem of intrinsic symmetry detection is the focus of most recent work in this area, so we refer the reader to [OSGo8, XZT\*09, LTSW09, KLCF10, MBB10] for more information on this interesting problem.

In general, symmetry detection algorithms require the specification of a set of possible rigid transformations that constitute a symmetry relation. This amounts to searching for evidence of these predefined symmetries (often combinations of translational, rotational, scaling and reflectional transformations) in some space of possible transformations. However, some models exhibit redundancies that cannot be fully described by these rigid transformations. Berner et

al. [BWM\*11] introduce the new symmetry detection concept of *subspace symmetries*. Their approach is a graph-based one where features (crease lines) are detected in the input model and represented with a graph, thus forming a high dimensional shape space to encode the connectivity of vertices. Subspace symmetries are then defined to be mappings between sub-shapes that lie within a low dimensional subspace of this high dimensional shape space, and can be thought of as abstract affine transformations. PCA<sup>10</sup> [Pea01] is used to retrieve these lower dimensional subspaces and a graph matching approach is employed to find symmetries. Another approach that does not rely on specific symmetry groups is that of *Symmetry Factored Embedding and Distance* as presented by Lipman et al. [LCDF10]. Rather than searching for points that conform to a symmetry transformation, they too work on a graph of correspondences between points, embedding points into a high dimensional space. It is then possible to use spectral methods to find symmetries within this high dimensional space.

### 2.4.3 Generating Grammar

Inverse procedural modelling aims to generate an algorithmic system of some sort to reproduce example content but with variation. Following from the recent algorithms presented in Section 2.2, a production rule set is the desired representation. The automatic inference of rules is far from a solved problem. Some of the recent efforts are reviewed here.

An early approach to inverse procedural modelling is given in Hart et al. [HCF97]. They use geometry hashing to detect a fractal's morphological self similarity, thereby inferring branching rules for an iterative function system [Hut81]. However, these are limited to very simple fractals with a small number of rules. More recent approaches address the problem of inferring rules from more complex and general examples that are more applicable to modelling architecture.

Aliaga et al. [ARBo7] propose a method to construct a grammar from multiple photographs of a building. A user aligns edges in the photograph to that on a manually created model. The system can then construct a style grammar that can be applied to models of arbitrary shapes. The system requires a lot of manual input to construct the first model but it is then trivial to produce new buildings in the same style.

The edge based subdivision algorithm presented in [MZWVG07] naturally leads to a derived rule set based on the split grammar proposed in [WWSR03]. However, their subdivision algorithm fails for many architecture styles that have a variation between different floors. More recent approaches are capable of recovering grammars for arbitrary patterns.

Ijiri et al. [IMIMo8] present an example based framework that can synthesise large patterns of elements given an input seed. The user inputs a small arrangement of elements and the system can automatically determine the connectedness and synthesise large patterns with

---

<sup>10</sup>Principle Component Analysis



different variations by applying noise. The user can also guide the synthesis directions. Stava et al. [vBM\*10] produced a system that can generate an L-System rule set given a vector image. Their work is inspired by the recent work on symmetry detection reviewed in Section 2.4.2. They can produce L-Systems that model complex patterns and allow a high level of control over synthesised results.

Merrell and Manoch [Mero7, MMo8, MMo9] present a method that takes an example model as input and can produce varying models that resemble it. Their approach is inspired by ideas from texture synthesis [WLKT09, ELS08, LHo6], whereby image data is extrapolated by incrementally adding bits of the image that best match a small neighbourhood. They use adjacency constraints to produce a model that has connected features similar to the input and restrict the output model. Output models are still very random and lack the fine control that production systems provide. While this approach does not attempt to derive an underlying structure to the input model, it does provide parameterised constraints over synthesis. This data driven approach provides some insight into how symmetric subsets of the model can be naturally interchanged due to local coherency. Bokeloh et al. [BWS10] take the first step towards recovering shape grammars from 3D objects. They make the observation that in many geometric models there are some non-symmetric regions separated from the rest of the model by symmetric regions. They label these regions as *dockers* and a cut through the symmetric region as a *docking site*, as illustrated in Figure 2.7. The observation is that any two docking sites that are similar under some allowed transformation can be interconnected. This allows reshaping of the model by interchanging dockers, thereby producing new models similar to the exemplar. They uncover a context free grammar, denoting docking sites as non-terminal symbols and dockers as terminal symbols. Production rules in the recovered grammar correspond with permissible shape operations, i.e., the connection of docking sites that are similar under some transformation. This is the first approach to inverse procedural modelling of arbitrary 3D geometry.

An approach to reconstruct buildings from calibrated aerial images using grammars based on the Manhattan-world assumption is given in Vanegas et al. [VABa10]. The Manhattan-world assumption [CY99] states that there are three orthogonal directions in the scene and, in the case of building reconstruction, all facades must be aligned to these directions. From an initial building footprint, a volume is extruded and then refined using a linear string rewriting grammar (identical to an L-System with only one successor symbol per rule). Parameters of the grammar are learnt from pixel intensities in the aerial image, successively generating one floor after another from the bottom up. This is a fully automated technique that can learn production rules to generate a building volume provided it adheres to the Manhattan-world assumption.

Nan et al. [NSZ\*10] present an interactive technique for the reconstruction of buildings from point clouds. They introduce the use of *smart boxes* that fit over point samples, snapping



to their proper locations. The boxes are smart in that they can learn contextual relations with other boxes, while automatically resizing themselves to fit the point data. Taking advantage of the orthogonality and regularity of architectural models, smart boxes can be used to quickly reconstruct the input building, thus sensibly reconstructing areas of missing data. Smart boxes could possibly be used in future work to determine procedural descriptions of architectural models from LIDAR data. Using boxes to fit the data is compatible with CGA shape grammar descriptions of building models.

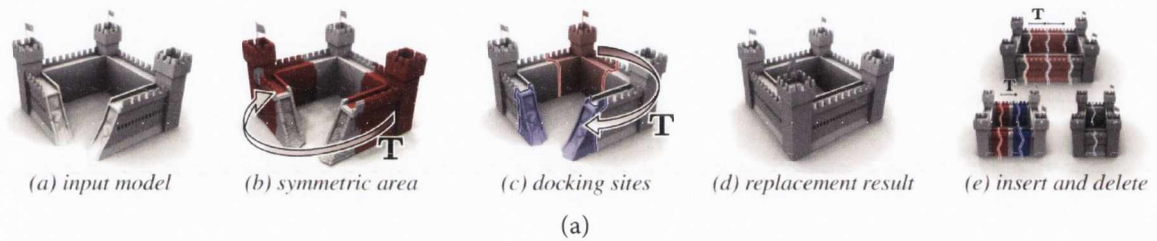


Figure 2.7: (a) Input model. (b) Symmetric regions marked in red. (c) A set of symmetric curves that cuts the model into two pieces yields a docking site that corresponds to (d) a replacement operation. (e) A similar construction yields insertions and deletions. Bokeloh et al. [BWS10]

#### 2.4.4 Towards Structured Hybrids

Inferring production grammars from input data greatly simplifies the modelling process. However, these inferred grammars can only reproduce the given input, albeit offering parameters to reshape the output. A far more powerful application of inverse procedural modelling would be a framework capable of learning a single production set from multiple examples. Such a system would preserve similarities between examples and offer combinatorial variations where differences are observed. One requirement of such a system would be the capability to incrementally learn from new examples. This would require a standardised representation of all detected structural relationships within the example, i.e., similar examples should have a similar structural representation. This interesting direction has not been directly explored in the literature. However, there has recently been some work that may provide insights into this problem.

Wang et al. [WXL\*11] propose representing an object by a graph of symmetries. Their observation is that many man-made objects are composed of a hierarchy of different symmetry relations. They propose a set of precedence rules governing the order in which symmetries are grouped to construct the hierarchy graph that represents relations between connected parts of the object. Using these precedence rules, a standard representation is created, whereby similar objects should have similar graphs.

Fisher et al. [FSH11] propose a graph representation for structural relationships between objects in a scene. They encode both semantic and geometric relations into a graph and present a kernel based approach for graph comparison. Using this representation, they can effectively classify arrangements of objects (e.g., chairs appear under tables) and find matching scenes. This approach is quite similar to our symmetry hybrid approach [CO11], whereby we compare similar patterns based on a graph of symmetry relations.

Yu et al. [YYT\*11] propose a similar scene representation for furniture arrangement. Hierarchical spatial relationships are learnt from example furniture layouts and a new scene can be synthesised by minimising a cost function. The cost function can take into account a number of contextual relationships and is minimised by simulated annealing using a Metropolis-Hastings state search step (See Chib and Greenberg [CG95]). This approach also has similarities to the goal of generating hybrid buildings by representing structural and contextual relationships between objects.

## 2.5 Discussion

Procedural generation of cities can be broken down into two different problems. Generating building models as discussed in Section 2.2 and procedural generation of street layouts as discussed in Section 2.3. We have reviewed previous work that demonstrates how shape grammars can generate realistic building models and are adequate to generate buildings in any style or shape. Likewise, algorithms inspired by the generation of trees have been demonstrated to successfully create plausible street layouts. However, it is very cumbersome to manually write the rule sets, and unintuitive since the output model is not seen until the end. Inverse procedural modelling attempts to solve this problem by inferring procedural rules from some example model. Previous work towards this goal were reviewed in Section 2.4, with a particular emphasis on inferring production grammars from images.

Despite efforts to simplify control over procedural systems, writing production rules still remains a difficult task. This is due to the trial and error approach a user must endure, which can be frustrating and non-intuitive. Entire production systems must be written before results can be visible, while intended local changes could result in unintended global alterations in the model. This problem is two-fold:

1. Changing parameters can have a chaotic effect as they propagate through the system. This is particularly common with L-Systems. Complex systems like road networks may have hundreds of parameters that need to be fine-tuned.
2. Changing production rules for non-terminal symbols may invalidate subsequent rules, either conditionally or semantically. This would cause unexpected changes to the design



of a building, destroying its strict hierarchical layout.

While the former has been addressed in much previous work, the latter problem remains a very open area of research, and is the goal of inverse procedural modelling, which is the process of inferring production rules from example models. While this problem remains unsolved, particularly for the general case, sufficient progress has been made towards inferring deterministic shape grammars to describe building facades. The general process is to determine some understanding of an input model's structure, including its symmetries and contextual relations, and to formalise this structure by inferring a grammar. In the case of images, this amounts to a detailed segmentation of the structural components one wishes to model.

Currently the standard approach to generate buildings is to write highly deterministic production sets for different building styles. This allows building styles to be applied to arbitrarily shaped building volumes, which may be parametrised, thus offering local control over the size and aspect ratios of architectural elements. Deterministic production sets are simpler to visualise and do not suffer as much from the chaotic propagation of parameters. Similarly, inverse procedural modelling can only infer a deterministic production grammar for a given input (since no variation in structure can be observed in a single instance).

Despite recent progress, however, writing specialised deterministic production sets for different building styles is highly redundant as there are many similarities between buildings. This also results in a lack of variation amongst output models, as modellers struggle to write hundreds of different production grammars to generate different buildings. This problem may be addressed by uncovering structural similarities between buildings and expressing their variation through stochastic production rules. This requires a standard representation of all structural relations within the building that we wish to model. Such a standard model would allow inverse procedural modelling to incrementally learn from input examples, thereby automatically inferring stochastic production grammars capable of generating hundreds of hybrid variations. We will address these problems in later chapters.



## Chapter 3

# Procedural Modelling

**P**ROCEDURAL MODELLING of urban environments has become an important topic in computer graphics. With the ever increasing demand for large scale realistic content in games and movies, the time and cost involved in modelling urban content by hand is becoming unfeasible. Apart from the entertainment industry, large urban models are also useful for urban planning applications and emergency response training.

This chapter will explore procedural modelling of urban environments and outline some of the key issues that arise. We start by reviewing the different grammars that have been proposed for architectural design in Section 3.1. Many such grammars have been proposed and each has its own advantages and disadvantages. A good grammatical model should be easy to design with and create many variations while offering control over the output. We hope to tackle this problem with the introduction of an object oriented production system. We will also explore how buildings thus generated can be made look more realistic using texture variations and weathering effects. These problems will all be explored in Section 3.2.

Many previous techniques have been concerned with procedurally generating city layouts. Procedural modelling can also be used to aid in the reconstruction of an existing city. It is often desirable to combine manually created landmark buildings (e.g., created by a 3D artist) and other points of interest with procedurally generated content. Section 3.3 will discuss how GIS data could be used for this purpose.

Many interactive applications could benefit from real-time procedural generation of cities. Online mapping applications such as Google Maps have shown how simple 3d models can enrich the user's experience. However, due to bandwidth restrictions it is unfeasible to have an interactive experience with highly detailed content. We will show how procedural modelling may be used with such a client server system, requiring only descriptors of buildings to be downloaded by the client, thereby reducing the required bandwidth. The client's system would then generate detailed building and terrain meshes capable of being rendered in real-time. Section 3.4 introduces a caching approach and shows how the performance of such a system

may be improved. While many cache based approaches have been proposed for rendering large terrains, the use of such techniques has not been explored for procedural generation of urban models. Numerous problems occur as rendering the buildings takes much less time than generating them. We aim to tackle this problem with a simple solution that can be used with existing techniques for terrain paging.

Finally we will discuss the limitations of these approaches and suggest some solutions that will be explored in later chapters.

### 3.1 Grammars

Procedural modelling is a technique used to automatically generate content from a set of algorithmic rules. Often these rules take the form of production systems which are carefully laid out by the modeller. Many forms of grammars have been proposed to tackle different problems, from modelling plant and multicellular organisms to generating abstract fractal patterns. Recently, there has been growing interest in generating architecture from production rule sets. These systems are based on early work with string rewriting systems and shape grammars. This Section will review the foundations on which recent grammars designed to generate urban content are based.

#### 3.1.1 L-System

L-Systems are parallel string rewriting systems, first proposed for the modelling of multicellular organisms [PL90]. An L-System grammar is defined as  $G = \langle V, \omega, P \rangle$  where the vocabulary  $V$  is a set of symbols,  $\omega$  is the starting string (composed of symbols from  $V$ ) and  $P$  is a set of production rules acting on the alphabet  $V$ . A production rule is defined as:

$$predecessor \rightarrow successor$$

The symbol *predecessor* is replaced with the symbol *successor* wherever it occurs in the string. Unlike context free grammars, all applicable production rules are applied simultaneously to each symbol in the string.

#### Interpretation of L-System Strings

Many applications of L-Systems use simple commands to control a *turtle graphics* cursor. The cursor stores its current state (i.e., its position and orientation) and the commands describe relative Euclidean transformations to move the cursor. The following set of commands summarise how movement of the turtle may be encoded into an L-System string [SQ79, Pru86]:

$+(\theta)$  Turn clockwise by angle  $\theta$ .

$-(\theta)$  Turn anti-clockwise by angle  $\theta$ .

$f(d)$  Move forward a distance  $d$ .

$[$  Push the turtle state onto the stack.

$]$  Pop the turtle state from the stack.

### Parametric and Stochastic L-Systems

The L-System production rules described above are deterministic and not of much interest for procedural modelling, as no variation is possible. However, a stochastic system can be defined by allowing multiple production rules for a single predecessor, each with a given probability of being chosen. Derivation can be further controlled by passing parameters to the production rule [Han92]. The parametric stochastic production rules then take the following form:

$$\text{predecessor}(\text{param}): \text{condition} \rightarrow \text{successor}: \text{prob}$$

Where *condition* is a relation that determines if the rule is valid given some input parameters *param*. For example, the following set of production rules could be used to generate simple streets in a city.

$$\text{Road}(\text{length}, d): \text{length} < 1 \rightarrow f(0.1)\text{Road}(\text{length} + 0.1, d): 0.8$$

$$\text{Road}(\text{length}, d): \text{length} \geq 1 \rightarrow \text{Intersection}(d): 0.8$$

$$\text{Road}(\text{length}, d): \text{length} \geq 0.6 \&\& d > 5 \rightarrow \text{End}: 0.2$$

$$\text{Intersection}(d) \rightarrow [-(90^\circ)\text{Road}(0.1, d + 1)][+(90^\circ)\text{Road}(0.1, d + 1)]: 1$$

A road must grow provided its length is less than 0.6km. If the street has grown longer than 1km it must either end or create an intersection for new roads. Parameter *d* forces the road network to grow if it is less than a certain size. Figure 3.1 shows the generated road network.

### 3.1.2 Shape Grammars

Shape grammars pioneered by George Stiny and James Gips [SG72] are generative grammars that operate on sets of lines. A *shape* is defined as follows:

**Definition** A *Shape* is any finite arrangement of lines with respect to a coordinate system.



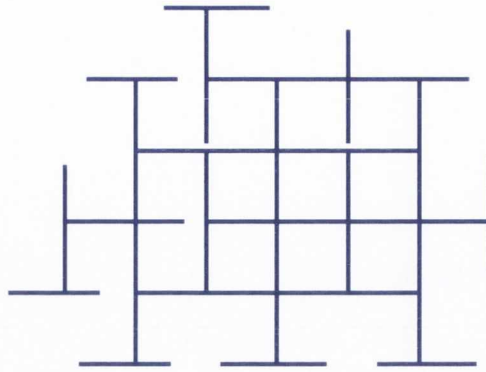


Figure 3.1: Example road network generated using a simple L-System production set.

New shapes can be formed by adding and subtracting two existing shapes. Under addition a new shape is formed consisting of the union of lines belonging to both shapes. By subtracting one shape from another a new shape is formed consisting of the lines in the first shape that are not in the second. A new shape may also be formed consisting of a combination of addition and subtraction of two shapes under a Euclidean transformation.

A shape grammar is defined as  $G = \langle S, i, R \rangle$  where the vocabulary  $S$  is a set of shapes,  $i \in S$  is the initial shape and  $R$  is the set of shape rules acting on  $S$ . A rule  $r$  is defined as,

$$r: a \rightarrow b | a, b \in S^*,$$

where  $S^*$  is the universe of  $S$  consisting of all possible concatenations of shapes in  $S$  under addition or subtraction. If we have some shape  $u \in S^*$  then the rule  $r$  can be applied to  $u$  if and only if the shape  $a$  occurs in  $u$  under some Euclidian transformation  $T$ . A new shape  $v$  is then formed where  $T(a) \subseteq u$  is replaced with  $T(b)$ . Formally:

$$v = u - T(a) + T(b) | T(a) \subseteq u$$

An example set of shape grammar rules and possible derivations is given in Figure 3.2.

### 3.1.3 Split Grammar

L-Systems have had much success in the generation of plant models (See Prusinkiewicz et al. [PHHM96]). This is due to their parallel string rewriting system that can accurately simulate a growth-like process. While these systems have been shown to work well with street generation [PM01], they are less than ideal for applications with spatial constraints such as architecture generation. Shape grammars, on the other hand, have been used for architectural design and analysis (See case studies [Duao2, Fle87, SM78, KE81, Sti77]) and can generate complex designs in a given spatial area. However, shape grammars are difficult to implement and a set of

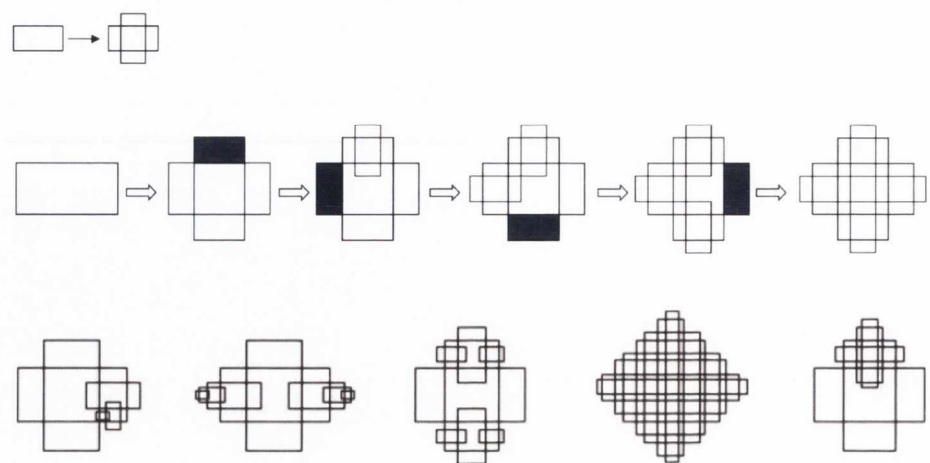


Figure 3.2: Top Row: A simple shape grammar rule. Middle Row: A possible derivation sequence. Bottom Row: Possible end results.

production rules is limited to a specific design.

Wonka et al. [WWSR03] proposed *Split Grammars* to address these problems with architectural generation in mind. Split grammars are a specialised set of shape grammars, which define a shape to be a primitive building block such as a cuboid or prism. A split grammar is therefore defined similarly to a shape grammar  $G = \langle S, i, R \rangle$ . Split grammars differ from general shape grammars specifying the following shape operations:

- Split** Defined as  $a \rightarrow b$  where  $a$  is a connected subset of shapes in the vocabulary  $S$  and  $b$  contains the same elements as  $a$  except for one element which is split.
- Replace** Defined as  $a \rightarrow b$  where  $a$  contains one element from the vocabulary  $S$  and  $b$  is a different shape from  $S$  that replaces  $a$  while being contained in the bounding volume of  $a$ .

An example set of split rules and the result of their derivation is illustrated in Figure 3.3. Attributes are assigned to each shape, specifying the texture and position of the shape and may contain other values that can be used to control the derivation. These attributes are propagated during each step of the derivation.

3.1.4 CGA Grammars

CGA grammars proposed by Müller et al. [MWH\*06] are an extension to split grammars. A CGA grammar simplifies the use of *splits* by combining the use of L-Systems and split grammars. Shapes are identified as either terminal symbols or non-terminal symbols and production

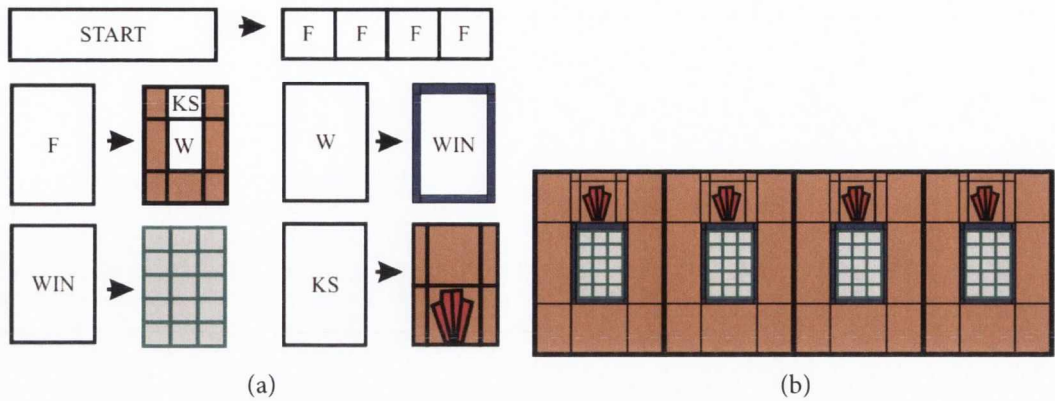


Figure 3.3: (a) An example set of split grammar rules. (b) The result of derivation using the rules in (a). Image from [WWSRo3]

rules are defined similarly to parametric L-System rules:

$$\text{Predecessor: Condition} \rightarrow \text{Command}(\text{params})\{\text{Successor}\}:\text{Prob}$$

Each shape stores a set of geometric attributes, namely its position, orientation and scale, which describe a local coordinate system. These attributes define a bounding box called the scope of the shape. Müller et al. [MWH\*06] define a set of commands that take a non-terminal shape as input and produce one or more shapes as output.

**Component Split:** Breaks a shape down into the lower dimensional shapes it is composed of.

For example, a 3D building volume can be broken down into 2D planes representing the facades:

$$\text{building} \rightarrow \text{Comp}(\text{"sidefaces"})\{\text{facade}\}:1$$

Individual edges and vertices can also be accessed using this command, providing a mechanism to easily detail specific subparts of shapes while using the same representation.

**Shape Subdivision:** Splits a shape along a given axis. Parameters are given to specify the size of the new successor shapes, which can be given either absolute or relative sizes. For example, a facade can be split into a unique ground floor and a block of upper floors, providing the facade is large enough:

$$\text{facade: Scope.height} > 2 \rightarrow \text{subdiv}(\text{"y"}, 2, 1r)\{\text{ground\_floor}|\text{upper\_floors}\}:1$$

This provides an intuitive way to subdivide building parts.

**Repeat:** Splits a planar shape several times to fit many new shapes of a given width. For exam-



ple, a block of floors can be split into as many floors that will fit:

$$upper\_floors \rightarrow repeat("y", 1.5)\{floor\}:1$$

Using this rule allows buildings to be scaled to different heights without needing to modify the rule set.

**Insert** Replaces a primitive shape with an external model. This can be used to replace parts of a building with highly detailed models that could not be procedurally generated. For example, a planar shape representing a window can be replaced with a detailed model:

$$window \rightarrow insert("detailedWindow.mesh")\{floor\}:1$$

The inserted model is scaled to fit the scope of the predecessor shape.

In addition to these modelling commands, the CGA grammar also allows Euclidean transformations to *translate*, *scale* and *rotate* shapes.

### Occlusion and Snapping

Some parts of a building may occlude others so it is necessary to check for these occlusions before generating further details. For example, an extension to the building shape will occlude parts of the wall. The CGA grammar allows the system to query for occluding shapes before a rule is applied. Occlusion queries can return whether a shape is fully, partially or not occluded by another shape.

$$tile: Shape.occ("all") == "none" \rightarrow window:1$$

$$tile: Shape.occ("all") == "part" \rightarrow wall:1$$

$$tile: Shape.occ("all") == "full" \rightarrow \epsilon:1$$

Apart from occlusion queries, the CGA grammar also makes it possible to create snap lines that allow shapes to align themselves in an aesthetic way. In many buildings the ground floor is specified differently than the other floors, which can cause misalignment of windows and doors between different floors. Using snap lines overcomes this problem.

## 3.2 Building Generation

In this section we will detail some problems with procedural generation of building models and possible solutions. Most notably, these problems involve difficulties in writing production

rule sets and lack of variation among generated buildings. We will introduce the idea of object oriented building styles, and show how they can simplify the writing of production systems to generate buildings. In addition we will discuss how simple techniques to allow variation among building textures can result in more realistic building models. The use of recent algorithms to simulate the effect of weathering on facade materials is also explored.

### 3.2.1 Object Oriented Buildings

One notable problem with using production systems such as the CGA shape grammar discussed in Section 3.1.4 is the difficulty of creating new building styles from previously written scripts. It is very cumbersome to rewrite entire building scripts just to make a specific change. This practice results in a lot of redundant code between different building scripts.

We propose the use of object oriented buildings as a solution to this problem. Figure 3.4 illustrates this idea. Buildings inherit everything from more abstract styles and only respecify certain aspects of the style. This is achieved by encapsulating semantically relevant production rules in labelled blocks. Each block is given a list of variables that can be changed at runtime or redefined by a child style. The building script for generating *Building<sub>1</sub>* from Figure 3.4 is given in Listing 3.6. It can be seen how production rules are grouped into semantic blocks (e.g., Rules defining how floors are created are grouped together in the *Floor* block). To create a new building that is similar to this building, it is only necessary to redefine the block that is changed. For example, to generate *Building<sub>2</sub>* from Figure 3.4 it is only necessary to specify how windowsills should be added, thereby drastically reducing the size of the script (See Listing 3.7). Each building can inherit from multiple parents, thus combining different variations. Buildings also inherit their parents' elements (i.e., 3D models that are used to replace certain terminal symbols) and can add or remove from their parents' element set. As illustrated in Listing 3.5, multiple meshes can be specified, corresponding to different levels of detail for the rendering system. Meshes are swapped with different level of detail meshes depending on their distance to the camera. The use of object oriented building styles can simplify the writing of new styles and can link building styles together in a meaningful way.

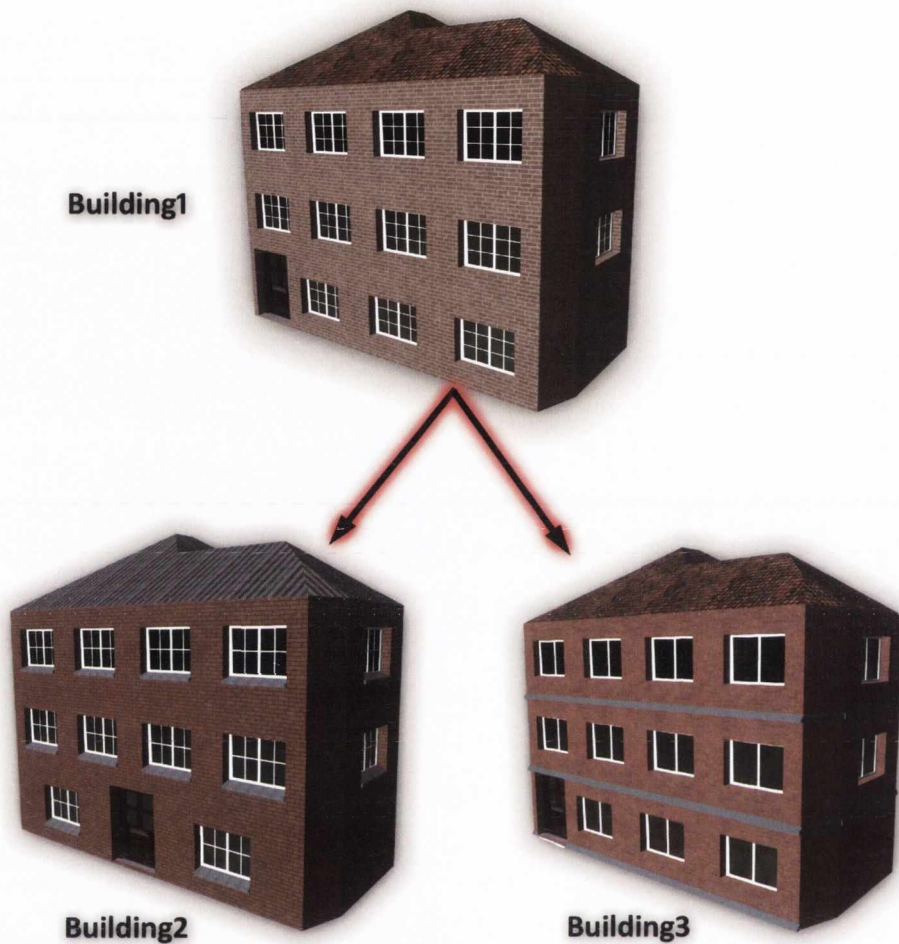


Figure 3.4: Building2 inherits from Building1, specifying how windowsills should be added. Building3 also inherits from Building1, adding a ledge to each floor. Example code to generate these buildings is provided in Listings 3.5 - 3.7.

---

```

class ElementPack
{
  Elements
  {
    WINDOW:
      "window1LOD1.mesh" "window1LOD2.mesh" : 0.5
      "window2LOD1.mesh" "window2LOD2.mesh" : 0.5

    DOOR:
      "door1.mesh" : 0.2
      "door2LOD1.mesh" "door2LOD2.mesh" : 0.8

    LEDGE:
      "windowLedge1LOD1.mesh" "windowLedge1LOD2.mesh" : 1
  }
}

```

---

Listing 3.5: Listing for elements: Multiple meshes can be defined to add variation, and different levels of details can be specified for each mesh.



