LEABHARLANN CHOLÁISTE NA TRÍONÓIDE, BAILE ÁTHA CLIATH | TRINITY COLLEGE LIBRARY DUBLIN
Ollscoil Átha Cliath | The University of Dublin

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.

Texture Synthesis Based Hybridisation for Images and Geometry

by Eric Risser

A Thesis

Submitted to the Office of Graduate Studies

of

Trinity College Dublin

in Candidacy for the Degree of

Doctor of Philosophy

October 31, 2012

TRINITY COLLEGE
2 4 MAY 2013
LIBRARY DUBLIN

Thesis 9838

Declaration

This thesis has not been submitted as an exercise for a degree at any other University. Except where otherwise stated, the work described herein has been carried out by the author alone. This thesis may be borrowed or copied upon request with the permission of the Librarian of Trinity College, University of Dublin.

Signature of Author

©2012 Eric Risser, Trinity College Dublin

Abstract

Texture synthesis deals with the example based creation of highly repetitive information. It is an important and widely studied problem in the computer graphics community due to the ever increasing need for art. Texture synthesis offers a way to automate the creative process so that artists are free to focus on the core aspects of their project, defining a look and style, and free from the mundane and tedious creative tasks of producing many unique instances of similar, repetitive content. The work presented in this thesis takes as input some form of example data and attempts to automatically imagine what more of it could look like.

Specifically, we explore the idea of hybridisation, taking multiple unique objects and mixing them to create new instances of those objects which share similar statistical features both locally and globally with the input. Hybrids occur in most textures but thus far have not been studied as a texture synthesis problem. In this thesis hybridisation is identified as a sub-problem of texture synthesis and hybrids are synthesised independent of the texture they would exist in. A new texture synthesis based hybridisation solution is proposed that generates near infinite unique instances of a hybrid from just a small handful of examples.

Texture, being a kind of information (highly repetitive information), can exist for any type of multimedia data (e.g. images, audio, geometry, animation, etc.). While some work has been done with regards to customizing texture synthesis algorithms into other content domains such as audio, animation and geometry, the bulk of the work done for texture synthesis has focused on images. In fact the challenges specific to the image domain have heavily influenced the direction

texture synthesis research has taken over the past years. We explore an interesting and practical new geometry synthesis approach that allows us to perform texture synthesis and hybridisation directly on 2D curves and 3D meshes, a previously open problem with vast academic and commercial appeal.

Given a handful of unique but similar images, 2D curves or 3D meshes, nearly infinite more similar but unique instances are automatically created. While we do not claim that such an algorithm reproduces the phenomena known as human imagination, we do believe that it mimics the final result and offers an autonomous approximation that is immediately useful to industry for the purpose of art creation. We hope that this thesis opens up an interesting new direction for texture synthesis, hybridisation and artificial imagination.

Acknowledgments

- First and foremost I would like to thank my co-advisors Rozenn Dahyot and Eitan Grinspun for their years of support and guidance. They have been the keel and the sail on this journey. Rozenn, thank you for keeping me balanced and pointed in the right direction whenever I started to tip over or steer off course. Your mathematical expertise and insight has carried me farther than I could have ever hoped. Thank you for giving me the freedom to make mistakes, while always ready to save me whenever I'd start to drown. Eitan, thank you for your positivity and excitement. Your constant influx of good ideas gave me both the raw material and the motivation to keep moving forward. I could not have hoped for two better advisors.

- A big thanks to Charles Han who is not only a collaborator, but also a mentor. It would be a big understatement to say that he influenced the direction of this thesis - he introduced me to texture synthesis in the first place.

- I'd like to thank my internal and external thesis review committee of John Dingliana and Niloy Mitra. This thesis owes much to your incisive questions and urgings to dig deeper. Thank you for your direction and perspective.

- Many thanks to Ravi Ramamoorthi for co-advising me during my masters studies along with Eitan and giving me the initial opportunity to get involved with research at Columbia. You really helped me get my foot in the door.

- I owe a big debt of gratitude to Carol O'Sullivan for inviting me to Trinity

and setting me up with a fellowship. On that note I'd also like to thank Trinity College for giving me years of funding and the freedom to pursue my own research interests and pretty much tinker with whatever I thought was cool - can't ask for a better deal than that! I'd like to thank my lab buddies Brian Cullen, Anton Gerdelan and Yann Morvan for sharing countless pints and laughs, and the good folks of GV2 (Graphics Vision and Visualization Group).

- I'd also like to acknowledge Li-Yi Wei, Hugues Hoppe, Greg Turk, Alyosha Efros and Connelly Barnes for creating the foundation this thesis is built from. They're the giants who's shoulders I'm standing on!

- Finally I'd like to thank my parents for making me wake up early to go to school even when I insisted that I did not want to. You were very persuasive - 20 years later and I'm still here.

Contents

1 Introduction **13**

1.1 Problem Formulation 14

1.1.1 What is Texture? 14

1.1.2 What are Hybrids? 15

1.1.3 What is Structure? 16

1.1.4 Hybridisation from Texture Synthesis 17

1.2 Applications 19

1.2.1 Images 19

1.2.2 Vector Graphics 19

1.2.3 3D Mesh Geometry 20

1.3 Contributions 20

1.3.1 (a) Hybridisation 20

1.3.2 (b) Geometry 21

1.4 Organization 21

2 Background **23**

2.1 Introduction 23

2.2 Image Texture Synthesis 24

2.2.1 Physical Simulation 24

2.2.2 Per-Pixel and Patch Based Synthesis 24

2.2.3 Markov Random Field 25

2.2.4 Coherence 26

2.2.5 Multiscale 28
2.2.6 Multiscale Texture Synthesis 30
2.2.6.1 Upscale 32
2.2.6.2 Jitter 33
2.2.6.3 Correction 33
2.2.6.4 Inconsistency Correction 34
2.2.6.5 Results 35
2.2.7 Symmetry 36
2.2.8 Global Structure 38
2.2.9 Optimisation Based Synthesis 38
2.2.10 Inverse Texture Synthesis 39
2.2.11 Image Inpainting 39
2.3 Geometry Synthesis 40
2.3.1 Modeling by Example 40
2.3.2 Geometry Texture Synthesis 42
2.3.3 Element Based Texture Synthesis 43
2.3.4 Mesh Morphing 43
2.4 Audio Texture Synthesis 45
2.5 Texture Analysis 47
2.5.1 Multiscale 48
2.5.2 Discrete Texton Identification and Extraction 48
2.6 Conclusion 50

3 Synthesising Structured Image Hybrids **51**
3.1 Introduction 51
3.2 Framework 52
3.3 Analysis 53
3.3.1 Gaussian Stack 54
3.3.2 Multiscale descriptors 55
3.3.3 Nearest Neighbour Search 58

3.4 Synthesis 58
3.4.1 Upscale 60
3.4.2 Jitter 60
3.4.3 Correction 62
3.4.4 Exemplar standardisation 64
3.4.5 Symmetric synthesis 65
3.4.6 Exemplar Weighting 66
3.5 Evaluation 66
3.5.1 Output Filtering 72
3.6 Results and Analysis 73
3.7 Discussion 75
3.7.1 Hybrids: Texton Synthesis 75
3.8 Conclusion 76

4 Curve Textures and Hybrids **79**
4.1 Introduction 79
4.2 2D Shape Representation 83
4.2.1 Geometric Input and Output 83
4.2.2 Shape Texels 84
4.2.3 Neighbourhoods 86
4.2.4 Averaging Neighbourhoods 86
4.2.5 Comparing Neighbourhoods 87
4.3 Analysis 90
4.3.1 Multiresolution 90
4.3.2 Nearest Neighbour Search 91
4.4 Synthesis 93
4.4.1 Solver 96
4.4.2 Upscale 98
4.4.3 Jitter 101
4.4.4 Correction 101

4.5 Open vs. Closed Curves . . . 103
4.6 Symmetric Synthesis . . . 104
4.7 Multi-Layer Synthesis . . . 105
4.8 Results . . . 107
4.9 Discussion . . . 111
4.9.1 Limitations . . . 113
4.9.2 Future Work . . . 113
4.10 Conclusion . . . 115

5 Synthesising Geometry Textures and Hybrids 117

5.1 Introduction . . . 117
5.2 3D Shape Representation . . . 118
5.2.1 Geometric Input and Output . . . 118
5.2.2 3D Shape Texel . . . 119
5.2.3 Neighbourhoods . . . 120
5.2.4 Averaging Neighbourhoods . . . 121
5.2.5 Comparing Neighbourhoods . . . 122
5.3 Analysis . . . 126
5.3.1 Multiresolution . . . 126
5.3.2 Nearest Neighbour Search . . . 128
5.4 Synthesis . . . 130
5.4.1 Solving . . . 130
5.4.2 Upscaling . . . 136
5.4.3 Jitter/Correction . . . 138
5.4.4 Symmetric Synthesis . . . 140
5.5 Non-Uniform Multiscale . . . 141
5.5.1 Non-Uniform Neighbourhood Sampling . . . 142
5.5.2 Vertex Activity . . . 144
5.6 Image Geometry . . . 148
5.7 Results . . . 151

5.7.1 Image Geometry Results 151
5.7.2 Geometry Results 155
5.8 Conclusion 157
5.8.1 Limitations and Future Work 159

6 Conclusion **161**
6.1 Summary and Final Words 161
6.2 Future Directions 162

A Image Hybrids **165**

B Curve Hybrids **173**

C Mesh Hybrids **185**
C.1 Semi-Regular Remeshing 185
C.1.1 Related Works 185
C.1.2 Method 187
C.1.2.1 Creating Base Mesh 188
C.1.2.2 Subdivision 189
C.1.3 Results 193
C.2 Supplemental Results 195

Chapter 1

Introduction

Texture is the characteristic physical structure given to an object by the size, shape, arrangement, and proportions of its parts. The world we experience around us can in essence be defined as a collection of textures. Whether it be the grain on a plank of wood, the sound of a baby crying, the movement of a flag in the wind, or even the physical makeup of the stars in the sky.

In this thesis we introduce hybrids as a component of texture and the concept of hybridisation as an adaptation of texture synthesis. The previous work in image texture synthesis is re-purposed and extended for the problem of synthesising structured hybrids of images and geometric objects. This includes identifying those aspects of the algorithm that are central to the texture synthesis process, and those which are specific to a certain type of multimedia data. Extensions of the fundamental texture/hybrid synthesis algorithm from the image to the geometry domain are presented. First the core ideas are presented through their simplest application, 2D curves. Given this foundation the idea is further developed to account for higher dimensions and more sophisticated geometry, specifically the application of hybridising 3D meshes. Figure 1.1 shows some of the different types of multimedia data that can be synthesised; it also identifies and illustrates the specific types of multimedia data we concentrate on in this thesis, images and

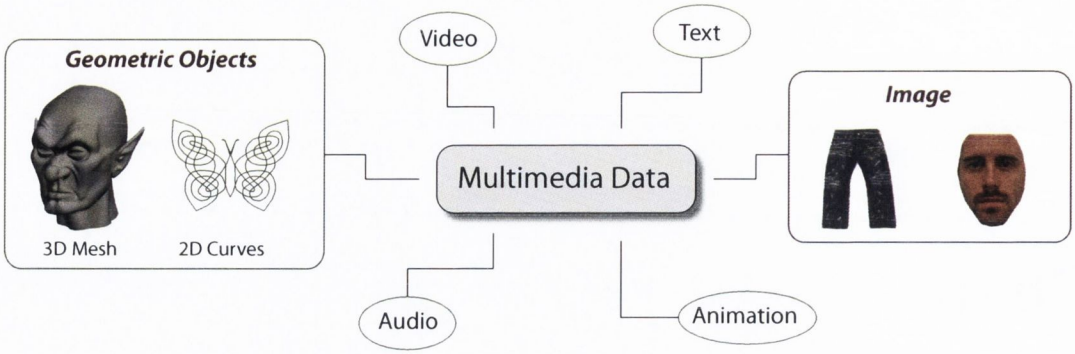


Figure 1.1: Multimedia Data. *While multimedia data is a general term that encompasses many things, we focus on images and geometric objects in this thesis.*

geometric objects.

The rest of this chapter is organised as follows. Section 1.1 poses the problem of texture synthesis, hybrid synthesis and how they are related. Section 1.2 describes the application domains that we explored. Section 1.3 briefly summarises the contributions of this thesis and Section 1.4 outlines the organisation of this thesis.

1.1 Problem Formulation

1.1.1 What is Texture?

Texture can be thought of as highly redundant information. A texture model is a texture that has been distilled down to its unique essence. Although a texture can take up an infinite amount of memory in a computer, the unique information it contains is finite and can be efficiently summarized. Using this unique texture model, an infinite and highly redundant instance of a texture can be created. This is the key concept behind texture synthesis and the focus of this thesis.

Texture has many meanings depending on the context its used in. Webster's English dictionary offers over 10 different definitions for texture, from which we

adopt the two most general for our purposes.

Texture, *noun*

(a) *The essential quality of something*

(b) *The disposition of the several parts of any body in connection with each other, or the manner in which the constituent parts are united; structure; as, the texture of earthy substances or minerals; the texture of a plant or a bone; the texture of paper; a loose or compact texture*

While texture has largely been seen in computer graphics as a property of images and surfaces, this thesis considers a more general meaning of texture so that a theoretical look can be taken at texture as a type of information, described as a statistical model. This definition offers the flexibility to adapt the theory behind texture for the problem of hybridisation and decouples texture synthesis from the image multimedia type, so that it can be generalised to any type of content, be it geometry objects, audio, or animation.

1.1.2 What are Hybrids?

In the most general sense a hybrid is a mixture of two or more things. In our case hybrids refer to the mixing of the individual unique atomic components that comprise a texture. Textures themselves are often composed of many hybrids arranged in some pattern. For example, a crowd of faces is a texture whereby the individual faces are hybrids with some semi-regular spacial offset between them. Since textures can contain hybrids, it follows that hybrids are a component of texture.

Not to be confused with interpolation or morphing (which are a limited subset of hybrids between two or more things), in our case hybrids refer to any possible mixing between two or more objects.

Returning to the crowd example; A single human face contains unique information, the shape of the head, the arrangement of the eyes, nose and mouth. These features comprise the core aspects of a face. Two human faces share those same core aspects but with small variations among them. One hundred human faces start to define a statistical probability model for human faces. This can be thought of as a typical average human face and the amount of variation observed from those average features. This would be a hybrid model for human faces. Hybridisation is the process whereby the statistical features of multiple objects are mixed together semi-randomly such that new unique variations or hybrids of the original inputs are created while the overall local features at any given scale in any of our new examples statistically resemble the inputs.

1.1.3 What is Structure?

Structure, *noun*

(a) *Organization of parts as dominated by the general character of the whole*

(b) *Something arranged in a definite pattern of organization*

(c) *The aggregate of elements of an entity in their relationships to each other*

Structure is a central concept in this thesis. The definition of structure is similar to that of texture and "structure" is used in the definition of texture, as one of its conceptual components. Understanding structure is key to understanding texture and its relationship with hybridisation.

Structure as it relates to texture synthesis can mean many different things. Symmetry patterns in the form of translation, rotation, scale, reflection and skew can all be considered structures. Any unique feature can be considered a structure, as can a repetition of a unique feature. Rather than trying to understand and account for all the ways in which structure can be present in a data set, structure in

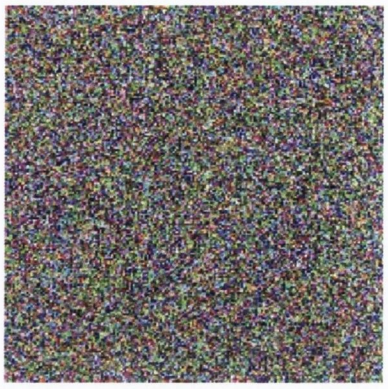


Figure 1.2: Random Image - Noise. *The colour for each pixel is chosen randomly from the set of all possible colours. Each pixel is independent from all other pixels in the image. Structure in this thesis is viewed as the absence of randomness.*

this thesis is considered as the absence of randomness (the inverse of structure). Figure 1.2 illustrates a fully random noise image where each pixel is given a random colour, and no two pixels influence each other's colour values. Structure is preserved in hybrid synthesis by not adding more randomness to output data than exists in the input data being modeled.

1.1.4 Hybridisation from Texture Synthesis

The unique atomic components of a texture are called textons. Any texture can be considered a collection of textons arranged in some way.

The objective of texture synthesis is to take some amount of texture information and create more of the same texture without introducing new information. The problem of texture synthesis consists of (1) the hybridisation of textons and (2) the arrangement of those texton hybrids with respect to each other. The degree to which the arrangement of textons are randomly laid out or follow some kind of symmetry pattern dictates how stochastic vs. structured a texture is. The majority of work in texture synthesis has focused on the statistical variation of

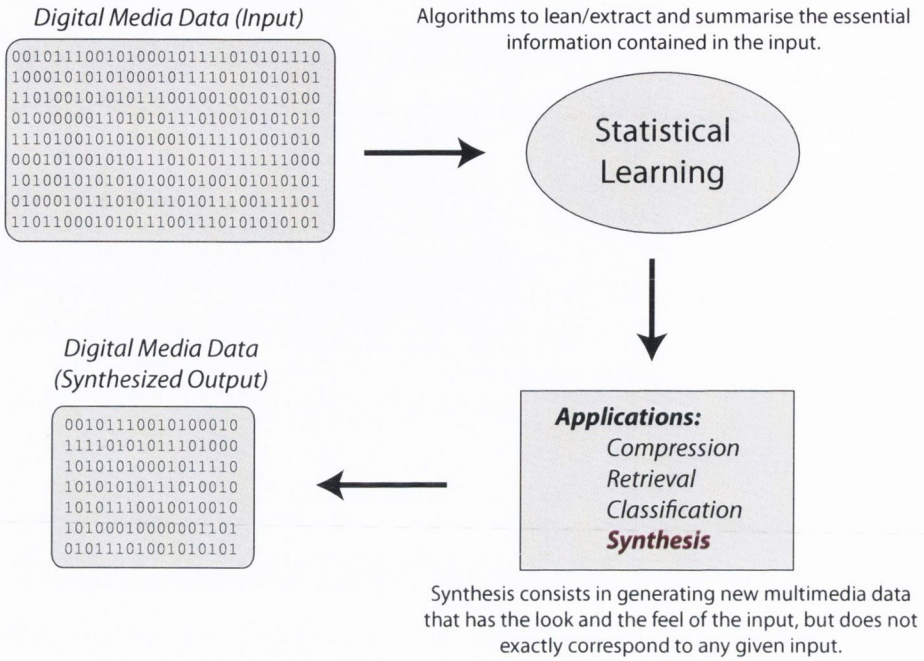


Figure 1.3: Example Based Texture/Hybrid Synthesis. *Given some input example data, the essential information is extracted and summarized through a statistical learning process. This information is then used in synthesis to generate a new unique output that is similar to the input in terms of look and feel.*

the arrangement of textons in a texture, ignoring the statistical variation among textons themselves. Thus, hybridisation is an overlooked sub-problem of texture synthesis. In both cases, though, the algorithm is given some example data and must reproduce the statistical characteristics of that data in a new form. This thesis explains the texture synthesis problem in Chapter 3 and then delves into how the same algorithm can be modified and extended to also synthesise hybrids of textons.

In both cases, some input example data is fed into a statistical learning process that summarises the essential information, and this information is then used to generate a new similar instance of that data as shown in Figure 1.3.

1.2 Applications

Texture synthesis while primarily being developed for images, has been applied to various other multimedia data types such as audio, animation, and geometry. Texture has traditionally been viewed as a highly stochastic process with little focus on the structured variations between the individual textons that make up the texture. Because our work focuses on hybridising these textons, we open up a new class of examples to synthesise. Given a set of textons taken from a texture sample, ignoring the relationship between textons, the focus is shifted to the statistical variations between the textons themselves. We synthesise a previously overlooked aspect of texture as an independent problem and by doing so we solve a long-standing open problem in computer graphics - how to synthesise limitless instances of an object from some handful of examples, hybridising them in all possible ways.

1.2.1 Images

Texture synthesis has had the largest impact in the image domain by generating texture maps that can be rendered on top of geometry giving it more detail. Hybridisation solves the open problem of generating limitless variations of highly structured texture maps, such as the faces on character models, or the wings of a butterfly.

1.2.2 Vector Graphics

While many images contain large noisy areas and are suitable for classic texture synthesis, very few geometric shapes exhibit those characteristics and are amenable to this type of approach. The concept of hybridisation however is very practical for geometric shapes as many physical objects in the world are structured variations of a common template. We show how some types of vector graphics consisting of one or more open or closed 2D curves can be hybridized using this approach.

1.2.3 3D Mesh Geometry

Manifold 3D meshes comprised of networks of vertices have always been a difficult problem for texture synthesis. This area has been largely overlooked due to the limited application for such an approach (highly stochastic terrains, tree bark, etc.). Hybridisation of 3D meshes however is a highly sought after application due to the commercial usefulness. Most 3D objects in the world can be considered variations on some common template for that object. This means that hybridisation makes 3D mesh synthesis a very sensible and practical topic of study.

1.3 Contributions

Our contributions can be organised into two categories.

(a) The introduction of and solution to hybridisation as a sub-problem of texture synthesis.

(b) A completely geometric alternative to the common image based texture synthesis framework.

1.3.1 (a) Hybridisation

We introduce a new more general definition of texture that includes and accounts for hybrids. Hybridisation has been a previously overlooked sub-problem of texture synthesis. We present new approaches to extend previous texture synthesis methods to the problem of hybrid synthesis. We present these new concepts in the image domain and then apply them to the geometry domain, both in the form of vector graphics as well as 3D meshes. While work has been done to synthesise geometry textures in the past, all previous work treats geometry as a function that is discretised into a regular grid of samples so that image texture synthesis

approaches can be used directly. Our contribution is the introduction of a native geometry texture representation such that texture and hybrid synthesis can be performed on geometry natively without the need for discretization or any other intermediate representation.

1.3.2 (b) Geometry

In this work we introduce new concepts for performing geometry-based texture synthesis directly on geometric information. For simplicity these ideas are introduced through the problem of 2D curve synthesis and show how we can solve the problem of synthesising curves hybrids and textures by example, a previously open problem in computer graphics. Given the basic concepts we develop for hybridisation and 2D curve synthesis, we show how to extend our formulations to three dimensions such that 3D mesh hybrids can be natively synthesised.

1.4 Organization

Chapter 2 establishes a background in the fields of image texture synthesis, geometry synthesis, audio texture synthesis, animation texture synthesis and texture analysis. This offers a comprehensive discussion of previous research in texture synthesis as well as a full background on all the subjects relevant to this thesis. Chapter 3 opens the discussion on structured image hybrids by detailing the approach taken to customize a texture synthesis algorithm for the purpose of synthesising hybrids. These extensions are agnostic with respect to the texture synthesis framework they are presented with and could be applied to other frameworks.

Armed with a new hybridisation approach, key concepts for geometry hybrids are introduced in Chapter 4 by targeting the problem of texture and hybrid synthesis of 2D curves. In Chapter 5 these concepts are extended to higher dimensions and applied to 3D mesh data for model texture and hybrid synthesis.

Chapter 2

Background

2.1 Introduction

Texture Synthesis is a popular and mature field. The background focuses mostly on texture synthesis topics while also looking at areas that are peripheral to texture synthesis or appropriate to hybridisation or other topics of this thesis. This chapter can be organised into the following categories:

- Image Texture Synthesis 2.2
- Geometry Synthesis 2.3
- Audio Texture Synthesis 2.4
- Texture Analysis 2.5

2.2 Image Texture Synthesis

2.2.1 Physical Simulation

One way to synthesise image textures is to directly simulate their physical generation processes. Biological patterns such as fur, scales, and skin can be modeled using reaction diffusion [WK91] and cellular texturing [Wor96]. Some weathering and mineral phenomena can be faithfully reproduced by detailed simulations. These techniques can produce textures directly on 3D meshes so the texture mapping distortion problem is avoided. However, different textures are usually generated by very different physical processes, so these approaches are applicable to only limited classes of textures.

2.2.2 Per-Pixel and Patch Based Synthesis

Image texture synthesis can be divided into two categories. (1) Algorithms that synthesise single pixels individually with respect to other pixels and (2) algorithms that synthesise entire patches of pixels with respect to other patches. While per-pixel approaches are generally more powerful and popular, early approaches ran in sequential order and did a poor job at preserving highly non-stationary objects in an image.

Thus, due to the running time disadvantages of synthesising a texture pixel by pixel in scan-line order, especially utilising a Markov Random Field (MRF) [PLM98] approach on early computers, patch based approaches were explored due to their ability to synthesise large portions of a texture at once, drastically improving speed at the sacrifice of variety, flexibility and in some cases quality. Image Quilting [EF01] is a simple and widely utilised example of this idea. An image is synthesised by taking large square-sized patches from the example texture and placing these blocks in the synthesised image in scan-line order where each

new block is placed, such that it slightly overlaps with the blocks to the top and left of it. The most appropriate patch from the exemplar is found by taking a sample that minimises the perceptual difference between the new and old pixels in the overlap region. Within the overlap region a border-detecting algorithm is used which finds the continuous border of pixels within the overlap that minimises seam error and the two blocks are edited appropriately in order to remove overlapped pixels. Several other approaches utilise similar strategies [GB00; LLX*01] by copying whole patches of texture at a time. The general strategy of these types of algorithms is to choose the most appropriate matching patch and then find an optimal boundary that minimises the noticeable seam between the new patch and the rest of the image. One of the most successful strategies [KSE*03] introduced an optimisation-like approach where new patches can be strategically placed over regions in the synthesised image which display poor seam matching, especially in regions where several patches meet and have multiple seams converging at a point causing noticeable artifacts. By iteratively copying over visible seams with new better-matching patches, the global seam error is minimized.

In general while these approaches are less computationally expensive in comparison to per-pixel techniques, they have yet to be explored in a parallel framework as was done for per-pixel methods [WL02; LH05; LH06] making them inappropriate for modern hardware and unable to run on GPUs, thus contemporary per-pixel methods are generally much faster due to their parallel nature. Also, patch based methods lack the fine scale variation seen in per-pixel approaches and the algorithmic flexibility to achieve effects such as Image Analogies [HJO*01].

2.2.3 Markov Random Field

While many texture synthesis approaches exist, so far the most successful and numerous of these techniques are based on the Markov Random Field (MRF). The MRF is an undirected graphical model in which a set of random variables (the

nodes) have the Markov properties of being *local* and *stationary*. The stationary property dictates that a variable in our MRF or a pixel in our image is characterised by a local set of spatially neighbouring pixels, and this characterization is uniform across all pixels. The locality property dictates that the value of a pixel is predictable by the values of its neighbours only, and is independent of both the values in the rest of the image and by its position in the image.

The goal, then, is to synthesise an output image in which each pixel has a spatial neighbourhood similar to at least one neighbourhood in the input image. Generally, the neighbourhood size is user specified according to the feature size of patterns in the image. This is limited however in that different texture features can exist at different scales. To account for this, a fixed neighbourhood size is now standard with the use of a multiscale framework, as discussed in the Multiscale section. The assumption behind the MRF model is that if each local neighbourhood in the synthesised output image matches some local neighbourhood in the input image, then the final output will be perceptually similar to the input.

2.2.4 Coherence

Given the MRF assumption, output pixels must be synthesised independently of their position in the input image. This requirement has been relaxed in recent works starting with [Ash01] in favour of synthesising coherent groups of pixels together. The reasoning behind this favouring of coherent pixels is that the least mean squared error metric used in texture synthesis is not a perceptually driven method and in fact fails to preserve important high frequency information such as lines and boundaries that the human eye is very adept at recognising, resulting in synthesis artifacts. Coherence can also be thought of as an intermediate step between a purely MRF based per-pixel texture synthesis method and patch based synthesis discussed earlier. Coherent texture synthesis relies on the use of jump maps [ZG02] where the output texture is synthesised indirectly not as colour

values but as pointers to pixels in the input image, and colour values are sampled indirectly.

Coherence simply favors the selection of neighbouring pixels in the output that are also neighbours in the input. So during synthesis, when a pointer to an input pixel is being tested for how well it fits with the rest of the synthesised texture with respect to the input texture, if that pixel has a neighbour whos pointer is assigned to the same neighbour of our current pointer being tested, then that link will be favoured as a match.

This results in coherent patches forming despite each pixel being synthesised separately. This concept has also appeared in Jump Maps [ZS04] and k-coherence [TZL*02]. K-coherence has been widely utilised due to its high quality and speed.

The k-coherence algorithm consists of two parts, a pre-processed analysis step followed by a synthesis step. First, a similarity-set is built for each input pixel where the similarity-set consists of other pixels in the input that share a similar neighbourhood to our input pixel, but are physically distant from our input pixel and from each other in the input image. During synthesis a texture coordinate is used to represent each output pixel rather than a colour value. A candidate set is generated by taking the union of similarity-sets from the neighbourhood surrounding our output pixel. This candidate set is then used to search for a new pixel coordinate with the best current neighbourhood match.

The idea of coherence has also been key to the development of a subproblem within texture synthesis, the efficient searching and identification of approximate nearest neighbours [BSFG09]. Patchmatch quickly finds for each pixel in image A, a link to the most similar pixel in image B as determined by some neighbourhood matching approach. Rather than searching for a pixel in image A, every pixel in image B exhaustively, Patchmatch first tries to propagate strong coherent matches between images by checking for a pixel in image A, its neighbouring pixels in image A, then for their nearest matches in image B, the corresponding neighbour. Thus,

coherent patches of pixels in image A will naturally link to similar coherent patches of pixels in image B. In addition to propagation, Patchmatch also looks for new nearest neighbours for each pixel by checking random pixels in image B that fall within a window around the current best match. Starting with a window the size of image B, the window resolution is iteratively decreased by half until the window is one pixel. The idea of exploiting image coherence has been further explored across scale and orientation [BaBGF10].

2.2.5 Multiscale

Texture exists at potentially infinite scales. Within a single image of a given width, texture can exist at any scale, from the width of the image itself down to a single pixel. While texture can exist at any scale, some textures in fact result from texture features that exist across multiple scales. Scale is a crucial aspect of texture and has been studied in great detail in previous works.

The first use of filtering was proposed in [HB95] where the original input image is down-sampled into a steerable image pyramid through the use of Gaussian filters. A histogram is built for each pyramid sub-band and a noise image is modified so it conforms to the statistics of the input. In this approach, all scale levels are optimised together.

True coarse-to-fine synthesis was introduced with the *Synthesis Pyramid* [DeB97], [PLM98] and became immediately popular. This manner of synthesis is used in the bulk of modern texture synthesis algorithms and is key to our work. In coarse-to-fine synthesis the input is first analysed and separated into scale levels. During synthesis each scale level is synthesised in its own pass where the coarsest level is synthesised first, and then finer levels are synthesised subsequently using the previous coarser level as a guideline for how the current level should be synthesised. This approach was made popular by Wei and Levoy [WL00] where a neighbourhood descriptor spans two scale levels at a time, taking a complete 3×3 neighbourhood

from the previous generated scale level, and a partial 5×5 neighbourhood from the current scale level being synthesised in scan-line order. This style of synthesis has become popular because it produces textures of a high level of quality and is computationally efficient. It allows for coarse, large-scale image features to be synthesised at the top of the pyramid, and then be inherited as finer scale texture is added to it at lower levels of the pyramid, thus achieving true multiscale synthesis.

A subtle but significant advance in the use of multiple synthesis scales, the Gaussian stack, was introduced by Lefebvre and Hoppe [LH05]. The precursor of this data structure, the Gaussian pyramid, consisted of images created from a single source image acting as the foundation of the pyramid and each level stacked above it is half the resolution in each dimension of the stack level below it. Theoretically, for an image of resolution m in both dimensions, there exists $m \times m$ possible Gaussian pyramids that can be formed by translating along the finest resolution of the image. [LH05] makes the observation that the samples of all $m \times m$ exemplar pyramids can be compacted to a single stack of $\log(m)$ images of size $m \times m$, which they refer to as the Gaussian stack. The advantage of the Gaussian stack is that they down-sample without sub-sampling, thus they ensure that two identical image features will produce identically down-samples at a coarser resolution regardless of their translation between each other. This is an important concept both for the following Multiscale Texture Synthesis to be presented, as well as the topic of this thesis and the already published work it includes [RHDG10].

In addition to the main line of multiscale research using Gaussian filters and image/synthesis pyramids, there are several lines of research exploring other multiscale texture representations. One such branch involves the use of wavelets [PS00; BJEYLW01], [GK05] as they are a convenient multi-resolution data structure and simplification approach that is also invertible. The Gabor filter was also explored by [PNN*96] and [NP96] as it approximates the same process that occurs in the

human visual system allowing for a perception inspired synthesis.

2.2.6 Multiscale Texture Synthesis

The current state of the art for utilising scale information for texture synthesis in a parallel controllable framework [LH05] is Multiscale Texture Synthesis [HRRG08]. While this paper is previous work and is not presented as part of this thesis, it will be discussed in detail as it is closely related to and in large part an inspiration to the work presented here.

Multiscale Texture Synthesis focuses on the study of texture across a high range of scales, specifically where the information is presented as a collection of different textures that relate to each other through a distinct space and scale transform. This work addresses the challenges associated with synthesising these multiscale textures. The key idea being a new exemplar data representation, referred to as the Exemplar Graph. This is a directed weighted graph where nodes contain a source texture and directed weighted edges denote both the scale hierarchy as well as the scale difference, where an edge going from node A to node B denotes that B contains at least in part the same texture as A, however at a finer scale. The weight of this edge tells us the scaling transform between the two nodes, where scale $= 2^{weight}$. An example Exemplar Graph is shown in Figure 2.1.

Because multiscale textures tend to be very large, it is necessary to use a hierarchical and fast parallel approach for synthesis. The approach given by Lefebvre and Hoppe [LH05] serves as a framework to build from and the three synthesis steps, *upscale*, *jitter* and *correction* are adopted in addition to coarse-to-fine synthesis using an image pyramid $S_0, S_1, ..., S_T$, where T depends on the desired output image size. Note that the images are not represented by colour values, instead they store pointers $S_t[p] = (i, k, u)$, of some stack level texel, $E_k^i[u]$.

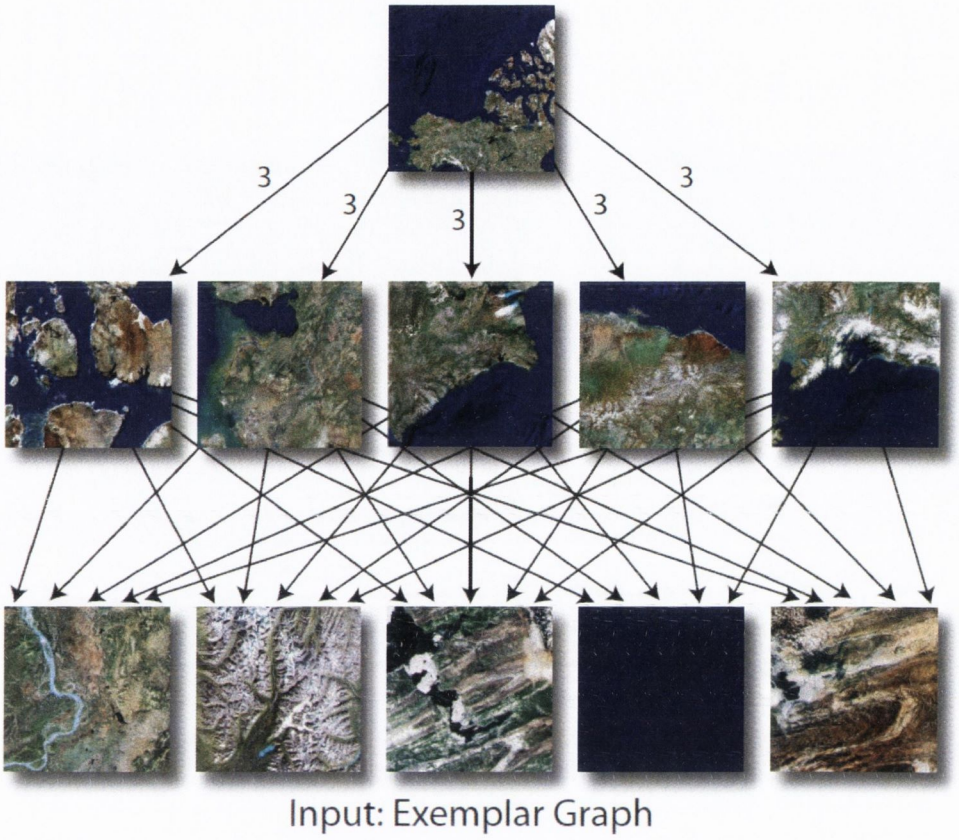


Figure 2.1: Alaska Exemplar Graph. *This exemplar graph contains features of Alaska across a wide range of scales.*

Gaussian stacks

For each exemplar E^i, its Gaussian stack [LH05], $E^i_0, E^i_1, ..., E^i_L$ is computed.