---

```

class Building1 : ElementPack
{
  Footprint
  {
    FOOTPRINT ~> extrude(BUILDING_HEIGHT){ BuildingVol } : 1
    BuildingVol ~> comp ("facades"){ FACADE } : 1
  }

  Facade
  {
    var GroundFloorHeight 1

    FACADE : H > (GroundFloorHeight + 1) ~>
      subdiv("Y",GroundFloorHeight,1r){ GROUND_FLOOR | UPPER_FLOORS } : 1
  }

  Ground_Floor
  {
    var EntranceWidth 0.75
    var DoorDepth 0.1

    GROUND_FLOOR: W > (EntranceWidth+1) ~>
      subdiv("X",1r,EntranceWidth,0.1){ FLOOR | EntrancePanel | WALL } : 0.3
    GROUND_FLOOR: W > (EntranceWidth+1) ~>
      subdiv("X",1r,EntranceWidth,1r){ FLOOR | EntrancePanel | FLOOR } : 0.44
    GROUND_FLOOR: W > (EntranceWidth+1) ~>
      subdiv("X",0.1,EntranceWidth,1r){ WALL | EntrancePanel | FLOOR } : 0.3
    EntrancePanel ~> subdiv("Y",0.02,1r){ WALL | Entrance }
    Entrance ~> detrude(DoorDepth){ DOOR | WALL } : 1
  }

  Upper_Floors
  {
    var FloorHeight 1.0

    UPPER_FLOORS ~> repeat("Y",FloorHeight){ FLOOR } : 1
  }

  Floor
  {
    var TileWidth 1.1

    FLOOR ~> repeat("X",TileWidth){ TILE } : 1
  }

  Tile
  {
    var WindowDepth 0.1
    var WindowWidth 0.75
    var WindowHeight 0.5

    TILE ~> subdiv("X",1r,WindowWidth,1r){ WALL | Tile | WALL } : 1
    TileC ~> subdiv("Y",1r,WindowHeight,1r){ WALL | WindowPlane | WALL } : 1
    WindowPlane ~> detrude(WindowDepth){ WINDOW | WALL } : 1
  }
}

```

---

Listing 3.6: Listing for Building<sub>1</sub> of Figure 3.4: Production rules are grouped into semantic blocks that correspond to different subparts of the building.

```
class Building2 : Building1
{
  Tile
  {
    var LedgeHeight 0.075
    var WLedgeHeight (WindowHeight+LedgeHeight)

    TILE ~> subdiv("X",1r,WindowWidth,1r){ WALL | TileC | WALL }: 1
    TileC ~> subdiv("Y",1r,WLedgeHeight,1r){ WALL | WindowPlane | WALL }: 1
    WindowPlane ~> detrude(WindowDepth){ WindowPlaneInner | WALL }: 1
    WindowPlaneInner ~> subdiv("Y",LedgeHeight,1r){ LEDGE | WINDOW }: 1
  }
}
```

Listing 3.7: Listing for Building2 of Figure 3.4: Building2 inherits everything from Building1 but specifies how windowsills should be added.

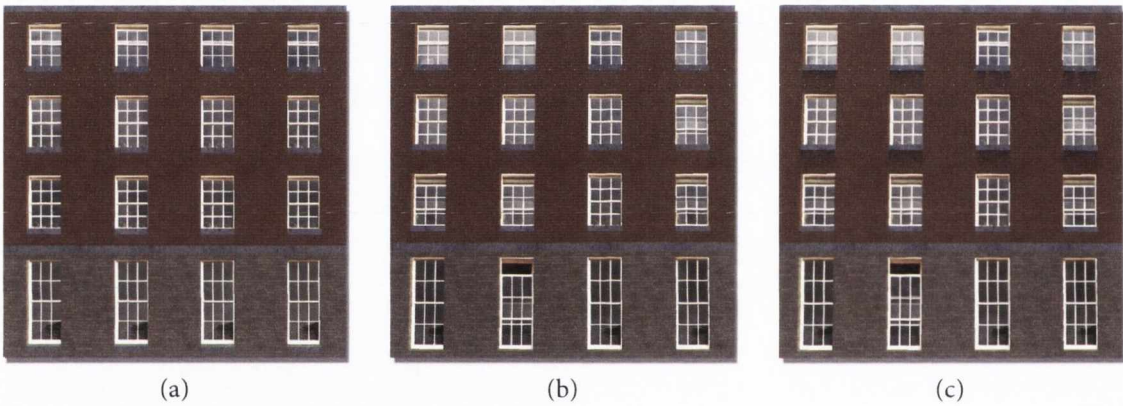


Figure 3.8: (a) No variation between windows of the same type. (b) Windows vary with some open and some closed, adding more realism to the facade. (c) Adding weathering effects such as stains under the windows further increases realism.

3.2.2 Texture Variations

In addition to the shape operations available in the CGA shape grammar, we allow a *texture* command to set the texture of a given shape. All derived shapes then inherit this texture.

Generated building facades can look very artificial, exhibiting many repeated elements. This lack of variation needs to be addressed in order to produce more convincing results. More variation can be achieved by allowing multiple variations of the same texture within a building. For example, several textures can be defined for a single window type, thereby allowing some windows to be open and others closed or with window blinds and curtains in different configurations. Figure 3.8 shows how this simple extension can add to the realism of a generated building facade.

## Weathering Effects

Real world buildings are exposed to harsh weathering that alters their appearance over time. Pollution gradually darkens building facades, contributing to the dirty appearance that is characteristic of older buildings. Rain water flowing down a building's facade deposits dirt, thus staining the building according to its flow pattern. Large variations in these weathering effects occur, depending on the exposure, shape, and material of which the building is composed. These effects can be simulated to generate more variation among buildings, thereby adding to the realism of the procedural city.

Bosh et al. [BLR\*11] propose a method that can realistically simulate the characteristic washing and staining effects of water flowing over a building's facade. The simulation utilises a set of parameters that can accurately reproduce stains from example images. For the purpose of generating random stains, we are only concerned with their stain simulation algorithm and not with learning the correct parameters from examples. Therefore, the set of parameters for stain generation as given in Table 3.1, can be randomly chosen within some given bounds. The stain simulation algorithm, as shown in Algorithm 3.9, models individual water particles flowing over a surface. Each particle  $i$  has a position  $(x_i, y_i)$ , velocity  $v_i$ , mass  $m_i$  and stain concentration  $S_i$ . At each time step  $\Delta\tau$ , particles are moved according to their velocity and deposit some stain material on the facades surface, denoted  $D(x, y)$ . The amount of material deposited depends on the surface properties given in Table 3.1, along with the particle's changing mass and concentration of stain material. The roughness  $r$  of the material deflects water particles in a random direction  $\xi$ , perpendicular to the vertical flow. At each step the particle accelerates according to a gravitational constant  $g$ , and a model of surface friction  $f(x, y)$ , which takes into account the surface wetness  $w(x, y)$ .

While this simulation seems overly complex for the simple purpose of generating random stains, it is in fact simple to implement. Windowsills are automatically tagged as stain generators during the generation of building meshes. Stain could alternatively be specified in the production rule set, thereby allowing the user to choose what facade elements will generate stains. To optimise building generation and rendering, stains need not be generated for each new building. Instead, an atlas of stain textures could be generated and applied using a shader. Results of applying stains to a building facade are given in Figure 3.8c.



| Symbol        | Parameter                   |
|---------------|-----------------------------|
| $S_i(o)$      | Initial stain concentration |
| $m_i(o)$      | Initial particle mass       |
| $k_S$         | Stain adhesion rate         |
| $k_D$         | Stain solubility rate       |
| $r$           | Target roughness            |
| $a$           | Target maximum absorption   |
| $k_a$         | Target absorptivity rate    |
| $N$           | New particles per time step |
| $\Delta \tau$ | Size of time step           |
| $T$           | Total time                  |

Table 3.1: Parameters for weathering algorithm from Bosch et al. [BLR\*11].

```
Initialise  $D(x,y)=0$  and  $w(x,y)=0$  for all target locations
Foreach water particle  $i$  do
  Set position  $(x_i, y_i)$  at stain generator location
  Initialise particle attributes:  $v_i=0, m_i=1, S_i=S_i(o)$ 
  Foreach time step  $\Delta \tau$  until  $T$  do
     $f(x_i, y_i) = \max\left(0.2, \frac{a-w(x_i, y_i)}{a}\right)$ 
     $v_{i,x} = v_{i,x} + (2\xi - 1) r |v_i| \Delta \tau$ 
     $v_{i,y} = v_{i,y} + g \Delta \tau$ 
     $x_i = x_i + (1 - f(x_i, y_i)) v_{i,x} \Delta \tau$ 
     $y_i = y_i + (1 - f(x_i, y_i)) v_{i,y} \Delta \tau$ 
     $m_i = m_i - \left(k_a \frac{a-w(x_i, y_i)}{a m_i}\right) \Delta \tau$ 
     $w(x_i, y_i) = w(x_i, y_i) + \left(k_a \frac{a-w(x_i, y_i)}{a m_i}\right) \Delta \tau$ 
     $S_i = S_i + (-k_S S_i + k_D D(x_i, y_i) m_i) \Delta \tau$ 
     $D(x_i, y_i) = D(x_i, y_i) + (k_S S_i - k_D D(x_i, y_i)) \Delta \tau$ 
  end
end
```

Algorithm 3.9: Algorithm for stain generation. Bosch et al. [BLR\*11]

### 3.2.3 Roof Generation

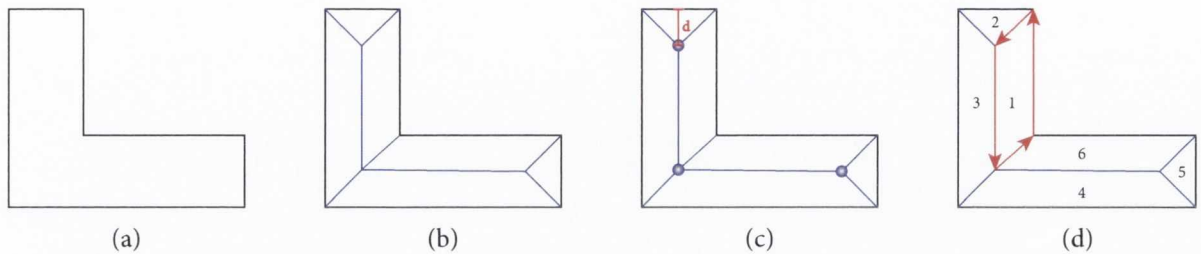


Figure 3.10: (a) Roof generation from building footprint. (b) Straight skeleton of the building footprint. (c) Points along skeleton are equidistant from the shape's boundaries. Internal vertices can be raised by distance  $d$ . (d) Determine roof faces.

Roofs can be modelled using the straight skeleton algorithm of Felkel and Obdržálek [FO98]. The straight skeleton is a shrunk version of a shape where all points are equidistant to the shape's boundaries (See Figure 3.10). A standard hipped roof can be modelled as follows:

1. Calculate straight skeleton of building footprint (Figure 3.10b).
2. Raise the internal vertices by their distance  $d$  to the shape boundary (Figure 3.10c).
3. Determine roof faces by traversing edges, taking the smallest interior angle at vertices (Figure 3.10d).
4. Triangulate roof faces using a suitable triangulation method for convex polygons.

Other roof types can be modelled by translating the internal vertices of the straight skeleton. For example, a gable roof can be modelled by translating the internal vertices to the footprint boundary as illustrated in Figure 3.11. Methods to create more roof types from the straight skeleton are presented in Laycock and Day [LD03].

## 3.3 Data Extraction from GIS

Since we are interested in reconstructing an existing city, we are not concerned with algorithms to procedurally generate new street layouts. Urban planning information, including road networks and building footprints, among other data, is made available by mapping agencies for use in GIS technologies. This GIS data often takes the form of raster and vector maps. We will make use of this data to reconstruct an existing city layout, and discuss how to extract useful data from these vector maps for this purpose.

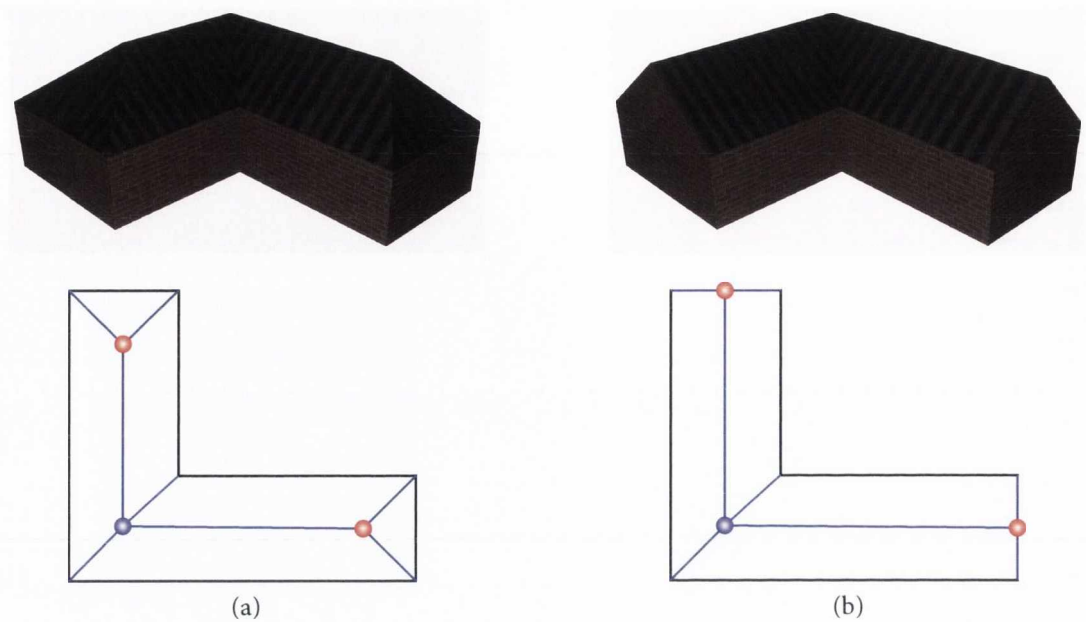


Figure 3.11: Different roof types can be created from the straight skeleton: (a) Hipped roof. (b) Gable roof can be created by translating the red vertices to the boundary of the building’s footprint.

3.3.1 Extracting Building Footprints

The GIS data recorded contains detailed urban planning information, which is stored in different semantic layers that make it easy to access the building layouts. However, since the data is simply represented by a set of poly-lines, it is necessary to determine which lines belong to which buildings. Figure 3.12 illustrates this process. The following algorithm describes how to extract the building layouts:

- 1. Create a graph representing all the vertices and edges.
- 2. Start at the bottom left node which contains two or more edges.
- 3. Follow the smallest interior angle edges until the starting node is reached again, thus creating a cycle.
- 4. Decrement the degree of every node along the cycle.
- 5. Repeat from Step 2 until no nodes with a degree greater than one remain.

The extracted building footprints are then loaded into a database for quick referencing by the system. A similar technique is used to extract the roads and insert the road network graph into a database.





Figure 3.12: Extracting building footprints from GIS data (left). Layer containing buildings is first chosen by the user (middle), while buildings are then extracted by finding loops in the data (right).

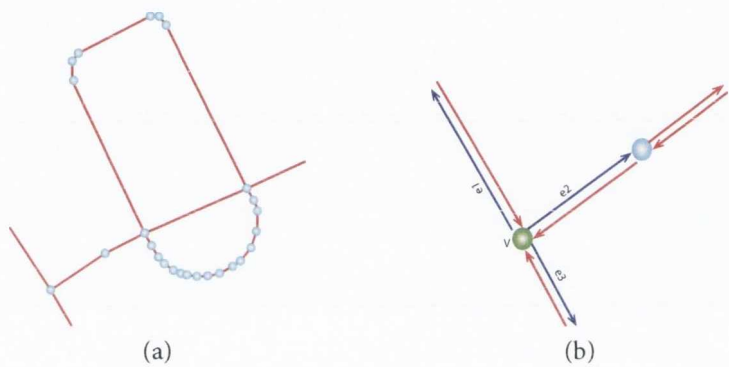


Figure 3.13: (a) Road data is represented in the GIS data as a set of lines connected at vertices. (b) A more useful representation where each vertex has a set of directional edges leaving itself.

3.3.2 Extracting Roads

Roads in the GIS data are represented as simple line segments. We would like to construct a road network graph which will later be used to model roads and possibly be used for traffic simulation in future applications. Each vertex in the data represents either a road intersection or a point along a curved road (See Figure 3.13a). For simplicity, we do not distinguish between the two types of vertices and simply create a set of directional edges leaving each vertex. This is illustrated in Figure 3.13b, where a vertex  $V$  has a set of directional edges  $e$  leaving itself. Each road segment contributes to two edges oriented in opposite directions. Using this approach simplifies traversing the graph for generating road meshes and could also be used to control the direction traffic may take.

Generating Road Meshes

For real-time rendering, it is often useful to divide a large terrain area into a grid of smaller geometric pages. Each page is a rectangular area within the scene that must be populated with polygon meshes. Given a page boundary, we would like to construct a mesh to model the roads and terrain within this boundary. Since roads are simply represented with a set of edges we must

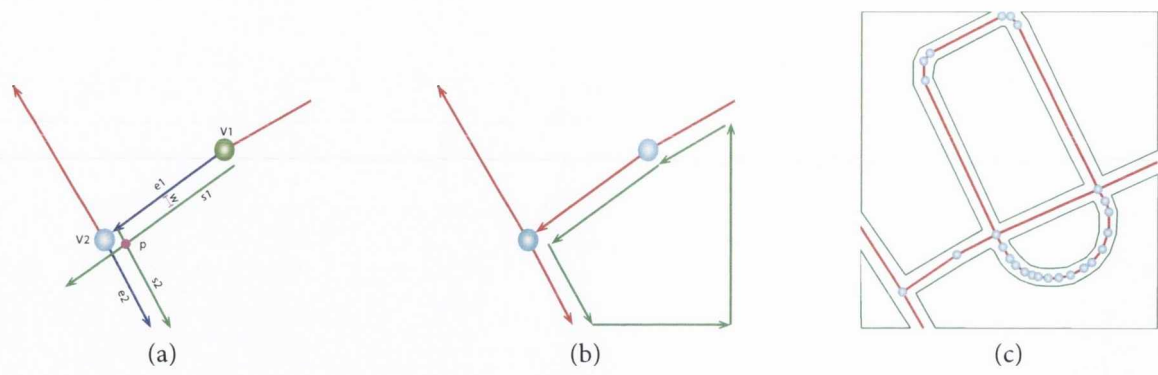


Figure 3.14: (a) Roadside edges  $s_1$  and  $s_2$  are created a distance  $w$  (the road width) to the left of their respective edges. The intersection point  $p$  between these roadside edges determine the terrain polygon vertices. (b) Roadside edges may intersect the page boundary. (c) The result is a set of polygons describing the page terrain.

calculate the geometry of the road sides. This task is simplified by the fact that all directional edges in the road network graph represent one side of a road. If edge is given an attribute  $w$ , the road width we can then calculate the mesh geometry as follows:

1. Take any edge  $e_1$  with a point within the page boundary.
2. Create a new parallel edge  $s_1$  at a distance  $w$  to the left side of  $e_1$ .
3. Take the connecting edge  $e_2$  with smallest interior angle to  $e_1$ .
4. Create  $s_2$  and calculate intersection point  $p$  of  $s_1$  and  $s_2$ .
5. Set the end point of  $s_1$  to  $p$  and the start point of  $s_2$  to  $p$ .
6. If  $p$  lies outside the page boundary then connect  $s_1$  to the page boundary edge.
7. Repeat from step 1 until all edges are processed.

This process is illustrated in Figure 3.14. After all roadside edges  $s$  have been created we can then extract the terrain polygons. This is simply done by traversing a roadside edge  $s$  until it loops back on itself, giving an anti-clockwise set of points defining the polygon boundary. Polygons can then be triangulated using an existing triangulation method [DZo8]. The same approach is applied to create the road polygons by simply traversing in the opposite direction as illustrated in Figure 3.15

Road polygons are placed lower than the terrain polygons and roadside curbs can be created by making planar shapes for each roadside edge connecting the road polygons to the terrain polygons, thereby creating a mesh. An example of the resulting road meshes is shown in Figure 3.16.

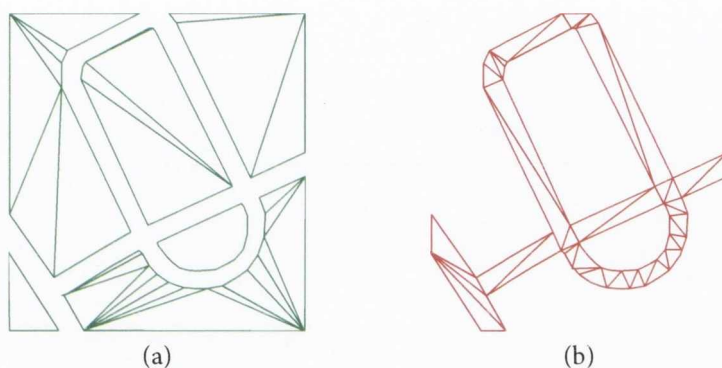
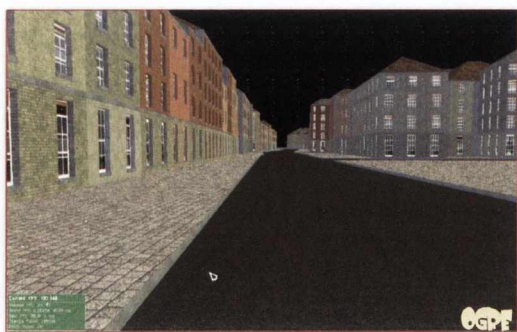


Figure 3.15: (a) Traversing anti-clockwise along the road edges gives a set of vertices that define the terrain polygons. (b) Similarly, road polygons can be created by simply traversing in the opposite direction. In both cases, polygons can be triangulated using existing methods.



(a)



(b)

Figure 3.16: Resulting road network: (a) Road meshes are placed lower than the terrain geometry to create curbs. (b) By utilising GIS data, road meshes can be created for the entire city.



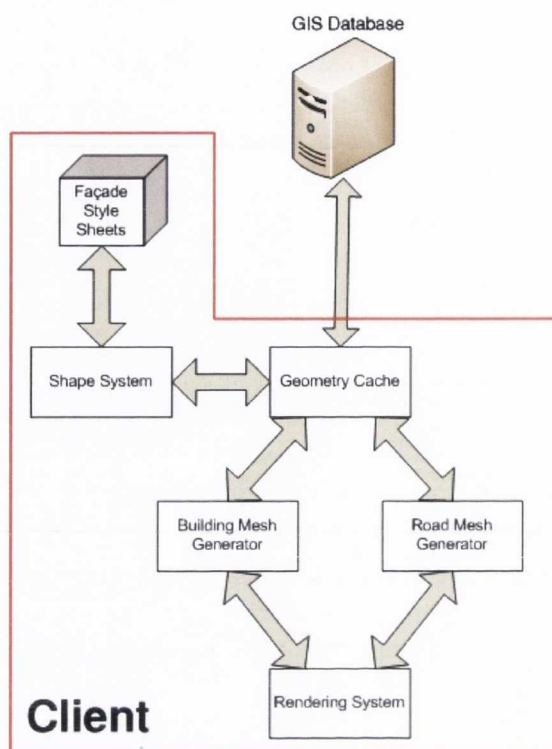


Figure 3.17: System Overview. GIS data along with building descriptors are stored on an external data server. Production rule sets are stored on the clients system and procedurally generated by the *Shape System*. The *Geometry Cache* determines what geometry is to be generated and at what level of detail. Building and road meshes are then generated as a final step before being sent to the *Rendering System*.

### 3.4 Real-Time Generation

In this section we present a system that can produce large detailed virtual cities in real-time using GIS data. Previous approaches focus on either pre-generating large cities or are limited to simple grid layouts and building geometry with random styles. The system presented continuously updates the city by streaming GIS data from a server along with style descriptions for every building, without interrupting the rendering system.

Urban GIS data is preprocessed and stored in a database along with style descriptions of every building for quick referencing. This preprocessing step was explained in Section 3.3. Style sheets that control the facade generation are loaded at run-time and are stored in a hash-table on the client's system. The geometry cache updates itself based on the camera's position in the environment, downloading the surrounding environment data from the GIS database. The cache controls what geometry is procedurally generated based on its distance from the camera. Meshes for the roads and buildings are then batched together for efficient rendering and sent to the render system. An overview of the system is shown in Figure 3.17.

---

```

while NewPage = getPageFromQueue()
    NumBldPerThd = NewPage.NumBlds/NumProc
    for x = 0 to numProcessors-1
        Thread[x] = ForkThread()
        Thread[x].MemoryPool = new MemoryPool
        Thread[x].BuildingList = distBuildings(NumBldPerThd)
        Thread[x].GenerateBuildings()
        NewPage.setBuildingMeshes(Thread[x])
    end for
    SynchroniseThreads()
end while
NewPage.BatchMeshes()

```

---

Algorithm 3.18: Algorithm for procedurally generating buildings in parallel.

### 3.4.1 Parallel Geometry Cache

In order to maintain a constant and high frame rate, building generation should not interrupt the rendering system. We achieve this by introducing a multi-state cache that stores geometry that is currently being generated. The system is based on the idea of paging geometry for rendering large terrains. The world is split into a regular grid as illustrated in Figure 3.19. The data in the cache have the following three states:

- State 1 Geometry descriptions are downloaded from the database and the area is procedurally generated. (Outer white area in Figure 3.19).
- State 2 Meshes are constructed and sent to the graphics card but are not yet rendered (Middle blue area in Figure 3.19).
- State 3 Meshes currently being rendered (Inner green area in Figure 3.19).

Depending on the camera motion, grid squares that are likely to become visible in the near future are loaded. Geometry descriptions are downloaded from the GIS database, procedurally generated and inserted into the cache. This is done in a separate thread from the rendering system. Only squares that are close to the camera are rendered. If a square is not yet generated, the rendering thread will put it on the end of a queue and try to retrieve the next square.

### 3.4.2 Parallel Building Generation

With the trend in computing power drifting towards multi-processor architectures, it is desirable to take advantage of parallel computation. It is possible to procedurally generate multiple buildings at the same time by utilizing parallel processing techniques. Algorithm 3.18 presents a simple algorithm that can speed up building generation on multiprocessor systems.

Each thread maintains a memory pool that is reused for every building it generates, which reduces memory allocation bottlenecks. Threads must synchronise before writing to the cache so that buildings can be batched together for fast rendering.

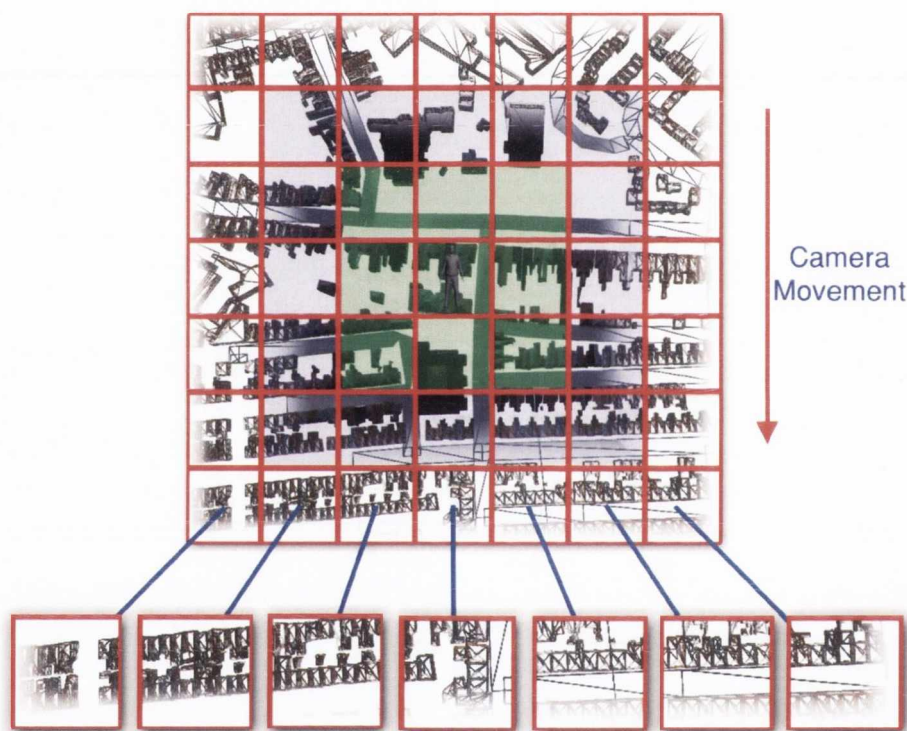


Figure 3.19: As the camera moves towards the geometry, new pages must be loaded. The pages are organised into a queue and processed in order of their distance to the camera. The buildings within each page should be shared among parallel executing threads.



**Benchmark**

To test the system, we conducted two separate benchmarks, which were performed on a machine with the following specifications:

|     |                                            |
|-----|--------------------------------------------|
| CPU | Intel(R) Core(TM)2 Duo CPU E8500 @ 3.16GHZ |
| RAM | 4GB                                        |
| GPU | NVIDIA GeForce 9800GT                      |

First, a frame rate analysis of the system was taken while the camera was moved between two preset points, both with and without the parallel geometry cache (Section 3.4.1). The results are given in Figure 3.20. While the camera travelled a distance of 800m in the scene, exactly 2,038 buildings were created. This had a significant effect on the frame rate of the system without a parallel cache. The sudden drops in frame rate correspond with new geometry pages being loaded and cause a jerk in the camera motion. In the system with the parallel cache there is much less jerking when pages are loaded and the overall frame rate stays within acceptable levels.

The second experiment performed was a multi-threaded processing benchmark. Four pages were generated consisting of 10, 100, 1000 and 10,000 buildings respectively. Processing time was logged for each of the pages with building generation distributed over a different number of threads. The average results over ten repetitions are shown in Figure 3.21. A configuration with two threads running in parallel yielded the best performance on the dual core machine. Running the experiment with more threads than processors led to worse results because of the overhead of thread switching. However, this result suggests better performance could be achieved with a greater number of processing cores. Better results were obtained using larger page sizes with 10,000 buildings leading to a 27.48% increase in performance (We suspect that this is due the initial memory pool allocation assigned to each thread). Table 3.2 shows the number of buildings generated per second for the 10,000 building page test. Each building was set to be strictly the same shape, contained an average of 980 vertices and required 610 shape operations to generate. Figure 3.24 demonstrates the type of architecture and scale of the city generated in the tests.

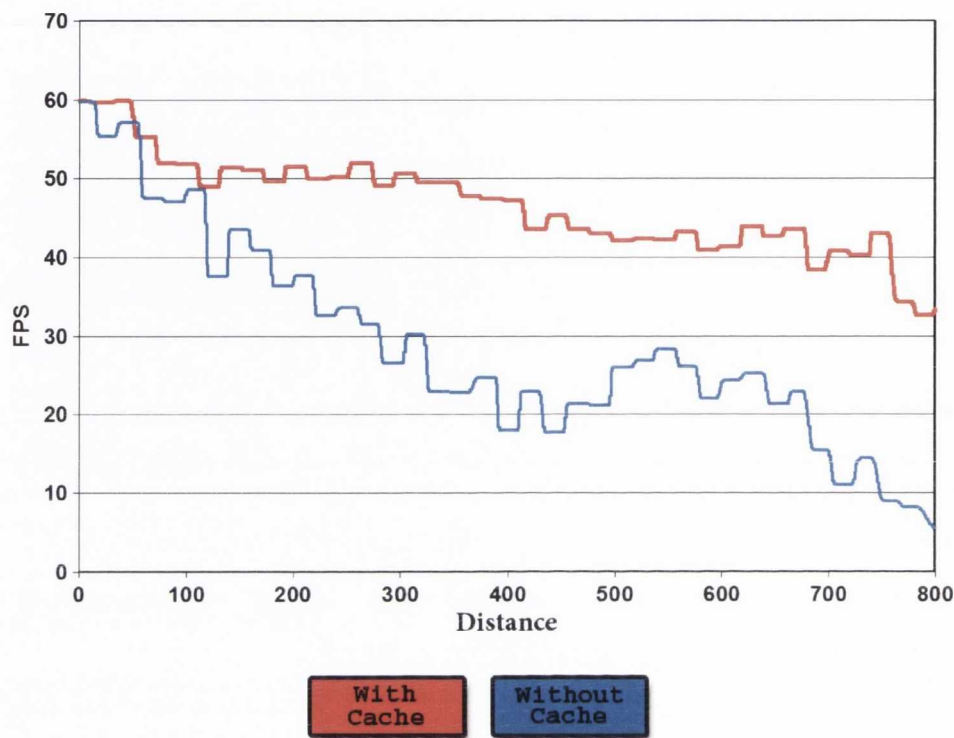


Figure 3.20: A comparison of results with and without the cache described in Section 3.4.1. The system with the cache has a much higher frame rate and less jerky movements of the camera. There was an average of 1,922 buildings in the scene at any time with 2,038 buildings created and destroyed over the distance.

|           | Buildings/Sec | Shape Operations/Sec | Percent Increase |
|-----------|---------------|----------------------|------------------|
| 1 Thread  | 255.56        | 15,586.34            |                  |
| 2 Threads | 325.78        | 19,868.86            | 27.48%           |
| 3 Threads | 277.47        | 16,922.73            | 8.57%            |

Table 3.2: Benchmark of multi-threaded processing on the 10,000 building data set. Results are shown for the number of buildings generated per second, the number of shape operations (discussed in Section 3.2) performed per second and the percentage performance boost over a single threaded configuration.