Admissible candidates

Due to the multiscale nature of this algorithm the concept of admissible candidates is used to account for exemplars that match features across scales. While each exemplar stack E^i has its own absolute stack levels where features can be matched internally, for each of those levels $E^i_0, E^i_1, ..., E^i_L$, features can be matched to stack levels in other exemplars than i based on the exemplar graph edges emanating from E^i, and their associated scaling relations. The *admissible candidates* for

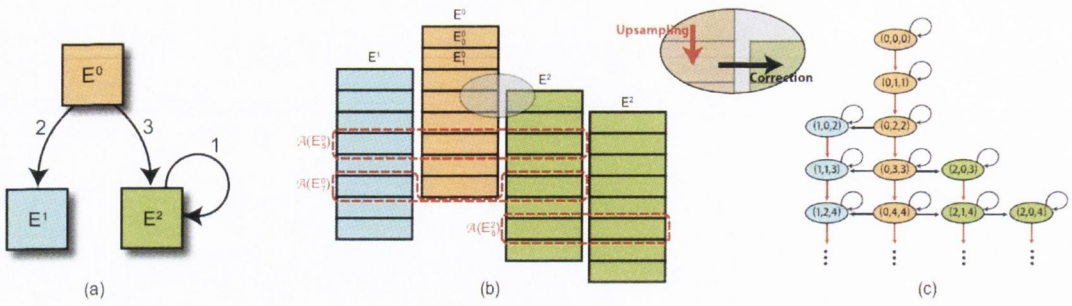


Figure 2.2: Simple Exemplar Graph. *This figure gives a more detailed view of an exemplar graph. (a) shows a simple three image exemplar graph where the root node E^0 points to E^1 and E^2 with a scale offset of 4 and 8 respectively and E^2 recursively points to itself with a scale offset of 2. (b) Gives the resulting Gaussian stack representation of the exemplar graph where stack levels between exemplars which are at the same global scale level are called admissible candidates of one another and during correction new jump map coordinates can be taken from anywhere within an admissible set. Notice that once the algorithm is at the finest scale level of E^0, that scale level is removed from the admissible set, as it no longer has upsampled information. The algorithm must jump to E^1 or E^2. (c) shows the super-exemplar expansion of illustration (a). Nodes correspond to stack levels and red and black edges correspond to upsampling and correction passes respectively. The node labels give the exemplar index, stack level and depth respectively.*

stack level E_k^i are gathered by,

$$A(E_k^i) = E_l^j | \exists (i, j, k-l) \in E, 0 \leq l < L, \tag{2.1}$$

2.2.6.1 Upscale

For each pixel in the image S_{l-1} a coherent 2×2 patch is formed in S_l through upsampling. The operation is defined as the scaling of the parent's coordinate plus

a child-dependent offset:

$$S_l[2p + \Delta] := (2S_{l-1}[p] + h_l\Delta) \bmod m, \tag{2.2}$$

$$\Delta \in \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\} \tag{2.3}$$

Note that when using Gaussian stacks the *physical scale* needs to be accounted for relative to the current synthesis level. $h_k = 2^{L-k}$ is used to denote the regular spacing of a texel in level k of a given stack.

2.2.6.2 Jitter

Jitter in this case is a type of Perlin noise [Per85]. It combines noise across several different scales. Rather than adding noise to pixel colours directly, jitter follows the [LH05] precedent of spatial randomisation on jump map coordinates

$$S_l[p] := (i, k, u + J_t(p)(\mathrm{mod} m)), \text{where } J_t(p) = \lfloor h_l H(p) p_t + (0.5, 0.5) \rfloor \tag{2.4}$$

using the hash function H, and level dependent randomness coefficient p_t.

2.2.6.3 Correction

In order to coerce local statistical similarity between the output and the exemplars, for each synthesised pixel $S_t[p] = (i, k, u)$ the correction step seeks out the point in any admissible stack levels $E_l^j \in A(E_k^i)$, a texel $E_l^j[v]$ whose 5×5 exemplar neighbourhood best matches the local 5×5 neighbourhood. Formally,

$E_l^j[v]$ is the minimizer of the error functional

$$\sum_{\delta \in \{-2 \ldots +2\}^2} \| * S_t[p + \delta] - E_l^j[v + \delta h_l] \|^2$$

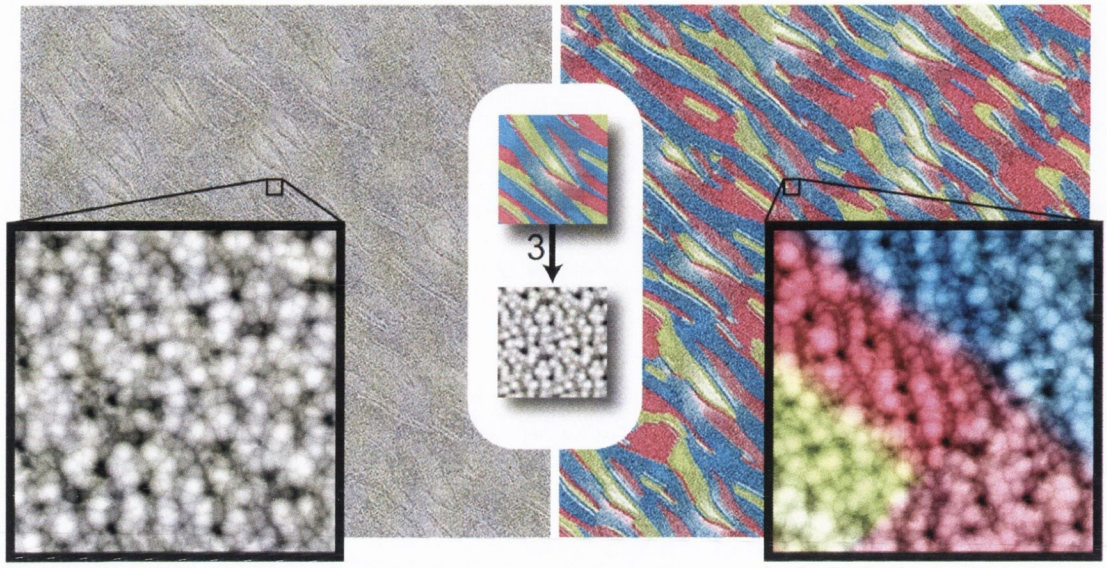


Figure 2.3: Inconsistency correction. *(middle) An exemplar graph with inconsistent colours. (left) Without inconsistency correction coarse level colour statistics are lost since the fine level exemplar provide poor matches for those in the coarse exemplar. (right) Inconsistency correction maintains colour transfer for each pixel and recovers coarse exemplar colours.*

over $E_l^j \in A(E_k^i)$ and $v \in 0...2^L - 1^2$ where $*S_l[p]$ dereferences the pixel pointer $S_l[p]$. Correction pass runs in parallel through the use of 8 subpasses [LH05].

2.2.6.4 Inconsistency Correction

A problem arises through the use of multiple exemplars which is illustrated in Figure 2.3. When two exemplars fail to share the same colour space, inconsistency arises in the synthesised output. Notice that a colourful coarse exemplar points to finer exemplar with no colour. Doing nothing will result in a final output that is no longer consistent with the fine level exemplar as shown on the left.

Given the coarse-to-fine synthesis direction the visual appearance of a coarser synthesis level constrains the visual appearance of the next finer level, and by induction, all finer synthesis levels. Because single exemplars are self consistent,

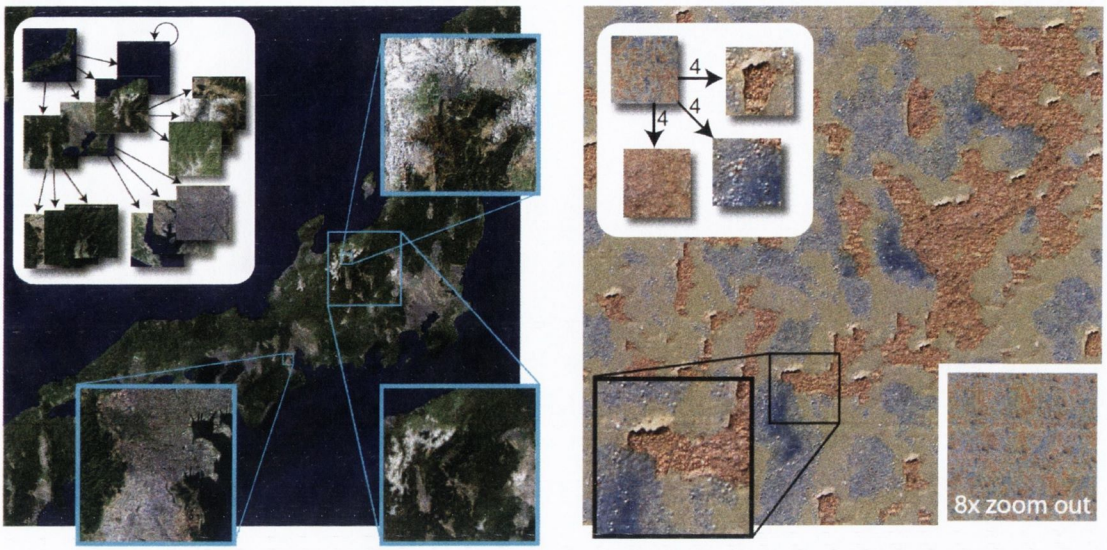


Figure 2.4: Results. *Super-resolution (left) uses fourteen exemplars of the Japan island at multiple scales. By disabling jitter at the coarsest levels, large features such as forests, mountains and cities are spatially preserved from the root image such that the fine scale details can then be filled in from the finer resolution textures. Multiscale Texture Synthesis is also useful as a compact representation (right) where only four 128 x 128 images are able to fully represent a 8k x 8k final synthesised result.*

a transfer function is introduced that captures the overall change in appearance between source and destination neighbourhoods when jumping between exemplars. By accumulating this change in colour and applying it to the final result, colours from finer exemplars can be preserved, as shown on the right.

2.2.6.5 Results

Multiscale Texture Synthesis boasts gigapixel results (Figure 2.4) where a handful of well chosen exemplar images can synthesise vastly large textures across a multitude of scale levels.

An interesting property was noticed in several real world and human created

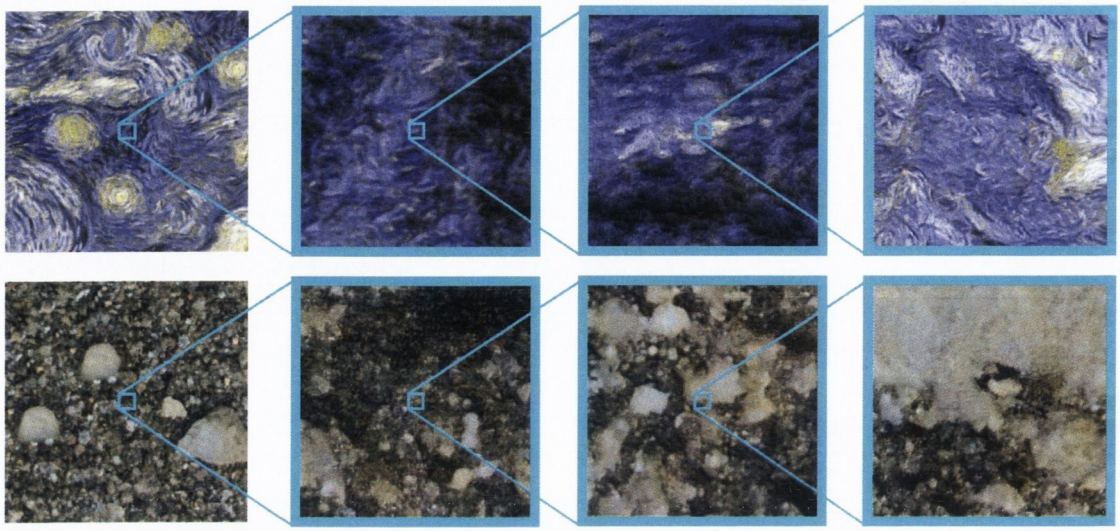


Figure 2.5: Recursive Images. *Starry Night (top) and rocks (bottom) are two examples of images that are perceptually similar across wide ranges of scale.*

images, a self similarity across wide range of scale, or a recursive scaling property, where an image looks the same at multiple resolutions, as was shown mathematically with fractals. Figure 2.5 shows two such example images where the finest scale level corresponds to features at coarser levels and allows the use of a cyclical edge in the exemplar graph, where a node can point to itself. Thus, a single image exemplar can be up-sampled to a theoretical infinite resolution.

2.2.7 Symmetry

Symmetry can be described as patterned self similarity. Given a set of objects, or subparts of a single object, their geometrical relationships to each other can take on the form of translational, rotational, scale, reflection, helical, and glide symmetry, or any combination of those six. Symmetry is an important aspect of structure and the proper analysis and preservation of symmetries is crucial when synthesising globally varying textures.

A great deal of work has been done to detect symmetry in 3D models [MGP06].

Given a 3D model, first transform local neighbourhoods of the model into a new space by using a translation, rotation, reflection, and scale invariant descriptor. Once a large set of local neighbourhoods are projected into this space, clusters of points will appear for similar shapes in the model, despite them possibly varying drastically across orientation, scale and position.

Going beyond simply detecting symmetries in a 3D model [PMW*08], often 3D scanning and vision techniques result in point cloud models with sparse or completely missing regions that the user would like to fill. By detecting global patterns in the present data, the algorithm can automatically determine what should exist in the missing gaps. If a regular pattern occurs then it is also possible to extend the model to fill a larger space.

While both of these techniques are powerful, they assume perfect symmetry and do not explore the idea of partial symmetry or statistical variations on symmetry. Early research in synthesising partial symmetry for texture synthesis [LT02], [LL03] has focused solely on regularity (translational symmetry). In the context of image textures, it has been observed that statistical variations on a common feature exist at regular or repeated patterns in an image. Thus research with regards to regular textures has mostly dealt with the detection and synthesis of this pattern. [LLH04] nicely defines a near regular texture as a statistical distortion of a regular, wallpaper-like congruent tiling, possibly with individual variations in tile shape, size, colour and lighting.

Recognising that in many man made and natural textures both a stochastic as well as regular pattern can occur, [HH09] identifies and models these separate components independently using methods that are optimal to each. Creating near regular textures through the combination of both independent models.

2.2.8 Global Structure

Several pixel-based approaches have explored synthesising globally varying textures by preserving global structure and inhomogeneity. These works fall under several general strategies. One such approach treats texture synthesis as a stationary process but constrains and directs it globally based on user input [HJO*01; RB07; WHZ*08]. Alternatively, specialised domain knowledge has been utilized by some approaches to determine the global structure using customized procedures [LLH04; DZ06; MPK09]; by automatically encouraging formation of coarse features in coarsely varying textures [KFCO*07; HRRG08; RCOL09] it is possible to preserve coarse features at fine levels, which is also a form of global structure. These ideas, along with another line of works designed to combine multiple exemplars [WY04; MZD05; HRRG08] inspires our own focus on hybridisation.

2.2.9 Optimisation Based Synthesis

Texture optimisation [KEBK05] marks a return to the strict MRF model popular in earlier texture synthesis approaches, away from reliance on coherence. Optimisation in some ways replaces the need for coherence which is an ad-hoc method for patching over an inherent shortcoming in the previous texture synthesis framework where each pixel operates in a greedy approach such that it chooses the best neighbourhood match for itself without consideration of what this selection means for its neighbour. Instead, by minimising a global error metric we can in effect eliminate this greedy approach and choose a set of new pixels that produce a globally more consistent image without the need to explicitly give the algorithm coherent pixel information. This optimisation approach uses an EM type algorithm to iteratively reduce global neighbourhood error through multiple passes.

2.2.10 Inverse Texture Synthesis

Inverse Texture Synthesis extends the optimisation framework [KEBK05] and is inspired by [BFK03] and [KP02]. Inverse texture synthesis attempts to create a smaller texture from a larger one, while preserving as much of the global and local characteristics of the texture as possible. As a texture relates to information theory, it contains a set amount of information necessary to describe itself. Texture synthesis creates redundant self similar information. In other words, it creates more of the same information, while creating no new information. Inverse texture synthesis works on the assumption that most textures people use already contain this redundant information. Thus, it builds a new texture only consisting of the minimal amount of information necessary to describe a texture. Inverse Texture Synthesis is the first paper that explores compression for image textures.

2.2.11 Image Inpainting

Image Inpainting techniques, where the aim is to replace missing regions in an image, are also concerned with maintaining structure [MGVC00; APK04; DI03]. However, the image itself is not rebuilt or changed in any way, so that the missing holes are filled in based on the surrounding known area. The general approach is to iteratively fill in the gap by adding layers of pixels into the missing region, maintaining gradients and edges from the known into the unknown. Texture synthesis methods are also used to propagate the surrounding texture into the missing region. More recently Image Inpainting has adopted texture synthesis concepts [YW07; BSFG09], where the source image along with missing region are down-sampled into an image pyramid, and going from coarse-to-fine, the missing hole is filled in by matching neighbourhoods of known border pixels to neighbourhoods of the image that are completely known. Both papers rely on optimisation based texture synthesis at each scale level.

2.3 Geometry Synthesis

Geometry synthesis is a widely studied topic in computer graphics. Approaches have been explored which utilize sets of example models in order to assist artists in the creation of new geometry. These approaches range from simple techniques which allow users to manually create models by selecting and mixing elements from example models, to more complicated approaches which automatically or semi-automatically analyze and build new geometry.

2.3.1 Modeling by Example

These techniques promote the idea that an artist can be assisted during the modeling process with the help of a database of existing models. Several creativity support approaches have been explored which assist a user in recombining parts of existing models together in new ways [FKS*04; CK10; CKGK]. The paper by Funkhouser et. al. introduces this concept where the user manually selects components of a model from a set of example choices of a specific category (e.g. chairs, horses, cars, etc). The algorithm then automatically deforms those chosen components in order to fit them together as a new hybrid model. The latter two papers by Chaudhuri et. al. extend the original idea by Funkhouser making it a semi-automatic approach where the input models are analyzed and the approach makes meaningful and creative suggestions to the user as to which components or sets of components should be combined, and how they should be combined.

Recently the focus of modeling with components has shifted toward full automation [KCKK12; FRS*12] where probabilistic modeling has been utilized to remove the user from the process of component based model synthesis. Both approaches make use of Bayesian networks to learn how objects should be arranged based on an input training set of models. A Probabilistic Model for Component-Based Shape Synthesis specializes in hybridizing sets of similar objects of a specific

type (e.g. airplane, boat, quadruped) by recombining the common elements among that set of objects. Example-based Synthesis of 3D object Arrangements focuses on synthesizing the arrangements of a collection of individual objects which are often found together. Control over the amount of diversity in a gallery of synthesised models can be influenced by the user through a genetic algorithm approach [XZCOC] where the fitness of individuals in the population is explicitly set by the user. Given a population of models, the algorithm randomly crosses pairs from the population introducing mutations as well. The user then chooses which models they like from this child population. The process is then repeated using the chosen models. Over several generations the user can direct the evolution of their population towards a desired result.

An equally challenging problem is automatically determining semantically meaningful controllers for updating a single model without deforming its shape in an undesirable way. iWires [GSMCO09] are based on the argument that man-made models can be structurally summarized using a few defining wires and their mutual relationships. By updating a model's shape in such a way that it maintains the properties of these wires, the defining characteristics of the entire model are preserved. The concept of iWires has been expanded to include component-wise controllers [ZFCO*11]. These are high level characteristics such as symmetry and parallelism which are organized into a hierarchy of relationship dependencies. During editing this high level structural template is preserved, maintaining the characteristics of the model as a whole.

The discovery of structural regularity in input geometry [MGP06; PMW*08] can aid in the automatic reconstruction of models with damaged or missing parts. Many natural and man made objects follow some kind of symmetry pattern. By detecting these global symmetries in a partially complete input model, the missing regions can be identified and filled in using information from the regions that are complete. Complete but potentially incorrect regions of a model can be tested against an averaging of all other regions that comprise the same symmetry pattern.

This leads to the detection and replacement of potentially damaged or incorrect model information. Laser scanners and other devices designed to scan large real world objects have the common problem that they produce noisy and incomplete models, where the target object is occluded by objects in the environment such as trees, signs or power lines. Measurements can also be distorted slightly by undesirable atmospheric conditions. Automatic reconstruction using structural regularity detection offers the best solution to this common problem.

2.3.2 Geometry Texture Synthesis

While automatic and semi-automatic geometry synthesis is a widely studied problem [BWW10], very few papers have explored a texture synthesis based solution. A key reason behind this could be that the field of texture synthesis has mainly focused on stochastic (noisy) examples while almost all interesting geometry is highly structured. Several papers [BP04; AL05; kLmHGM05] have explored creating stochastic geometry textures on the surface of an already defined mesh, or plane, much the way two dimensional textures are synthesised over the surface of a mesh or image, but extended to voxels in the third dimension or treated as image displacement texture. The approach presented in [AL05] processes a geometric model into voxel grids storing displacement vectors to a surface. Building a Gaussian pyramid out of this voxel data Lagae and Dumont are able to re-synthesise their exemplar into a stochastic texture in a multi-resolution procedure.

Mesh Quilting [ZHW*06] divides a mesh exhibiting repetitive or stochastic features into a subset of patches and then synthesises a new collection of these patches in a similar approach as its image space precursor [EF01]. Geometry texture synthesis has also been explored for the niche area of human hair [WYZG09], where a hybrid 2D texture synthesis approach is used for synthesising clusters of hair spawning from the scalp, and then hair strands are synthesised geometrically as thin strands.

2.3.3 Element Based Texture Synthesis

While the bulk of texture synthesis has focused on synthesising data on a regular grid, such as images, sounds or volumes, work has been done to extend the concepts of texture synthesis to the positional arrangement of discrete elements or objects. Texture Particles [mDMLG02] laid groundwork by viewing texture as a spatial arrangement of atomic texture objects, and placing new instances of those objects with relation to each other following the example given. This concept was further explored by Stroke Pattern Analysis and Synthesis [BBT*06], which took into account concepts from human vision research concerning perceptual organisation. Ijiri [IMIM08] introduces both the concept of controllability as well as multi-resolution. While the previous work starts with a small seed pattern and grows outward in the same manner as [EL99], an Example based Procedural System for Element Arrangement starts with multiple seed points which designate at a coarse resolution the space to be filled, and then fine pattern is grown between these coarse seed points to fill in the space. User controllability is introduced through the placement of these seed points and the growing procedure. Seed point placement and growing orientation can be warped due to an underlying vector field, or restricted by a bounding area.

2.3.4 Mesh Morphing

The techniques known as morphing or 'metamorphosis' were originally developed for the image domain. Image morphing papers created the illusion of a smooth transition between two images by approximately interpolating colours and structured shapes between two images. A detailed survey of these techniques is available in 'Warping and Morphing of Graphical Objects' [gDCV]. Motivated by this line of research but targeted towards 3D mesh data, the topic of mesh morphing has grown to become its own field of study. While the underlying principle is the same between the two areas of research, to smoothly interpolate or 'morph' be-

tween two or more inputs, mesh morphing introduces its own unique challenges for which many different solutions have been discovered. Multiple surveys are available on this topic [gDCV; LV98].

Volumetric Metamorphosis

Conceptually, the simplest approach to extend an image technique to meshes is to take samples from the mesh object and build a volumetric representation. Most image techniques can be easily extended to work in three dimensions on the intermediate volumetric representation, then a final mesh can be reproduced from that volumetric representation using the marching cubes algorithm [LC87]. This effectively simplifies the mesh morphing problem to volume morphing, where 3D voxel grids are morphed instead of their geometric counterparts.

One approach to volume morphing utilizes signal processing to separate a volume into high and low frequency components. By separating these components, the low frequency features can be smoothly interpolated over time as the high frequency details are slowly added back in. This multi-resolution approach morphs between the coarse scale shapes of the two objects, reducing distortion caused by the interpolation of high frequency components. This idea was introduced using the Fourier transform [Hug92] and later extended to Wavelets for performance reasons [HWK94].

Another approach [COLS98] is to transform volumes into distance fields - DFI (Distance Field Interpolation). This is a two step process. In the first step, the 'warp' step, each object is decomposed into contour level volume sets. The volume space is then deformed so that two sets of user defined feature points are brought into coincidence with each other. In the second step, the 'interpolation' step the volumes are actually converted into distance fields and a linear interpolation is performed between the two distance fields.

These volumetric approaches are attractive because they build on the previous

work done in the image domain and they sidestep the problem where two meshes have different geometric topologies and a direct one-to-one mapping between mesh regions is not possible. Despite these advantages, the volumetric approach is not very popular due to the high memory and processing requirements necessary to work in the volumetric domain. In general, when working with an intermediate volume representation instead of a mesh directly, it is expected that sacrifices are made in terms of fine scale details and running time performance.

Mesh-Based Metamorphosis

Due to the inefficiency of working with the intermediate volume representation, approaches have been developed for interpolating two mesh models directly [LDSS99]. This problem can be broken down into two sub-problems, setting up correspondences between points in two or more meshes, and interpolating these corresponding points to create intermediate representations. In general, mesh-based metamorphosis is unrelated to texture synthesis. However, this family of approaches are directly relevant to this thesis because they face the same challenges faced for geometry hybridisation and they give insight into the problem of hybridising two or more meshes with different geometric topologies.

2.4 Audio Texture Synthesis

Similar to images, audio is a continuous function consisting of similar statistical properties and encoding information. Work has been done to explore adapting texture synthesis concepts to the audio domain and in fact the evolution of the audio texture field closely follows that of image textures. One of the earliest works [SaP95] modeled audio texture as a two-level phenomenon where an audio sample could be decomposed into atomic sound clips, and at a higher level the audio texture is described by the distribution and arrangement of these atoms.

By clustering the transition of feature vectors they build a probability distribution function that they can then use to re-synthesise using a binary tree structured quadrature mirror filter bank (QMF).

[HP01] also discretised an audio signal into atomic components that they refer to as *natural grains*. They utilise a wavelet transform over windowed frames in order to find natural grains and then perform speech recognition to determine transition points. A table of similarity between segments is computed and synthesis is then performed with a Markov Random Chain where the next natural grain in the sequence is determined by finding a nearest match between the last several natural grains synthesised and a similar pattern in the input.

Synthesising Sound Textures through Wavelet Tree Learning [DDEY*02], models sound grains as short, distinct bursts of energy. By decomposing the input sound into wavelets, they analyse the joint statistics of the coefficients across both time and scale. Then synthesis builds a statistically similar wavelet tree to produce a new audio texture. In this approach a Markov Random Chain is used to ensure statistical similarity to the input.

Spectral Sound Gap Filling [AFY04] takes a new direction in audio texture synthesis and targets hole filling akin to image Inpainting rather than synthesising larger samples of their audio texture. They transform their audio signal into a spectrogram and perform a similar approach as image quilting from both the left and right ends of a hole simultaneously, iteratively filling it in. They repeat this process iteratively going from low to high frequency bands in their spectrogram, using the already synthesised lower frequency information to choose good higher frequency matches. They base this synthesis order on the fact that more energy exists at the lower frequencies and they are perceptually dominant. Similar to image quilting, blocks are overlapped with already synthesised information and seams are found that minimise transition error.

Audio textures [LWZ04] take a patch or clip based approach to audio synthesis

similar to image quilting. First, the signal is processed based on the Mel-Frequency Cepstral Coefficients, then the audio clip is analysed in terms of its structure and the audio clip is segmented into several basic building patterns that can be recombined to produce new signals, minimising error between transitions.

The dual tree complex wavelet transform was used by O'Regan and Kokaram [OK07] in order to simplify a complex waveform into descriptive coefficients. By following the synthesis procedure made popular by Efros [EL99] on coarse scale coefficients O'Regan and Kokaram effectively synthesise an audio texture at a very coarse level of the multi-resolution pyramid, and then utilise the inverse complex wavelet transform to reconstruct the higher frequency information so that it's similar to the original input.

2.5 Texture Analysis

While work in texture spans many subjects and fields, two of the main areas of relevance for this thesis are computer graphics, and computer vision. While the computer graphics field has largely focused on the synthesis of textures, primarily in the form of images, the computer vision community has looked at texture from an analysis standpoint. They focus on several interesting applications of texture such as identifying textures in an image by their statistical appearance, and then using identified texture regions to extract other visual information from a scene such as lighting, shape, distance from viewer. Beyond using texture as a tool for image analysis, fundamental models for what texture is have been proposed which are more general, robust, and principled than those proposed in the synthesis literature.

2.5.1 Multiscale

Much like synthesis, the multiscale aspect of texture has been studied for analysis as well. Specifically, it has been observed that the spatial patterns that characterise a texture, exist only at a specific scale level, and that the globally varying arrangement of different textures are themselves a texture [New07]. Hierarchical Texture Motifs identifies multiscale textures in a bottom up fashion where discrete texture regions are identified using the texture motifs identification algorithm [NM04], then a new label map is created where pixels at the location of a texture are replaced with the label ID number for that texture. In this way, a gray-scale image is created that can be viewed as a coarser scale texture of the textures, and the process is recursively repeated.

While not strictly a texture analysis approach, A Stochastic Grammar of Images [cZM06] explores the automatic segmentation of a scene into simpler components embodied in a simple And-Or graph representation. This approach identifies stochastic texture regions and makes the observation that globally varying texture regions can collectively form structured objects or texture elements at a higher graph level.

2.5.2 Discrete Texton Identification and Extraction

Recall from Chapter 1 that any texture at a given scale can be thought of as the arrangement of textons, where a texton is a texture element, or an atomic unit of texture which by itself does not contain internal feature redundancy. The field of texture synthesis has focused almost exclusively on reproducing the arrangement of these textons, where textons themselves are either cloned directly from the input, or randomly deformed to add variation, naively ignoring the common structure shared between them.

The vision field has extensively researched the problem of dividing an image

into smaller low level primitives. Although there are variations in the literature of what the low level building blocks should be (as well as named), the differences between texels, textons, primitives, patches, and fragments are small and subtle [cZM06]. The goal of automatically decomposing a texture into an assortment of low level texture pattern features is an interesting problem and has even led to a more complete definition and mathematical framework for texture synthesis [cZLW00].

An early attempt at defining the basic components of an image was the "Texton" concept, referring to the fundamental micro-structure present in natural images that are analogous to atoms but for visual perception [Jul81]. The key idea being that the pre-attentive human vision system is sensitive with respect to some fundamental image features, while being blind to others. In Julesz work, textons are treated as a product of human perception and studied through a psychological perspective which measures human response time for identifying textons amongst a distracting background. While the first real work in identifying image textons was tied to human perception and offered no rigorous mathematical definition for textons and how they relate to texture, this concept inspired further research that is central to this thesis.

Building on the texton concept, Song-Chun Zhu wrote several important follow up papers [ZM98; ZGWX05] that further explored the concept of textons as a component of texture, as well as proposed the first rigorous mathematical definition of textons as well as ideas for extending the texton concept to other domains such as particle motion 'Motons' [WZ02] and 3D relief structures 'Lightons'.

While texton identification has dominantly been studied as a texture analysis tool, two techniques have been proposed that take analysis a step farther and synthesise texture based on textons. Nonparametric Bayesian Texture Learning and Synthesis [ZCFT09] represents a texture image as a 2D Hidden Markov Model where hidden states correspond to texton labels, and a transition matrix is used

to encode their spatial layout. This work first detects textons as collections of image patches and attempts to produce a regular layout pattern for these textons. Larger textures can be synthesised by propagating the learned regular layout. While this approach only works for regular or near regular textures, and quality is comparable to early patch based synthesis approaches, this work takes a bold step towards utilising textons for texture synthesis.

Texture Splicing [LWX*09] performs a new type of mixed texture synthesis from multiple exemplars where the textons from one image are chosen as the textons for the synthesised output, and the spatial pattern for the texton arrangement is chosen from another exemplar. This work extracts a texton dictionary for each exemplar as well as a mapping of the texton patterns. A new larger pattern map can be synthesised with textons from a different exemplar copied over the synthesised location markers.

2.6 Conclusion

This was a broad overview of the area of texture synthesis and work relevant to this thesis. The review of image based texture synthesis 2.2 gives a general idea of how this type of algorithm works and the review of Multiscale Texture Synthesis 2.2.6 provides a solid texture synthesis framework which is built upon in this thesis. Non-image texture synthesis approaches are reviewed 2.3, 2.4 as well as the mathematical concepts behind texture 2.5.

Chapter 3

Synthesising Structured Image Hybrids

Figure 3.1: Jeans. *New jean hybrids are synthesised from only four examples.*

3.1 Introduction

In this chapter we introduce the concept of hybridisation, a core contribution of this thesis. We show how the problem of hybridisation can be framed as a type of texture synthesis and we offer insight into the underlying theory that drives both[1].

Hybridisation is an important problem. We rarely experience the same thing twice. No two beetles have the exact same form and markings, no two people have

[1]Most of this chapter has been published in ACM SIGGRAPH 2010 [RHDG10].

the same fingerprints and no two birds have the exact same song. Synthesising the infinite diversity in the world requires a new perspective on the old problem of texture synthesis.

Contributions

We introduce a multiscale descriptor to measure neighbourhood distance across both image space and image scale and show that it is necessary for detecting highly structured features in images. Second, we propose a structure-preserving jitter step that operates in the appearance rather than spatial domain. The first extension helps maintain and repair synthesised image consistency; the second introduces variation without breaking structure. The combined extensions make for an algorithm that handles images far outside the domain of traditional texture synthesis.

3.2 Framework

Example based texture synthesis algorithms generate novel texture images from example data. A popular hierarchical pixel-based approach uses spatial jitter to introduce diversity, at the risk of breaking coarse structure beyond repair. We propose a multiscale descriptor that enables appearance-space jitter, which retains structure. This idea enables re-purposing of existing texture synthesis implementations for a qualitatively different problem statement and class of inputs: generating hybrids of structured images.

We build on the method of Lefebvre and Hoppe [LH05], and follow the notations established therein. Following the hierarchical synthesis paradigm, the output is generated in a series of images, $S_0, S_1, \ldots, S_L$, of increasing resolution. The input is given as a set of I images $E^0, E^1, \ldots, E^{I-1}$ where each exemplar image is processed into a multiresolution stack of L images where a given exemplar

at a given stack level is indexed by E_l^i. The image pixels, $S[p]$, are not represented as RGB values but rather as coordinate pointers into the exemplar, such that the rendered colour at pixel p is $E[S[p]]$ where $S[p]$ consists of two values, u which references pixel coordinates in an image, and i that references which exemplar should be sampled from. $*S_l[p]$ is used to dereference the actual colour value stored in $E_l^i[u]$.

Synthesis operates from coarse to fine resolution. Three steps are executed at each scale level: **upsample** the previously generated level, **jitter** the coordinate image to add variation, and **correct** local appearance through a series of relaxations. The correction step requires a search for similar texture neighbourhoods—an expensive high-dimensional nearest-neighbour search—which is accelerated in two ways: An offline **precomputation**, where k nearest-matching patches are selected as *candidates* for an exemplar patch, reduces the requisite number of comparisons during the online synthesis. Truncated principal component analysis (PCA) reduces the high-dimensional pixel neighbourhoods to six dimensions. Following Han Multiscale Texture Synthesis [HRRG08], this framework is extended to multiple exemplars by augmenting the coordinate pointers with exemplar indices.

3.3 Analysis

Following the design of the adopted framework, first a Gaussian stack E [LH05] is built for each of the input images. This grooming of the exemplars into a multiresolution representation is necessary for coarse-to-fine multiscale synthesis. In the case of hybridisation the need for a multiresolution representation of the input goes beyond accelerating synthesis and is crucial to the goal of preserving structure. While texture features have traditionally been taken as a coherent 2D patch of image pixels in the space of the image, this chapter introduces a new multiscale descriptor that utilizes both space and scale to provide a more powerful analysis tool for detecting structural similarity in images.