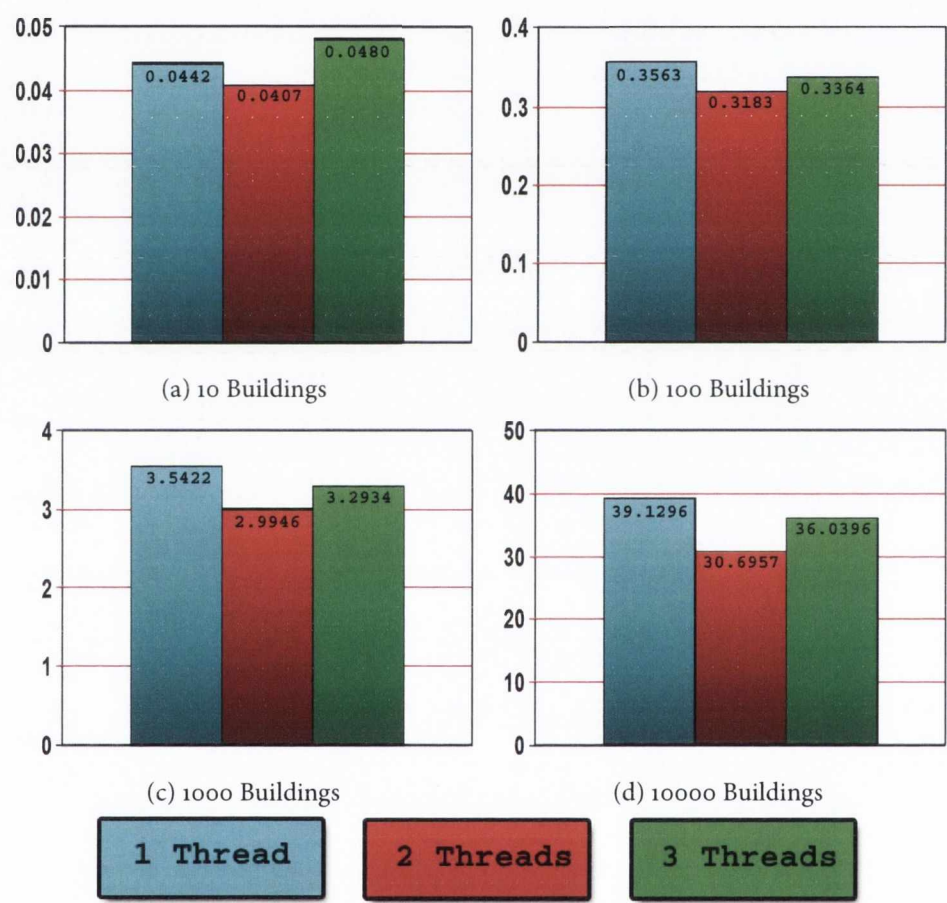


Figure 3.21: Time in seconds to generate buildings with different levels of multithreading. On the dual core machine two threads yielded the best performance.



### 3.4.3 Level of Detail

Buildings can be automatically generated with different levels of detail. We define three levels of detail as follows:

- LOD 1:** Building generation stops after the building footprint is extruded.
- LOD 2:** Building generation completes, skipping detrude and extrude commands.
- LOD 3:** Building generation completes, building has full detail.

These three different level of detail meshes are illustrated in Figure 3.22.

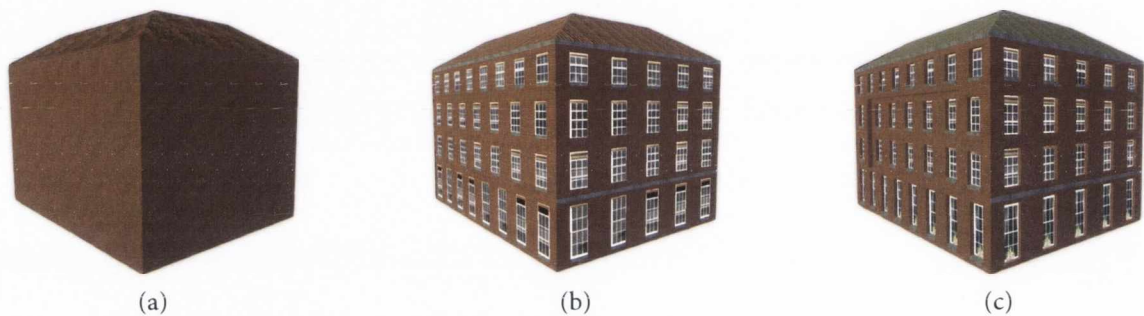


Figure 3.22: (a) Lowest level of detail. (b) Mid level of detail. (c) Highest level of detail.

Lower level of detail meshes not only improve rendering speeds, but also reduce the time required to generate the building itself. A comparison of generation times for the different levels of detail is given in Figure 3.23. For each of the three levels of detail illustrated in Figure 3.22, 10,000 identical buildings were generated. The time taken to generate *LOD 1* is almost negligible compared to the full detail mesh.

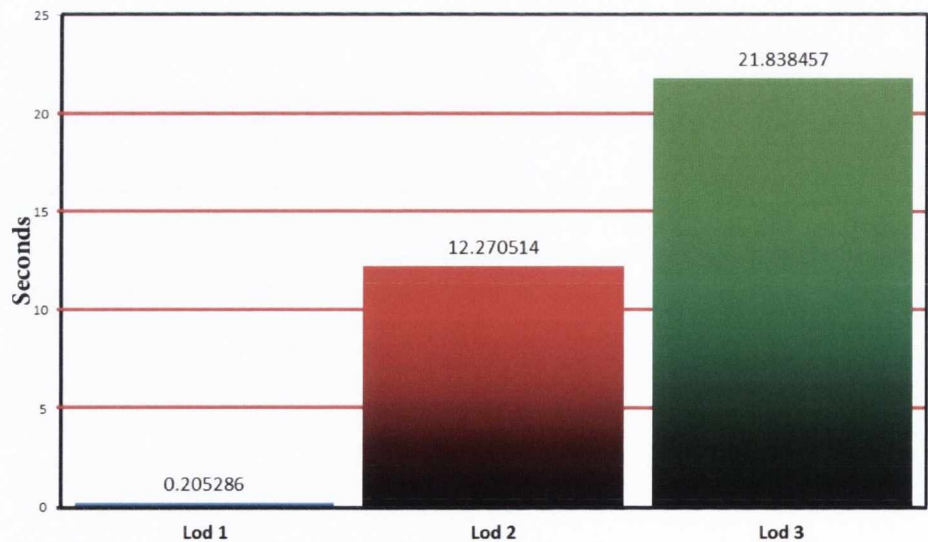


Figure 3.23: Time taken in seconds to generate 10,000 buildings at three different levels of detail.

### 3.4.4 Other Optimisations

Many building facades are occluded by other buildings. We can make further optimisations by considering that most of the buildings in our city are of similar height. We can then decide not to generate facades that are in close proximity to other buildings. We detect building footprints that share a common edge in the GIS data. We can then provide a set of indices to determine which footprint edges should be generated and which footprint edges can be replaced by a wall. This effectively reduces the number of facades that need to be generated in full detail by about 20% per geometry page, thereby improving generation and rendering speeds.

There are also many small footprints in the GIS data that correspond to garden sheds and other small structures. We remove all footprints with an area less than 10% of the average footprint area for the city.

## 3.5 Discussion

Section 3.1 reviewed various grammars that can be used for procedural generation of urban environments. Some approaches were introduced in Section 3.2 with the goal of tackling the difficulty of writing production systems and the lack of variation that can result from using such systems. The use of GIS data to reconstruct an existing city was explored in Section 3.3. We discussed some approaches in Section 3.4 that can improve performance enough to allow real-time generation of detailed virtual cities from GIS data. The system can be run over a network while allowing multiple clients with only one data set. We also presented a set of benchmarking statistics calculated with different configurations of the system.

The output of our system is given in Figure 3.24. The procedural city was built around manually created models of the Trinity College Dublin campus. GIS data was used to generate the buildings and roads in real-time. Buildings can be accurately overlaid on existing aerial maps and satellite imagery. Procedural buildings act as an extension to manually created content, thereby allowing a dramatic expansion of the city. Users can reshape and restyle the buildings in real-time or swap them with externally created models.

Writing production sets to model buildings still remains a cumbersome task. It is difficult to create a lot of variation between building facades without writing many different production sets. The introduction of object oriented buildings provides a limited solution to this problem, but the user is still required to imagine what the result will be while writing the code. This is very much a trial and error approach that can be very time consuming. In later chapters we will further explore this problem with the goal of hiding these production sets from the user. We will explore inverse procedural modelling techniques that automatically derive production rules from real world photographs. We will look at recent symmetry detection algorithms and how





Figure 3.24: City generated in real-time. Procedural building models are integrated with user created 3D CAD models of the Trinity College Campus. By utilising GIS data, buildings may be placed on top of satellite imagery.



they may be used to create thousands of varying hybrid buildings from a few input examples.

## Chapter 4

# Inverse Procedural Modelling

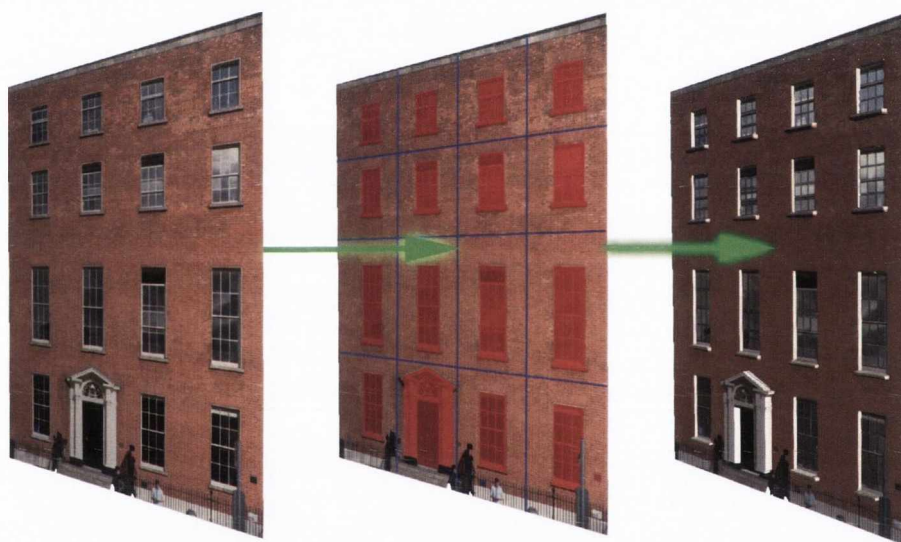


Figure 4.1: Procedural rule sets can be learnt from photographs of building facades. This involves using image processing techniques to segment architectural elements and assigning depth to image regions, thereby creating a 3D model.

**W**RITING production rule sets to generate architectural models can be a difficult task. Such an approach requires the modeller to imagine the end result while writing the rules. This naturally leads to a trial and error process that can be both time consuming and frustrating. Inverse procedural modelling aims to address this problem by taking an example model and automatically inferring a set of rules to describe it. An automatic system would radically simplify the creation of procedural models, hiding the complex rule sets from the user.

There are many data sources that we could use as input models for inverse procedural modelling, including: lidar data, photographs and structure from motion techniques. This chapter will focus on inferring rules from photographs as they are by far the most readily available data.

A depiction of this process is given in Figure 4.1, where a photograph is taken as input, processed and finally a 3D model is created. The structure of this model is determined in such a way that it may be represented using a production rule set. This involves hierarchically subdividing the facade until its architectural elements are obtained. This process is similar to the method in which procedural building facades are created using CGA shape grammar (as discussed in Chapter 3). We must utilise computer vision techniques to segment important architectural elements, in order to obtain this structural subdivision.

To simplify the subdivision algorithms, we take as input a rectified photograph void of perspective distortion. This is briefly discussed in Section 4.1. Section 4.2 will discuss methods to create an initial subdivision of the rectified facade image. This will be the foundation for detecting repetitions that will later be used to support algorithms for more accurate subdivision discussed in Section 4.4. Deriving production rule sets from the subdivided facade is a parallel process to this subdivision and will be detailed in Section 4.5. Finally we will discuss the limitations of deriving rule-sets from photographs.

## 4.1 Rectifying

Most photographs of buildings are taken from ground level, looking up, and therefore exhibit a large perspective distortion. To simplify image processing algorithms and the derivation of split grammars, it is necessary to rectify this distortion. This problem is well understood and there are many automatic techniques based on vanishing lines that can achieve good perspective correction [Bar83]. Commercial software is available for this purpose and Figure 4.2 shows a facade image before and after rectification using Adobe Photoshop.

## 4.2 Subdividing Facades

The first step in deriving a production system for a rectified facade image is to determine where repetition in the image occurs. Determining repeating translational symmetries will later improve subdivision algorithms, while defining how a facade may be expanded to an arbitrary sized building volume. This is accomplished by first subdividing the facade horizontally and vertically into a set of facade tiles. Repetitions can then be found among these tiles. We will discuss algorithms to determine these subdivisions and how to detect translational symmetries among these facade tiles. An overview of the process is given in Figure 4.3.

Facades are split based on two functions corresponding to some vertical and horizontal





Figure 4.2: (a) Input facade image has perspective distortion. (b) Rectified facade image.

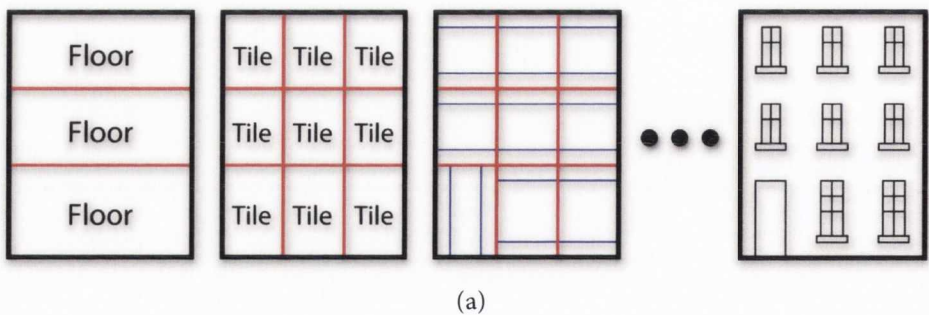


Figure 4.3: Facades are hierarchically subdivided (Müller et al. [MZWVG07]). First the facade is divided into a set of tiles that can be used to find repetitions. Repetitions in these tiles can later be used to further subdivide the facade into its composing elements.

accumulation of image features for some input image  $I(x, y)$ :

$$Vert(x) = \sum_{y=0}^{I_{width}} features(I, x, y)$$

$$Hor(y) = \sum_{x=0}^{I_{height}} features(I, x, y)$$

Splitting points correspond to local minimums of these functions, designating lines that are least likely to traverse facade elements. Features could correspond to edges in the image or some other method that gives a strong response where facade elements are located.

### 4.2.1 Edge Based

Müller et al. [MZWVG07] introduce an approach to iteratively divide a facade into its components. Vertical and horizontal edges are found in the input image using the Sobel edge detector. Two functions which represent the strength of vertical and horizontal edges crossings are as follows:

$$Vert(x) = \sum_{y=0}^{width} GradVert(I, x, y) - \kappa GradHor(I, x, y)$$

$$Hor(y) = \sum_{x=0}^{height} GradHor(I, x, y) - \kappa GradVert(I, x, y)$$

Where  $GradVert$  and  $GradHor$  are the gradient images returned from the Sobel operator in the vertical and horizontal directions respectively (See Figure 4.5).  $\kappa$  is a small constant that can promote snapping onto horizontal and vertical lines when dividing the image in the horizontal and vertical directions. Please refer to Figure 4.4. The function to the right of the image represents  $Vert(x)$  and the function under the image represents  $Hor(y)$ . It can be seen from the image that low frequency local minimums correspond to ideal places to split the image.

In order to consider only low frequency components, we propose the following technique:

1. Approximate the function with a line using linear least squares approximation.
2. Divide the function into multiple high frequency sub-functions by cutting the function along the line.
3. Take the global maximum of each sub-function and divide the image along those points.

This is a much faster technique than trying to remove high frequency components using the Fourier transforms. Each sub function also retains the sharp peaks corresponding to the local maximums that would be lost if the function was over-smoothed.

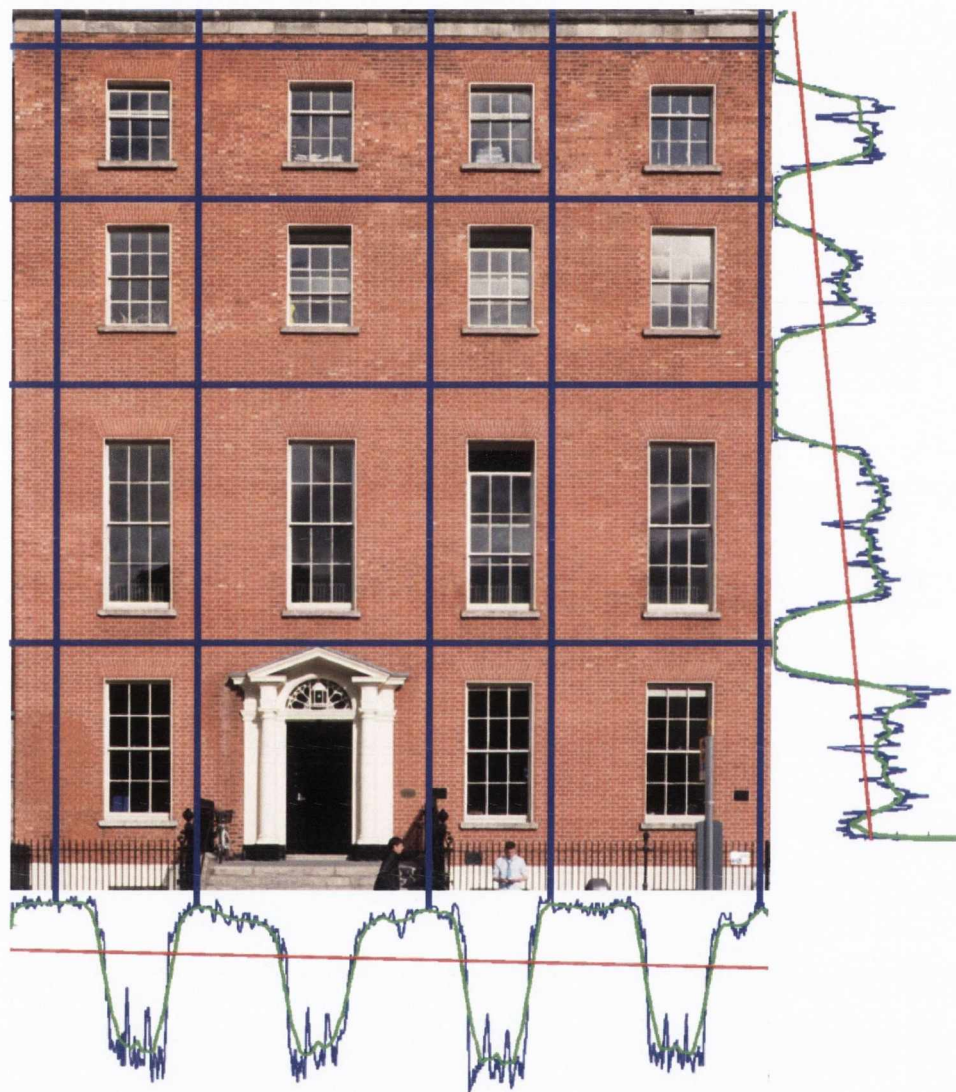


Figure 4.4: Facades can be divided into floors and columns using edge detection. Blue lines correspond to the accumulation functions  $Vert(x)$  and  $Hor(y)$  respectively. Green lines correspond to a smoothed version of these functions, removing high frequency components. The red lines are a linear fit through these functions that can be used to subdivide the functions for splitting.



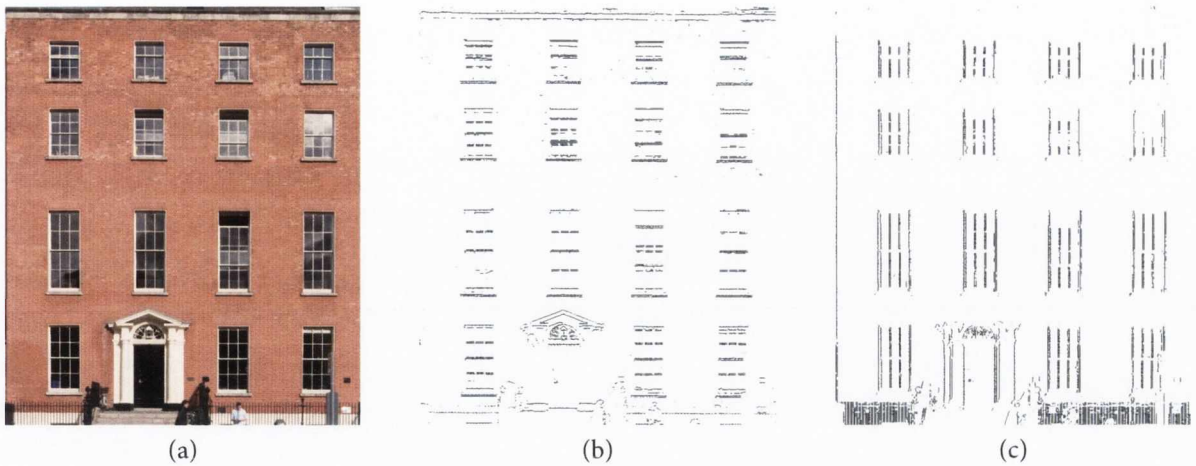


Figure 4.5: (a) Input image to be segmented. (b-c) Horizontal and vertical edge responses are considered separately when subdividing the facade in the horizontal and vertical directions respectively.

A number of problems occur if we use facades that contain strong textures such as brick patterns. Accumulation of edge responses of a brick texture may be stronger than that of desirable edges such as windows and doors. This makes it difficult to determine where to split the facade. The edge responses of these textures can be reduced by down sampling the image using a pyramid scheme. Nevertheless, many edges remain and suitable thresholds must be selected to obtain a good subdivision. Selection of thresholds is difficult and often different values are required on a per image basis. Figure 4.6 illustrates this fact.

### 4.2.2 Region Based

Dividing the facade image based on edge detection requires careful tuning of threshold values, often on a per image basis. This leads to a tradeoff between having spurious edges and reducing the number of desired edges. Considering that architectural elements generally have a different intensity to the texture of the surrounding walls, we may view the problem as region based segmentation. We have tried a number of approaches to segmenting architectural elements and will discuss each in depth.

#### Binary Thresholding

Architectural elements in photographs often appear darker than surrounding walls with pixels belonging to the same region sharing intensity values. With this in mind a simple binary thresholding approach seems like a good place to start. Binary thresholding separates an image into background and foreground regions based on pixel intensity values. Taking a greyscale

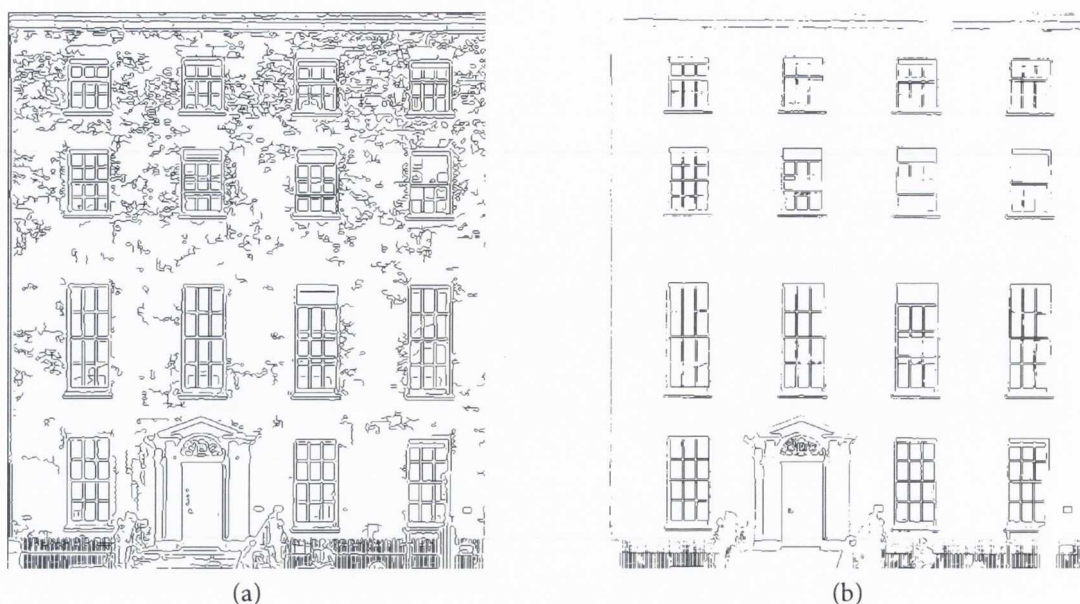


Figure 4.6: Facades are divided into floors and columns using edge detection. Suitable thresholds must be chosen to accurately divide the facade into floors and columns. (a) A bad choice of threshold gives many edge response due to textures in the image. (b) A better choice can remove these edges at the expense of degrading the outline of architectural elements.

input image  $I$ , a binary image  $B$  is given as,

$$B(x, y) = \begin{cases} 1 & \text{if } I(x, y) \geq T \\ 0 & \text{otherwise} \end{cases},$$

where  $T$  is some threshold value. The choice of threshold  $T$  determines which pixels belong to the foreground and which belong to the background. Fortunately, there are algorithms to automatically choose an optimal value for  $T$  based on pixel intensity distribution.

We use the Otsu method [Ots79] for selection of a suitable threshold. A histogram image is created representing the distribution of  $L$  grey-levels,  $[0, 1 \dots, L]$ . Let  $n_i$  equal the number of pixels of intensity  $i$ , so that,

$$\sum_{i=0}^L n_i = N,$$

where  $N$  is the total number of pixels in the image. A probability distribution  $p(i)$  can be created by normalising the histogram:

$$p(i) = n_i/N, \quad \sum_{i=0}^L p(i) = 1$$

Selecting the ideal threshold  $T$  can then be thought of as minimising the combined (within-

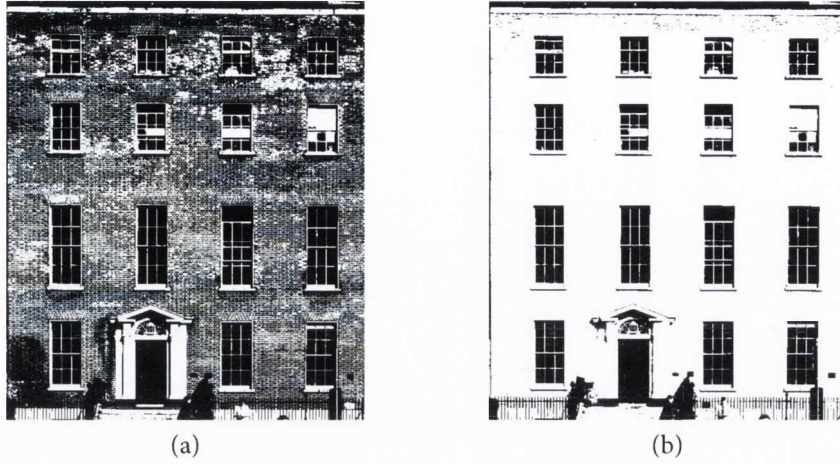


Figure 4.7: The result of binary thresholding the input facade in Figure 4.7a. Black and white correspond to the foreground and background regions respectively. (a) Much of the wall texture is segmented with the windows. (b) Thresholding on the red image channel alone gives better results.

class) variance of the foreground and background clusters. We select  $T$  to minimise,

$$\sigma_{within}^2 = \omega_B(T)\sigma_B^2(T) + \omega_F(T)\sigma_F^2(T), \quad (4.1)$$

where  $\sigma_B^2(T)$  and  $\sigma_F^2(T)$  are the variances of the pixels grouped into the background and foreground clusters respectively (i.e., below and above the threshold), and  $\omega_B(T)$  and  $\omega_F(T)$  are the probability of the background and foreground clusters given a threshold  $T$ :

$$\omega_B(T) = \sum_{i=0}^{T-1} p(i), \quad \omega_F(T) = \sum_{i=T}^L p(i) \quad (4.2)$$

Otsu [Ots79] present an iterative approach to select a value of  $T$  that minimises this total variance.

The result of applying a binary threshold to the input image in Figure 4.5a is given in Figure 4.7a. Much of the wall texture is segmented along with the windows and doors. We can obtain better results by considering the red image channel only as windows are likely to have a low intensity in the red image channel. According to Mayer and Reznik [MR07b], this is because red light passes more easily through glass and the reflected sky is mostly blue. Applying a binary threshold to the red image channel gave the result in Figure 4.7b. While this result is much improved, there are still parts of the image misclassified (such as the door frame and windowsills). It is apparent that thresholding on intensity levels alone is not sufficient and a more general clustering approach that can take colour information into account is required.



## K-Means

To take colour information into account when segmenting the input image, a multivariate clustering algorithm that can work with a user defined number of clusters is required. K-Means clustering proposed by Macqueen [Mac67] is one such algorithm that fits this criteria. Similar to Otsu's method for binary threshold selection, k-means clustering minimises the within-class variance. For  $k$  clusters  $C = [C_1, C_2, \dots, C_k]$ , the position of cluster centres that minimises within-class variance is given as:

$$\operatorname{argmin}_C \sum_{i=1}^k \sum_{x \in C_i} \|x - \mu_i\|^2 \quad (4.3)$$

Where  $C$  is a collection of clusters and  $\mu_i$  the mean of cluster  $C_i$ . The algorithm is summarised as follows:

1. Initialise cluster centres by randomly placing them.
2. Assign all data points to their nearest cluster centre.
3. Calculate the mean position of all data points within each cluster.
4. Move cluster centres to their means and repeat from step 2 until convergence.

K-means clustering will converge on a local minimum and may not give the optimum result. The initial choice of cluster centres largely determines where they will converge. Several iterations of the algorithm with different choices of initial cluster centres may be needed to achieve the best result.

Our results for segmenting the input facade image using k-means clustering are illustrated in Figure 4.8. For a choice of two clusters, our results are similar to the binary threshold (See Figure 4.7). Using more clusters, we can obtain a better segmentation of architectural elements of differing textures, including the door frame and windowsills that could not be obtained using binary thresholding. However, the optimal number of clusters to correctly segment architectural elements from the background is dependent on the image. More detail (and noise) is segmented with a greater number of clusters and we may simply chose the background to be the cluster with the greatest number of pixels (depicted in white in Figure 4.8). While this approach yields good results, we may reinterpret the problem as finding the probability that a pixel belongs to the background, thereby removing the need to select a suitable number of clusters.

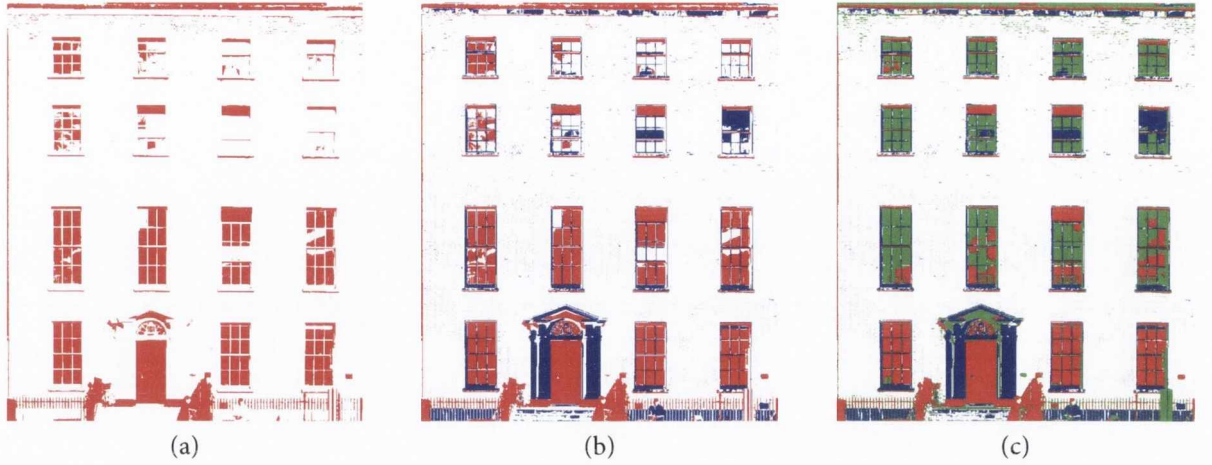


Figure 4.8: Segmentation using k-means clustering for a choice of two, three and four clusters respectively. The greater the number of clusters the more detail that can be segmented from the background walls (depicted in white).

### Gaussian Mixture Model

Segmenting building facades using clustering techniques such as binary thresholding or k-means clustering has led to unsatisfactory results. As demonstrated in Figure 4.8, increasing the number of clusters improves segmentation. However, it is difficult to tell which clusters should belong to the background walls and which correspond to facade elements. Making the assumption that only one cluster (the largest one) corresponds to the background is fundamentally flawed. Wall textures can be composed of many different colours, rendering a single Gaussian cluster unsuitable to accurately model the distribution. To correctly segment facade elements from the wall texture, a better model is required.

We may view the problem of segmenting facade elements as determining the distribution that models the background texture. Since a single Gaussian is unsuitable to describe this distribution, a Gaussian mixture model is required. Gaussian mixture models are among the most mature statistical methods for clustering (cited as far back as the late 19<sup>th</sup> century), and have been successfully used to model colour distributions for both soft and hard segmentation of images [RT00, RKBo4]. Given some known sample of the wall texture, it is possible to calculate a Gaussian mixture model to accurately describe its colour distribution. Let us first consider the case of a single multivariate Gaussian with mean  $\mu$  and covariance  $\Sigma$ ,

$$p(X) = N(X|\mu, \Sigma), \quad (4.4)$$

where  $X = [X_r, X_g, X_b]^T$  is a random vector corresponding to the image pixels with red green

and blue colour channels. For  $n$  sample pixels  $S = [x_1, x_2, \dots, x_n]$ , the mean  $\mu$  is given as:

$$\mu = \begin{bmatrix} \mu_r \\ \mu_g \\ \mu_b \end{bmatrix} \quad \mu = E[X] = \frac{1}{n} \sum_{i=1}^n (x_i)$$

The covariance matrix  $\Sigma$  can also be calculated from the sample pixel colours:

$$\Sigma = \begin{bmatrix} E[(X_r - \mu_r)(X_r - \mu_r)] & E[(X_r - \mu_r)(X_g - \mu_g)] & E[(X_r - \mu_r)(X_b - \mu_b)] \\ E[(X_g - \mu_g)(X_r - \mu_r)] & E[(X_g - \mu_g)(X_g - \mu_g)] & E[(X_g - \mu_g)(X_b - \mu_b)] \\ E[(X_b - \mu_b)(X_r - \mu_r)] & E[(X_b - \mu_b)(X_g - \mu_g)] & E[(X_b - \mu_b)(X_b - \mu_b)] \end{bmatrix}$$

Having computed the mean and covariance matrix from the sample image, the pdf<sup>1</sup> of the multivariate normal distribution is calculated as:

$$p(X = x) = \frac{1}{(\sqrt{2\pi})^d \sqrt{\det(\Sigma)}} \exp\left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)\right), \quad (4.5)$$

where  $d = 3$  in the case of three dimensional colour data. The calculated covariance matrix may sometimes be singular (or close to it), i.e., non-invertible resulting in a non-computable pdf. This case can be avoided by adding a small value to its diagonal elements.

The Gaussian mixture model is a set of  $k$  weighted multivariate Gaussian distributions:

$$p(X) = \sum_{i=1}^k \omega_i N(X | \mu_i, \Sigma_i) \quad (4.6)$$

Such that:

$$\sum_{i=1}^k \omega_i = 1 \text{ with } \omega_i \geq 0 \forall i$$

The set of parameters  $\Theta = (\{\omega_i\}, \{\mu_i\}, \{\Sigma_i\})$  that define this distribution needs to be calculated from the sample image. We accomplish this using the binary tree quantisation algorithm described in Orchard and Bouman [OB91]. The algorithm starts with one cluster initialised to contain all sample points. The eigenvector of the covariance matrix with greatest eigenvalue corresponds to the first principle component, i.e., the direction of most variation (See Pearson [Pea01]). The cluster is partitioned by a plane through the mean and perpendicular to this eigenvector, giving two new clusters. The process is repeated until  $k$  clusters  $C_1, C_2, \dots, C_k$  have been created, each time choosing to split the cluster with greatest variance. The algorithm is summarised as follows:

1. Initialise  $C_1$  to include all points in the sample image  $S$ .

---

<sup>1</sup>probability density function



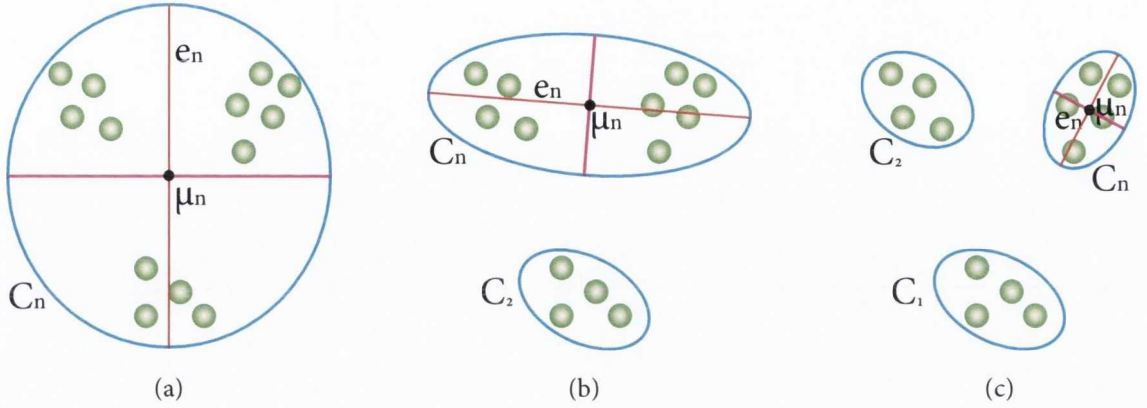


Figure 4.9: Binary quantisation algorithm. (a) Initial cluster contains all points. The cluster  $C_n$  is split through its mean  $\mu_n$ , perpendicular to its eigenvector  $e_n$  with greatest eigenvalue as indicated by the purple line. (b) A new cluster  $C_n$  is chosen with the greatest eigenvalue. (c) The process repeats until  $k$  clusters have been created.

2. Calculate the mean  $\mu_1$  and covariance  $\Sigma_1$  of  $C_1$ .
3. For  $i = 2$  to  $k$ 
  - (a) Find the set  $C_n$  with largest eigenvalue and take its corresponding eigenvector  $e_n$ .
  - (b) Create a plane perpendicular to  $e_n$  intersecting  $\mu_n$ .
  - (c) Split  $C_n$  into two sets,  $C_i$ , the set of points to one side of the plane, and  $C_n = C_n - C_i$ .
  - (d) Compute the new values of  $\mu_n, \Sigma_n, \mu_i, \Sigma_i$ .

The algorithm is illustrated in Figure 4.9 for  $k = 3$  clusters. Increasing the number of clusters provides a more accurate model at the expense of more computations. As recommended in Rother et al. [RKBo4], we set  $k = 5$  for modelling the background distribution. Each cluster  $C_i$  provides one Gaussian component in the Gaussian mixture model. The weights  $\omega_i$  are set to the number of points in each cluster normalised with the total number of sample points.

Having determined the Gaussian mixture model for a user marked sample of the facade image, we can then proceed with our segmentation. Assigning each pixel a probability of belonging to the sample distribution, i.e., the background wall texture, we get the result in Figure 4.10a. We can then binary threshold this probability image to achieve a segmentation of the facade elements (See Figure 4.10b). The result shows much improvement over the previously discussed segmentation methods. The functions  $Vert(x)$  and  $Hor(y)$  calculated on image  $I(x, y)$  can then be taken to be,

$$Vert(x) = \sum_{y=0}^{I_{width}} p(x, y)$$

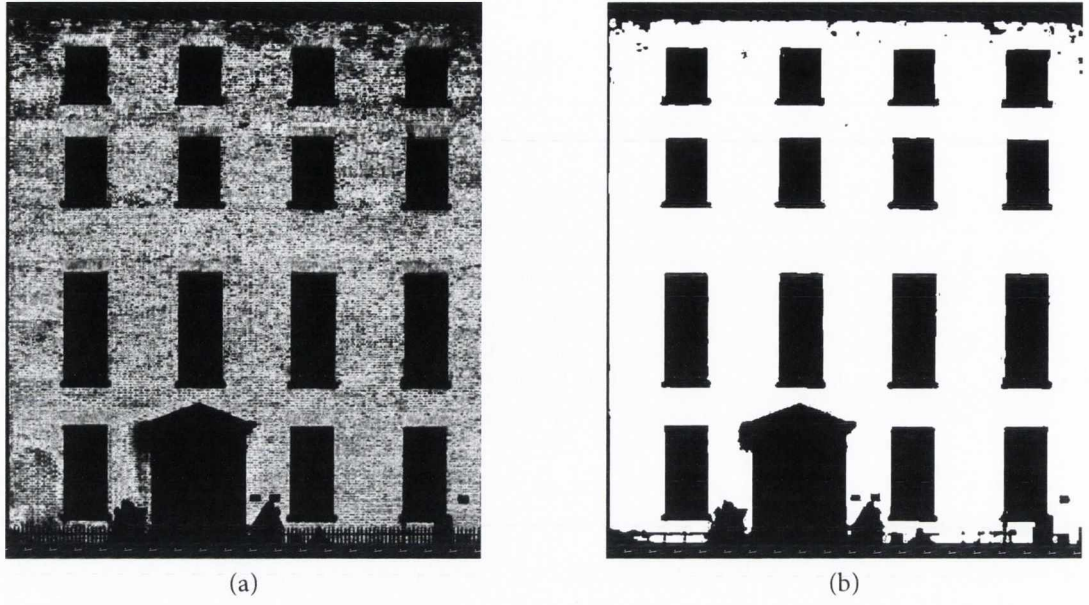


Figure 4.10: (a) Probability that pixels belong to the wall texture. White values indicated high probabilities. (b) Binary threshold of the probability image with threshold  $T = 0.1$ .

$$Hor(y) = \sum_{x=0}^{I_{height}} p(x, y)$$

Where splitting points of these functions can be found using the same approach proposed in Section 4.2.1.

### 4.3 Symmetry Matching

After subdividing the input facade into tiles, it is then possible to find symmetry relations among them. Many tiles will contain the same architectural elements and it is necessary to detect this if we are to represent the facade using a production rule set. Template matching can be used to detect repeating tiles in the facade. Taking one tile to be the template image  $T$  and a neighbouring tile to be the search image  $S$ , normalised cross correlation can be used to see if they match:

$$NCC(x, y, S, T) = \frac{1}{n-1} \sum_{i=1}^{T_{rows}} \sum_{j=1}^{T_{cols}} \frac{(S(x+i, y+j) - \bar{S})(T(i, j) - \bar{T})}{\sigma_s \sigma_t}$$

Where  $\bar{T}$ ,  $\sigma_t$ ,  $\bar{S}$  and  $\sigma_s$  are the means and standard deviations of the template and search images respectively. Parameters  $x$  and  $y$  specify an offset between the template and search image to be matched. Since the template and search tiles are not guaranteed to overlap directly, we must

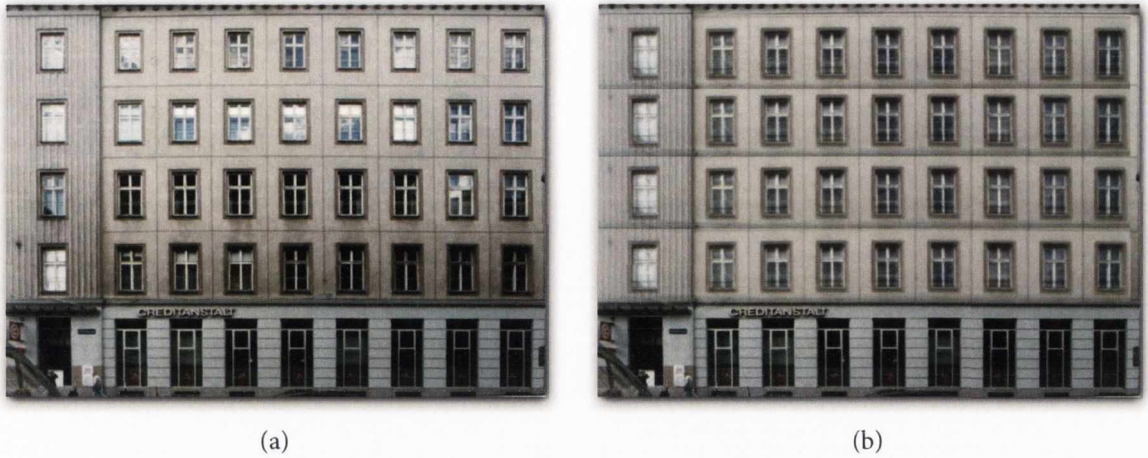


Figure 4.11: (a) Input facade image. (b) Matching tiles can be averaged to remove lighting variation and noise.

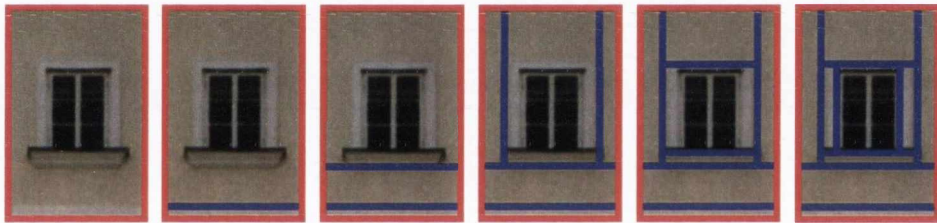


Figure 4.12: Facade tiles are iteratively subdivided into *regions* according to their gradient image. These *regions* are then recursively subdivided. Müller et al. [MZWVG07]

translate the template within the search image:

$$M(S, T) = \sum_{x=-m}^m \sum_{y=-m}^m NCC(x, y, S, T),$$

where  $m$  specifies how much the template can move about in the search image. Having determined repeating tiles within the facade image, we can use this symmetry information to assist further processing algorithms. Figure 4.11 illustrates how we can remove lighting variation by averaging the values between matching tiles.

## 4.4 Extracting Elements

Further subdivision of the facade is necessary to extract individual facade elements. This requires a detailed segmentation. Müller et al. [MZWVG07] propose an edge based approach to hierarchically split facade tiles. This idea is illustrated in Figure 4.12 and summarised as follows:

1. Initialise all tiles.



2. Find the best split candidate for each region.
3. Synchronise splits between tiles.
4. Subdivide regions into new regions and mark non-subdivided and small regions as leafs.
5. Repeat from step 2 until all regions are marked leafs.

The main steps of the algorithm involve detecting a suitable split location and synchronising among detected translational symmetries within the image. Detecting these translational symmetries was discussed in Section 4.3 and involved finding similar tiles within the facade image. Steps two and three of the algorithm will now be discussed in detail.

### Find Split Candidates

Split candidates correspond to horizontal and vertical edges in the facade tile. This can result in either a top, bottom, left or right split, corresponding to the nearest edges parallel to the top, bottom, left or right tile border. Out of these four candidates, the longest is chosen with the restriction that it must not cross a different candidate edge. This is illustrated in Figure 4.13.

### Synchronise Splits

The split candidate selection step discussed above is very sensitive to image noise, often leading to undesirable splits. By synchronising splits among similar tiles, much better results can be achieved. Split candidates are chosen for each tile in a group of similar tiles. The most common split choice among these tiles is then forced onto all tiles within the same group. Vertical and horizontal splits are then aligned for each column and row of tiles. This is akin to calculating split candidates on the averaged facade image of Figure 4.11.

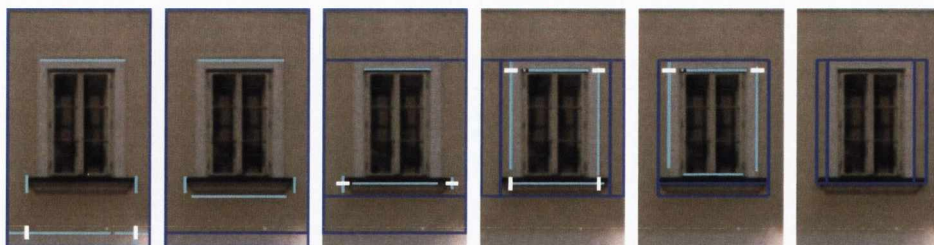


Figure 4.13: Split candidate edges (cyan) are chosen according to their length and proximity with the image border. Some candidate edges cross other edges (white) and cannot be chosen. Facade regions are then iteratively subdivided by choosing the best split candidates (blue). Müller et al. [MZWVG07]

Like all edge based methods, this approach is sensitive to background texture edge responses.

Furthermore, the criteria to terminate the splitting procedure need to be better defined. The algorithm will often split through edges belonging to window panels, overly dividing the facade tile. We improve on this by only considering edges belonging to the boundary of segmented facade elements.

#### 4.4.1 Detailed Segmentation

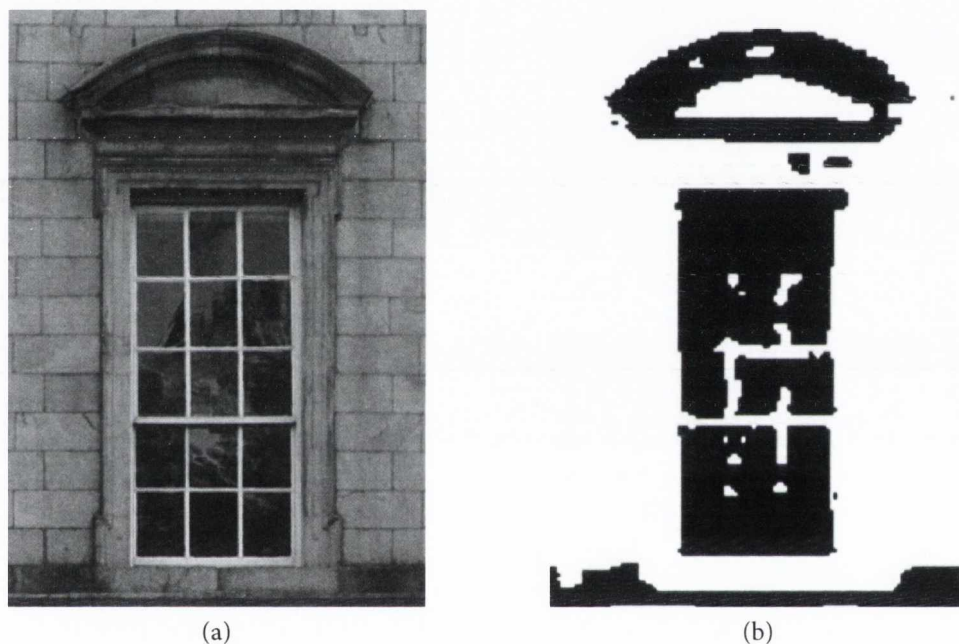


Figure 4.14: Binary segmentation using the Gaussian mixture model approach from Section 4.2.2 fails to segment facade elements with similar colour distribution to the wall. An approach that takes gradient information into account is required for an accurate segmentation.

We wish to determine the contours of facade elements within each facade tile. This amounts to performing a detailed segmentation of facade elements within each tile. One might be tempted to choose the initial segmentation based on Gaussian mixture models that we used to subdivide the facade into tiles, as discussed in Section 4.2.2. However, in some cases where the facade elements are the same colour as the background walls, a detailed segmentation is not obtained. This is illustrated in Figure 4.14 where the window border and cornice have the same colour distribution as the surrounding walls. Given the ill-posed nature of segmentation, some user interaction is typically required to fix incorrectly classified regions. Recent segmentation techniques recognise this problem and allow the user to guide a semi-automatic process. These algorithms take into account gradient information as well as the colour distribution to provide a robust segmentation with minimal user interaction, and are thus more suitable for the purpose of facade segmentation than an unsupervised approach.



## Graph Cuts

Image segmentation using graph cuts, as proposed in Boykov and Jolly [BJ01], is an interactive method that takes both gradient and intensity information into account. The technique has proven to be very successful in segmenting foreground objects from background textures, with minimal user interaction, and thus seems appropriate for the problem at hand.

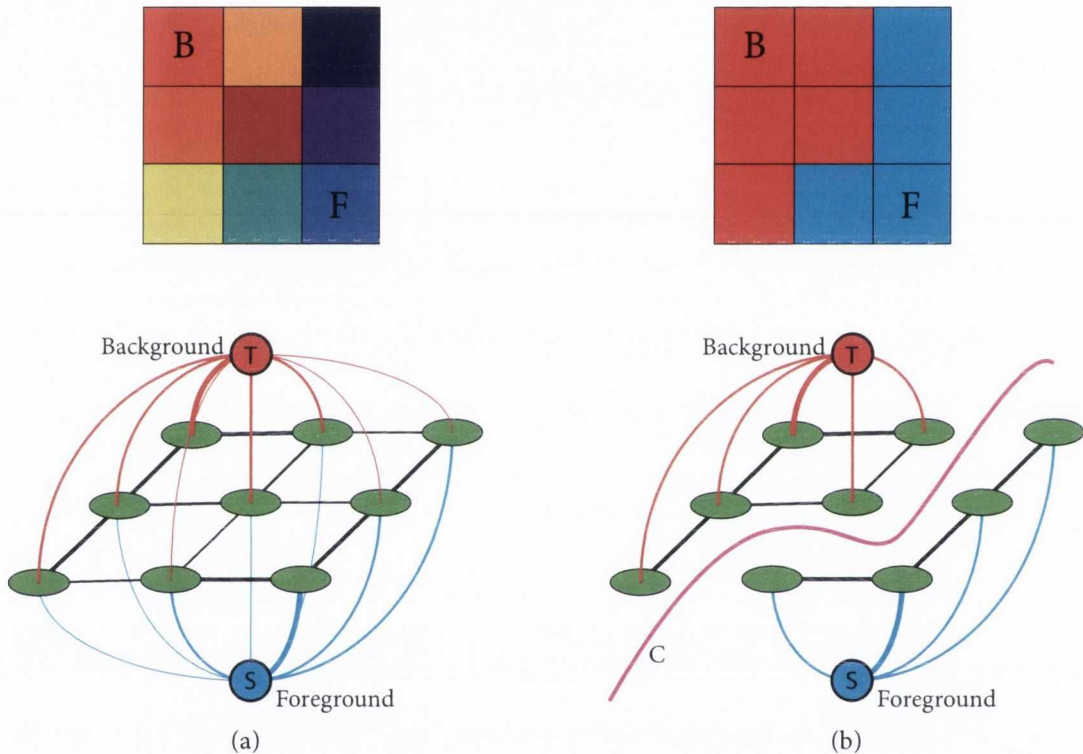


Figure 4.15: (a) Graph is constructed from image. N-links are defined between neighbouring pixels (black edges) and T-links connect the pixel nodes to a background (source) and foreground (sink) nodes. Weights are assigned for both N-links and T-links, specifying an energy function for the segmentation. (b) An optimal segmentation can be found as a minimum cut  $C$  through the graph, separating the background and foreground nodes.

For a given image, an undirected graph  $G = (V, E)$  is constructed with nodes  $V$  and edges  $E$ . Each pixel contributes one node in the graph. Neighbouring pixels are connected with edges called N-links, so that each pixel is connected to eight neighbouring pixels. As illustrated in Figure 4.15a, there are also two additional special nodes called terminals with edges (T-links) connecting to every pixel. These terminal nodes represent the background and foreground in a binary segmentation. Each edge  $e$  in the graph is assigned a weight  $w_e$ . Figure 4.15b illustrates that a cut  $C$  in the graph separates the two terminal nodes and has a cost  $|C|$ , given as:

$$|C| = \sum_{e \in C} w_e \quad (4.7)$$



The weights  $w_e$  depend on whether or not an edge  $e$  is an *N-link* or a *T-link*. N-links represent a cost for placing a boundary between pixels, which should be low for regions of high gradient and high for a regions of low gradient. T-links represent the probability that a pixel belongs to some sample distribution that models either the foreground or background respectively. Sample distributions are learnt from a user defined seed, i.e., a marked sample of the foreground and background regions. In Boykov and Jolly [BJ01] a normalised intensity histogram is learnt from user marked strokes in the image, while Rother et al. [RKBo4] use a GMM. Like Rother et al. [RKBo4], we use a GMM to model the foreground and background distributions. The cost for cutting foreground and background T-Links are  $R_{fore}(p)$  and  $R_{bkg}(p)$ , the log likelihood of pixel  $p$  belonging to the foreground and background GMMs respectively:

$$R(p) = -\log \sum_{i=1}^k \omega_i N(p|\mu(i), \Sigma(i)) \quad (4.8)$$

This defines a regional condition, encouraging segmentations to favour regions of homogenous colour. Weights given to N-links between pixels  $p$  and  $q$  are given by the boundary function  $B(p, q)$  as:

$$B(p, q) = \frac{\gamma}{\text{dist}(p, q)} \exp\left(-\frac{\|p - q\|^2}{2\sigma^2}\right) \quad (4.9)$$

Where  $\gamma$  is a constant that can adjust how much the segmentation favours the boundary or regional conditions,  $\text{dist}(p, q)$  is simply the distance between neighbouring pixels, and  $\sigma^2$  is the variance of neighbouring pixels (i.e., an estimation of noise). This boundary function encourages segmentation boundaries to snap to gradient contours. With weights defined for all edges in the graph, the optimal segmentation can be found by finding the minimum cost graph cut  $\hat{C}$  that separates the terminal nodes. This can be found using the max-flow algorithm of Ford and Fulkerson [FF62], which is given as follows:

1. Initialise graph so that every edge  $e$  has a weight  $w_e$ .
2. Find a path  $p$  from source node  $T$  to sink node  $S$ .
3. Calculate the flow  $f_p = \min \{w_e : e \in p\}$ .
4. Subtract  $f_p$  from all forward edges  $e = \{u, v\} : e \in p$  and add  $f_p$  to all backwards edges  $e = \{v, u\} : e \in p$ .
5. Repeat from step 2 until no new path with flow  $f_p > 0$  can be found.

Finding the max-flow of a graph is identical to finding the minimum cut. The minimum cut  $\hat{C}$  is simply the set of edges with leftover weights  $w_e = 0$  as calculated from the max-flow algorithm (See Figure 4.16). Graph cuts provide detailed segmentation but require the user to

select suitable foreground regions, i.e., segments of facade elements. Better results can be obtained by iteratively refining graph cuts using the GrabCut technique of Rother et al. [RKBo4].

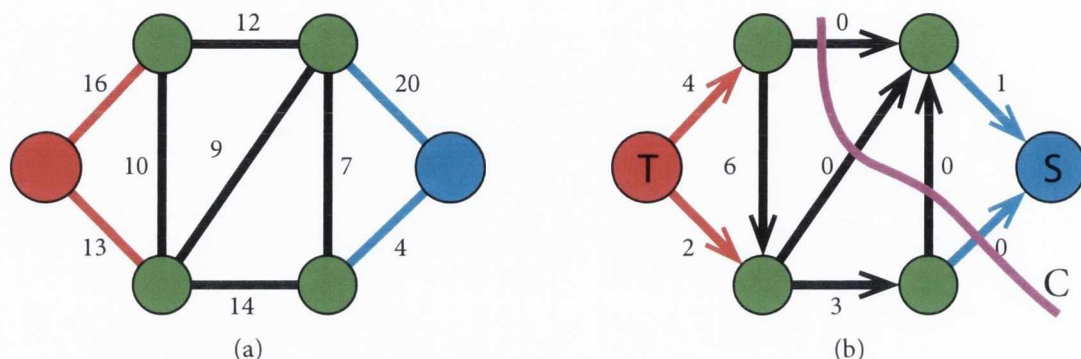


Figure 4.16: (a) Initial graph with source node  $T$  and sink node  $S$ . Edge weights  $w_e$  are given adjacent to each edge. (b) After calculation of max-flow, the min-cut is the set of edges with  $w_e = 0$ .

### GrabCut

GrabCut as proposed by Rother et al. [RKBo4] is an iterative graph cut technique that provides optimum segmentations with little user interaction. Only a sample of the background region needs to be indicated by the user and foreground regions are automatically detected. The approach works using a trimap  $T$  that indicates whether a pixel is part of the background  $T_{bkg}$ , foreground  $T_{fore}$  or unknown  $T_{unk}$ . The user provides a background sample by creating a rectangle around an object to be segmented. All pixels outside of this rectangle are initialised as the background and everything else is marked as unknown. The algorithm is summarised as follows:

1. Initialise trimap  $T$  by selecting a rectangle around an object to be segmented, every pixel outside the rectangle is labelled as  $T_{bkg}$  and every pixel inside is labelled as  $T_{unk}$ .
2. Calculate two Gaussian mixture models, one for pixels in the  $T_{unk} \cup T_{fore}$  class and one for pixels in the  $T_{bkg}$  class.
3. Assign the most likely Gaussian component to each pixel belonging to the same class as the GMM.
4. Recalculate the Gaussian components using the newly assigned pixels.
5. Perform a graph cut using the technique discussed in the previous Section, assigning suitable pixels from  $T_{unk}$  to the  $T_{bkg}$  class.



6. Repeat from step 2 until convergence.

The user may in addition mark segments of the image as known foreground regions by labelling them as  $T_{fore}$ . These will not be changed in the segmentation but will change how the GMMs are calculated, thereby updating the segmentation. We can achieve highly accurate segmentations using this approach. Results are given in Figure 4.17.



Figure 4.17: Facade elements segmented using GrabCut. Left: Input facade images. Right: Can achieve very accurate segmentations even when facade elements resemble the wall texture.

### Contour

Having obtained accurate segmentations of facade elements, we can proceed to obtain contour edges for use with the subdivision algorithm of Müller et al. [MZWVG07]. Contours are extracted by following the boundary of segmented objects. However, some facade elements might be connected together that need to be manually separated. We achieve this by letting the



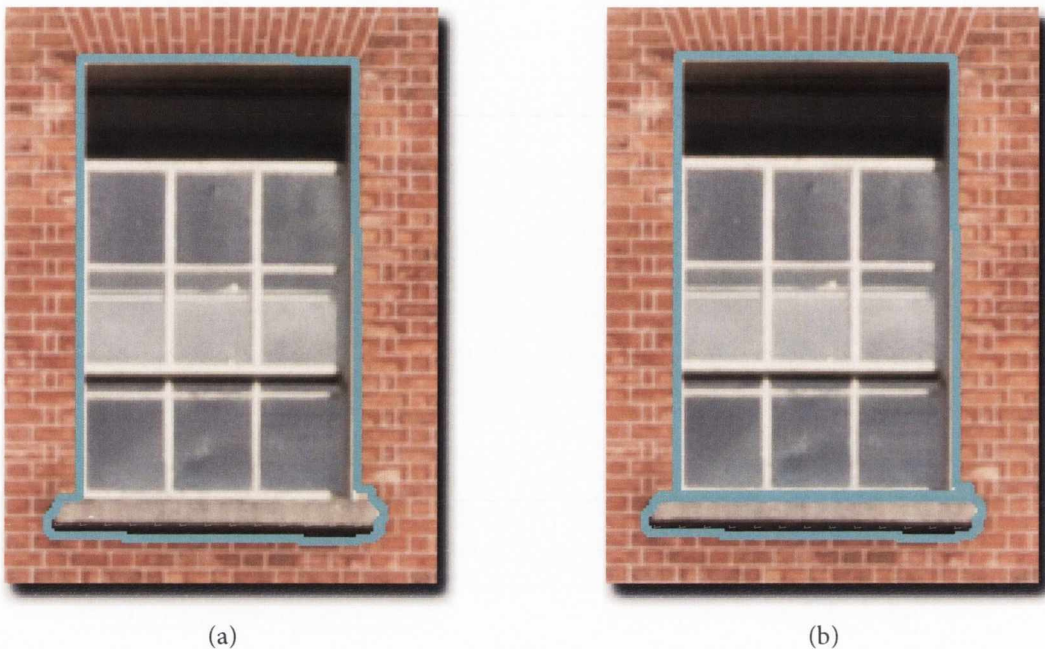


Figure 4.18: (a) Segmentation results in the window and windowsill being joined together. (b) User can select a split point to separate the two facade elements.

user select a split point on the segmented foreground, as depicted in Figure 4.18. This selection is guided by snapping to dominant edges in the image as calculated using the Canny edge detection technique from [Can86]. Some segmented facade elements are not rectangular, and in order to facilitate the splitting process we represented them with their bounding box. The resulting contour image for an input facade image is shown in Figure 4.19.