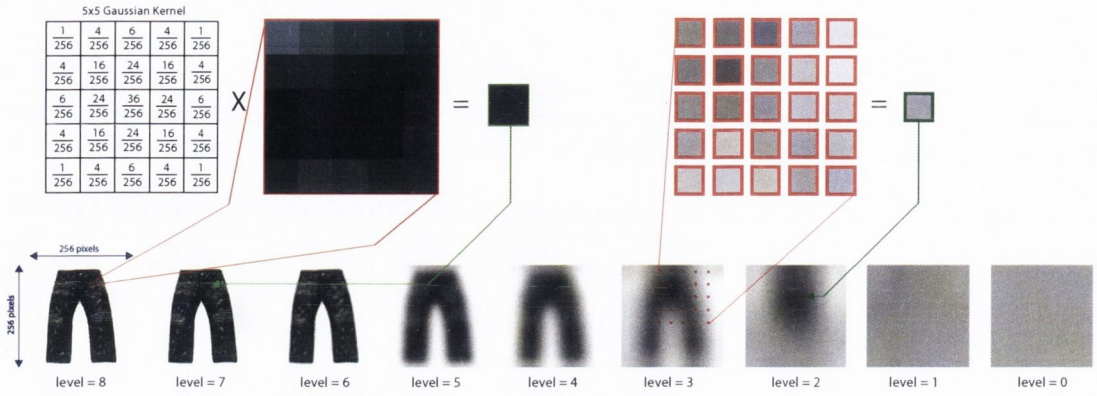


Figure 3.2: Gaussian Stack Creation. *Given a square input image (shown as level = 8), build a Gaussian stack by iteratively generating the next coarser level. This is done by convolving every* 5×5 *neighbourhood by the Gaussian Kernel as illustrated in the top left. The resulting averaged pixel value is placed at the location of the centre of the neighbourhood in the next level. The level dependent offset* h_l *is used to offset neighbourhood sampling (level 3) where* $h_3 = 32$ *pixels between samples.*

3.3.1 Gaussian Stack

The first stage of analysis is to build Gaussian stacks for each of the input hybrid images. The stack representation has been chosen over the traditional Gaussian image pyramid. Even using a handful of randomly sampled Gaussian pyramids [ZRDM01] is problematic as the pyramid aligns features at the coarser scales to the quantized 'grid' of the pyramid. Rather, the stack representation sidesteps this problem by downsampling without subsampling, preserving the full image resolution at all levels of the stack. In effect, given an $m \times m$ pixel image, it creates the full family of m^2 Gaussian pyramids, one for each translation of the input hybrid image at its finest level. The stack is a tightly packed realization of the family of m^2 Gaussian pyramids stored in a stack of $\log_2 m$ images.

The stack is built by iteratively creating the next coarser level from the current

level as shown in figure 3.2. At each level a 5×5 neighbourhood is gathered around every pixel in the image. This neighbourhood is then convolved with the Gaussian kernel, resulting in a single new averaged pixel colour that is placed at the pixel location in the next scale level. The Gaussian stack incurs the added complication that neighbourhoods corresponding to the Gaussian pyramids are no longer tightly packed. In order to gather neighbourhoods that reflect pyramid sampling, a level dependent offset $h_l = 2^{L-l}$ is introduced where h_l pixels are skipped between samples. This neighbourhood gathering process is illustrated for levels eight and three. Notice that at the finest level $h_8 = 1$ and tightly packed neighbourhoods are sampled. At level three $h_3 = 32$ and as such neighbourhood samples are 32 pixels apart from each other in the image. At the coarsest level the Gaussian stack produces a single solid colour.

Each neighbourhood gathered for downsampling during stack creation will be used repeatedly throughout the algorithm. As such each $5 \times 5 \times 3 = 75$ dimensional neighbourhood vector $N_{E_l}(p)$ is projected down to 6 dimensions using PCA. This is standard practice [LH05] for accelerating the neighbourhood matching process throughout the algorithm.

3.3.2 Multiscale descriptors

During the precomputation of nearest neighbours, a multi-resolution descriptor is utilised which embodies a compromise between expressiveness and simplicity. Drawing inspiration from more sophisticated models, we ultimately found that our simple PCA-based descriptor is (surprisingly) adequate for synthesis, while being simple to implement and naturally compatible with our implementation of hierarchical synthesis.

Both the new jitter step and the correction step rely on precomputed sets of best-match candidates for each pixel location in the input. The notion of "best-match" is not fixed, however, and depends on the choice of local descriptor; almost

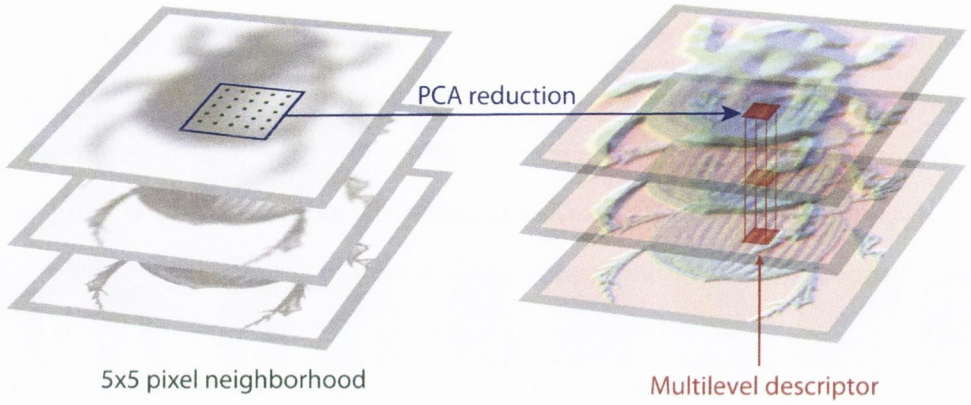


Figure 3.3: Schematic of descriptors. *Principal component analysis (PCA) is used at each stack level to reduce the 75D ($5 \times 5 \times 3$) pixel neighbourhoods to a 6D representation (shown here as RGB, right). PCA vectors of adjacent levels are then concatenated to form the multilevel descriptor. For illustration, only three intermediate levels of the Gaussian stack are shown (left).*

all pixel-copying texture synthesis methods use the single-scale local pixel neighbourhood as the descriptor (Lefebvre and Hoppe use 7×7 neighbourhoods).

We extend the notion of a single-scale pixel neighbourhood to a multiscale counterpart (see Fig. 3.3). If $\tilde{N}_{E_l}(p)$ is the neighbourhood vector at pixel p in level l, our multilevel descriptor, $\hat{N}_{E_l}(p)$, is defined as the concatenation of neighbourhood vectors at p from L successive finer levels (we use $L = 8$):

$$\hat{N}_{E_l}(p) = \left[\tilde{N}_{E_l}(p)\ \tilde{N}_{E_{l+1}}(p)\ \ldots\ \tilde{N}_{E_{l+L}}(p)\right]. \tag{3.1}$$

Note that we use the 6D PCA-projected neighbourhood vectors (hence the use of tilde notation) rather than the full 75D representations. Since our pre-computed Gaussian stack already includes a PCA projection, these are available to us essentially "for free." At the finest levels where $l+L$ exceeds the Gaussian stack height, we truncate the multiscale neighbourhood to include levels only up to the finest.

This multiscale descriptor improves on a straightforward starting point, the use of a larger neighbourhood at a finer scale. In particular, it improves performance, since it gathers fewer pixels, and involves shorter vectors in the PCA step. As an

added benefit, it avoids a potential pitfall of the large flat neighbourhood approach: it allows (e.g., equal) weighting of each scale, so that in particular fine features do not overpower information at coarse scales, as might occur in the most naive flat implementation.

We provide some intuition for the role of multiscale descriptors within the overall method. Synthesis proceeds in a coarse-to-fine direction, at each point building on the information contained in the finest synthesised scale; thus we measure features of the synthesised image using only a single-scale descriptor. On the other hand, analysis has all scales available at once. Here a multiscale descriptor enables the analysis algorithm to make better choices at coarse levels, in effect anticipating finer levels. Candidate sets precomputed using the multiscale descriptor include candidates that match well both at the current level and at finer levels (see Figure 3.5).

What is the ideal depth for a multiscale descriptor? An overly deep descriptor is costly and may introduce noise from its finest levels, while shallower descriptors retain the disadvantages of the single-scale setting. Heuristically, we have found that eight scale levels works well for all images we test, yet is efficient. It would be interesting to explore an adaptive method for choosing the descriptor depth, in particular one that focuses attention on non-stationary features, defined as follows. Points in an image at different scales can be classified on a gradient from completely stationary (stochastic) to completely non-stationary. When maintaining structure during synthesis the goal is to replace highly non-stationary features with similar non-stationary features. To this end, these similar features must be properly identified. However, since images do not maintain the same stationary properties across scales (see Fig. 3.4) it is only necessary to use enough scale levels to capture all non-stationary information. If the descriptor looks deeper than necessary then it includes stochastic information into the neighbourhood which acts as noise with regards to describing structure. We have found that in practice this is not a severe problem and non-stationary features generally have more extreme

Figure 3.4: Stationary variation across scales. *A human face is highly non-stationary at a scale that captures facial features. However, at very fine scales skin features such as pores become dominant and the image becomes highly stationary.*

differences than stationary features and thus overpower them in terms of feature distance. To illustrate the use of different scale levels we include examples of multiscale neighbourhoods using one, two, four and eight scales as shown in Figure 3.6.

3.3.3 Nearest Neighbour Search

The PatchMatch [BSFG09] algorithm is used for nearest-neighbour search during precomputation, slightly altered so that matches found in each candidate set are separated by at least 5% of the image size [ZG02]. To encourage a diversity of choices at synthesis time, we construct candidate sets with equal numbers of candidates *per exemplar* (choosing one or two candidates per exemplar usually suffices, in our experience).

3.4 Synthesis

The algorithm starts with a single pixel at the coarsest level of the synthesis pyramid, representing the entire synthesis image if averaged to a single value. Following the three step synthesis procedure established in [LH05] (upscale, jitter, correc-

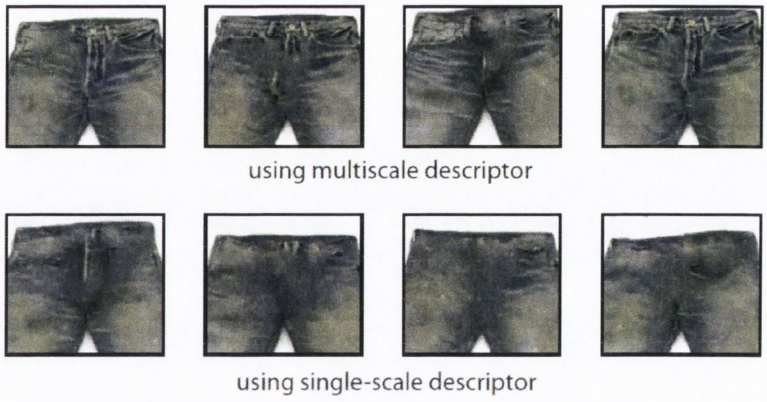


Figure 3.5: Multiscale vs single-scale descriptor. *Synthesis results using our multiscale descriptor* (top) *compared to the same example run with a single-scale descriptor* (bottom). *The addition of information at finer scales guides candidate selection to more structurally meaningful location matches, better resolving sharp features such as a zipper.*

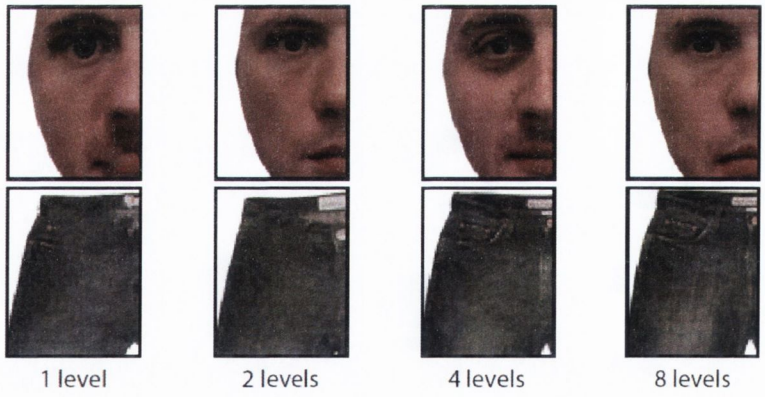


Figure 3.6: Progressively deeper multiscale descriptors. *Synthesis results using the multiscale descriptor with depths of one, two, four, and eight for human faces (top) and jeans (bottom). Jean quality improves up to the eighth level, and face quality improves up to the fourth level.*

tion), subsequent pyramid levels are iteratively created by doubling the resolution of the previous level in the upscale step, adding randomisation to the image in the jitter step and finally fixing any newly introduced jitter artifacts in the correction step.

3.4.1 Upscale

For each pixel in the image S_{l-1} a coherent 2×2 patch is formed in S_l by upsampling its coordinates. Intuitively, the four newly upsampled pixels will represent the same information as their single parent, but at a finer resolution sampled from the next finer level of the Gaussian stack. The operation is defined as the scaling of the parent's coordinate plus a child-dependent offset:

$$S_l[2u + \Delta] := (2S_{l-1}[u] + h_l\Delta) \bmod m, \tag{3.2}$$

$$\Delta \in \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\} \tag{3.3}$$

3.4.2 Jitter

Structure-preserving jitter

Jitter introduces variation during synthesis. The standard, spatial approach to jitter adds a small randomised offset to each coordinate (Fig. 3.7a). Jitter is applied independently to each texel, thus the resulting image typically breaks or distorts features, which the correction step then attempts to repair. When jitter is applied too strongly (Fig. 3.7b), correction fails to restore structure. *Structure-preserving jitter* judiciously introduces variation without breaking structure (Fig. 3.7c,d). It serves as a drop-in replacement for spatial jitter in an existing implementation of Lefebvre and Hoppe's framework.

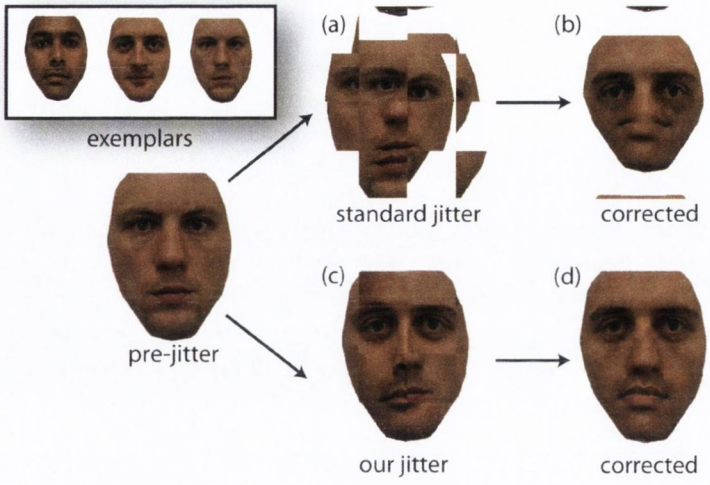


Figure 3.7: Jitter. *The traditional jitter method* (a) *causes breaks in image structure that may not be repairable by correction* (b). *Structure-preserving jitter* (c) *adds variation while maintaining structural coherence* (d). *Note that, although jitter occurs here at a coarse level, the prejitter and jitter steps are shown upsampled to full resolution for clarity.*

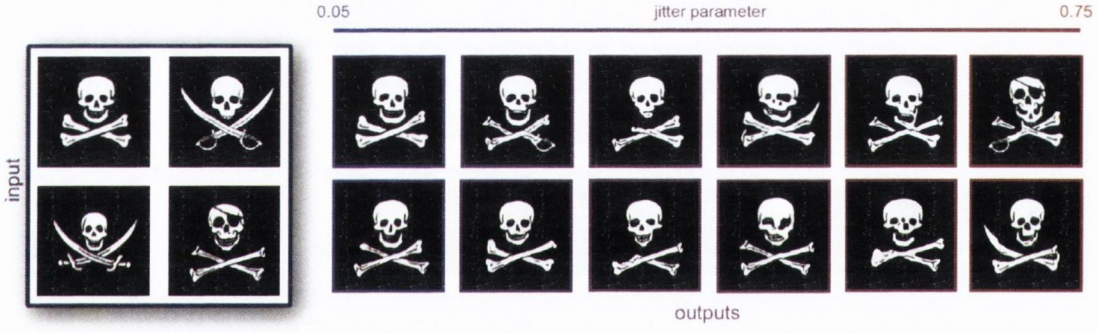


Figure 3.8: Variation control. *Qualitatively, higher values of the jitter parameter J (increasing towards the right) permit more variation in synthesised hybrids.*

To understand structure preserving jitter, note that the correction step points each synthesised pixel to the closest-matching exemplar neighbourhood's centre. Similarly, our jitter randomly chooses *some* exemplar neighbourhood, among those "sufficiently similar," an attribute governed by the jitter parameter, J. This ap-

proach is related in spirit to earlier works in nonparametric texture synthesis that use neighbourhood similarity to build candidate sets from among which random winners are chosen [EL99].

At each synthesis pixel, p, the 5×5 colour neighbourhoods $N_S(p)$ are collected and projected into a truncated 6D PCA basis to obtain the reduced neighbourhood $\tilde{N}_S(p)$. Precomputed candidate locations $\{c_i\}$ are collected from each of the 3×3 neighbouring texels. Employing k-coherence search [TZL*02], we evaluate the error $d_i = \|\tilde{N}_s(p) - \tilde{N}_E(c_i)\|^2$ of each candidate with respect to the local neighbourhood.

Letting $d_{\min}$ and $d_{\max}$ be the minimal and maximal errors over all candidates, respectively, candidates whose normalised error $(d_i - d_{\min})/(d_{\max} - d_{\min})$ exceeds the given jitter threshold $J \in [0, 1]$ are disqualified. We update $S[p]$ selecting uniformly at random among the qualified candidates. Intuitively, higher values of J cull fewer candidates, thus encouraging variation (Fig. 3.8). This process is illustrated in figure 3.9 where a single neighbourhood from the source image is projected down to a point in 3D space for visualization (black). Given some collection of other projected neighbourhoods (green dots), the jitter parameter becomes the radius of a sphere around the point of interest and any other neighbourhoods within that radius (red dots) are suitable candidates for jitter.

Observe that structure preserving jitter is heterogeneous and data-sensitive in behaviour: it adds variation when safe to do so. For regions with many plausible matches, jitter selects among multiple candidates; at unique features having no close matches, jitter selects the best matching candidate—acting essentially as a correction step, discussed in the next section.