## 4.5 Rule Derivation

The splitting hierarchy is recorded in the form of a derivation tree. This allows a one to one mapping between facade image splits and split grammar production rules. The symmetry information from Section 4.3 is used to define *repeat* rules in the derived production set. We allow the user to assign labels to segmented facade elements, which are then used as terminal symbols in the production system. Since there is no depth information in the facade images, we require the user to manually adjust the depth of facade elements. Each depth adjustment defines one branch on the derivation tree, corresponding to an *extrude* command. The derived production system is capable of reproducing facades and extending them to fit buildings of any shape, providing there is enough repetition in the facade image.

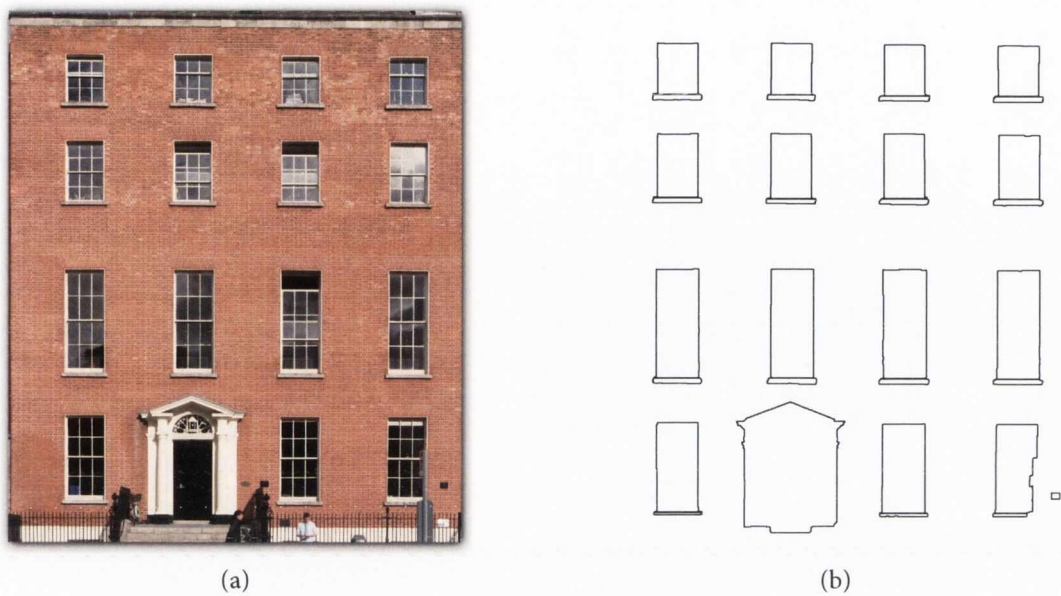


Figure 4.19: (a) Input facade image. (b) Extracted contour image can be used with the splitting technique of Müller et al. [MZWVG07].

## 4.6 Summary

A summary of our facade segmentation approach for inverse procedural modelling is illustrated in Figure 4.20. The four main steps are as follows:

1. Rectify and crop input facade photograph.
2. Select area of wall texture and compute its GMM. From the GMM calculate foreground and background estimates. Compute the accumulation functions in both horizontal and vertical directions, and split the image along maximums.
3. Perform the GrabCut image segmentation technique on each facade tile. Find repetitions in the image using normalised cross correlation on foreground regions. If necessary, split foreground regions into separate facade elements. Give each facade element an identifying label.
4. Iteratively split along foreground contours using the method of Müller et al. [MZWVG07]. CGA shape grammar rules can be inferred with each split, and terminal symbols correspond to facade element labels. Depth can be manually adjusted for each facade element.

The results of applying this method on real world photographs can be seen in Figure 4.21.

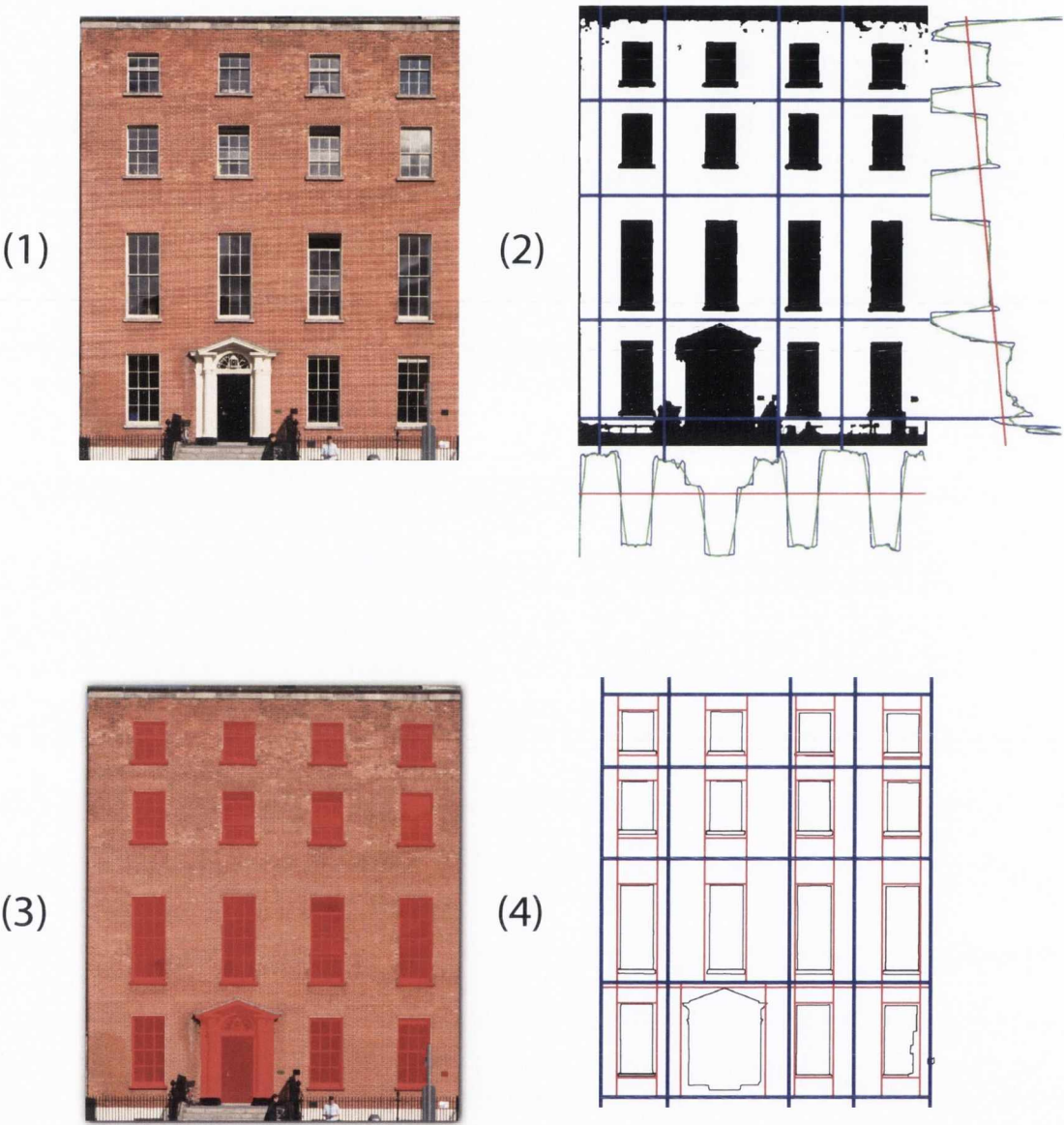


Figure 4.20: Summary of segmentation method: (1) Rectify input photograph. (2) Compute background GMM and subdivide the facade into tiles. (3) Perform the GrabCut segmentation technique on each facade tile. (4) Iteratively split along foreground contours using the method of Müller et al. [MZWVG07].





Figure 4.21: Results of applying our segmentation method to three different facades. Left: Input rectified photographs. Middle: Segmented facade elements. Right: Resulting 3D facade models after manual depth adjustment. Shaders were added to the windows to provide reflection effects.

## 4.7 Discussion

We have investigated how image segmentation techniques may be used to derive a production grammar capable of reproducing an input facade image. We have explored several segmentation algorithms for this purpose. Unsupervised methods such as binary segmentation, k-means clustering and GMM were suitable to give initial clues as to where facade elements are likely to be in the image. However, since there are many different styles of architectural elements to consider, it is unlikely that any unsupervised method could give correct segmentations of the individual facade elements in all possible input images. Shadowing and differing lighting conditions adds to this problem, prompting the use of (semi) supervised methods to give robust segmentations through minimal user interaction. Our segmentation approach complements the split based decomposition of facade images proposed in Müller et al. [MZWVGo7], utilising modern segmentation algorithms based on graph cuts to determine the structure of complex facades. A facade decomposition using the discussed techniques allows a one to one mapping with the production rules of the CGA shape grammar discussed in Section 3.1.4.

This inverse procedural modelling approach removes the need for modellers to write complex production rules to generate building facades. A photograph is all that is needed, and a production rule set can be derived within minutes with minimal interaction. The results in Figure 4.21 show how this process can be applied to different building styles to produce 3D models. The production rules derived from these models allow the facade structure to be applied to arbitrarily shaped buildings. However, this approach can only reproduce the given input facades, albeit with precision. In order to generate more variation, the system should learn how to produce new facade structures from examples. In the next Chapter we will investigate how symmetry relations between facade elements may be exploited to infer stochastic production grammars capable of generating new hybrid variations.

## Chapter 5

# Symmetry Hybrids

**S**YMMETRY is prevalent in both man-made and natural objects and plays a key role in aesthetics. Indeed, how we perceive the world is largely governed by symmetry [Koh92]. Patterns of repeating elements are present all around us from architecture to flower gardens.

Inverse procedural modelling involves detecting symmetry relations and representing them with a set of production rules. Procedural descriptions can accurately reproduce the input pattern while offering a highly compressed representation. The rules provide high level editing capabilities and can produce variation if desired.

Many patterns have a similar appearance despite containing different repeating elements. Building facades are composed of differing architectural elements, whereas the symmetry relations between them are very similar. Cornices above windows have widely different appearance among architectural styles and yet we can identify them based on their symmetry relation with the windows.

This chapter explores how symmetry can be recognised and used to generate hybrid patterns from examples. By interchanging similar symmetry relations between patterns, we can produce many variations while preserving the overall look of the input patterns. We propose this a solution to the difficulties of writing production rules to create many varying building facades.

Methods to detect the hierarchical symmetry relations among shapes are presented in Section 5.1. We propose the use of a symmetry tree to represent such patterns. In Section 5.2 we investigate how we may graft different symmetry trees together to produce hybrid patterns. We will look at how certain contextual relationships between shapes may be preserved with the introduction of an indexed grammar. The results of our methods are presented in Section 5.3 and the drawbacks and possible directions for future work is discussed in Section 5.4.



## 5.1 Symmetry Detection

Symmetry plays an important role in the aesthetic appeal of artistic patterns. Familiar geometric symmetry types in 2D are illustrated in Figure 5.1 and listed as follows:

**Rotational Symmetry** An object has  $n$ -fold rotational symmetry if it can be rotated by  $360/n$  degrees about a point and not change its appearance.

**Reflectional Symmetry** An object has reflectional symmetry if its appearance does not change with a reflection about a line.

**Translational Symmetry** An object has translational symmetry if its appearance does not change with a particular translation.

**Glide Reflection** An object has glide reflection symmetry if its appearance does not change with a translation and reflection.

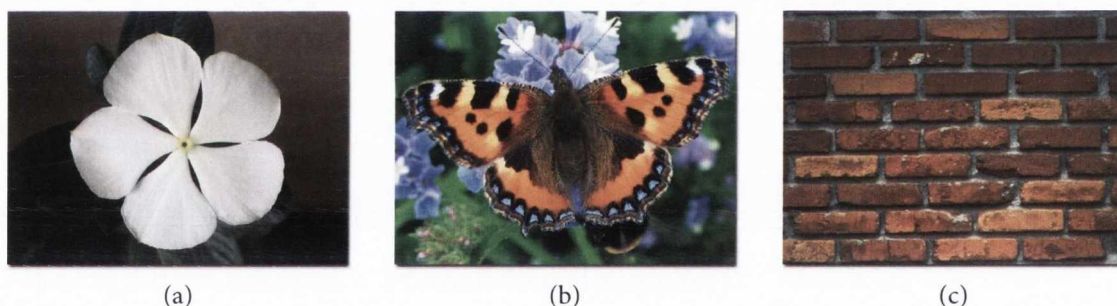


Figure 5.1: (a) The flower has a 5-fold rotational symmetry. (b) The butterfly has reflectional symmetry. (c) Bricks have translational symmetry.

Geometric symmetries are subgroups of isometries (distance preserving transformations) that are invariant under composition of the operation. Detecting such symmetry relations within patterns or objects has long been of interest in Computer Science. Of the many techniques proposed, recent voting algorithms have had the most success with real data. These techniques consider all pairwise transformations between data points, where each transformation has a vote in the space of all possible transformations. Clustering among these points reveals transformations that occur many times, suggesting possible symmetry relations.

### 5.1.1 Transformation Voting

We take as input a set of labelled shapes with known coordinates representing the atomic elements within an input pattern, e.g., segmented windows and doors from a photograph of a

building, see Chapter 4. Following from Stava et al. [vBM\*10], we define transformations between pairs of identical shapes  $a$  and  $b$  as  $t_{ab} = t_a^{-1}t_b$  where  $t_a$  is a 3-tuple  $\langle \alpha, d, \beta \rangle$  representing the position in polar coordinates (angle  $\alpha$  and distance  $d$ ), and orientation  $\beta$  of shape  $a$  (see Figure 5.2a). As in Mitra et al. [MGPO6], we take the transformations between every pair of shapes and place a point in transformation space. Unlike Mitra et al. [MGPO6], however, we are only concerned with 2d patterns, thus creating a 3d transformation space with each point representing a transformation  $t_{ab} = \langle \alpha, d, \beta \rangle$ . This is illustrated in Figure 5.2.

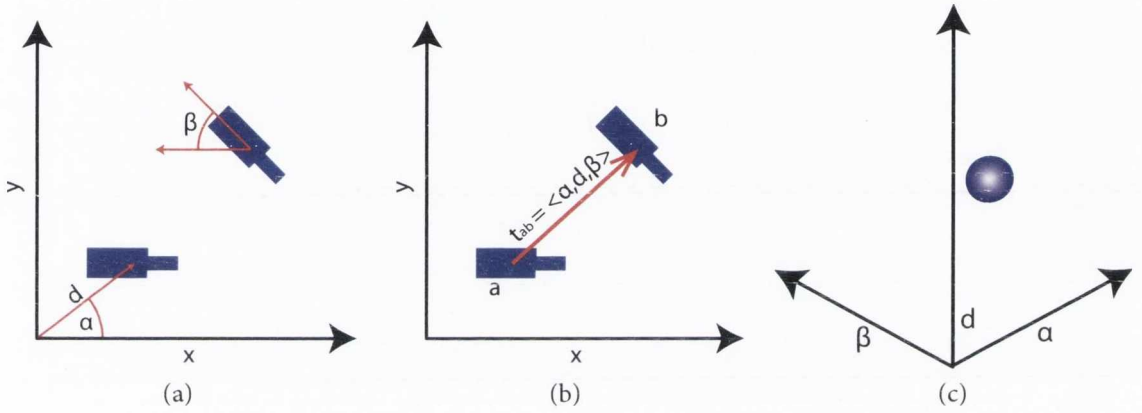


Figure 5.2: (a) Shapes are represented with polar coordinates  $(\alpha, d)$  and orientation  $\beta$ . (b) Pairwise transformations between shapes are represented with a 3-tuple  $t_{ab} = \langle \alpha, d, \beta \rangle$ . (c) Each transformation corresponds to one point in transformation space.

### 5.1.2 Clustering

Given a transformation space populated with points that represent every possible transformation between input shapes, it is necessary to determine which transformations correspond to the strongest symmetry relations. Each point provides evidence of a symmetry relation between pairs of shapes, and can be thought of as voting for that particular relation. Due to noise in the input data, transformation points will rarely coincide with each other, thus giving rise to a clustering problem. Transformation points representing similar transformations must be grouped together if they are to agree on a specific symmetry relation. Any solution to this clustering problem must take into account the following:

1. We do not know the number of clusters, i.e, the number of possible symmetry relations present.
2. We can make no assumptions about the shape of the clusters.

These constraints render many standard parametric clustering methods, such as k-means clustering (MaQueen [Mac67]) unsuitable. Mean-Shift clustering is a non-parametric method

that satisfies these constraints. It was first proposed in 1975 by Fukunaga and Hostetler [FH75] and was extended for use in computer vision by Cheng [Che95] and Comaniciu and Meer [CMo2]. The algorithm is summarised as follows:

1. Create a window around each point in the dataset.
2. Calculate the mean of the data within the window.
3. Shift the window towards the mean and repeat until the window converges on a point.

We can calculate the mean of the window using kernel density estimation. A kernel is a function used to estimate the pdf of a random variable. There are many possible kernel functions  $K(x)$ , which must satisfy  $\int_{\mathcal{R}^d} K(x) dx = 1$  and  $K(x) \geq 0$ , where  $x$  is a vector of  $d$  dimensions. Some commonly used kernels are as follows:

Uniform: 
$$K(x) = \begin{cases} \frac{1}{2} & \text{if } \|x\| \leq 1 \\ 0 & \text{else} \end{cases}$$

Gaussian: 
$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\|x\|^2}$$

Epanechnikov: 
$$K(x) = \begin{cases} \frac{3}{4}(1 - \|x\|^2) & \text{if } \|x\| \leq 1 \\ 0 & \text{else} \end{cases}$$

The kernel density estimator for a set of  $n$  points of dimension  $d$ , using a kernel function  $K(x)$  and bandwidth  $h$ , is then given as:

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right) \quad (5.1)$$

The bandwidth  $h$  determines the window size, and shifting the window can be thought of as a gradient ascent method operating on the kernel density estimator:

$$x_{t+1} = x_t + \kappa \nabla \hat{f}(x_t) \quad (5.2)$$

The position  $x$  of the window is moved in the direction of rising density of the kernel density estimator  $\hat{f}$ , i.e., in the direction of the gradient vector  $\nabla \hat{f}$ . A small constant  $\kappa$  sets the size of each step along the gradient vector, ensuring that  $\hat{f}(x_{t+1}) \geq \hat{f}(x_t)$ . The mean shift vector  $m(x)$  for moving the window is then:

$$m(x) = \frac{\sum_{i=1}^n \nabla K\left(\frac{x_i - x}{h}\right) x_i}{\sum_{i=1}^n \nabla K\left(\frac{x_i - x}{h}\right)} - x \quad (5.3)$$



All windows will then converge on local maximums of the kernel density estimator, which represent the cluster centres. Starting points that converge on the same local maximum are grouped into the same cluster. Comaniciu and Meer [CM02] recommend using the Epanechnikov kernel for density estimation but we found that the simple uniform kernel worked well for our purposes. The cluster grouping is largely dependent on the choice of bandwidth  $h$  and a suitable bandwidth must be chosen for the data. We found that simply normalising the data dimensions was sufficient for using a set bandwidth for all examples.

### 5.1.3 Regularly repeating elements

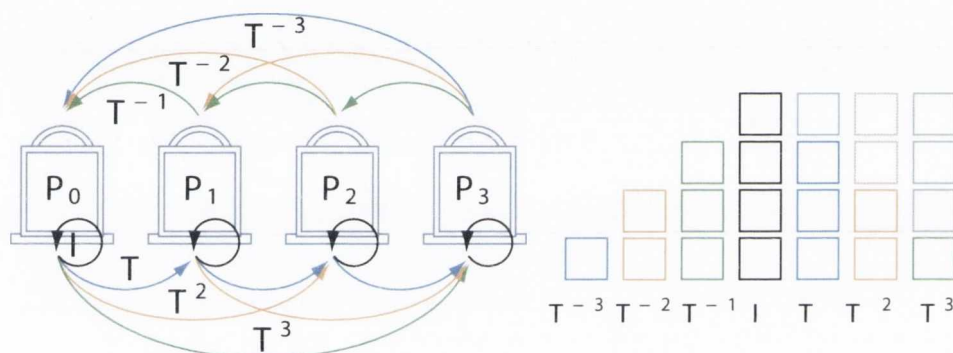


Figure 5.3: In a regular structure, transformations form an accumulative pattern. Image from Pauly et al. [PMW\*08]

Dense clusters in transformation space provide strong evidence of symmetry relations. However, the strongest cluster may not correspond exactly to the best symmetry relation that describes the pattern of the shapes. Consider the fish in Figure 5.4a, whose arrangement approximates a grid pattern with strong evidence of a translational symmetry. Clustering in transformation space reflects this grid layout, but no one cluster gives the optimum translational symmetry to fully describe the pattern (See Figure 5.4b).

Pauly et al. [PMW\*08] propose exploiting the composition invariant property of symmetries to better understand structures of regularly repeating elements. Transformations between regularly repeating shapes will consist of some composition of the symmetry operation that minimally describes the pattern. This is illustrated in Figure 5.3 where transformations between any two windows can be expressed as some composition of a minimal translation  $T$ . It is also shown how this leads to an accumulation pattern in the transformation space.

The optimal symmetry relation that describes the input pattern can be recovered from transformation space by considering all of the clusters together. If the shapes in the input pattern exhibited perfect regularity then the transformation clusters would form a perfect grid. Detecting the optimal symmetry relation then corresponds to finding the best fitting grid among

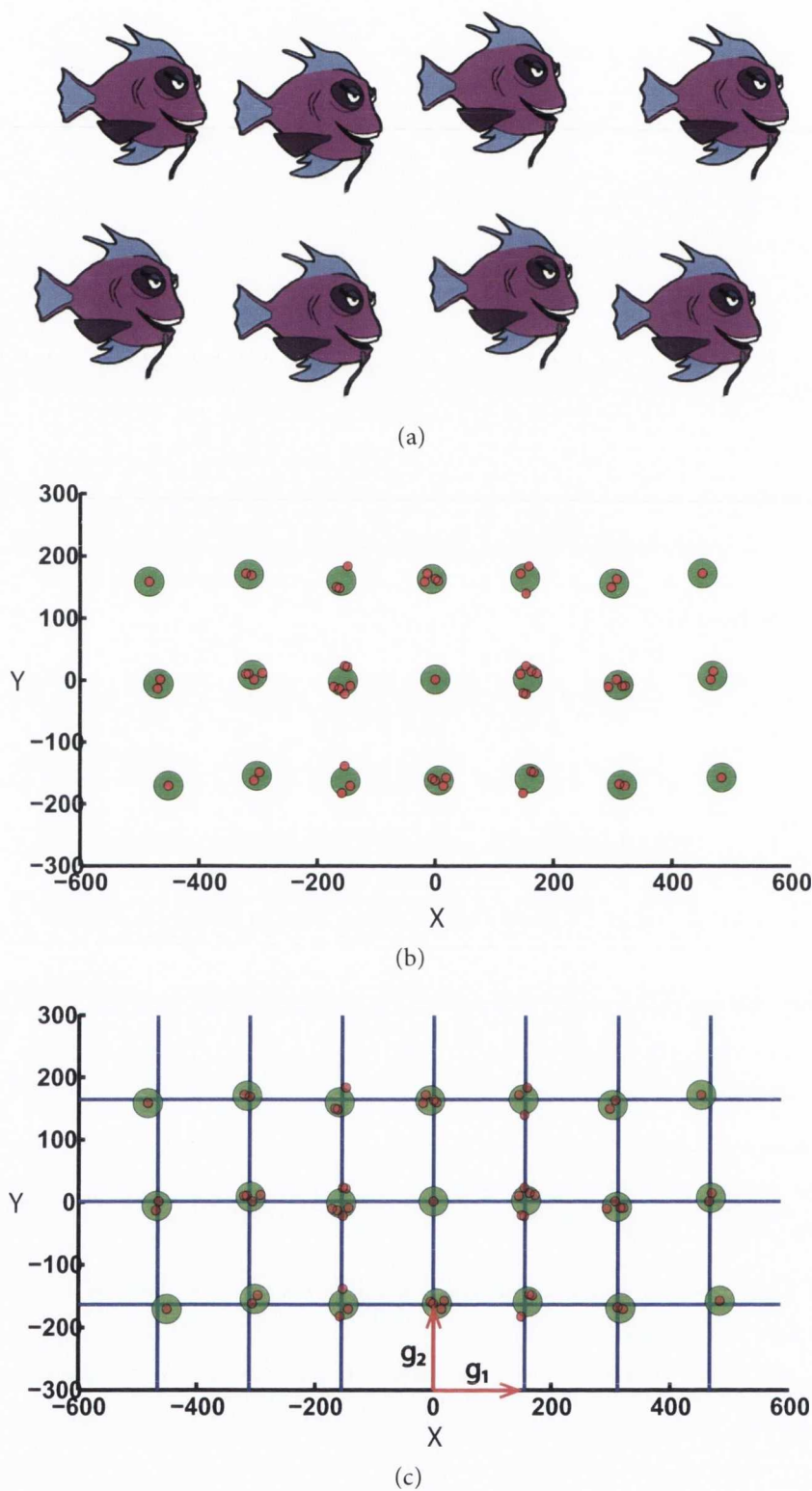


Figure 5.4: (a) The fish approximate a grid pattern. (b) Cluster layout reflects this pattern in transformation space. Each point represents a transformation between two fish (translation in Cartesian coordinates). (c) The optimal grid pattern with basis vectors  $g_1$  and  $g_2$  that describes the layout of the fish, i.e., how transformation clusters would be organised if the fish had perfect symmetry.

the clusters, i.e., how the transformation clusters would be organised if the pattern had perfect symmetry. Grid locations  $X = x_{ij}$  may be expressed parametrically using two basis vectors (grid generators)  $g_1$  and  $g_2$  so that  $x_{ij} = ig_1 + jg_2$ . The goal is to find the vectors  $g_1$  and  $g_2$  such that the distance between the grid locations and cluster centres is minimised. Pauly et al. [PMW\*08] achieve this using energy minimisation of a distance function  $E$ . The energy function  $E$  is composed of two terms  $E_{X \rightarrow C}$  and  $E_{C \rightarrow X}$ .  $E_{X \rightarrow C}$  is the sum of squared distances of the grid locations to the nearest cluster centres:

$$E_{X \rightarrow C} = \sum_i \sum_j \alpha_{ij}^2 \|x_{ij} - c(i, j)\|^2, \quad (5.4)$$

where  $c(i, j)$  is the cluster nearest to grid location  $x_{ij}$ . The second term  $E_{C \rightarrow X}$  is the sum of squared distances of the cluster centres to the nearest grid locations:

$$E_{C \rightarrow X} = \sum_{k=1}^{|C|} \beta_k^2 \|c_k - x(k)\|^2, \quad (5.5)$$

where  $x(k)$  is the grid location nearest to cluster  $c_k$ . Pauly et al. [PMW\*08] use the variables  $\alpha_{ij}$  and  $\beta_k$  as weights to measure the how reliably a grid location was mapped to a cluster, and require additional terms to be added to  $E$ . However, since we are dealing with labelled shapes, we are unlikely to have many outliers and found that setting  $\alpha_{ij} = 1$  and  $\beta_k = 1$  was sufficient, giving:

$$E = E_{X \rightarrow C} + E_{C \rightarrow X} \quad (5.6)$$

Grid generators  $g_1$  and  $g_2$  can be found by minimising  $E$ :

$$g_1, g_2 = \underset{g_1, g_2}{\operatorname{argmin}} (E) \quad (5.7)$$

The function can be minimised using non-linear optimisation. We used the Levenberg-Marquardt algorithm [Lev44] to achieve this. The algorithm is prone to converging on local minimums and the initial choice of  $g_1$  and  $g_2$  is very important to obtain the global minimum. We can make good estimations of  $g_1$  and  $g_2$  by simply taking the strongest clusters to initialise. Pauly et al. [PMW\*08] suggest taking clusters with  $g_i$  minimal on dominant lines through the origin. We found that simply applying weights to the clusters that favour shorter transformations with many points (according to the accumulation pattern), and taking the heaviest clusters to initialise, also led to good results.



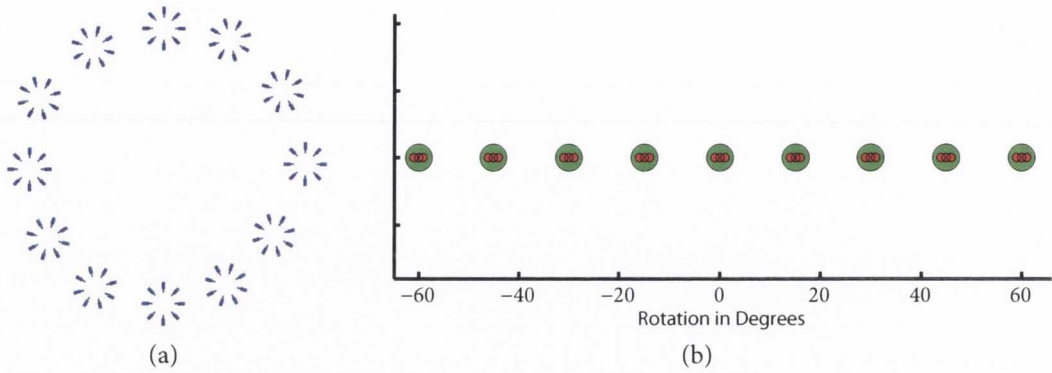


Figure 5.5: (a) Patterns can be composed of hierarchical symmetry relations between shapes. The blue shapes form cyclic groups of  $45^\circ$  which in turn form a cyclic group of  $30^\circ$ . (b) Grid pattern appears in transformation space (projected onto rotation subspace) with the x-axis giving the rotation component  $\beta$  (see Figure 5.2) of transformations in degrees. However, symmetry grids from different hierarchical levels are overlaid on top of each other, giving the appearance of a  $15^\circ$  dominant rotational symmetry, which is incorrect.

#### 5.1.4 Hierarchical Symmetries

Many patterns, including the layout of building facades, are formed from a hierarchy of symmetry relations. Detecting symmetry relations that minimally describe such patterns is not as straightforward as previously discussed. Consider the pattern in Figure 5.5a, composed of a hierarchy of rotational symmetries. Hierarchical structures like these produce many dense clusters of unimportant symmetries in transformation space (i.e., they do not minimally describe the structure). Figure 5.5b illustrates how simply detecting a grid of clusters in transformation space is not sufficient to detect the dominant symmetry relations. This is because grids belonging to different symmetry levels are overlapping.

Higher level symmetry relations do not present themselves until lower level symmetry relations are discovered. As in Pauly et al. [PMW\*08], we too will take advantage of the composition invariant property of symmetry. Consider the subset of shapes in Figure 5.6. Some transformations can be viewed as chains of smaller transformations that all provide evidence of the same symmetry relation. This is illustrated in Figure 5.6 where the transformation  $t_2$  is a power of  $t_1$ :

$$t_2 = t_1 t_1 t_1 = t_1^3$$

We can then divide transformation space into sets of transformations that are all powers of each other. This in effect declutters transformation space, thereby exposing the grid structure for different symmetry relations. However, it is still unclear which one of these sets represents the most informative symmetry relation.