3.4.3 Correction

From a high level perspective, upscaling adds new details to the synthesising image, jitter modifies those details to form new images, and correction then "fixes" the image so that it maintains statistical similarity to the exemplar input. For each

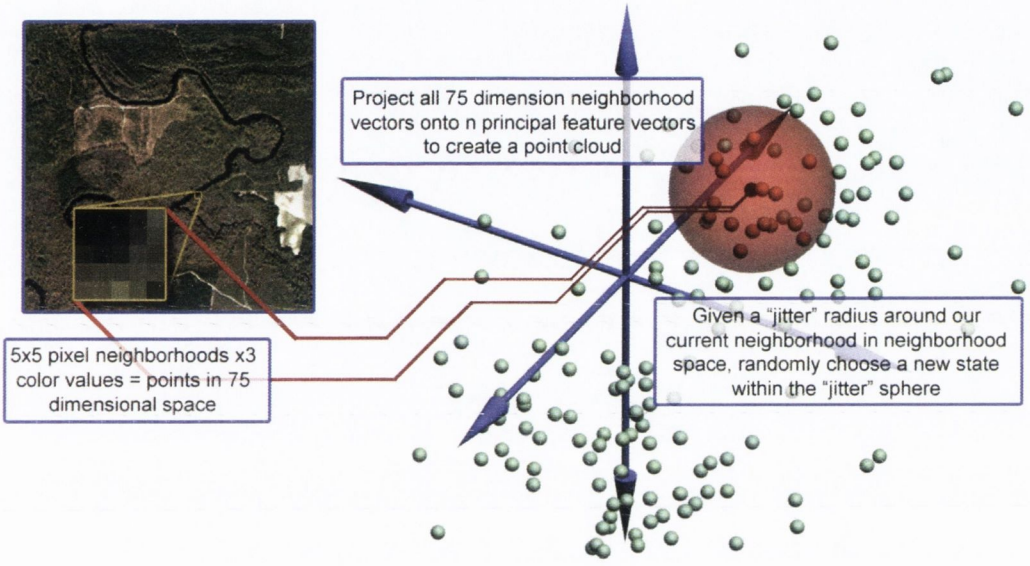


Figure 3.9: Structure Preserving Randomization. *Given some image pixel to be jittered, collect the* 5×5 *neighbourhood of pixels around that location. Project the point in* 75 *dimensional space down to a lower dimensional space using PCA (in this case 3 dimensions for visualization). The other dots (right) represent the neighbourhoods around other pixels in the image which have been projected into the same feature space. The jitter parameter becomes a radius in this feature space and forms a sphere around the current neighbourhood's location. Any other neighbourhoods that fall within this sphere (coloured red) are considered similar enough to the current neighbourhood in terms of feature distance to be candidates for jitter. The pixel of interest is jittered to the centre pixel of any of the jitter candidates' neighbourhoods.*

synthesised pixel $S_l[p] = (u, i)$ the correction seeks out the point in any exemplar image $E_l^j[v]$ whose 5×5 exemplar neighbourhood best matches the local 5×5 neighbourhood. Formally,

$E_l^j[v]$ is the minimizer of the error functional

$$\sum_{\delta \in \{-2 \ldots +2\}^2} \| * S_l[p + \delta] - E_l^j[v + \delta h_l] \|^2$$

As a reminder, $*S_l[p]$ dereferences the texel pointer $S_l[p]$. This process follows directly from [HRRG08] with the exception that multiple exemplar images all exist at the same stack level as the exemplars share the same scale. The correction pass runs in parallel through the use of 8 subpasses [LH05].

The correction scheme is based on coherence [Ash01], which states that nearest neighbour selection is not always the best method for finding perceptual similarity. Rather coherence, or the relationship between neighbouring pixels, plays a large role in structure and perceptual similarity. We seek to find the minimum error between our current synthesis neighbourhood, and the minimum of the set of precomputed candidates. Mirroring the jitter step 3.4.2, precomputed candidate locations $\{c_i\}$ are collected from each of the 3×3 neighbouring texels. Employing k-coherence search [TZL*02], the error $d_i = \|\tilde{N}_s(p) - \tilde{N}_E(c_i)\|^2$ is evaluated for each candidate with respect to the local neighbourhood.

During correction we have opted to test structure at the current scale alone rather than reconstructing and filtering finer scales. This is due to the efficiency benefits of testing at the current scale and the fact that the precomputed candidates are already known to be structurally similar. Thus a better candidate is chosen from a set of already good choices.

3.4.4 Exemplar standardisation

In many applications such as object detection and recognition [MP97], the scale, orientation or illumination variations across images are removed by a pre-normalisation

step, because they do not characterise the intrinsic properties of the object but rather the conditions of the acquisition. Similarly, our exemplars are manually normalised for scale and orientation variations not modeled in our estimated probabilistic manifold. In addition, exemplars share a similar colour palette in these experiments (e.g., faces, jeans, chess pieces, etc.): the intensity has been automatically normalised in all exemplars by computing the PCA on their (RGB) pixels and dividing by the dominant eigenvalue. More advanced automatic standardisation techniques could be adopted following the object recognition literature.

3.4.5 Symmetric synthesis

For some examples (Fig. 3.16, butterflies, chess, faces) an additional scheme to encourage symmetry has been implemented. During the correction step (jitter step is unmodified), the usual neighbourhood distance measure is augmented with a term that encourages symmetric formation of features:

$$d_i = \|\hat{N}_s(p) - \hat{N}_E(c_i)\|^2 + \|\hat{N}_s(W_S - p) - \hat{N}_E(W_E - c_i)\|^2. \tag{3.4}$$

Here, W_S and W_E are the respective widths of the synthesis and exemplar images. The additional neighbourhood comparison is taken from the mirrored pixel location along the horizontal axis. It is assumed that all bilaterally symmetric exemplar images are manually classified as symmetric and centered on the axis of reflection. Exemplar pointers for the mirrored pair of synthesis texels are updated together to the best matching mirrored texel pairs from the exemplar data. As such, only one of the two symmetric halves of the image needs to be corrected as the mirrored pixels are updated as well. This symmetry approach was chosen because it can account for approximate symmetry such as a human face, in addition to perfect symmetry as is seen in the chess example.

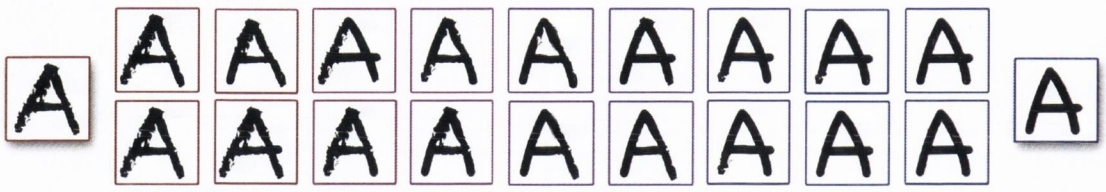


Figure 3.10: Exemplar Weighting. *Synthesis results from linear exemplar weight interpolation from leftmost exemplar image to rightmost.*

3.4.6 Exemplar Weighting

Individual exemplar images can be given special emphasis during synthesis through the use of per-image weighting. The mechanism for this is the correction step where new link coordinates are chosen from the candidate set based on a neighbourhood similarity error metric. By weighting the neighbourhood comparison error values, based on the exemplar image, which the target neighbourhood is taken from, certain images features can be favoured or discouraged in the final synthesised result. Exemplar weighting is a controllability mechanism that allows the user to direct the algorithm towards desired results. Exemplar weighting can exist for each image, for each image at each scale level, or per-pixel in each image using a weight map. It is also possible to randomly weight exemplars in order to promote diversity between outputs. Figure 3.10 illustrates the use of exemplar weights to favour specific images.

3.5 Evaluation

This evaluation system is based on the texture energy metric (neighbourhood distance) introduced in Texture Optimization for Example-based Synthesis [KEBK05] and expanded upon by Inverse Texture Synthesis [WHZ*08]. All neighbourhood based texture synthesis approaches operate on the concept that if an output shares the same local neighbourhood features as some input, then it is the same

texture as that input. Optimisation based synthesis starts from a random image (inspired by the input) and updates each neighbourhood in the output to better resemble its closest neighbourhood in the input. By running this process in an Expectation-Maximization (EM) type algorithm, the global error between all neighbourhoods in the output and their closest neighbourhoods in the input is reduced. This process is repeated until the global error plateaus and can no longer be reduced by further optimisation. This global energy metric serves many state of the art texture synthesis approaches [KEBK05; WHZ*08; BaBGF10] as a quality measure for the synthesised texture.

We adopt the same idea to create our own quality measure for texture synthesis and hybridisation. Rather than using this metric as the guiding force during synthesis, it is applied after synthesis for validation purposes only. Previous approaches operate under the assumption that if the output is fully optimised, then it is as similar as it can be (from its starting state) to the input texture, and thus can be considered the same texture. In practice the global neighbourhood distance is just the summation of the individual neighbourhood distances from each pixel in an image to its nearest neighbour. Because structured images are evaluated, the multiscale descriptor (introduced in this chapter, in section 3.3.2) is used for neighbourhood comparison. For each scale level PCA is used to reduce each 5×5 neighbourhood down to its three principle components, which are compared instead.

We look at neighbourhood distances for several different synthesised textures in order to establish a ground truth range of distances which can be used as a frame of reference for evaluating hybridisation. Figure 3.11 evaluates several texture synthesis images produced by Lefebvre and Hoppe [LH05]. Each image is paired with its distance map. The distance map visualizes the neighbourhood distance from each pixel to its most similar match in the exemplar. For the case of an exemplar images, its nearest neighbours are choses among all other pixels five percent of the image size away. In the case of the results, each pixel's distance is

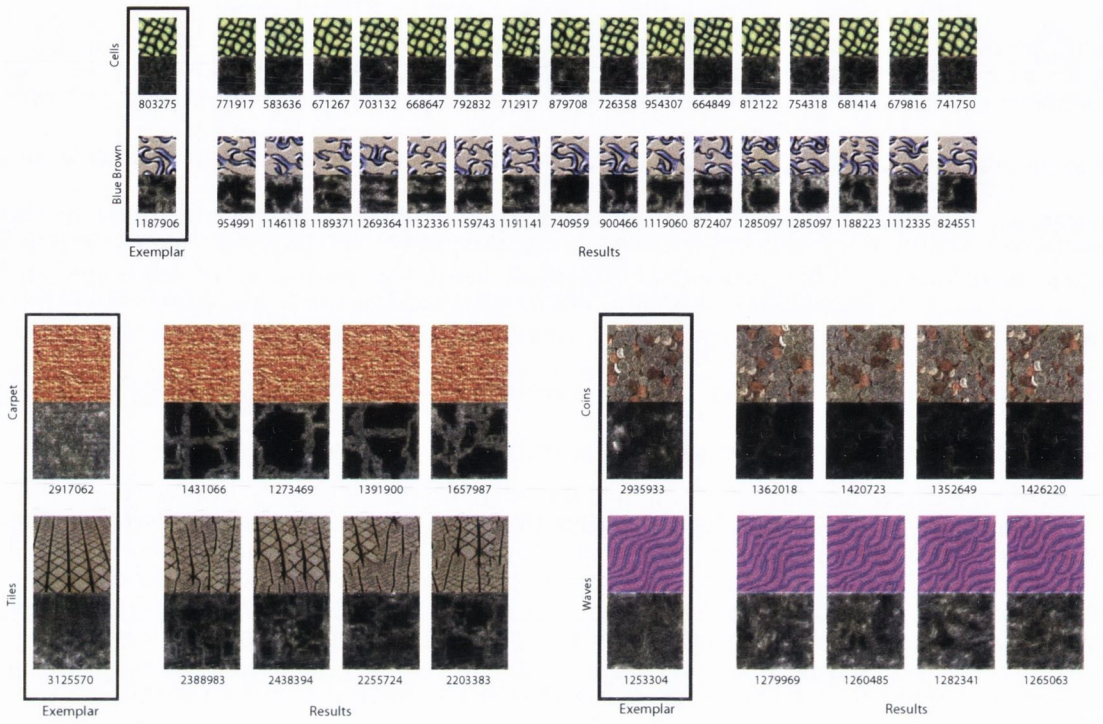


Figure 3.11: Texture Neighbourhood Distance. *This illustration shows the neighbourhood distance values for six textures taken from the supplemental results of Parallel Controllable Texture Synthesis. The first two examples, cells and blue brown both use 64×64 pixel exemplars and display a single 256×256 pixel output texture cropped into 16 smaller blocks for comparison against the exemplar. The remaining four examples start from a 128×128 pixel exemplar and the resulting 256×256 pixel output is chopped up into 4 smaller blocks for comparison. In each case the texture images are shown with their corresponding distance maps given below. In the case of texture synthesis where there is only one exemplar, each pixel searches for its nearest neighbour among all other pixels in the image at least five per-cent away, which is standard practice. For each pixel in the results, the nearest neighbour is found among all the pixel locations in the exemplar. low to high distance is mapped from black to white. The summation of all neighbourhood distances for every pixel in the map is given below.*

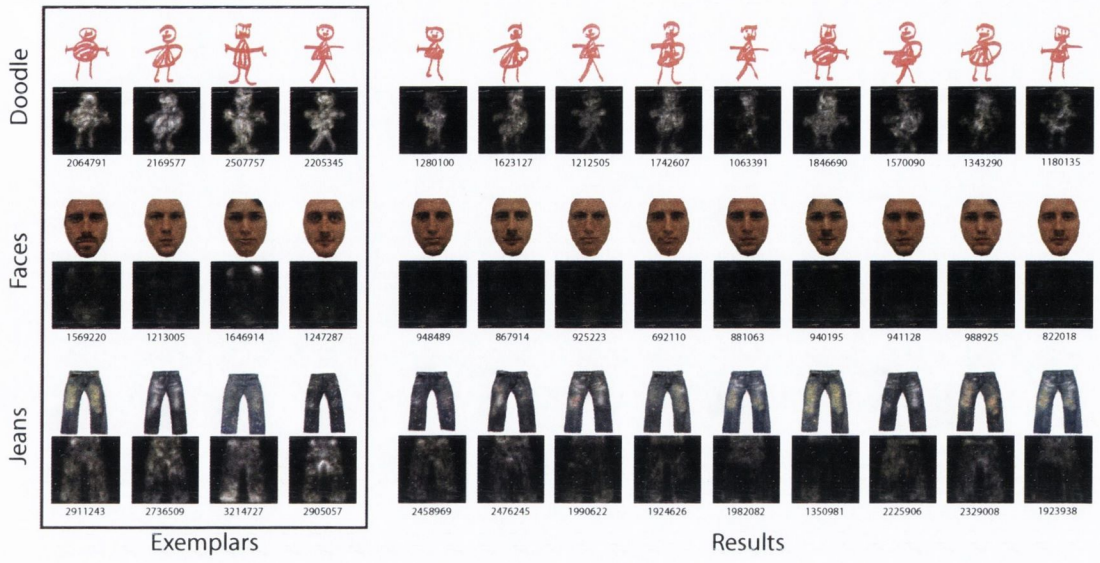


Figure 3.12: Hybrid Neighbourhood Distance. *Three examples are given with four exemplars and nine results each. The distance map and total distance is given below each image. In the case of hybridisation, for each pixel in each exemplar image, its nearest neighbour is found among any of the pixels in any of the other exemplar images. In the case of the results, its nearest neighbour is found among all of the exemplar data.*

computed for the nearest exemplar pixel chosen from among all of them. Low to high neighbourhood distances are mapped for each pixel as black to white, with the summation of all neighbourhood distances for an image given below the map.

Hybrid evaluation uses the same formatting as texture evaluation. Figure 3.12 gives three hybrid examples with four input exemplars and nine synthesised hybrids each. For each pixel in an exemplar, its nearest neighbour is chosen from any pixel in any other exemplar. Each pixel in the result has a nearest neighbour pixel taken from any of the input exemplars.

Graphical comparison of the two approaches is shown in figure 3.13. Pink bars measure exemplar neighbourhood distance while green bars convey the averaged result neighbourhood distance to the exemplar information. Note that in the case

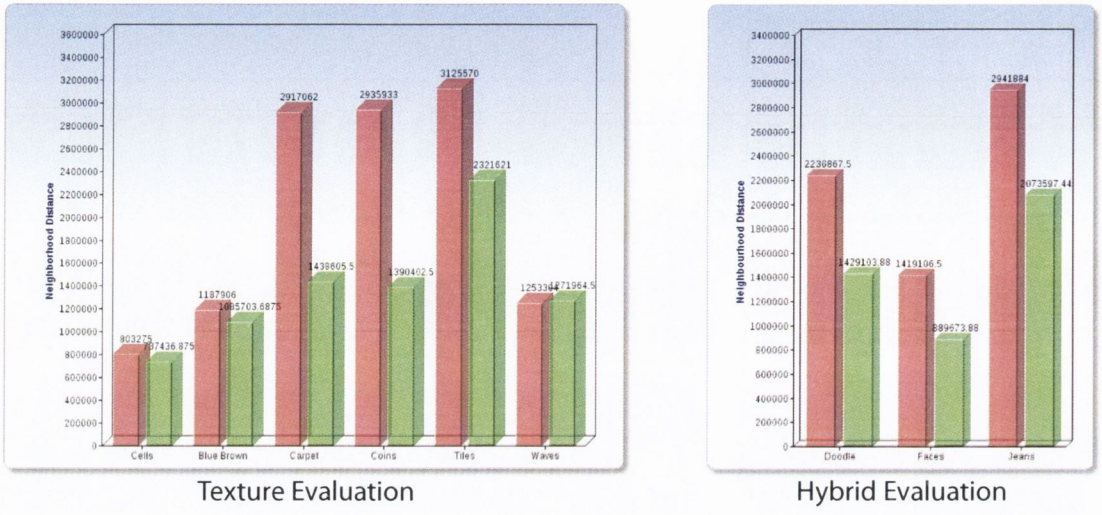


Figure 3.13: Graphical Comparison of Texture Synthesis and Hybridisation. *Left: texture evaluation illustrates the total neighbourhood distance between exemplars and results for the six examples given in figure 3.11. Right: hybrid evaluation illustrates the same thing for the three hybridisation examples shown in figure 3.12. Pink bars on the left measure average exemplar distance while green bars to the right show the averaged result distance.*

of texture synthesis the cells, blue brown and waves examples all show neighbourhood distances which are reasonably similar between results and exemplars. The carpet, coins and tiles examples all show exemplar distances which are lower than their exemplars, this however is not surprising based on the distance maps given in figure 3.11 where large coherent patches of the exemplar have been copied into the output, and in fact synthesis only occurs at borders between these patches. In the case of carpet and coins, these border regions appear to account for roughly half the image, and in the case of tiles it only accounts for three quarters, explaining the relative distances shown in the graph. It follows that for regions of an image being synthesised, it is common for the neighbourhood distance to roughly equal that internal to the exemplar itself. Hybrid evaluation on the other hand shows that the average hybrid displays less variation than the exemplars do with respect

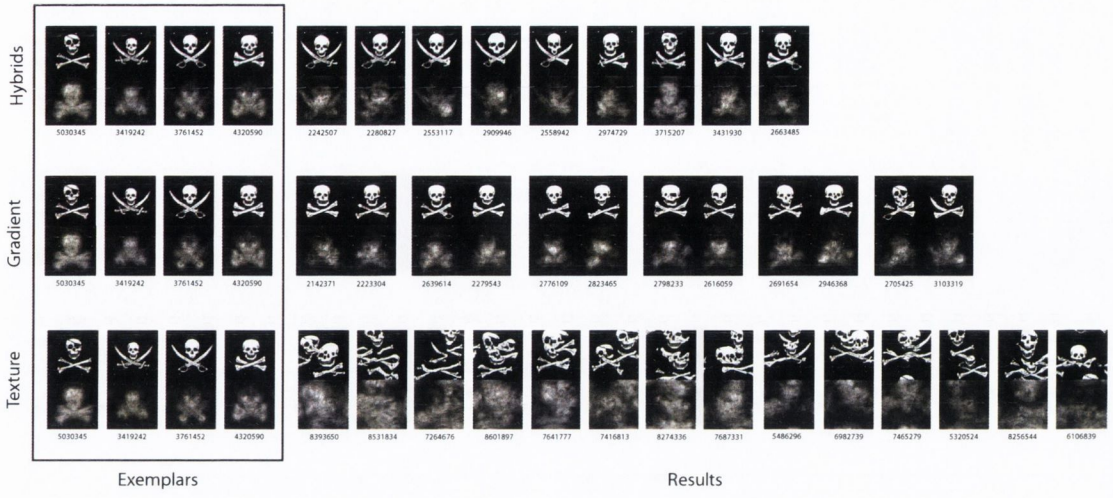


Figure 3.14: Pirate Comparison. *Hybrid evaluation is given for the pirate example (top) to show that pirates are consistent with the rest of the hybrids evaluated. The jitter spectrum example is analyzed (middle) to show that as jitter is increased, so is neighbourhood distance. Spatial jitter is used (bottom) to illustrate that texture synthesis yields a much larger total neighbourhood distance than texture synthesis and thus is not capable of generating the same types of results.*

to each other. This makes sense conceptually - input exemplars define the extreme bounds of a hybrid set, and the synthesised results fall within those bounds by blending features from the inputs.