Transformations of lesser magnitude generally represent more important (from the point of view of reconstructing the pattern) symmetry relations than those of higher magnitude. In

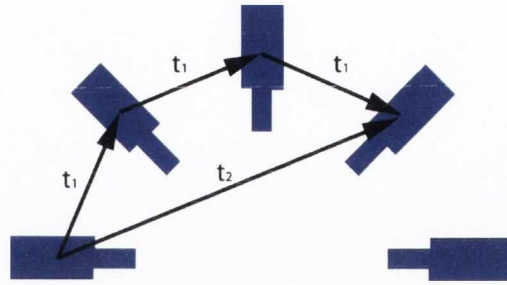


Figure 5.6: Larger transformations are composed of chains of smaller transformations within the same symmetry group. Here, transformation  $t_2$  is equivalent to performing  $t_1$  three times.

order to exploit this law of proximity, weights are assigned to transformation clusters according to their magnitude,  $w(t_i) = \sqrt{\alpha^2 + d^2 + \beta^2}$ . Let  $T$  be the list of all transformation cluster centres, sorted by their assigned weights:

$$T = (t_1, t_2, t_3, \dots, t_n) \mid w(t_i) \leq w(t_{i+1})$$

We iteratively divide transformation space into sets of transformations, with each set providing evidence of some symmetry relation. A new set  $S_i \subseteq T$  is created containing all transformations  $t_k$  which are some power of  $t_1$  (the symmetry with lowest magnitude). To account for variance in the data we define two transformations  $t_k$  and  $t_1^m$  to be equivalent if the distance between them is less than some small threshold  $\epsilon$ :

$$t_k \in S_i \mid d(t_k, t_1^m) < \epsilon$$

These cluster centres are then removed from the list so that  $T = T \setminus S_i$  and the process is repeated until  $T$  is empty. We are left with a collection of sets  $S$  which correspond to possible symmetry relations among the input shapes. Each set will expose a grid structure that may be exploited to find the optimal symmetry relation. We take the set  $S_i$  with highest cardinality and lowest weight and perform the energy minimisation technique of Pauly et al. [PMW\*08] (as discussed in Section 5.1.3) on its members to find the symmetry relation that minimises error.

Shapes are then grouped into patches according to this symmetry relation. This whole process is repeated by constructing a new transformation space consisting of transformations between patches, and terminates when only one patch remains (see Figure 5.7). This technique effectively describes the pattern from the lowest level symmetry relation up to the highest. We represent the detected hierarchical structure of the pattern using a tree, with each iteration creating one node that describes a symmetry relation.



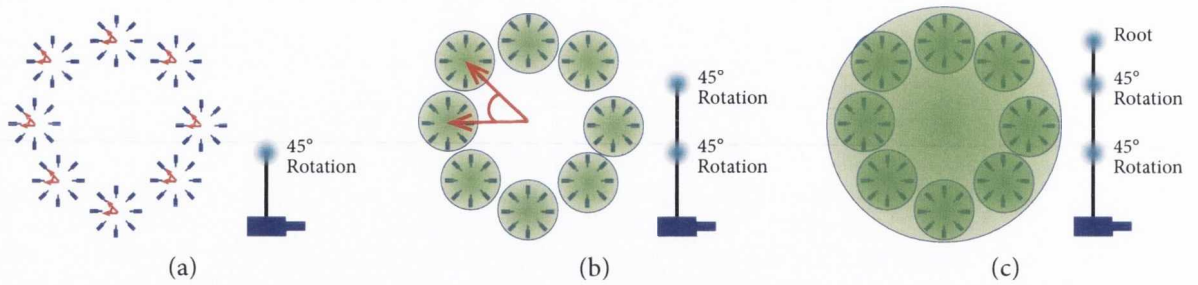


Figure 5.7: (a) Lowest level symmetry is detected first, creating a node in the symmetry tree. (b) Shapes are grouped into patches according to the detected symmetry. The next symmetry level is detected and the tree grows from the bottom up. (c) The process keeps iterating until only one patch is left, defining the root node of the tree.

### 5.1.5 Grouping

Patterns are usually composed of more than one type of shape. In the case of a building facade, there are many different architectural elements that have important semantic relations between them. For instance, windowsills accompany windows and can be grouped together. We wish to find a general way of grouping shapes together to synthesise new plausible patterns. It is therefore necessary to find these relations between different types of shapes in order to fully describe the pattern. Wang et al. [WXL\*11] propose a set of precedence rules that may be invoked to group sets of shapes together according to their symmetry relations. Shapes can either be grouped together with other identical ones, thereby forming a symmetry clique, or joined with other non-identical shapes, dubbed an assembly. Figure 5.8 demonstrates these two cases. Wang et al. [WXL\*11] set the following precedence rules describing when, and in what way shapes should be grouped:

**M1 (Grouping before assembly):** Identical shapes sharing the same symmetry relation should be grouped together before grouping with other shapes. This rule follows from the Gestalt law of symmetry [Koh29], which describes how humans perceptually group objects together according to symmetry. Grouping by symmetry removes redundancies, thus creating a compact representation of the pattern.

**M2 (Assembly before grouping):** Non-identical shapes are assembled together first if they possess equivalent grouping symmetries. In this instance the Gestalt law of proximity takes precedence over the law of symmetry. It allows symmetry to apply to a larger group of shapes, and forces all shapes in the group to be generated together.

We require that the symmetry trees of two similar patterns be ordered in a very definite way. The reason for this will become clearer when we discuss creating hybrid patterns later. Wang et al. [WXL\*11] define precedence rules for grouping by symmetry:



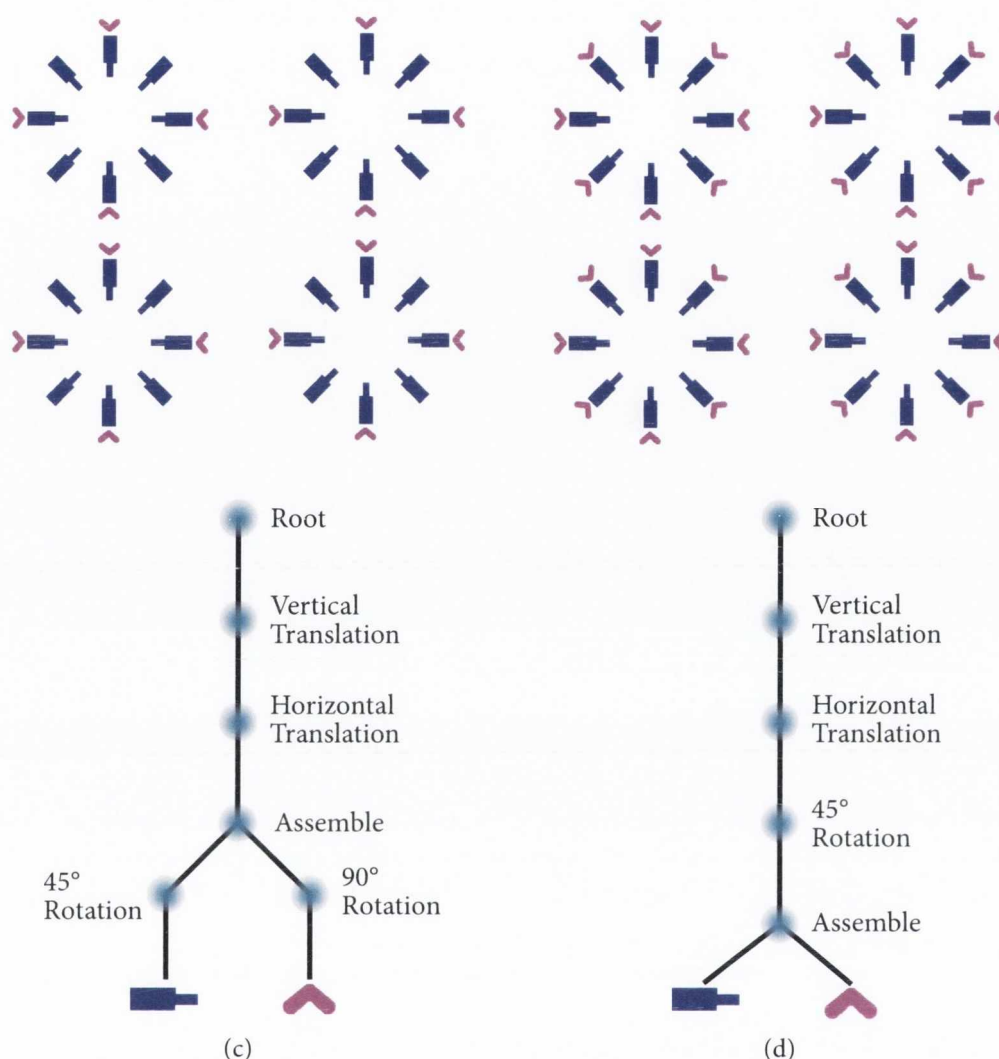


Figure 5.8: Two cases for grouping shapes: (a) Shapes must be grouped into their individual symmetry groups before being grouped together. (b) Shapes must be grouped together before being grouped into the same symmetry group.

**G1 (Clique order):** Symmetry cliques of higher order should be grouped before those of lower order.

**G2 (Reflectional Symmetry):** Symmetry cliques of order-2 should be grouped by reflectional symmetry before rotational and translational symmetries.

**G3 (Proximity):** If no other precedence can be set, then groups with shapes of closer proximity should be grouped first.

These symmetry grouping rules assure that perceptually similar patterns are grouped in a defined way and result in similar symmetry trees. Example situations of when each rule applies are illustrated in Figure 5.9. Likewise there are some precedence rules for assembling shapes

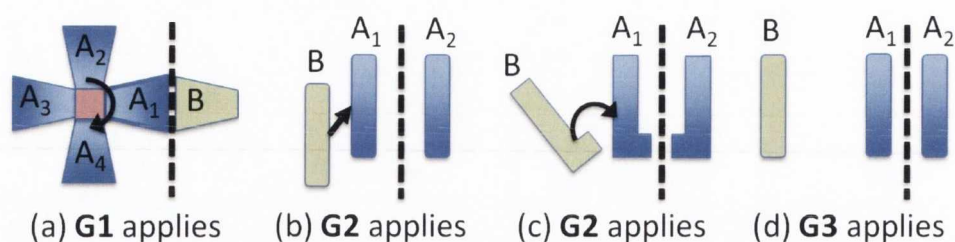


Figure 5.9: Precedence rules for grouping by symmetry.  $A_i$ 's are grouped before being grouped with  $B$ . Wang et al. [WXL\*11]

belonging to different symmetry cliques. However, these rules largely deal with connectivity of geometric shapes that are actually joined together, such as parts of a model. Since we are dealing with disconnected shapes belonging to patterns, we will not discuss these rules. It must be pointed out that when we refer to *assembly* rules, we take a more general definition than the strict assembling together of meshes in Wang et al. [WXL\*11]. We regard an *assembly* as simply a grouping of two shapes (or patches of shapes) with some transformation separating them. We simply use the term *assembly* to be consistent with the terminology used in Wang et al. [WXL\*11].

The grouping precedence rules discussed above allow groups of shapes to overlap. While this is desirable for many general patterns, building facades are an exception. If we wish to model building facades using split grammars, we cannot allow overlapping groups. Figure 5.10a demonstrates one case where overlapping of groups will result if we follow these precedence rules. We add the following constraint to all precedence rules:

**Overlap exception:** Any precedence rule that results in an overlapping group loses its precedence order, and a rule of lower precedence is allowed to precede it.

Taking this constraint into account, we ensure that no groups can overlap, thus resulting in the grouping order of Figure 5.10b.

## 5.2 Generating Hybrids

We have shown how a symmetry tree can be constructed to describe the structure of a pattern of shapes. This symmetry tree has a direct parallel with a context free deterministic grammar, and can be utilised to reproduce the input pattern exactly. However, this is not what we find most interesting about symmetry trees, but rather the fact that perceptually similar patterns will have very similar trees. Consider Figure 5.11, where both input patterns have similar symmetry trees and reflect a dominant rotational symmetry. The terminal nodes of both patterns serve the same *function*, i.e., the structure of the patterns will not change much if we interchange them.

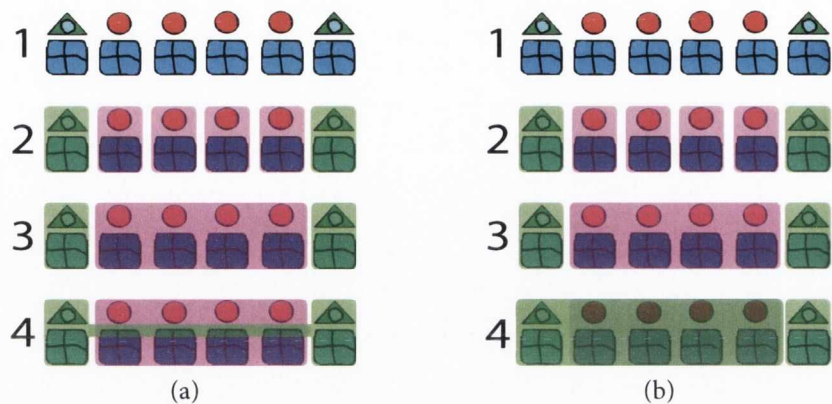


Figure 5.10: Iterative grouping of facade representation. (a) Precedence rules result in overlapping groups at stage 4. (b) By including the overlap exception constraint, overlapping no longer occurs, resulting in a more suitable grouping for split grammars.

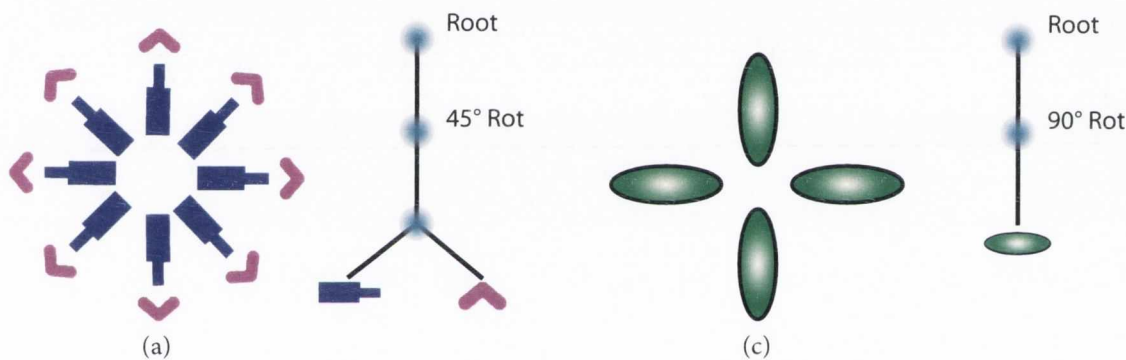


Figure 5.11: Input patterns are both composed of a dominant rotational symmetry. Their corresponding symmetry trees are very similar.

This brings us back to the discussion in Chapter 3, notably the difficulty in writing production rules to generate variations. By interchanging nodes in symmetry trees, we can create many new hybrid variations, each structurally similar to the input examples, without ever having to look at a production rule. This seems like an ideal solution to infer stochastic production rules capable of generating many varying outputs.

5.2.1 Hybridising Symmetry Tree

Having determined the symmetry trees for a set of input patterns, it is then possible to create a hybrid symmetry tree. This hybrid tree will be used to generate a production system capable of synthesising new varying patterns. Any algorithm to generate hybrids should satisfy the following constraints:

- 1. Hybrid patterns should exhibit the same symmetry relations as their input examples, thus generating perceptually similar patterns.



2. There should be a high level constraint to control the extent to which hybrids are allowed to vary from their input examples.

The first constraint ensures that terminal nodes in the hybrid tree have the same *function* as those in the input examples. For example, windowsills in all building facades occur directly underneath windows and do not appear under doors. We would state that according to this observed symmetry relation, all windowsills provide the same *function* and this should not be broken in the output hybrids. We generalise this *function* to have meaning for any abstract pattern, stating it as follows:

**Definition** The *function* of a shape (or group of shapes) is the set of observed symmetry relations between itself and other shapes.

Therefore, a shapes *function* is a subtree within the patterns symmetry tree, describing how each node relates to its neighbours. Under very strict conditions, the shapes *function* will incorporate all observed symmetry relations, thereby restricting hybrids to be identical to the input examples. To achieve variation we must be less strict, only considering the shapes *function* within a local subtree surrounding its node in the symmetry tree. This directly relates to the second constraint. The amount of variation can be controlled by adjusting how strictly a shapes *function* is defined. In other words at what distance from the node in the symmetry tree is it permissible to break a symmetry relation.

Deciding which nodes may be interchanged between two symmetry trees now corresponds to matching local subtrees surrounding the nodes (the nodes will be leaves within these local subtrees). We use the following matching criteria to match subtrees:

1. Nodes that represent rotational symmetry are matched with other rotational symmetry nodes, regardless of their angle of rotation.
2. Nodes that represent translational symmetry can only be matched with nodes that represent collinear translations.
3. Terminal and assembly nodes match each other.

By specifying the minimum depth that subtrees must match, the number of matching nodes is constrained, thereby controlling the amount by which a hybrid pattern varies from the input examples (See Figure 5.13). The algorithm is given in Algorithm 5.12, while the effect of changing the matching depth is demonstrated in Figure 5.14.

---

```

hybridizeTrees(tree, searchTree, minDepth)
  foreach subtree in searchTree
    [matchTree score] = find(subtree, tree);
    if (score > minDepth)
      matchTree.addSibling(subtree);
      subbTree.addSibling(matchTree);
    end
  end
end

```

---

Algorithm 5.12: Algorithm to control the amount of variation a hybrid can exhibit. The subrou-tine *find* searches for a subtree within a bigger *tree*, returning the matching nodes and matching score.

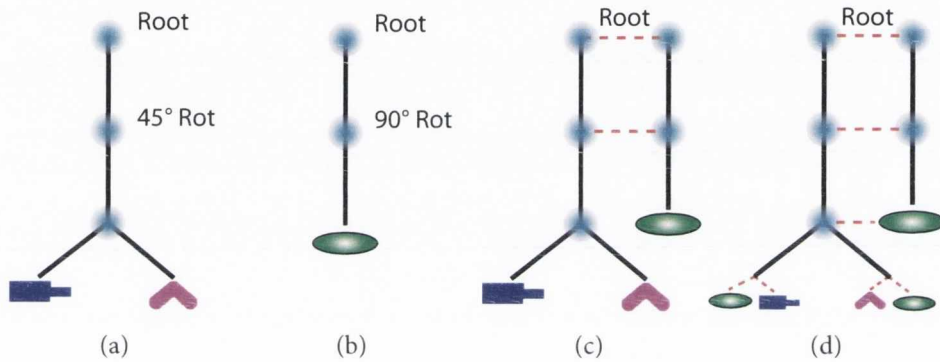


Figure 5.13: (a–b) Graphs before hybridisation. (c) Subtree matching with a matching depth of 2. (d) Subtree matching with a matching depth of 1. Broken red line indicates where nodes have been connected as siblings.

### 5.2.2 Production System

Symmetry trees can be directly represented using a production system. This allows our method to act as a layer on top of any existing stochastic production system of the form:

$$Predecessor \rightarrow Command(params)\{Successor\}:Probability$$

A rule is created for each node in the symmetry tree. *Predecessor* is given the node name (a unique identifier), while the *Command* and *Successor* symbols are determined from the symmetry relation the node represents and its children. Several rules may have the same *Predecessor* corresponding to sibling nodes in the hybrid symmetry tree (sibling nodes represent variation and are assigned the same identifier). We will first discuss the general case, which is capable of generating any abstract pattern. We will then discuss some of the problems associated with generating hybrid building facades and propose suitable additions to the CGA shape grammar commands discussed in Chapter 3.

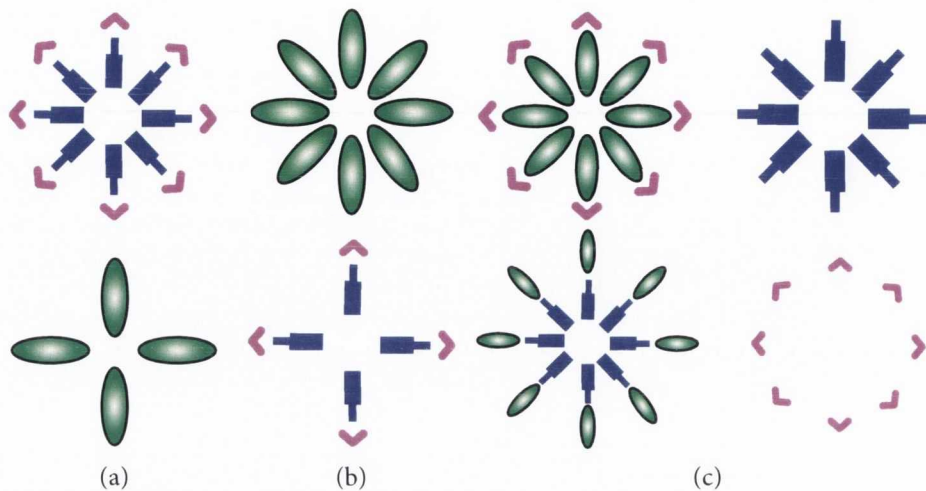


Figure 5.14: (a) Input patterns. (b) Subtree matching depth set to 2. Some variation is possible while preserving the input structure. (c) Subtree matching depth set to 1. Much more variation is possible as less of the original structure is preserved.

### General Case

In general, patterns cannot be divided up into spatially confined regions such as the split grammars used to generate buildings. Many shapes can belong to overlapping symmetry cliques, thereby requiring a grammar that can generate overlapping symbols. L-Systems can achieve this but lack the necessary control to restrict spatial growth. We propose the following two commands to achieve both overlapping symbols and to provide restrictions on the size of symmetry groups:

**Assemble( $\alpha, d, \beta$ ):** Assembles two successor shapes together with a transformation between them.

**Symmetry( $\alpha, d, \beta, n$ ):** Repeats the transformation  $n$  times, thus creating  $n$  successor symbols. If  $n < 1$  the transformation is repeated until there is no space left on the canvas.

These simple commands follow directly from the grouping method discussed in the previous section. They are sufficient to generate a wide range of patterns but lack the spatial subdivision properties required to generate buildings of arbitrary sizes. We will consider building facades as a special case where spatial overlapping is not permissible.

### Spatially Constrained Case

Building facades require rules that are spatially constrained. Recall the process of iteratively subdividing a facade through the use of split grammars from Chapter 3. We can consider a



symmetry tree for a building facade to be a special case, whereby all symmetry relations are translational symmetries, restricted to be either horizontal or vertical. Observing this, the *Assemble* and *Symmetry* commands in the general case are synonymous with the *Subdivision* and *Repeat* commands of the CGA shape grammar.

However, since we are aiming to produce large variations among output buildings, the sizes of successor shapes can be radically different depending on which rule is chosen. In fact, we do not know what terminal shapes will be chosen in the end and so cannot assign absolute or even relative sizes to parameters for some shape operations. For example, we do not know the aspect ratio of the windows when we are placing the floors and hence, cannot use the traditional approach of setting size parameters to the rules. In order to tackle this problem, we allow the use of a “don’t care” size parameter denoted by a colon (:). This parameter instructs the system to forward generate all of its children and then propagate their sizes back up to that level of the hierarchy. The system will then try to automatically determine a suitable split size that would preserve the learnt symmetry relations of the chosen terminals. This gives a best fit approach that can be easily altered by changing the desired spacing between elements. Such an approach also reduces the number of productions needed to correctly place output shapes.

### 5.2.3 Conditional Dependence

We have shown how symmetry trees may be grafted together to produce a hybrid symmetry tree capable of synthesising varying hybrid patterns. We proposed that groups of shapes may be interchanged if they have a similar *function* within their respective input patterns. However, some shapes may have various aesthetic properties that, despite having the same *function* cannot be interchanged. Consider for instance elements of buildings belonging to two different architectural styles. While cornices and balconies may have the same shape *function*, the type of cornice or balconies used may be dependent on the window type or its position on the building’s facade. We therefore require some way to represent this contextual relationship between shapes.

We will investigate the use of a probabilistic model, whereby relationships between shapes can be expressed using conditional probabilities. Figure 5.15 illustrates the conditional probabilities between the elements of a simple input example. We can see that a windowsill has a 50% chance of occurring underneath a window. Using Bayes’ theorem it can be shown that, given a windowsill there must be a window above it:

$$\begin{aligned}
 P(Win|WSill) &= \frac{P(WSill|Win)P(Win)}{P(WSill)} \\
 &= \frac{0.5 \times 0.5}{0.25} \\
 &= 1
 \end{aligned}$$

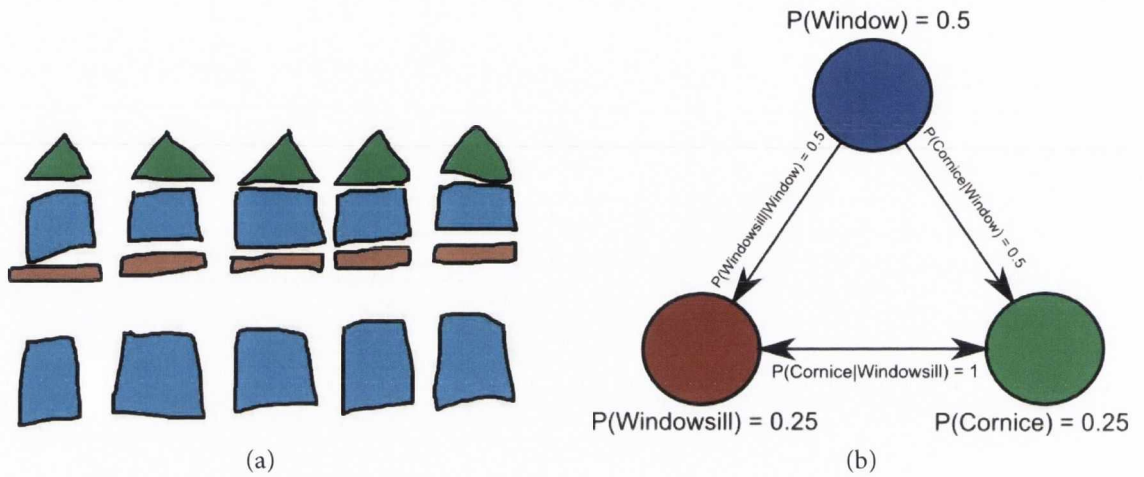


Figure 5.15: Object Dependencies

In order to represent contextual relationships between productions, we modify the CGA shape grammar. In a similar approach to indexed grammars [Sta93], we represent context sensitivities using a stack. Each rule has an identifier and a numeric index, which is used to define conditional probabilities on the rules. A production rule then has the following form:

$$\text{Identifier}[X] \rightarrow \text{rule}(\text{parameters})\{\text{successors}\}: P(R|X = x) = p$$

*Identifier* is the predecessor for the rule. The index  $X$  can be given any numerical value and conditional probabilities are defined for that number. We previously considered interchanging nodes on the symmetry tree if their *function* was similar, i.e, if the symmetry relations of their parent nodes were the same. We can represent a node's *function* by adopting a labelling scheme when grouping shapes. We label nodes from the bottom up, prepending a  $H$  or  $V$  depending on the direction of symmetry. This labelling process is illustrated in Figure 5.16. This is effectively the same as varying the subtree matching depth depending on the hierarchy level of the node in the symmetry tree. Low level symmetries can be replaced with other low level symmetries with little restriction, but there is greater restriction on how high level symmetries may be replaced, thus preserving the high level structure of example buildings. The index numbers will propagate down the symmetry tree, applying constraints on what terminal shapes may be selected. Using this approach, we can represent different contextual relations, depending on how we propagate these indices. We can adjust how low level nodes may vary while preserving high level structure. We tested a number of different indexing schemes and results are given in Section 5.3.



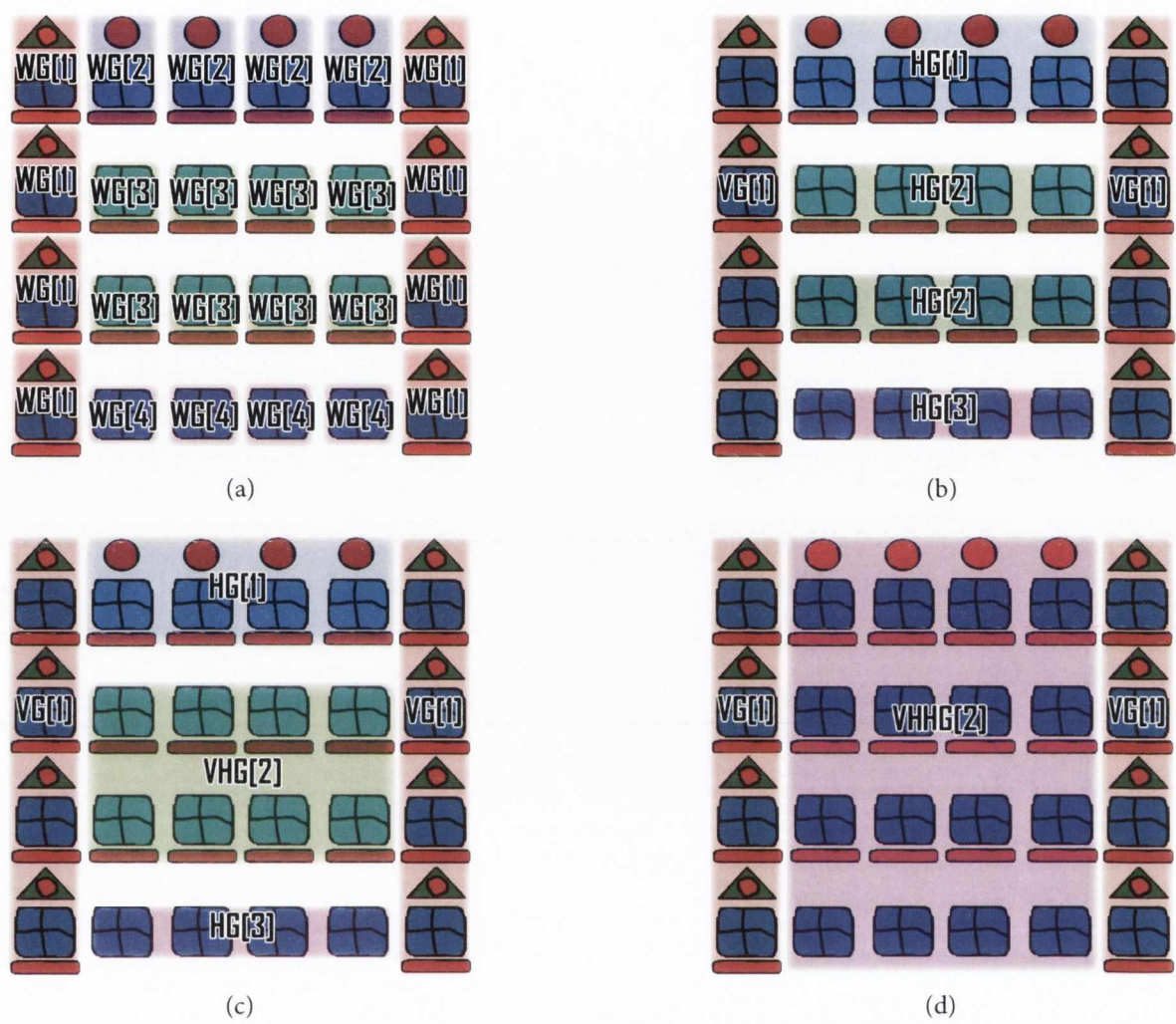


Figure 5.16: Groups of elements are labelled from the bottom up. Prefixing a *H* represents a horizontal grouping and *V* represents a vertical grouping.

### 5.3 Results

Generating hybrids from symmetry has a number of useful applications. We have tested this technique with a number of pattern types.

#### 5.3.1 Abstract Patterns

**Flower Pattern** We segmented some flower patterns into a set of oriented labelled shapes. We obtained many interesting hybrid flowers with characteristics of the input patterns. Results are given in Figure 5.17.

**Ornamental Designs** We segmented the parts of ornamental designs and achieved many new varying designs. The complexity of each pattern allowed hundreds of hybrid variations.



Results are given in Figure 5.18.

**Fractal** Some interesting and unexpected results occurred with some of the hybrids. If symmetries are matched between different hierarchy levels of the input patterns' symmetry tree, we can obtain some fractal like effects. This occurs if we allow loops within the symmetry tree. We have no way to adjust the depth of recursion within these fractal hybrids without manually adding some control parameter to the resulting production set. Results are given in Figure 5.19.

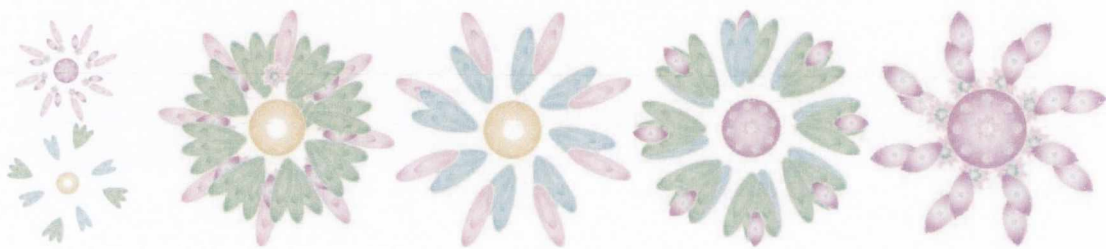


Figure 5.17: Left: Input flowers to be hybridised. Outputs reflect the rotational symmetry of the petals.

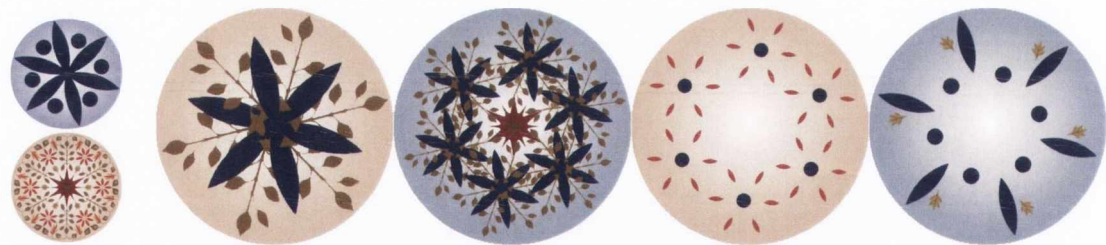


Figure 5.18: Left: Input ornamental designs to be hybridised. Many new output designs can be achieved (the background disk was added manually).

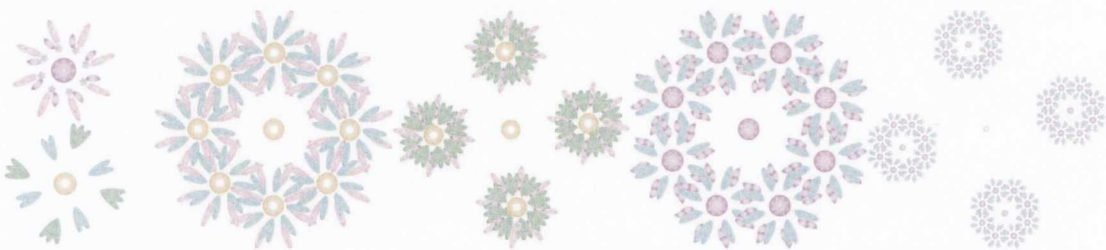


Figure 5.19: Fractal patterns can result if symmetry nodes are matched between different hierarchy levels.

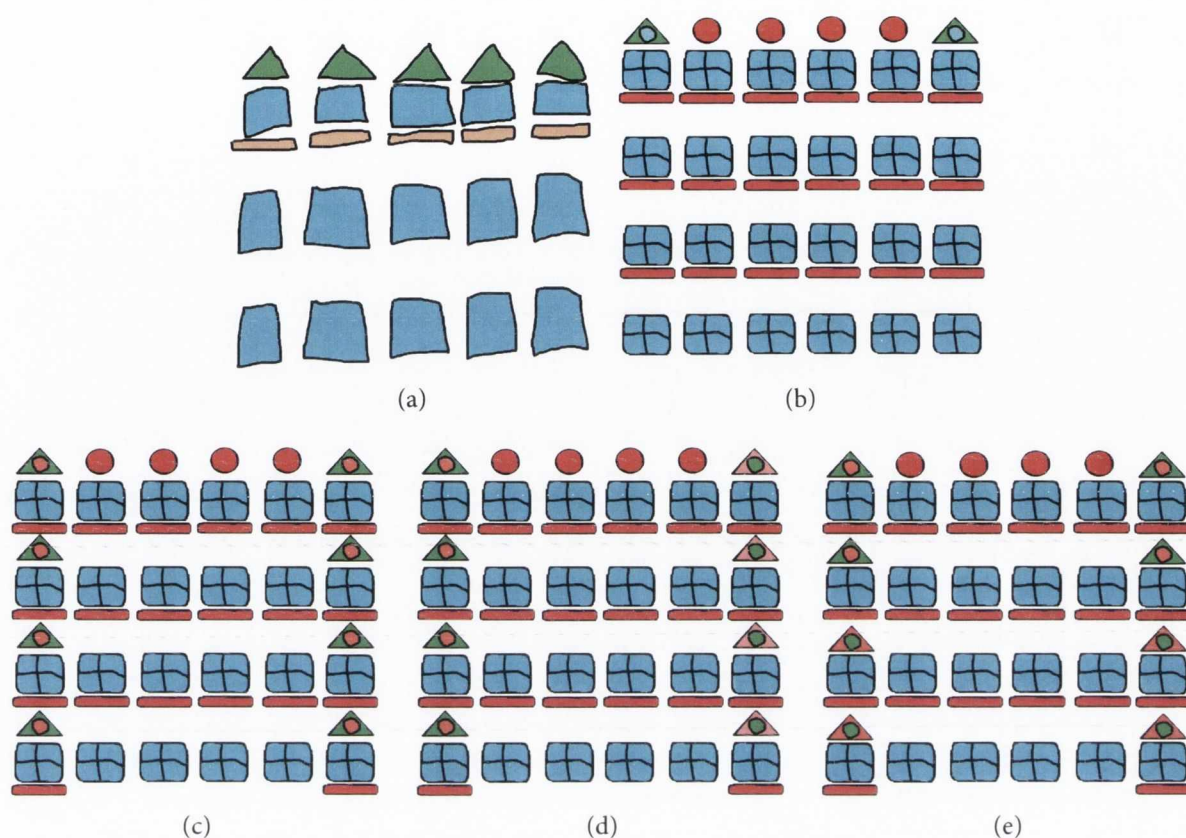


Figure 5.20: Input sketches used to test the preservation of contextual relations in generated hybrids.

### 5.3.2 Indexing Schemes

In order to preserve some contextual relationships between shapes, we applied semantic labels to symmetry groups. By passing index numbers between production rules we can assign conditional probabilities with the aim of preserving certain kinds of relationships. In each case we create hybrids from the input images in Figure 5.20. We explored different choices of terminal shapes based on their symmetry relations in the input images, symmetry relations between different elements and combinations of these two.

**Unique Index** Figure 5.21 shows the results of simply giving each new group a unique index number. Contextual relations between elements are preserved (e.g., a brown windowsill is only observed under a green cornice).

**According to Symmetry** Figure 5.22 shows the results of assigning indices according to the symmetry type of a group. It can be seen that vertically/horizontally repeating groups can only be replaced with other groups that have been observed to repeat vertically/horizontally. This scheme does not preserve contextual relations between elements and so windowsills and cornice types are chosen independently of each other.



**According to Symmetry and Unique Index** Figure 5.23 shows the results of assigning indices according to symmetry while giving a unique index to groups containing different elements. Vertical/horizontal groups are more likely to be replaced by vertical/horizontal groups. Conditional relations are better preserved between elements (e.g., the brown windowsill can only appear with a certain window type and green cornice). However, using this configuration forces multiple window types to appear on one building, which is not observed in the inputs.

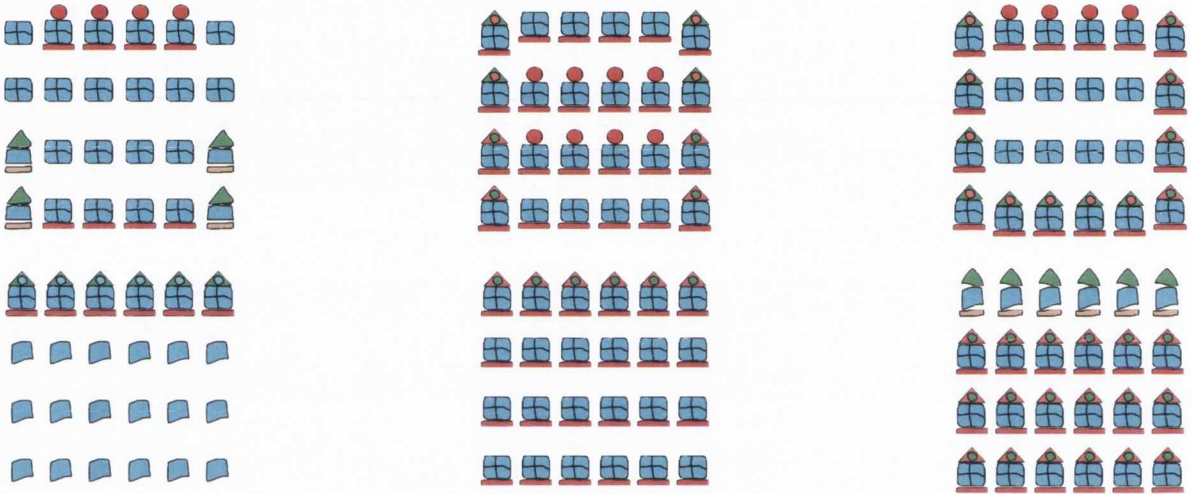


Figure 5.21: Unique Index: Contextual relations between elements of the same group are preserved (e.g., a brown windowsill is only observed under a green cornice).

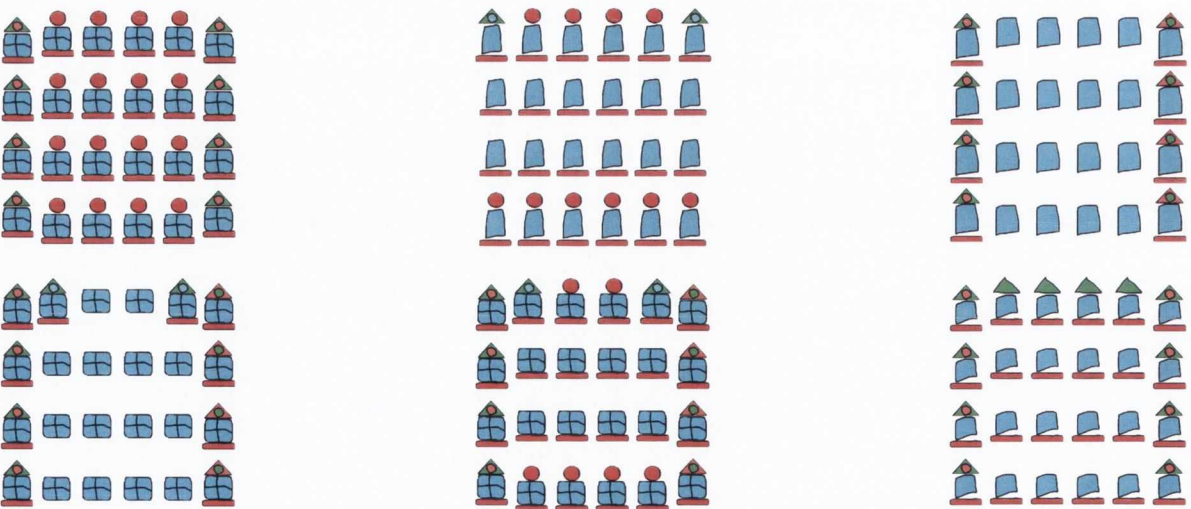


Figure 5.22: According to Symmetry: Does not preserve relations between elements (e.g., a red windowsill can appear under a different window type than that which was observed in the inputs). Building elements are chosen based on their symmetry layout in the input images.



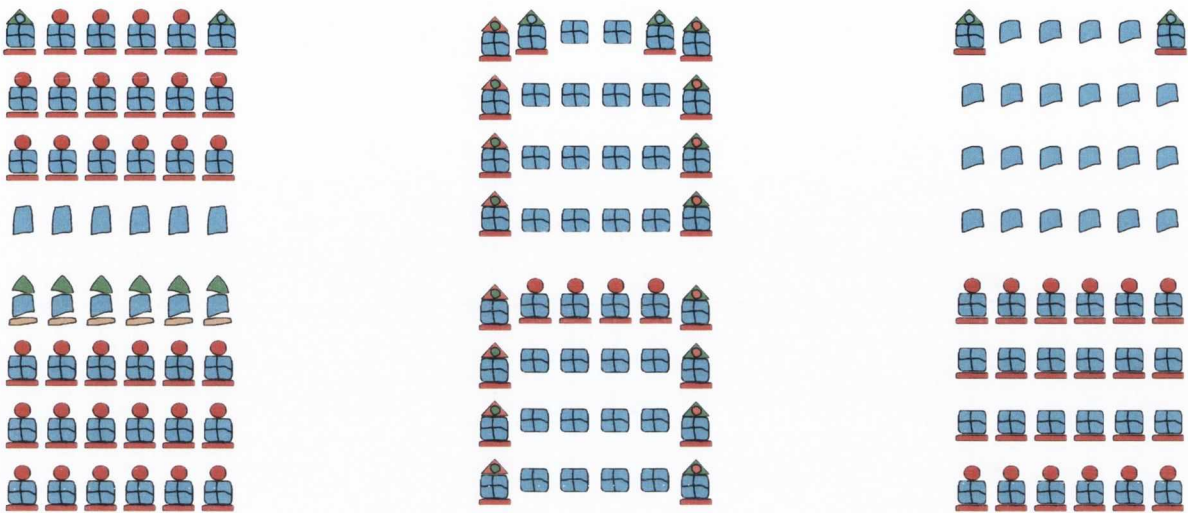


Figure 5.23: According to Symmetry and Unique Index: Preserve relationships between element types.



Figure 5.24: Input facade images to be hybridised. Facades were segmented using the techniques presented in Chapter 4. Resulting hybrid building facades are illustrated in Figure 5.25.

### 5.3.3 Building Facades

We tested our symmetry hybridisation algorithm on photographs of building facades. Facade elements were extracted and labelled using the technique presented in Chapter 4. Taking as input the facades in Figure 5.24 resulted in hundreds of varying hybrid buildings. A sample of the resulting output buildings can be seen in Figure 5.25. To improve realism, stains were then added to the output buildings using the weathering simulation algorithm discussed in Section 3.2.2. The results demonstrate that our hybridisation algorithm is able to generate stochastic production grammars capable of modelling the structural variations of common building facades. This technique could be utilised to add much more variation to procedurally generated cities, without much effort from the user. The resulting production grammars could be assimilated into the object oriented building styles presented in Section 3.2.1, thereby abstracting facade generation from the user, whilst allowing manually written production rules to control the building's overall shape.

## 5.4 Discussion

We have proposed some techniques to help solve the difficulties of writing production rules to generate many varying building facades, as discussed in Chapter 3. A building facade was viewed as a pattern of shapes, with intricate symmetry relations between them. Section 5.1 presented some techniques to reliably detect the hierarchical symmetry relations between shapes (building elements) and proposed a model to represent this in the form of a symmetry tree. In Section 5.2, we explored the idea of comparing these symmetry trees and grafting them together to generate hybrid patterns. We demonstrated the use of hybridisation to infer stochastic production grammars capable of generating a large number of varying buildings from a few inputs. This simplifies the writing of production systems to generate buildings, and has the potential to add a huge amount of variation to procedural cities.



Figure 5.25: Results for generating hybrid building facades from segmented photographs. The input photographs can be seen in Figure 5.24. Many new building facades can be created from just a few input images.



We have proposed that contextual relations should be preserved among generated hybrids, and have investigated the use of conditional probabilities to model these relations. Using index numbers with conditional probabilities can preserve relations between objects at the same hierarchical level. However, contextual relations between different hierarchy levels cannot be preserved using the model we have proposed. In many classical buildings such as the one depicted in Figure 5.26, certain architectural elements only appear between columns. If we wish



Figure 5.26: Contextual relations can be between elements on different branches of the symmetry tree. Balconies (blue) only appear between columns (green).

to create more accurate hybrid buildings, these contextual relationships should be preserved. However, it is unclear which contextual relationships are important and which can be ignored. Perhaps placing restrictions is not the right direction to take. Relaxing restrictions and creating as much variation as possible, while allowing the user to select which hybridisations are suitable, might be a better approach. The system could then prune branches in the hybrid symmetry tree that led to undesirable results. We believe this will be the direction of future work.

# Chapter 6

## Conclusion and Future Work

In this thesis we have addressed problems in the domain of procedural modelling of cities. This chapter will review the contributions made and suggest directions for future work.

### 6.1 Contributions

We have introduced a number of new ideas and algorithms that directly addressed the problem of writing production systems for building generation. We made improvements to existing methods for facade segmentation and, demonstrated how symmetry detection could be utilised to generate hybrid buildings.

#### 6.1.1 Building Generation

In Chapter 3 we highlighted problems with state of the art methods for procedural building generation. We introduced some new ideas to simplify the writing of production systems and to improve the realism and performance of generated buildings.

##### Object-Oriented Buildings

We introduced the new concept of *object-oriented building styles* to simplify the manual writing of production systems. Object-oriented building styles allow buildings to inherit their production rules from more abstract styles and only re-specify certain aspects of the style that need to be changed. This improvement was achieved by breaking down production systems into several subsystems that are each responsible for generating a different part of the building. By encapsulating these subsystems into semantic blocks, users are abstracted from the complexity of other blocks that do not need to be changed. As demonstrated in Listing 3.6 and Listing 3.7, this practice reduces code redundancies and improves the readability of styles.

## Building Variation

We demonstrated how simple texture variations can improve the realism of building models. By implementing a simple weathering simulation, even more variation was achieved among generated buildings. Stain generators were automatically defined through the semantic labelling of terminal symbols (building elements).

## Real-Time Generation

We introduced a caching approach to aid in the real-time generation of buildings. The multi-state cache exploits temporal coherence between frames so that buildings have been generated by the time a new geometry page needs to be rendered. We gradually apply different steps of the building generation process depending on distance to the camera. Memory intensive geometry meshes are only created for buildings that are likely to be rendered soon, while slow tasks such as database access and the evaluation of production rules are initiated much earlier on and distributed over multiple frames. Our benchmark results showed that this caching approach results in much smoother page loading compared to a lazy generation approach. We also demonstrated how different LOD building meshes can be automatically generated without altering the production systems. Our results showed that generation times are orders of magnitude quicker for lower LOD, thereby facilitating real-time generation of much larger terrain areas.

### 6.1.2 Inverse Procedural Modelling

In Chapter 4 we described our improvements to existing methods for segmenting facade images. This process is a necessary step towards understanding a building facade's structure and inferring production grammars. Popular techniques in the literature rely on iteratively subdividing facade images based on edge responses. We presented an approach that complements existing subdivision based segmentation methods and is thus able to infer split grammar rules to represent a facade. We showed that clustering algorithms based on GMM and graph cuts can achieve much better segmentation results than edge based approaches, and do not require per-image tuning of threshold values. Our results demonstrate clean segmentations on facade images that would prove difficult using edge based methods (such as facades with a strong wall texture or non-rectangular facade elements).

### 6.1.3 Symmetry Hybrids

In Chapter 5 we introduced new methods to infer stochastic production grammars from input patterns. The aim was to create new hybrid buildings from example images to address problems



in writing production rules to create many variations. We took the first step towards inferring hybrid production rules and will review our contributions.

### **Hierarchical Symmetry Detection**

We presented a new algorithm for *hierarchical symmetry detection*. Many patterns, including the layout of building facades, are formed from a hierarchy of symmetry relations. Previous transformation voting based methods for detecting structural regularity rely on detecting a regular grid in the space of pairwise transformations. However, in hierarchical patterns, this grid is obscured by overlapping symmetries and cannot be directly detected. We presented a new algorithm capable of grouping together transformation clusters that are likely to belong to the same symmetry clique. It is then possible to detect regular grids within each of these cluster groups and use a suitable criterion to detect the lowest level symmetry. A tree of symmetry relations is iteratively constructed that fully describes the input pattern. Our results demonstrate that this algorithm is a suitable approach for detecting hierarchical symmetry relations in many arbitrary 2D patterns (and should generalise to higher dimensions).

### **Generating Hybrid Patterns**

We introduced the new concept of *symmetry hybrids*, which are hybrid variations of input patterns that preserve structural similarities. Through the use of precedence rules based on Gestalt psychology [WXL\*11], we can detect the hierarchical structure of patterns composed of several different shapes. We make the observation that shapes (or groups of shapes) with identical symmetry hierarchies have the same perceptual grouping in their respective patterns. We exploit this phenomenon by allowing similar subtrees of different symmetry hierarchies to be interchanged, thereby creating new hybrid patterns. By adjusting the depth by which subtrees must match, we can control the degree to which hybrid patterns can vary from input examples. Our results demonstrate that symmetry hybrids can be useful for a wide variety of domains, including building facade generation.

### **Context-Aware Representation**

We experimented with a new form of production grammar capable of representing a limited set of contextual relationships between production rules. In this new production grammar, control variables are propagated with each derivation step. By defining conditional probabilities for these control variables, limited contextual properties can be preserved. Our results show that conditional probabilities could be learnt to preserve a number of different contextual properties such as: a shape's overall symmetry in the pattern, the type of shapes that can be grouped together, and a shape's visual dominance in the pattern.

## 6.2 Conclusion

In Chapter 1 we discussed our motivation for the research presented in this thesis. We investigated procedural generation as a tool to aid in the reconstruction of cities, and discussed the difficulty of manually writing production grammars to create building models. This led us to the following research question:

*“How can the writing of procedural systems that model variation between architectural types be simplified, with particular regard to inferring stochastic production grammars from images?”*

The research presented in this thesis demonstrates significant progress towards answering this question. We introduced the concept of *object-oriented building styles* to simplify the manual writing of production grammars for the generation of building models, and showed how more variation could be added by simulating weathering effects.

We proposed that symmetry detection could be utilised to create a standard representation of the hierarchical structure of a building, and demonstrated that this representation could be exploited to infer stochastic production grammars capable of generating hundreds of new hybrid building models. This inverse procedural modelling approach requires the user to simply input a few photographs, thereby removing the need to manually write production grammars altogether.

## 6.3 Future Work

In this section, we will discuss possible avenues of future work, along with the limitations of our current approaches for procedural modelling of cities. We also suggest starting points for future research to resolve these limitations.

### Hybrid Control

The hybridisation approach discussed in Section 5.2 only permits global control over variation among hybrids. We achieve this by adjusting how closely hybrids must conform to the symmetry hierarchy detected in input examples. However, this is not sufficient to achieve the maximum number of variations that seem plausible. For example, some combinations of architectural styles fit well together while others do not (e.g., cornices from classical architecture may look plausible on modern buildings, but modern windows do not look right on classical buildings). This implies that there is an aesthetic element that is not simply dependant on symmetry relations. As a result of this some variations do not look plausible, while others do, even though they follow the same hybridisation rules. This suggests that we need to enforce local



constraints over where variation may occur, which may not be learnt directly from examples. This could possibly be achieved with user interaction by generating many variations, and requesting the user to select which ones look plausible and which ones do not. By validating the results of this process it should be possible to determine which branches in the derivation tree result in unwanted variation, and to prune them. This is one method which should be explored in future work.

However, it seems that there are contextual constraints between different branches of the derivation tree that cannot simply be pruned. For instance, the choice of door that would look plausible is dependent on the choice of windows, even though they belong to different branches. This rules out simple pruning on such cases and a different representation is required to represent these contextual dependencies. In Section 5.2 we presented a new form of production grammar that is capable of representing contextual relations between production rules. However, the types of contextual relationships that can be modelled using this approach are limited. This is due to the Markovian nature of the derivation process, i.e., the choice of production rules depends only on their successor symbols and parameters. We can only model dominant symmetry relations in input examples and cannot represent direct relationships between different branches of the symmetry hierarchy. For example, we cannot model a direct relationship between the choice of windows on different floors. Currently, such relationships are indirectly modelled by propagating values from the top down but there is not much control over which specific relations are important. A better representation is required to allow control over these types of contextual relationships.

We believe a mass-spring model might be capable of representing these relationships. In this system all symmetry relationships between elements would be assigned different strengths (importance), depending on some user decided criterion. This would essentially allow feedback between different branches, so that placement of architectural elements would be dependent on their relationship with all previously placed elements. This would permit a user to change the choice of windows on the ground floor and to see a direct change on the layout of other floors to keep the system in equilibrium. Control over variation could then be achieved by adjusting how strong relations are between specific branches in the structural hierarchy. For instance, the user could decide that there is a strong dependency between the type of windows and doors, but not between windowsills and cornices. New buildings could then be designed using a blank canvas where users simply drag building elements from some collection onto the canvas. The building's structure would then be automatically determined so that it best represents learnt symmetries. This would be an interactive and simple way to design new building styles. Future work should explore whether or not this type of model could achieve better control over hybrid variations.



### **Hybrids From 3D Models**

Another avenue for future work would be to expand our symmetry hybrid framework to work with 3D geometric models. The recent work of Bokelah et al. [BWS10] has demonstrated progress towards inferring production rules from 3D models. It should be possible to integrate their approach with our symmetry hybrid algorithm. However, problems would arise due to the connectedness of 3D models. For two models to be successfully combined, they would need to undergo some transformation so that they would join together without discontinuities. Another possibility might be to simply transform one model to undergo the same symmetry relations of another model. For example, it should be possible to take two castle models and change one of them to assume the same shape as the other. This should be a case of changing high level symmetry relations, while preserving lower level symmetry relations. Future work should investigate how plausible this would be.

A less ambitious approach would be to first segment a 3D model into its structural components. Recent work by Nan et al. [NSZ\*10] shows that structure can be uncovered from LIDAR scans of buildings, and represented with boxes. This box based representation would simplify hybridisation as it is not subject to the same connectedness problems of generic models. Breaking the model into a set of boxes makes it easier to label building elements and hence, hybrids could be created in the same way as the method we proposed for 2D patterns of disjoint shapes.

### **Integration With Simulation**

In Chapter 3 we demonstrated that procedural modelling could be used in conjunction with manually created CAD models, to expand existing urban simulations. However, this expansion is purely geometrical and does not yet provide data to extend behavioural systems, such as pedestrian and traffic simulations. This enhancement would be interesting to achieve with real-time procedural generation because there would be no limit to how large scale a simulation could be. The results could be used to provide pedestrian and traffic simulations to online services like Google Maps. However, this is a particularly challenging problem for Metropolis. Metropolis implements the path planning technique of Paris et al. [PDBo6], which requires preprocessing of the environment map to create a set of semantic layers (e.g., roads, footpaths, buildings) that are used for path planning. Although it would be possible to preprocess a large portion of GIS data for this purpose, the approach does not scale well as it requires the entire environment map to reside in memory. For the purpose of Metropolis, future work should explore methods to incrementally expand this environment topology map as new pages of geometry are procedurally generated. One benefit of this approach is that the procedural system could place identifiers in different semantic layers of the environment topology map. For instance, the locations of building entrances and street side props could be placed in different

semantic layers. These layers could then be used to allow virtual humans to interact with their procedural environment, such as entering and exiting buildings.

Another challenge would be to keep the simulation persistent as new pages are loaded. For example, if a car drives into an unloaded area, it should still exist when that area is generated. This could potentially be achieved using a similar strategy to the caching approach presented in Section 3.4, so that behavioural simulations continue for a larger area than that which is currently visible.

# Bibliography

- [AA96] AICHHOLZER O., AURENHAMMER F.: Straight skeletons for general polygonal figures in the plane. In *Computing and Combinatorics*, Cai J.-Y., Wong C., (Eds.), vol. 1090 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 1996, pp. 117–126.
- [AAAG95] AICHHOLZER O., AURENHAMMER F., ALBERTS D., GRTNER B.: A novel type of skeleton for polygons. *J. UCS* (1995), 752–761.
- [ABVA08] ALIAGA D. G., BENEŠ B., VANEGAS C. A., ANDRYSCO N.: Interactive reconfiguration of urban layouts. *IEEE Comput. Graph. Appl.* 28, 3 (2008), 38–47.
- [ARB07] ALIAGA D. G., ROSEN P. A., BEKINS D. R.: Style grammars for interactive visualization of architecture. *IEEE Transactions on Visualization and Computer Graphics* 13, 4 (2007), 786–797.
- [Ata85] ATALLAH M. J.: On symmetry detection. *IEEE Trans. Comput.* 34, 7 (1985), 663–666.
- [Aur91] AURENHAMMER F.: Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Comput. Surv.* 23 (September 1991), 345–405.
- [Auro8] AURENHAMMER F.: Weighted skeletons and fixed-share decomposition. *Comput. Geom. Theory Appl.* 40 (July 2008), 93–101.
- [AVBo8] ALIAGA D. G., VANEGAS C. A., BENEŠ B.: Interactive example-based urban layout synthesis. *ACM Trans. Graph.* 27, 5 (2008), 1–10.
- [AYRW09] ALI S., YE J., RAZDAN A., WONKA P.: Compressed facade displacement maps. *Visualization and Computer Graphics, IEEE Transactions on* 15, 2 (march-april 2009), 262–273.
- [Bal87] BALLARD D. H.: Generalizing the hough transform to detect arbitrary shapes. In *Readings in computer vision: issues, problems, principles, and paradigms*. Morgan Kaufmann Publishers Inc., 1987, pp. 714–725.



- [Bar83] BARNARD S. T.: Interpreting perspective images. *Artificial Intelligence* 21, 4 (1983), 435 – 462.
- [BBW\*08] BERNER A., BOKELOH M., WAND M., SCHILLING A., SEIDEL H.-P.: A graph-based approach to symmetry detection. In *Symposium on Volume and Point-Based Graphics* (Los Angeles, CA, 2008), Eurographics Association, pp. 1–8.
- [BBW\*09] BOKELOH M., BERNER A., WAND M., SEIDEL H.-P., SCHILLING A.: Symmetry detection using line features. *Computer Graphics Forum (Proc. EUROGRAPHICS 2009)* 28, 2 (2009), 697–706.
- [Bel03] BELLMAN R.: *Dynamic Programming*. Dover Books on Mathematics. Dover Publications, 2003.
- [B]01] BOYKOV Y. Y., JOLLY M.: Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images.
- [BLR\*11] BOSCH C., LAFFONT P.-Y., RUSHMEIER H., DORSEY J., DRETTAKIS G.: Image-guided weathering: A new approach applied to flow phenomena. *ACM Trans. Graph.* 30 (May 2011), 20:1–20:13.
- [BM]\*11] BENEŠ B., MASSIH M. A., JARVIS P., ALIAGA D. G., VANEGAS C. A.: Urban ecosystem design. In *Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2011), I3D '11, ACM, pp. 167–174.
- [BR06] BRENNER C., RIPPERDA N.: Extraction of facades using rjMCMC and constraint equations. In *PCV06* (2006), pp. 155–160.
- [BvMM11] BENEŠ B., ŠTÁVA O., MĚCH R., MILLER G.: Guided procedural modeling. *Computer Graphics Forum* 30, 2 (2011), 325–334.
- [BWM\*11] BERNER A., WAND M., MITRA N. J., MEWES D., SEIDEL H.-P.: Shape analysis with subspace symmetries. *Computer Graphics Forum* 30, 2 (2011), 277–286.
- [BWS10] BOKELOH M., WAND M., SEIDEL H.-P.: A connection between partial symmetry and inverse procedural modeling. *ACM Trans. Graph.* 29 (July 2010), 104:1–104:10.
- [Can86] CANNY J.: A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.* 8 (November 1986), 679–698.
- [CEW\*08] CHEN G., ESCH G., WONKA P., MÜLLER P., ZHANG E.: Interactive procedural street modeling. *ACM Trans. Graph.* 27, 3 (2008), 1–10.

- [CG95] CHIB S., GREENBERG E.: Understanding the Metropolis-Hastings Algorithm. *The American Statistician* 49, 4 (1995), 327–335.
- [Che95] CHENG Y.: Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intell.* 17 (August 1995), 790–799.
- [CM02] COMANICIU D., MEER P.: Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.* 24, 5 (2002), 603–619.
- [CO11] CULLEN B., O’SULLIVAN C.: Symmetry hybrids. In *Proceedings of the International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging* (New York, NY, USA, 2011), CAe ’11, ACM, pp. 33–38.
- [CY99] COUGHLAN J., YUILLE A.: Manhattan world: compass direction from a single image by bayesian inference. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on* (1999), vol. 2, pp. 941–947 vol.2.
- [DTC04] DICK A. R., TORR P. H. S., CIPOLLA R.: Modelling and interpretation of architecture from several images. *Int. J. Comput. Vision* 60, 2 (2004), 111–134.
- [Duao2] DUARTE J.: *Malagueira Grammar - towards a tool for customizing Alvaro Siza’s mass houses at Malagueira*. PhD thesis, MIT School of Architecture and Planning, 2002.
- [DZ08] DOMITER V., ZALIK B.: Sweep-line algorithm for constrained delaunay triangulation. *Int. J. Geogr. Inf. Sci.* 22 (January 2008), 449–462.
- [Ebe03] EBERT D.: *Texturing & modeling: a procedural approach*. Morgan Kaufmann series in computer graphics and geometric modeling. Morgan Kaufmann, 2003.
- [EGO10] ENNIS C., GERDELAN A., O’SULLIVAN C.: Plausible methods for populating virtual scenes. *Crowd Simulation Workshop, St. Malo, France* (2010).
- [ELSo8] EISENACHER C., LEFEBVRE S., STAMMINGER M.: Texture synthesis from photographs. *CGF: Eurographics* 27, 2 (2008), 419–428.
- [EMO10] ENNIS C., McDONNELL R., O’SULLIVAN C.: Seeing is believing: body motion dominates in multisensory conversations. *ACM Trans. Graph.* 29 (July 2010), 91:1–91:9.