These observations about texture and hybrids are key for understanding figure 3.14. The top row shows another example of hybridisation where four exemplars are used and nine results are given. The relative exemplar and result neighbourhood distances are consistent with those evaluated thus far. The second row borrows the hybrid synthesis results generated in figure 3.8 which illustrates the jitter spectrum where the images from left to right have increasing levels of jitter used during synthesis. Images are grouped up in pairs because they share the same jitter parameters. Notice that as jitter increases, so does the relative neighbour-

hood distance between result and exemplar. This is consistent with the concept of jitter. In the final row, spatial jitter [LH05] is used instead of the structured jitter introduced in this chapter. These results are more akin to Parallel Controllable Texture Synthesis rather than hybridisation, even though they still utilize a multiscale descriptor and multiple exemplars. Contrary to the previous evaluation for textures which showed roughly similar neighbourhood distances between exemplars and results, this evaluation yields double the neighbourhood distance for the results as it does for the exemplar. An obvious reason for this is the nature of the input. Texture synthesis is designed to work for highly stochastic textures and as such does not account for highly structured image features. It follows that the hybridisation approach introduced in this chapter produces a range of synthesis results for highly structured images which is outside the range produced by traditional texture synthesis.

3.5.1 Output Filtering

Evaluation is beneficial beyond validating the quality and necessity of this hybridisation method. This evaluation approach can be easily integrated directly into the hybridisation framework and used to automatically filter out undesirable results. Keep in mind that jitter acts as a safety net, or an upper bound for the amount of randomness introduced during synthesis. There is no lower bound for the amount of randomness introduced, nor any guarantee that the correction pass will not remove more randomness than desired. As such, the major drawback keeping this approach from being viable for industry is that often times the results show insufficient amounts of randomness, or simply reproduce one of the inputs exactly. By incorporating the evaluation approach presented in this section as a final filtering step, results with little or no variation from any input can be automatically thrown away. So can results which vary too drastically from the desired range of hybrids. The preliminary results shown in this section give some useful

insight about the range of relative distances that should be expected for a typical case of hybridisation.

3.6 Results and Analysis

Figure 3.16 depicts typical results for a variety of inputs, including drawings, photographs, objects with textured surfaces, solid shapes, both natural and anthropogenic. This hybridisation system produces an unlimited number of results, representative examples from typical usage are shown.

Inspired by *Visio-lization* [MPK09], the performance of this approach is explored for images of human faces (FACES), a particularly challenging problem because of our attuned perception to facial structure. Despite that the algorithm does not "know" specifically about faces, we observe however that for highly standardised facial photographs, the resulting hybrids are surprisingly perceptually compelling. We attribute this primarily to the built-in prudence of our structure-preserving jitter. Likewise, in the other organic examples (BEETLES, BUTTERFLIES) our system was able to recreate the natural templates defined within a given species.

We also tested our system on highly structured test cases (CHESS, LETTERS), noting that the resulting variations retain the rigid structure of the input. On slightly less structured inputs (Fig. 3.1, PIRATES), the system discovered useful, interesting, and even creative variations. For even more loosely structured inputs (DOODLE), we were pleasantly surprised that not only the important structural variation was recovered, but also to some extent the artistic *style*.

Being a straightforward extension, our timings were roughly equal to those seen in our reference CPU implementation of the [LH05] algorithm. Specifically, hybridisation was found to yield a 30% computational overhead from the previous technique, which should follow for any implementation as well. This approach is

Figure 3.15: Colour Limitation. *The rudimentary colour normalisation scheme can struggle when exemplars have radically different colour palettes.*

also capable of real time performance on a GPU.

Limitations

As our rudimentary colour standardisation scheme mainly addresses intensity normalisation between the exemplars, it can struggle when the multiple exemplars have greatly disparate colour palettes as shown in Figure 3.15. Such an input with widely varying colours carries an inherent ambiguity when defining the desired space of output colours, which will result in potentially unwanted variations. One potential solution is to increase the number of exemplars to more densely cover the desired space; however, this process can be tedious and computationally expensive. Alternatively, more advanced colour transfer techniques [PKD07] can reduce colour variation by transforming all exemplars so that they map to the same colour palette.

The perceived randomness is controlled by a jitter parameter, J. The ideal range of values for J will vary depending on the particular input and desired randomness. Fortunately, since structure-preserving jitter degrades gracefully into correction when randomness is inappropriate, it still produces a usable, if not highly-varied image, for most reasonable settings.

3.7 Discussion

This chapter has shown how to hybridize two or more images using a texture synthesis algorithm. We now discuss the intuition behind this idea, and how hybrids relate to texture.

3.7.1 Hybrids: Texton Synthesis

An early attempt at defining texture resulted in the "texton" concept, referring to the fundamental micro-structures present in natural images which are analogous to atoms but for visual perception [Jul81]. Building on the texton concept, Song-Chun Zhu wrote several important follow up papers [ZM98; ZGWX05] that further explored the concept of textons as a component of texture, as well as proposed the first rigorous mathematical definition of textons. This model treated any texture as two components, a collection of textons, and the spatial relationship with which textons are arranged.

In plain English, a texton can be thought of as the largest coherent chunk of data that is unique, as in there is no redundant information. For example one human face conveys unique information, the shape of the head as well as the positions of the mouth, nose, and eyes. However a second human face, while being different still contains the same overall shape and arrangement of parts, hence most of the information in the second face is redundant. Once the sample size has been upped to one thousand faces, a probability model is defined for a class of textons, human faces in this case. As we showed earlier, the problem of texture synthesis is to produce unique output data that shares the same probability distribution function as the input data. Therefore, the goal of hybridisation is to synthesise a unique output that matches the probability distribution function defined by all the input textons.

Following Zhu's work, we point out that all Texture synthesis approaches fall

along a gradient from synthesising only textons removed from the problem of their arrangement [RHDG10] (the approach we introduce in this chapter) at one extreme, and only synthesising the arrangement of a single texton or static set of textons that never change 2.3.3 at the other extreme. Most texture synthesis approaches fall somewhere in-between where they don't account for the distinction between textons and their arrangements, and instead find some other statistic about an image such as colour values in neighbourhoods of pixels from which to try and match the output to the input. The pitfall of many of these approaches is that because they don't account for the textons in their texture explicitly, these approaches break complex structures and are only suitable for highly stochastic images (i.e. only very small and simple textons), or they simply mutate input textons arbitrarily such that the output textons have no relation to the communal statistics of the input textons.

While the approach presented in this chapter follows the Parallel Controllable Texture Synthesis framework from a technical standpoint (and neighbourhood based texture synthesis approach pioneered by Li-Yi Wei), in fact the theoretical intuition behind hybridisation as a texture synthesis problem follows from the texton concept proposed by Song-Chun Zhu.

3.8 Conclusion

This chapter introduced hybridisation, one of the core concepts and main contributions of this thesis. We introduce the concept of hybrids as an unexplored extension of texture synthesis. We propose a new hybridisation algorithm as a simple, easy to implement extension to a popular texture synthesis framework, introducing the multiscale descriptor as well as structure preserving jitter as the two main functional modifications to traditional texture synthesis. We offer insight into why hybrid synthesis works and where hybrids fit in the greater area of texture.

Having introduced Hybridisation as a new type of texture synthesis, we spend the rest of this thesis exploring how to incorporate geometric multimedia data into a texture synthesis framework as such types of data lend themselves nicely to hybridisation.

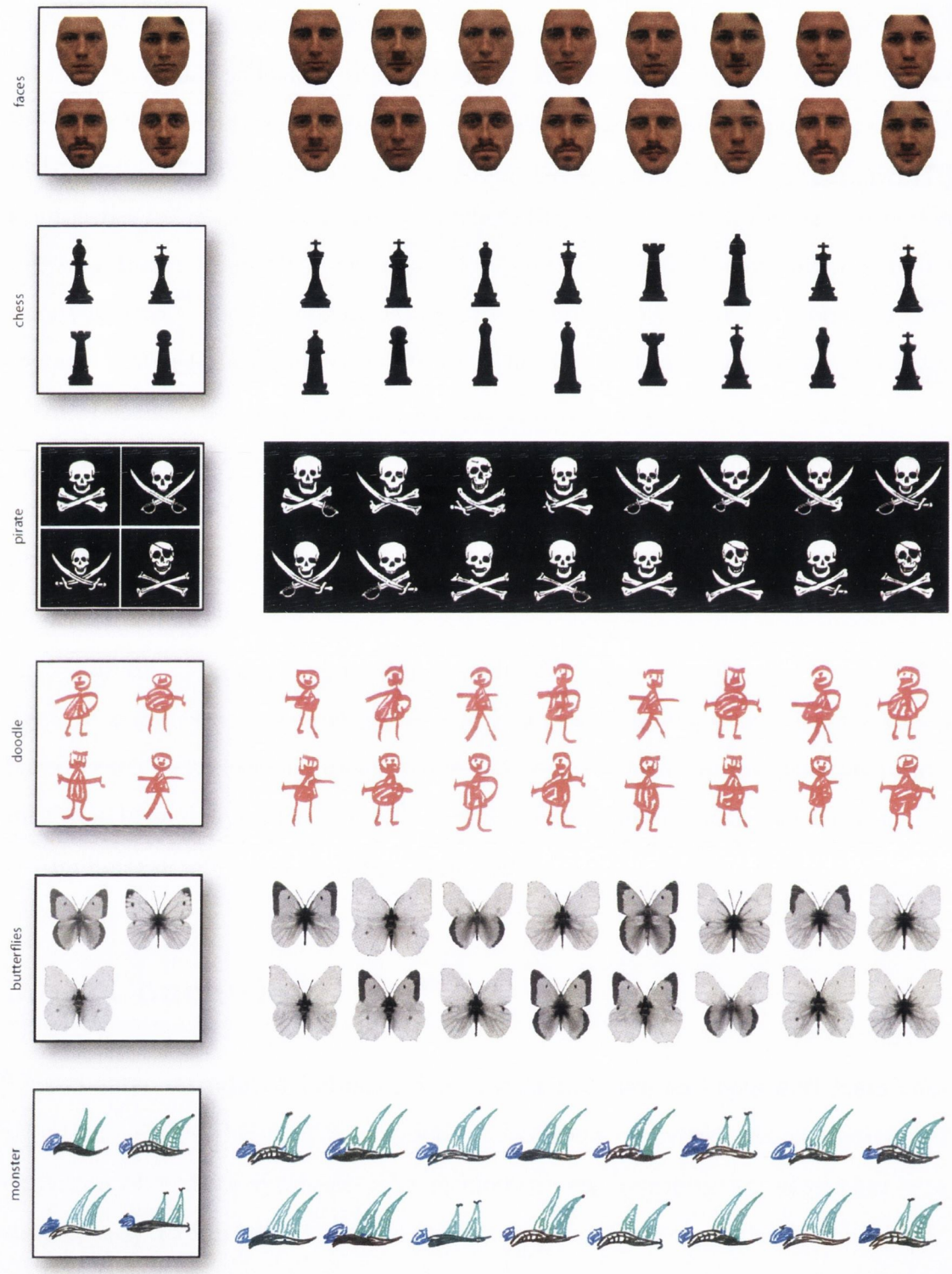


Figure 3.16: Synthesis results. *A series of example inputs and the results generated by our method.*

Chapter 4

Curve Textures and Hybrids

Figure 4.1: Multi-layer curve hybrids. *Using our approach, limitless unique instances of a cartoon drawing can be synthesised.*

4.1 Introduction

In this chapter we explore adapting texture synthesis and hybrids to the curve domain as it is especially suitable for the hybrid approach. Although texture synthesis has been widely studied across many domains, relatively little work has been done on synthesising geometry. The previous work in this area has been discussed in Section 2.3. The goal of this chapter is to introduce new ideas for applying texture based techniques directly to 2D curves.

We build our new ideas around a popular texture synthesis and hybridisation framework [LH05; RHDG10], making it possible for native geometric formats such as polylines to be synthesised and hybridized as quickly and easily as images. The goal of this chapter is to bridge the domain gap between pixel based images and geometric curves. We hope to lay the foundation for a new direction in automatic geometric content creation.

Bitmaps vs. Curves

Bitmaps (Pixel based images) have inherent characteristics which make them inefficient at representing certain images. Vector based images provide an alternative format where a picture is stored as a collection of geometrical primitives such as points, lines, curves and polygons. Each approach has its own relative merits. Bitmaps are a general format which can represent any image. They are practical when dealing with detailed or real world images (such as photographs or paintings) where each point on the image needs its own colour. Vectors become attractive when image complexity is relatively low, but resolution needs to be high. Vector graphics are resolution independent and can be rasterized to any desired scale, large bitmaps quickly become prohibitive due to high memory requirements.

In addition to the general strengths of vector based graphics over bitmaps, there are also properties which make them attractive to texture synthesis and hybridisation. Specifically, bitmap based images are comprised of a grid of pixels where the pixels are the primitive elements of the image. In neighbourhood based synthesis, coherent neighbourhoods of primitive elements are compared in order to measure image similarity around a point. This means that picture elements are compared based on the spatial region around them, not based on the actual image feature which that picture element contributes to. By using vector graphics, neighbourhoods can consist of the meaningful shape features that make up the image, independent of their size or how they span the image itself.

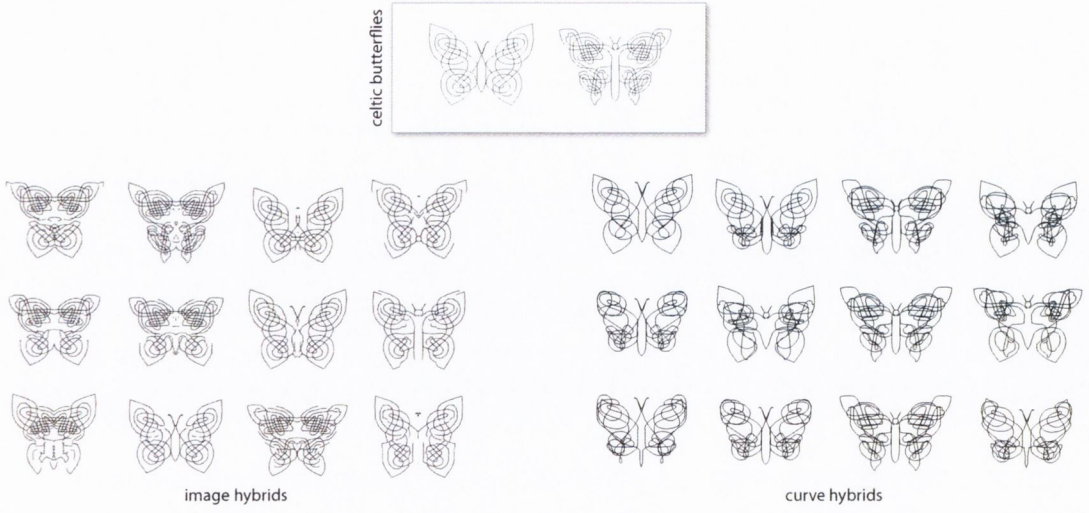


Figure 4.2: Image Hybrids vs. Curve Hybrids. *Given the two input exemplars (top), these hybrids were produced using the image hybrids approach introduced in the previous chapter (left) and the curve hybrids approach proposed in this chapter (right). Note that the curve results are resolution independent and do not have broken or disjoint line segments.*

Important features which span the image (like sharp lines) are not compared in their entirety by bitmap synthesis and shape information is not synthesised correctly. Line features are also lost at coarser stack levels as they are averaged into the surrounding pixels and lost. This can lead to continuous lines being broken during bitmap synthesis. These artifacts can bee seen in figure 4.2 where the results on the left are generated using the image hybrids approach, and the results on the right are created using the curve hybridisation presented in this chapter. In addition to being resolution independent, curve hybridisation can introduce large variations without breaking lines.

While this chapter does not tackle general vector graphics, it does deal with Curves, which are one of the common primitives used in vector images.

Contributions

We extend multiresolution texture synthesis and hybridisation to operate on planar polylines (i.e., curves in 2D). To better understand the goal of this work we review the general texture synthesis problem statement: given some input information, run a process that creates more similar but unique information as output. We try to segment our texture synthesis framework into those components that are specific to the process itself, and those which are specific to the type of data being processed (e.g. images, audio, animation, geometry, etc.). By separating these components, we have identified the aspects of the algorithm which are key to texture synthesis in general and those that are specific to a multimedia type. Given any type of information, our texture synthesis approach will need to segment that information up into coherent patches called neighbourhoods and be able to find a single numerical distance value between any two neighbourhoods. The modern texture synthesis/hybridisation framework requires a multiresolution representation for any input. The creation method for this multiresolution representation is also domain specific.

We propose the "shape texel", a discrete unit of shape and the geometric counterpart to the pixel. Like pixels, the shape texels can be averaged together in order to down-sample the curve. We also present a method for gathering neighbourhoods of shape texels and comparing them based on shape similarity.