- [EPO11] ENNIS C., PETERS C., O'SULLIVAN C.: Perceptual effects of scene context and viewpoint for virtual pedestrian crowds. *ACM Trans. Appl. Percept.* 8 (February 2011), 10:1–10:22.
- [FF62] FORD L. R., FULKERSON D. R.: *Flows in Networks*. Princeton University Press, 1962.
- [FH75] FUKUNAGA K., HOSTETLER L.: The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on* 21, 1 (jan 1975), 32 – 40.
- [FH04] FELZENSZWALB P. F., HUTTENLOCHER D. P.: Efficient Graph-Based image segmentation. *Int. J. Comput. Vision* 59, 2 (2004), 167–181.
- [Fle87] FLEMMING U.: More than the sum of parts: the grammar of queen anne houses. *Environment and Planning B: Planning and Design* 14, 3 (1987), 323–350.
- [FO98] FELKEL P., OBRZĄDEK S.: Straight skeleton implementation. In *SCCG: Spring Conference on Computer Graphics* (1998), p. 210–218.
- [Fre11] FREDERICKSON M.: Modeling the city of london in cars 2. In *Course: Modeling 3D Urban Spaces Using Procedural and Simulation-Based Techniques*. ACM SIGGRAPH, 2011.
- [FSH11] FISHER M., SAVVA M., HANRAHAN P.: Characterizing structural relationships in scenes using graph kernels. *ACM Trans. Graph.* 30 (August 2011), 34:1–34:12.
- [GHLY09] GAO J., HU Y., LIU J., YANG R.: Unsupervised learning of high-order structural semantics from images. In *ICCV* (2009), pp. 2122–2129.
- [GMB06] GLASS K. R., MORKEL C., BANGAY S. D.: Duplicating road patterns in south african informal settlements using procedural techniques. In *Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa* (New York, NY, USA, 2006), AFRIGRAPH '06, ACM, pp. 161–169.
- [GO10] GERDELAN A., O'SULLIVAN C.: A genetic-fuzzy system for optimising agent steering. *CASA* 21, 3–4 (2010), 453–461.
- [GPGB11] GALIN E., PEYTAIE A., GURIN E., BENES B.: Authoring Hierarchical Road Networks. *Computer Graphics Forum* 30, 7 (2011).



- [GPMG10] GALIN E., PEYTAVIE A., MARCHAL N., GURIN E.: Procedural generation of roads. *Computer Graphics Forum* 29, 2 (2010), 429–438.
- [GPSL03] GREUTER S., PARKER J., STEWART N., LEACH G.: Real-time procedural generation of ‘pseudo infinite’ cities. In *GRAPHITE: Computer Graphics and Interactive Techniques* (New York, NY, USA, 2003), ACM, p. 87–ff.
- [Gre95] GREEN P. J.: Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika* 82 (1995), 711–732.
- [Han92] HANAN J. S.: *Parametric l-systems and their application to the modelling and visualization of plants*. PhD thesis, 1992. AAINN83871.
- [HBW06a] HAHN E., BOSE P., WHITEHEAD A.: Lazy generation of building interiors in realtime. In *CCECE: Candian Conference on Electrical and Computer Engineering* (May 2006), pp. 2441–2444.
- [HBW06b] HAHN E., BOSE P., WHITEHEAD A.: Persistent realtime building interior generation. In *Sandbox: ACM Symposium on Videogames* (Boston, Massachusetts, 2006), ACM, pp. 179–186.
- [HCF97] HART J. C., COCHRAN W. O., FLYNN P. J.: Similarity hashing: A computer vision solution to the inverse problem of linear fractals. *Fractals* 5, 1 (1997), 35–50.
- [HCL09] HINKS T., CARR H., LAEFER D. F.: Flight optimization algorithms for aerial lidar capture for urban infrastructure model generation. *Journal of Computing in Civil Engineering* 23, 6 (2009), 330.
- [HEH05] HOIEM D., EFROS A. A., HEBERT M.: Geometric context from a single image. *IN ICCV 2005* (2005), 654–661.
- [HLEL06] HAYS J., LEORDEANU M., EFROS A., LIU Y.: Discovering texture regularity as a higher-order correspondence problem. In *9th European Conference on Computer Vision* (May 2006).
- [HMFN04] HONDA M., MIZUNO K., FUKUI Y., NISHIHARA S.: Generating autonomous time-varying virtual cities. In *Cyberworlds, 2004 International Conference on* (nov. 2004), pp. 45–52.
- [HO03] HAMILL J., O’SULLIVAN C.: Virtual dublin - a framework for real-time urban simulation. *Journal of WSCG* 11 (2003), 221–225.

- [Hou59]        HOUGH P. V. C.: Machine Analysis of Bubble Chamber Pictures. In *International Conference on High Energy Accelerators and Instrumentation* (CERN, 1959).
- [HS88]        HARRIS C., STEPHENS M.: *A combined corner and edge detector*, vol. 15. Manchester, UK, 1988, pp. 147–151.
- [HSEH07]     HOIEM D., STEIN A. N., EFROS A. A., HEBERT M.: Recovering occlusion boundaries from a single image. In *2007 IEEE 11th International Conference on Computer Vision* (Rio de Janeiro, Brazil, 2007), pp. 1–8.
- [Hut81]        HUTCHINSON J.: Fractals and self-similarity. *Indiana Univ. Math. J.* 30 (1981), 713–747.
- [HWA\*10]     HAEGLER S., WONKA P., ARISONA S. M., VAN GOOL L., MLLER P.: Grammar-based encoding of facades. *Computer Graphics Forum* 29, 4 (2010), 1479–1487.
- [IMIM08]     IJIRI T., MÊCH R., IGARASHI T., MILLER G.: An example-based procedural system for element arrangement. *Computer Graphics Forum* 27, 2 (2008), 429–436.
- [JFK03]        JOJIC N., FREY B. J., KANNAN A.: Epitomic analysis of appearance and shape. *Proceedings Ninth IEEE International Conference on Computer Vision* 1, Iccv (2003), 34–41 vol.1.
- [Joo08]        JOOST V. D.: Interior mapping - a new technique for rendering realistic buildings. *Computer Graphics International* (2008).
- [KE81]        KONING H., EIZENBERG J.: The language of the prairie: Frank lloyd wright's prairie houses. *Environment and Planning B: Planning and Design* 8, 3 (1981), 295–323.
- [KGBR10]     KEARNEY G., GORZEL M., BOLAND F., RICE H.: Depth perception in interactive virtual acoustic environments using higher order ambisonic sound-fields. In *Proceedings, 2nd International Symposium on Ambisonic and Spherical Acoustics* (Paris, France, May 2010).
- [KK11a]        KRECKLAU L., KOBELT L.: Procedural modeling of interconnected structures. *Computer Graphics Forum* 30, 2 (2011), 335–344.
- [KK11b]        KRECKLAU L., KOBELT L.: Realtime compositing of procedural facade textures on the gpu. In *3DARCH11* (2011), pp. xx–yy.

- [KLCF10] KIM V., LIPMAN Y., CHEN X., FUNKHOUSER T.: Mobius transformations for global intrinsic symmetry analysis. *Computer Graphics Forum (Symposium on Geometry Processing)* 29, 5 (July 2010).
- [KM07] KELLY G., MCCABE H.: Citygen: An interactive system for procedural city generation. In *Fifth International Conference on Game Design and Technology* (2007), pp. 8–16.
- [Koh29] KOHLER W.: Gestalt psychology. New York: H. Liveright (1929).
- [Koh92] KOHLER W.: *Gestalt Psychology: An Introduction to New Concepts in Modern Psychology*. Liveright Publishing Corporation, July 1992.
- [KPK10] KRECKLAU L., PAVIC D., KOBBELT L.: Generalized use of non-terminal symbols for procedural modeling. *Computer Graphics Forum* 29, 8 (2010), 2291–2303.
- [KR07] KORAH T., RASMUSSEN C.: 2d lattice extraction from structured environments. *2007 IEEE International Conference on Image Processing 2* (2007), II – 61–II – 64.
- [KW11] KELLY T., WONKA P.: Interactive architectural modeling with procedural extrusions. *ACM Trans. Graph.* 30 (April 2011), 14:1–14:15.
- [KWR07] KANNAN A., WINN J., ROTHER C.: Clustering appearance and shape by learning jigsaws. vol. 19, MIT Press, p. 657.
- [LBHL08] LIU Y., BELKINA T., HAYS J., LUBLINERMAN R.: Image de-fencing. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on* (june 2008), pp. 1 –8.
- [LCDF10] LIPMAN Y., CHEN X., DAUBECHIES I., FUNKHOUSER T.: Symmetry factored embedding and distance. In *ACM SIGGRAPH 2010 papers* (New York, NY, USA, 2010), SIGGRAPH '10, ACM, pp. 103:1–103:12.
- [LD03] LAYCOCK R. G., DAY A. M.: Automatically generating roof models from building footprints. WSCG (2003).
- [Lev44] LEVENBERG K.: A method for the solution of certain non-linear problems in least squares. *Quarterly Journal of Applied Mathematics II*, 2 (1944), 164–168.
- [Ley92] LEYTON M.: *Symmetry, Causality, Mind*. MIT Press, Apr. 1992.



- [LHo6] LEFEBVRE S., HOPPE H.: Appearance-space texture synthesis. In *ACM Trans. Graph.* (Boston, Massachusetts, 2006), ACM, pp. 541–548.
- [Lin68] LINDENMAYER A.: Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs. *Journal of Theoretical Biology* 18, 3 (Mar. 1968), 300–315.
- [LNo4] LEE S. C., NEVATIA R.: *Extraction and integration of window in a 3D building model from ground view images*, vol. 2. IEEE Computer Society, 2004, pp. 113–120.
- [Low04] LOWE D. G.: Distinctive image features from Scale-Invariant keypoints. *Int. J. Comput. Vision* 60, 2 (2004), 91–110.
- [LPDO10] LARKIN M., PARIS S., DOBBYN S., O’SULLIVAN C.: Every last detail: density based level of detail control for crowd rendering. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2010), I3D ’10, ACM, pp. 2:1–2:1.
- [LRW\*06] LECHNER T., REN P., WATSON B., BROZEFSKI C., WILENSKI U.: Procedural modeling of urban land use. In *ACM SIGGRAPH 2006 Research posters* (Boston, Massachusetts, 2006), ACM, p. 135.
- [LS04] LEIBE B., SCHIELE B.: Scale-invariant object categorization using a scale-adaptive mean-shift search. *Proc DAGM Symposium*, 1 (2004), 145–153.
- [LSWW11] LIPP M., SCHERZER D., WONKA P., WIMMER M.: Interactive modeling of city layouts using layers of procedural content. *Computer Graphics Forum* 30, 2 (2011), 345–354.
- [LTSW09] LASOWSKI R., TEVS A., SEIDEL H.-P., WAND M.: A probabilistic framework for partial intrinsic symmetries in geometric data. In *Twelfth IEEE International Conference on Computer Vision (ICCV 2009)* (Kyoto, Japan, 2009), Cipolla R., Hebert M., Tang X., Yokoya N., (Eds.), IEEE Computer Society, pp. 963–970.
- [LWH\*05] LIU H., WANG Q., HUA W., ZHOU D., BAO H.: Building chinese ancient architectures in seconds. *ICCS 2* (2005), 248–255.
- [LWW08] LIPP M., WONKA P., WIMMER M.: Interactive visual editing of grammars for procedural architecture. *ACM Trans. Graph.* 27, 3 (2008), 10, 1.

- [Mac67] MACQUEEN J. B.: Some methods for classification and analysis of multivariate observations. In *5th Berkeley Symposium on Mathematical Statistics and Probability* (1967), pp. 281–297.
- [Man83] MANDELBROT B.: *The fractal geometry of nature*. W.H. Freeman, 1983.
- [MBB10] MITRA N. J., BRONSTEIN A., BRONSTEIN M.: Intrinsic regularity detection in 3d geometry. In *Proceedings of the 11th European conference on computer vision conference on Computer vision: Part III* (Berlin, Heidelberg, 2010), ECCV’10, Springer-Verlag, pp. 398–410.
- [Mero07] MERRELL P.: Example-based model synthesis. In *I3D: Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2007), ACM, p. 105–112.
- [MGPO6] MITRA N. J., GUIBAS L. J., PAULY M.: Partial and approximate symmetry detection for 3D geometry. *ACM Trans. Graph.* 25, 3 (2006), 560–568.
- [MGPO7] MITRA N. J., GUIBAS L. J., PAULY M.: Symmetrization. In *ACM SIGGRAPH 2007 papers* (San Diego, California, 2007), ACM, p. 63.
- [MHC\*08] MORVAN Y., HINKS T., CARR H., LAEFER D., MORRISH S., O’SULLIVAN C.: Post-facto registration tools for urban modelling. In *Eurographics, Short Papers* (2008), pp. 29–32.
- [MIK93] MINOVIC P., ISHIKAWA S., KATO K.: Symmetry identification of a 3-D object represented by octree. *IEEE Trans. Pattern Anal. Mach. Intell.* 15, 5 (1993), 507–514.
- [Miro01] MIRANDA E.: *Composing music with computers*. No. v. 1 in Music technology series. Focal Press, 2001.
- [MJH\*09] McDONNELL R., JÖRG S., HODGINS J. K., NEWELL F., O’SULLIVAN C.: Evaluating the effect of motion and body shape on the perceived sex of virtual characters. *ACM Trans. Appl. Percept.* 5 (February 2009), 20:1–20:14.
- [MK09] MICUSIK B., KOSECKA J.: Semantic segmentation of street scenes by super-pixel co-occurrence and 3d geometry. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on* (2009), pp. 625–632.
- [MKG\*10] MASTERSON C., KEARNEY G., GORZEL M., RICE H., BOLAND F.: Optimised virtual loudspeaker reproduction. In *Proceedings, 21st Irish Signals and Systems Conference* (Cork, Ireland, June 2010).

- [MLD\*08] McDONNELL R., LARKIN M., DOBBYN S., COLLINS S., O'SULLIVAN C.: Clone attack! perception of crowd variety. *ACM Trans. Graph.* 27 (August 2008), 26:1–26:8.
- [MLH\*09] McDONNELL R., LARKIN M., HERNÁNDEZ B., RUDOMIN I., O'SULLIVAN C.: Eye-catching crowds: saliency based selective variation. *ACM Trans. Graph.* 28 (July 2009), 55:1–55:10.
- [MM08] MERRELL P., MANOCHA D.: Continuous model synthesis. *ACM Trans. Graph.* 27, 5 (2008), 1–7.
- [MM09] MERRELL P., MANOCHA D.: Constraint-based model synthesis. In *SPM '09: SIAM/ACM Joint Conference on Geometric and Physical Modeling* (New York, NY, USA, 2009), ACM, p. 101–111.
- [MMON09] MCHUGH J. E., McDONNELL R., O'SULLIVAN C., NEWELL F.: Perceiving emotion in crowds: the role of dynamic body postures on the perception of emotion in crowded scenes. *Experimental Brain Research* 204, 3 (2009), 361–372.
- [MO09] MORVAN Y., O'SULLIVAN C.: Handling occluders in transitions from panoramic images: A perceptual study. *ACM Trans. Appl. Percept.* 6 (October 2009), 25:1–25:15.
- [MO10] McDONNELL R., O'SULLIVAN C.: Movements and voices affect perceived sex of virtual conversers. In *Proceedings of the 7th Symposium on Applied Perception in Graphics and Visualization* (New York, NY, USA, 2010), APGV '10, ACM, pp. 125–128.
- [MP96] MĚCH R., PRUSINKIEWICZ P.: Visual models of plants interacting with their environment. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 397–410.
- [MP02] MICHALEK J., PAPALAMBROS P. Y.: Interactive design optimization of architectural layouts. *Eng. Opt.* 34 (2002), 485–501.
- [MP08] MITRA N. J., PAULY M.: Symmetry for architectural design. In *Advances in Architectural Geometry* (2008), p. 13–16.
- [MR05] MAYER H., REZNIK S.: Building facade interpretation from image sequences. *ISPRS* 36, 3 (2005), 55–60.



- [MRo6] MAYER H., REZNIK S.: *MCMC linked with implicit shape models and plane sweeping for 3D building facade interpretation in image sequences*, vol. 36. ISPRS Comm. III Symposium, IAPRS, 2006, p. 3.
- [MRo7a] MAYER H., REZNIK S.: Building facade interpretation from uncalibrated wide-baseline image sequences. *PandRS* 61, 6 (Feb. 2007), 371–380.
- [MRo7b] MAYER H., REZNIK S.: Building facade interpretation from uncalibrated wide-baseline image sequences. *ISPRS Journal of Photogrammetry and Remote Sensing* 61, 6 (2007), 371–380.
- [MRM\*10] MUSIALSKI P., RECHEIS M., MAIERHOFER S., WONKA P., PURGATHOFER W.: Tiling of ortho-rectified facade images. In *Proceedings of the 26th Spring Conference on Computer Graphics* (New York, NY, USA, 2010), SCCG '10, ACM, pp. 117–126.
- [MRPSo8] McDONALD P., RICE H., PILLA F., SIMON D.: Communicating environmental noise data using virtual soundscaping. In *Proceedings, Internoise* (Shanghai, China, 2008).
- [MSHSo6] MARTINET A., SOLER C., HOLZSCHUCH N., SILLION F. X.: Accurate detection of symmetries in 3D shapes. *ACM Trans. Graph.* 25, 2 (2006), 439–464.
- [MSK10] MERRELL P., SCHKUFZA E., KOLTUN V.: Computer-generated residential building layouts. *ACM Trans. Graph.* 29 (December 2010), 181:1–181:12.
- [MVW\*06] MÜLLER P., VEREENOOGHE T., WONKA P., PAAP I., GOOL L. V.: Procedural 3D reconstruction of puuc buildings in xkipché. In *VAST: Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage* (2006), p. 139–146.
- [MWH\*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., GOOL L. V.: Procedural modeling of buildings. *ACM Trans. Graph.* 25, 3 (2006), 614–623.
- [MWR\*09] MUSIALSKI P., WONKA P., RECHEIS M., MAIERHOFER S., PURGATHOFER W.: Symmetry-Based Facade Repair. In *VMV* (2009), Magnor M. A., Rosenhahn B., Theisel H., Magnor M. A., Rosenhahn B., Theisel H., (Eds.), DNB, pp. 3–10.
- [MZWVG07] MÜLLER P., ZENG G., WONKA P., VAN GOOL L.: Image-based procedural modeling of facades. *ACM Trans. Graph.* 26 (July 2007).
- [NSZ\*10] NAN L., SHARF A., ZHANG H., COHEN-OR D., CHEN B.: Smartboxes for interactive urban reconstruction. *ACM Trans. Graph.* 29 (July 2010), 93:1–93:10.

- [OB91] ORCHARD M., BOUMAN C.: Color quantization of images. *IEEE Transactions on Signal Processing* 39 (1991), 2677–2690.
- [OSGo8] OVSJANIKOV M., SUN J., GUIBAS L.: Global intrinsic symmetries of shapes. In *Proceedings of the Symposium on Geometry Processing* (Aire-la-Ville, Switzerland, Switzerland, 2008), SGP '08, Eurographics Association, pp. 1341–1348.
- [Ots79] OTSU N.: A threshold selection method from gray-level histograms. *Systems, Man and Cybernetics, IEEE Transactions on* 9, 1 (jan. 1979), 62–66.
- [PDBo6] PARIS S., DONIKIAN S., BONVALET N.: Environmental abstraction and path planning techniques for realistic crowd simulation. *Journal of Visualization and Computer Animation* 17, 3-4 (2006), 325–335.
- [Pea01] PEARSON K.: On lines and planes of closest fit to systems of points in space. *Philosophical Magazine* 2, 11 (1901), 559–572.
- [PGOo9] PARIS S., GERDELAN A., O'SULLIVAN C.: Ca-lod: Collision avoidance level of detail for scalable, controllable crowds. *Lecture Notes in Computer Graphics (Motion In Games)* (2009), 13–28.
- [PHHM96] PRUSINKIEWICZ P., HAMMEL M., HANAN J., MECH R.: L-systems: From the theory to visual models of plants.
- [PHO11] PRAŽÁK M., HOYET L., O'SULLIVAN C.: Perceptual evaluation of footskate cleanup. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2011), SCA '11, ACM, pp. 287–294.
- [PKM\*10] PRAZAK M., KAVAN L., McDONNELL R., DOBBYN S., O'SULLIVAN C.: Moving crowds: a linear animation system for crowd simulation. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2010), I3D '10, ACM, pp. 9:1–9:1.
- [PL90] PRUSINKIEWICZ P., LINDENMAYER A.: *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., 1990.
- [PMo1] PARISH Y. I. H., MÜLLER P.: Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM, pp. 301–308.
- [PMW\*08] PAULY M., MITRA N. J., WALLNER J., POTTSMANN H., GUIBAS L. J.: Discovering structural regularity in 3d geometry. *ACM Trans. Graph.* 27, 3 (2008), 1–11.

- [pro] Procedural inc. - 3D modeling software for urban environments. <http://www.procedural.com/>.
- [Pru86] PRUSINKIEWICZ P.: Graphical applications of l-systems. In *Proceedings on Graphics Interface '86/Vision Interface '86* (Toronto, Ont., Canada, Canada, 1986), Canadian Information Processing Society, pp. 247–253.
- [PSG\*06] PODOLAK J., SHILANE P., GOLOVINSKIY A., RUSINKIEWICZ S., FUNKHOUSER T.: A planar-reflective symmetry transform for 3D shapes. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 25, 3 (July 2006).
- [RHDG10] RISSE E., HAN C., DAHYOT R., GRINSFELD E.: Synthesizing structured image hybrids. *ACM Trans. Graph.* 29 (July 2010), 85:1–85:6.
- [RKBo4] ROTHER C., KOLMOGOROV V., BLAKE A.: "GrabCut": interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.* 23, 3 (2004), 309–314.
- [RRA08] RICARD J., ROYAN J., AUBAULT O.: Visualization of real cities based on procedural modeling. *Orange Labs* (2008).
- [RT00] RUZON M. A., TOMASI C.: Alpha estimation in natural images. *Proceedings IEEE Conference on Computer Vision and Pattern Recognition CVPR 2000 Cat NoPR00662 1*, June (2000), 18–25.
- [SB03] SCHINDLER K., BAUER J.: A model-based method for building reconstruction. In *Higher-Level Knowledge in 3D Modeling and Motion Analysis, 2003. HLK 2003. First IEEE International Workshop on* (oct. 2003), pp. 74 – 82.
- [SG72] STINY G., GIPS J.: Shape grammars and the generative specification of painting and sculpture. In *Information Processing '71* (Amsterdam, 1972), Friedman C. V., (Ed.), p. 1460–1465.
- [Sha74] SHAVIV E.: *A Model for space allocation in complex buildings: a computer graphics approach*. Faculty of Architecture and Town Planning, 1974.
- [SKG\*09] SMELIK R. M., KRAKER K. J. D., GROENEWEGEN S. A., TUTENEL T., BIDARRA R.: A survey of procedural methods for terrain modelling. In *Proc. of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMI-GAS)* (2009).
- [SM78] STINY G., MITCHELL W. J.: The palladian grammar. *Environment and Planning B: Planning and Design* 5, 1 (1978), 5 – 18.



- [S0009] SOON T. T.: Generalized descriptions for the procedural modeling of ancient east asian buildings. In *Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging(CAE'09)* (2009).
- [SP86] SEDERBERG T. W., PARRY S. R.: Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph.* 20 (August 1986), 151–160.
- [SQ79] SZILARD A., QUINTON R.: An interpretation for DoL-systems by computer graphics. *The Science Terrapin* 4 (1979), 8–13.
- [SRMG10] SMYTH R., RICE H., McDONALD P., GERDELAN A.: Simulation of vehicle noise in the virtual city. In *Proceedings, Internoise* (Lisbon, Portugal, June 2010).
- [SS97] SUN C., SHERRAH J.: 3D symmetry detection using the extended Gaussian image. *IEEE Trans. Pattern Anal. Mach. Intell.* 19, 2 (1997), 164–168.
- [SS06] SHAH M. I., SORENSEN D. C.: A symmetry preserving singular value decomposition. *SIAM J. Matrix Anal. Appl.* 28, 3 (2006), 749–769.
- [Sta93] STAUDACHER P.: New frontiers beyond context-freeness: DI-grammars and DI-automata. In *Proceedings of the sixth conference on European chapter of the Association for Computational Linguistics* (Utrecht, The Netherlands, 1993), Association for Computational Linguistics, pp. 358–367.
- [Sti77] STINY G.: Ice-ray: a note on the generation of chinese lattice designs. *Environment and Planning B: Planning and Design* 4, 1 (January 1977), 89–98.
- [Sti80] STINY G.: Introduction to shape and shape grammars. *Environment and Planning B: Planning and Design* 7, 3 (1980), 343 – 351.
- [Sti82] STINY G.: Spatial relations and grammars. *Environment and Planning B* 9, 1 (1982), 113–114.
- [SY99] SCHAPIRE R. E., YOEM S.: Machine learning. No. 3. 1999, pp. 297–336.
- [SYBG02] SUN J., YU X., BACIU G., GREEN M.: Template-based generation of road networks for virtual city modeling. In *Proceedings of the ACM symposium on Virtual reality software and technology* (New York, NY, USA, 2002), VRST '02, ACM, pp. 33–40.
- [TLL\*11] TALTON J. O., LOU Y., LESSER S., DUKE J., MĚCH R., KOLTUN V.: Metropolis procedural modeling. *ACM Trans. Graph.* 30 (April 2011), 11:1–11:14.

- [TSKP10] TEBOUL O., SIMON L., KOUTSOURAKIS P., PARAGIOS N.: Segmentation of building facades using procedural shape priors. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on* (june 2010), pp. 3105–3112.
- [VABa10] VANEGAS C., ALIAGA D., BENEŠ AND B.: Building reconstruction using manhattan-world grammars. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on* (june 2010), pp. 358–365.
- [VABW09] VANEGAS C. A., ALIAGA D. G., BENEŠ B., WADDELL P. A.: Interactive design of urban spaces using geometrical and behavioral modeling. In *ACM SIGGRAPH Asia 2009 papers* (Yokohama, Japan, 2009), ACM, pp. 1–10.
- [vBM\*10] ŠT'AVA O., BENEŠ B., MĚCH R., ALIAGA D. G., KRIŠTOF P.: Inverse procedural modeling by automatic generation of l-systems. *Computer Graphics Forum* 29, 2 (2010), 665–674.
- [VGZVDBM07] VAN GOOL L., ZENG G., VAN DEN BORRE F., MLLER P.: *Towards Mass-produced Building Models*. Institute of Photogrammetry and Cartography, Technische Universitaet Muenchen, 2007, pp. 209–220.
- [Wad02] WADDELL P.: Urbansim : Modeling urban development for land use , transportation and environmental planning urbansim : Modeling urban development for land use , transportation and environmental planning abstract. *Transportation* 68, 3 (2002), 297–314.
- [WBN\*03] WADDELL P., BORNING A., NOTH M., FREIER N., BECKE M., ULFARSSON G.: Microsimulation of urban development and location choices: Design and implementation of urbansim. *Networks and Spatial Economics* 3, 1 (2003), 4367.
- [WDB10] WENDEL A., DONOSER M., BISCHOF H.: Unsupervised facade segmentation using repetitive patterns. In *Proceedings of the 32nd DAGM conference on Pattern recognition* (Berlin, Heidelberg, 2010), Springer-Verlag, pp. 51–60.
- [Whio6] WHITE C.: King Kong: the building of 1933 New York City. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches* (New York, NY, USA, 2006), ACM.
- [WKM08] WHELAN G., KELLY G., MCCABE H.: Roll your own city. In *Proceedings of the 3rd international conference on Digital Interactive Media in Entertainment and Arts* (New York, NY, USA, 2008), DIMEA '08, ACM, pp. 534–535.
- [WLKT09] WEI L., LEFEBVRE S., KWATRA V., TURK G.: State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Reports, EG-STAR* (2009), Eurographics Association.

- [WMWG09] WEBER B., MÜLLER P., WONKA P., GROSS M.: Interactive geometric simulation of 4D cities. *Computer Graphics Forum* 28, 2 (2009), 481–492.
- [WOD09] WHITING E., OCHSENDORF J., DURAND F.: Procedural modeling of Structurally-Sound masonry buildings. *ACM Transactions on Graphics* 28, 5 (2009), 112.
- [WTT\*02] WANG X., TOTARO S., TAILL F., HANSON A. R., TELLER S.: Recovering facade texture and microstructure from Real-World images. In *In Proc. 2<sup>nd</sup> International Workshop on Texture Analysis and Synthesis at ECCV* (2002), p. 381–386.
- [WU04] WADDLE P., ULFARSSON G. F.: Introduction to urban simulation: design and development of operational models. *Handbook in Transport* 5 (2004), 203–236.
- [WVW85] WOO T., VOLZ R., WOLTER J.: Optimal algorithms for symmetry detection in two and three dimensions. *The Visual Computer* 1, 1 (July 1985), 37–48.
- [WVOH08] WANG H., WEXLER Y., OFEK E., HOPPE H.: Factoring repeated content within and among images. *ACM Trans. Graph.* 27, 3 (2008), 1–10.
- [WWSR03] WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant architecture. *ACM Trans. Graph.* 22, 3 (2003), 669–677.
- [WXL\*11] WANG Y., XU K., LI J., ZHANG H., SHAMIR A., LIU L., CHENG Z., XIONG Y.: Symmetry hierarchy of man-made objects. *Computer Graphics Forum* 30, 2 (2011), 287–296.
- [WZ02] WERNER T., ZISSERMAN A.: New techniques for automated architectural reconstruction from photographs. In *Proceedings of the 7th European Conference on Computer Vision-Part II* (2002), Springer-Verlag, pp. 541–555.
- [XBK09] XIAO J., BAY C. W., KONG H.: Multiple view semantic segmentation for street view images. No. Iccv, Ieee, pp. 686–693.
- [XFT\*08] XIAO J., FANG T., TAN P., ZHAO P., OFEK E., QUAN L.: Image-based facade modeling. *ACM Trans. Graph.* 27, 5 (2008), 1–10.
- [XFZ\*09] XIAO J., FANG T., ZHAO P., LHUILLIER M., QUAN L.: Image-based street-side city modeling. In *ACM SIGGRAPH Asia 2009 papers* (Yokohama, Japan, 2009), ACM, pp. 1–12.



- [XZT\*09] XU K., ZHANG H., TAGLIASACCHI A., LIU L., LI G., MENG M., XIONG Y.: Partial intrinsic reflectional symmetry of 3d shapes. *ACM Trans. Graph.* 28 (December 2009), 138:1–138:10.
- [YMo9] YEH Y.-T., MĚCH R.: Detecting symmetries and curvilinear arrangements in vector art. *Computer Graphics Forum* 28, 2 (2009), 707–716.
- [YYT\*11] YU L.-F., YEUNG S.-K., TANG C.-K., TERZOPOULOS D., CHAN T. F., OSHER S. J.: Make it home: automatic optimization of furniture arrangement. *ACM Trans. Graph.* 30 (August 2011), 86:1–86:12.
- [Zac07] ZACH C.: *High-Performance Modeling From Multiple Views Using Graphics Hardware*. PhD thesis, Graz University of Technology, 2007.
- [ZHT07] ZHANG E., HAYS J., TURK G.: Interactive tensor field design and visualization on surfaces. *IEEE TVCG: Transactions on Visualization and Computer Graphics* 13, 1 (2007), 94–107.
- [ZSW\*10] ZHENG Q., SHARF A., WAN G., LI Y., MITRA N. J., COHEN-OR D., CHEN B.: Non-local scan consolidation for 3d urban scenes. *ACM Trans. Graph.* 29 (July 2010), 94:1–94:9.