We discuss how to plug our geometric extensions into a parallel controllable texture synthesis [LH05] framework. Making geometry specific modifications to the algorithm so that synthesis texel values no longer take on the exact value of their exemplar counterparts, but try to mimic the neighbourhood shape instead. This entails a re-engineering of the three basic synthesis steps, upscale, jitter and correction, as well as adding a fourth step we call the "solver step" that is run in conjunction with each of the three original steps in the synthesis pipeline. Lastly, we present extensions necessary for applying our curve synthesis framework to the

problem of hybridisation [RHDG10]. We show a simple graph based procedural approach to synthesising multiple layers of discrete curves, inspired by Han et al. [HRRG08], which produces more complex visual results.

To summarize, our contributions include a new shape descriptor, a procedure for neighbourhood comparison, an approach to downsample vertices (along a polyline resulting in a coarser approximation of the shape) and finally an overhaul on a popular texture synthesis framework, tailored for the geometry domain.

4.2 2D Shape Representation

This section introduces the key ideas behind representing 2D curve information as a texture or hybrid.

4.2.1 Geometric Input and Output

We define the input and output information used in our synthesis process. This work focuses on 2D curves. These are represented in the algorithm by open or closed semi-regular polylines. A polyline is a continuous line composed of one or more line segments where the endpoints of each line are represented by vertices and the line segments are the edges between those vertices. A semi-regular polyline is a coarse polyline that has been repeatedly uniformly bisected, so that the final polyline has a number of edges that is a small integer times a power of two. The distinction between open and closed simply refers to whether the polyline as a whole does or does not have endpoints (e.g. a line is open while a circle is closed). As input, the method accepts one or more *exemplars*: semiregular polylines [WSBD00]. The output is always a single semiregular polyline.

In computer memory a polyline is represented as an array of coherent vertices $\{\mathbf{V}_i\}_{i=0,\cdots,I}$ where a vertex is an (x, y) coordinate (Note that the index i indicates

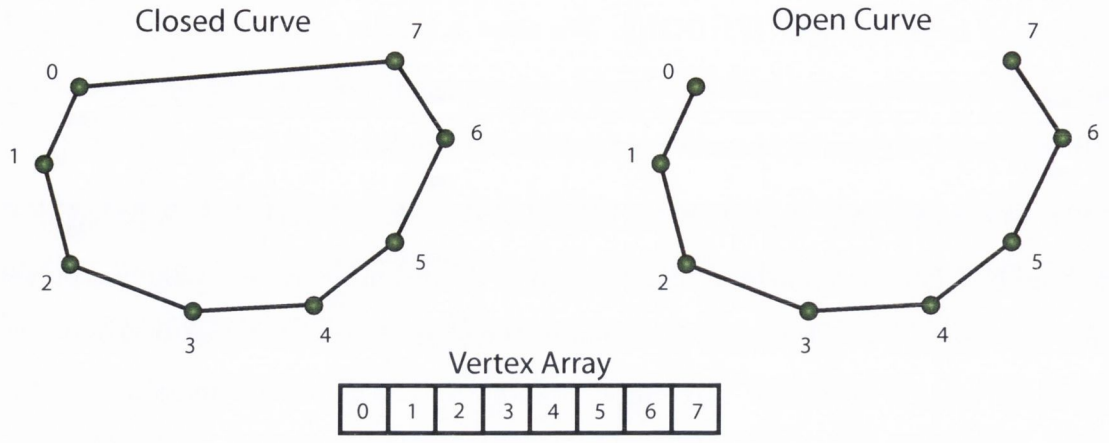


Figure 4.3: Open and Closed Curves. *A curve is stored in computer memory as an array of vertices where lines exist between adjacent array elements. The only distinction is whether a line is drawn between the first and last elements in the array, indicating whether we have an open or closed curve.*

the parameterization of the curve [HOCS02]). A line exists between the vertices of any element and the subsequent element of the array. The only distinction is whether a line is assumed to exist between the first and last elements as well. This distinguishes whether our curve is open or closed as illustrated in figure 4.3.

4.2.2 Shape Texels

While the input and output for the algorithm comes in the form of polylines comprised of vertices, internal to the algorithm each vertex is replaced by a more descriptive counterpart called a "shape texel". Polylines are stored in memory as a coherent array of shape texels $\{\mathbf{T}_i\}_{i=0,\cdots,I}$. Following the recent advances in 2D shape representation used in computer vision [DTT08], mathematically the shape texel is modeled by a 2D Gaussian function, this includes a mean $\boldsymbol{\mu}$ (i.e. (x, y) centre position, just like a vertex), and the covariance matrix (2×2) of a Gaussian probability distribution represented by Σ. Using Gaussians to represent shape is a well established idea. To the best of our knowledge however, we are the first

to utilise Gaussian functions to comprise regular geometry in the same way pixels are used to comprise images. We show that a Gaussian can act as a discrete unit of shape and behave like the geometric counterpart to the pixel, we thus find it appropriate to coin the term "shape texel".

Gaussians make a convenient mathematical model for shape texels. There are established methods for averaging multiple Gaussians together into a single coarser Gaussian. There are also approaches for comparing the distance between two Gaussians, both in terms of position in space as well as shape. Despite these reasons the real value behind using a Gaussian function as a shape texel goes far beyond the convenience with which they can be combined and compared. Although a curve exists in 2D space, it is algorithmically processed as a 1D function. Our polyline is a Gaussian mixture model that summarises the density of this shape in 2D space. In previous approaches [AL05], the space itself would have been discretised into an image. Each pixel models a point in space and the shape is summarized by storing the density value (or distance to the surface of the polyline) at every pixel in the image. This is highly wasteful as the majority of the space is empty or does not contribute directly to the shape being modeled. Conceptually, we take the opposite approach to the problem: rather than using a volumetric model in some space to represent a curve, instead, the curve itself summarises a volumetric model. Although this distinction is subtle, it is the most important idea of this chapter. Decoupling the dimensionality of the geometric information from the space it exists in offers the best of both worlds; The curve can be modeled as a higher dimensional shape (which is necessary to build a proper multiresolution representation), while keeping the computational and memory complexity of the lower dimensional shape itself. In fact, using shape texels a 1D polyline can exist and be properly described in three or higher dimensional space with no additional algorithmic cost.

4.2.3 Neighbourhoods

A neighbourhood is a set of five coherent shape texels. The neighbourhood around any given shape texel $\mathbf{T}_i$ can be sampled from the polyline by taking the two elements preceding the texel i and the two elements following. Formally this is defined as neighbourhood $\mathbf{N}_i = \{\mathbf{T}_{i+j}\}_{j=-2,\cdots,2}$.

4.2.4 Averaging Neighbourhoods

Having sampled a neighbourhood of shapes around texel i we now require a new method for mixing or averaging them into a single new coarser shape texel.

Considering the shape texel neighbourhood $\mathbf{N}_i$ which consists of the five Gaussian functions $\{\boldsymbol{\mu}_j, \Sigma_j\}_{j=0,\cdots,4}$, we model the probability density function of $\mathbf{x}$ to be on the shape near texel i as a mixture of normal distributions

$$p_i(\mathbf{x}) = \sum_{j=0}^{4} \omega_j \times \mathcal{N}(\boldsymbol{\mu}_j, \Sigma_j)$$

where the weights $\{\omega_j\}_{j=0,\cdots,4}$ control the importance of each mixture component. These weights are set as $\omega_0 = 0.061$, $\omega_1 = 0.242$, $\omega_2 = 0.383$, $\omega_3 = 0.242$, and $\omega_4 = 0.061$, being modeled from a Gaussian distribution summing to approximately 1.0.

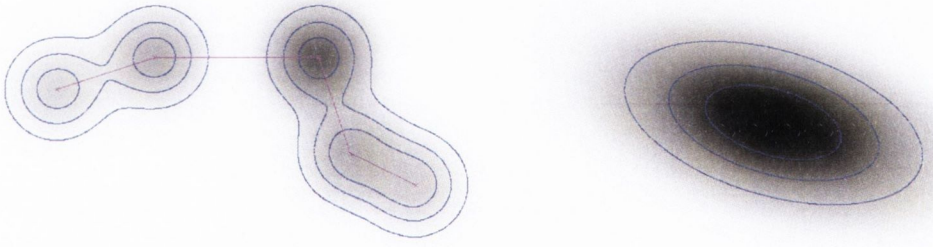
Averaging the neighbourhood of Gaussian distributions will approximate p_i by a simple normal distribution $\tilde{p}_i \equiv \mathcal{N}(\tilde{\boldsymbol{\mu}}_i, \tilde{\Sigma}_i)$ such that

$$\begin{cases} \tilde{\boldsymbol{\mu}}_i = \int \mathbf{x}\ p_i(\mathbf{x})\ d\mathbf{x} \\ \tilde{\Sigma}_{i,xy} = \int (x - \mu_x)(y - \mu_y)\ p_i(\mathbf{x})\ d\mathbf{x} \end{cases}$$

giving

$$\begin{cases} \tilde{\boldsymbol{\mu}}_i = \sum_{j=0}^{4} \omega_j \, \boldsymbol{\mu}_{i_j} \\ \\ \tilde{\Sigma}_{i,xy} = \sum_{j=0}^{4} \omega_j \left[\Sigma_{j,xy} + (\mu_{i_j,x} - \tilde{\mu}_{i,x})(\mu_{i_j,y} - \tilde{\mu}_{i,y}) \right] \end{cases} \tag{4.1}$$

where $(\mu_{i,x}, \mu_{i,y})$ indicates the two coordinates of vector $\boldsymbol{\mu}_i$ and $\Sigma_{i,xy}$ is the associated covariance. Equation (4.1) is sufficient to compute all coefficients of the covariance matrix Σ_i for 2D texel nodes $\mathbf{x} = (x, y)$. $(\tilde{\boldsymbol{\mu}}_i, \tilde{\Sigma}_i)$ defines a coarser texel. To illustrate this process, figure 4.4 shows the probability density function $p_i(\mathbf{x})$ for a neighbourhood of shape texels and the corresponding $\tilde{p}_i(\mathbf{x})$ shows the probability distribution function of the combined average (i.e. five Gaussian functions forming a Gaussian mixture model are approximated by a single Gaussian function).



$p_i(\mathbf{x})$ $\tilde{p}_i(\mathbf{x})$

Figure 4.4: Merging Texels. *Left: Texel $(\boldsymbol{\mu}_i, \Sigma_i)$ has four neighbours $(\boldsymbol{\mu}_{i_0}, \Sigma_{i_0})$, $(\boldsymbol{\mu}_{i_1}, \Sigma_{i_1})$, $(\boldsymbol{\mu}_{i_3}, \Sigma_{i_3})$ and $(\boldsymbol{\mu}_{i_4}, \Sigma_{i_4})$ that are combined in the mixture $p_i(\mathbf{x})$. Right: all five texels are merged into one texel $(\tilde{\boldsymbol{\mu}}_i, \tilde{\Sigma}_i)$ at the next level simplifying the local pdf to $\tilde{p}_i(\mathbf{x})$.*

4.2.5 Comparing Neighbourhoods

We find the distance between individual corresponding pairs of shape texels between two neighbourhoods and we find the total distance between two neighbourhoods by summing these five individual distances.

While many approaches exist to measure the distance between two Gaussian PDFs, we require an approach that measures both the distance between two Gaussian's shapes (Σ) as well as the distance between their spacial locations $\boldsymbol{\mu}$ in a single unified value. The Kullback-Leibler (KL) divergence [GGG03; JKH04; HV10] satisfies these requirements as well as offers an information-theoretically motivated measure between two PDFs. That is, the difference between two PDFs is measured in the expected number of bits required to code samples from A when using a code based on B. Therefore the PDF B represents an approximate model, or theory of the "true" distribution given by A. This is the relative information entropy between PDFs A and B. The closed form solution for KL distance is defined as

$$M(A,B) = (\boldsymbol{\mu}_A - \boldsymbol{\mu}_B)^T \Sigma_B^{-} 1 (\boldsymbol{\mu}_A - \boldsymbol{\mu}_B). \tag{4.2}$$

$$KL(A,B) = \frac{1}{2}\{\log \frac{|\Sigma_B|}{|\Sigma_A|} + (Tr(\Sigma_B^{-} 1 \Sigma_A) - 2) + M(A,B)\}. \tag{4.3}$$

The KL distance 4.3 is comprised of the Mahalanobis distance 4.2 along with terms that more accurately model the distance between PDF variance. As such, the Mahalanobis distance can be used as a faster albeit less accurate alternative to the full KL distance. The KL distance measures relative information entropy between PDFs and as such is not symmetric and thus not a true metric. It can be easily made symmetric by switching A and B and averaging the two results together. We chose a log base of 2 rather than e so that our entropy is measured in bits rather than nats.

When comparing two neighbourhoods we have an extra complication. Because we are looking for the difference in neighbourhood shape, we first have to remove the difference in global neighbourhood positions as that will add irrelevant distance and overpower the distance in shape, rendering our comparison of neighbourhoods meaningless. We treat the mean $\boldsymbol{\mu}$ of the middle neighbourhood texel as the origin

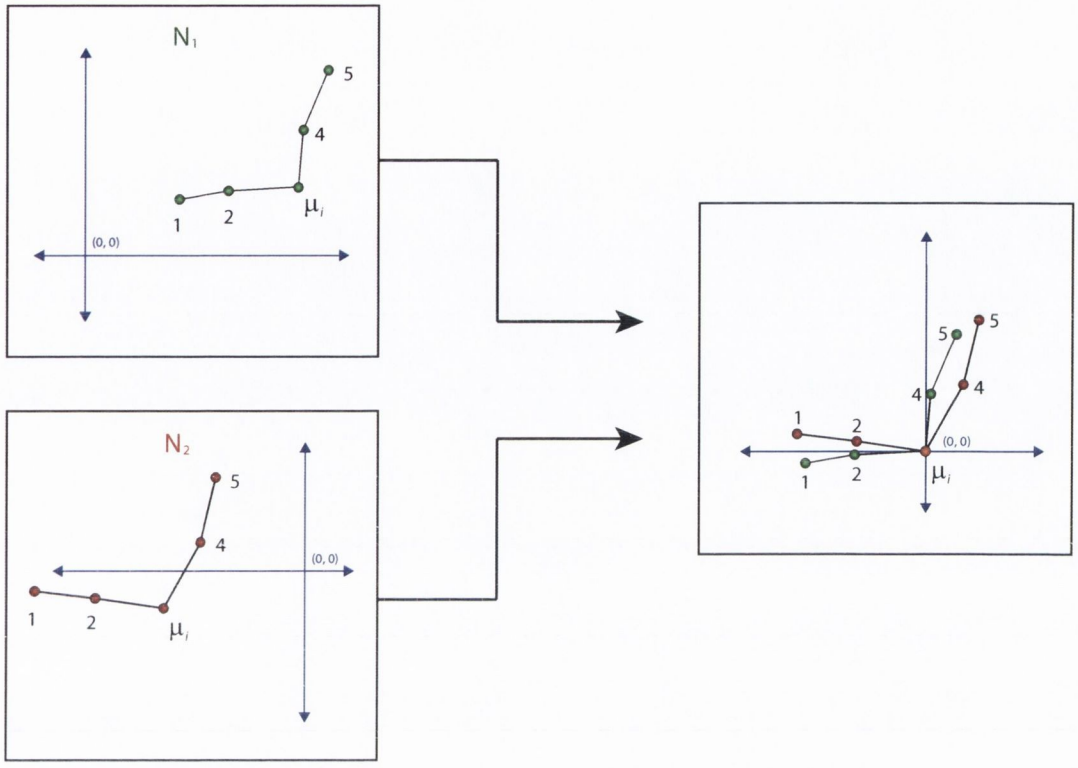


Figure 4.5: Neighbourhood Comparison. *Given two arbitrary neighbourhoods N_1 and N_2 (left), each shape texel's position is translated by the inverse of the centre texel's position. This brings the two neighbourhoods into alignment (right) so that the distance between corresponding neighbourhood elements can be determined.*

of the entire neighbourhoods local coordinate system

$$\{\boldsymbol{\mu}_j = \boldsymbol{\mu}_j - \boldsymbol{\mu}_2\}_{j=0,\cdots,4}$$

This removes the translation distance between the two neighbourhoods by centering each of them at the origin of their coordinate system as shown in figure 4.5.

4.3 Analysis

The Parallel Controllable Texture Synthesis [LH05] and Image Hybrids [RHDG10] framework synthesises its output in a coarse-to-fine manner. Given single scale inputs it is necessary to extract a multiresolution representation for each exemplar before synthesis can be performed. To accelerate synthesis our framework utilizes pre-computed nearest neighbour candidate sets for each shape texel. Both of these operations are performed once in an offline analysis phase of the algorithm. The results are stored so that synthesis can operate at interactive rates.

4.3.1 Multiresolution

First a stack is extracted for each exemplar as seen in Section 2.2.6. The only requirement on the input is that it come in the form of a semi-regular polyline. The reasoning behind that requirement now becomes apparent when building a multi-resolution representation. A semi-regular polyline contains a power of two edge count. The number of edges can be evenly divided by half at each scale level giving each level half the resolution as the previous. Parallel Controllable Texture synthesis [LH05] introduced the exemplar stack as a tightly packed combination of all multi-resolution pyramids created for every translation of the input exemplar at the finest level. Formally E_l^k indexes the exemplar stacks where l denotes the exemplar stack level and k indexes the exemplar itself. At the finest level L, the mean $\boldsymbol{\mu}_i^L$ of exemplar texel $E_l^k[i]$ is initialised to the position of the corresponding vertex in the input polyline, and the covariance matrix is initialised to $\Sigma_i^L = h\ \mathrm{I}$ (with h a chosen small scalar and I, the identity matrix). The covariance matrices $\{\Sigma_i^L\}$ are then isotropic (i.e. circular). $(\boldsymbol{\mu}_i^l, \Sigma_i^l)$ is denoted as *the texel* i at level l.

Because the exemplar stack is in fact a tightly packed representation of I multi-resolution pyramids, the neighbourhood gathering strategy must be revisited when sampling from exemplar stacks E^k. Rather than gathering a coherent set of five

texels, a level dependent offset $h_l = 2^{L-l}$ is added between the texels to be sampled. Now when gathering a neighbourhood for texel i at level l elements $E_l^k[i - h_l \times 2]$, $E_l^k[i - h_l]$, $E_l^k[i]$, $E_l^k[i + h_l]$ and $E_l^k[i + h_l \times 2]$ are gathered. Note that at the finest level L, the offset $h_l = 1$ and coherent neighbourhood gathering is used.

Downsampling

The exemplar stack is created through an iterative downsampling process in which each shape texel at a given level is downsampled by taking the neighbourhood, as shown in Section 4.2.3, around that texel with the level dependent offset $h_l = 2^{L-l}$, and combining their values into a single new shape as described in Section 4.2.4. This process is the one-dimensional equivalent to the Gaussian image exemplar stack [LH05]. For a polyline consisting of I texels, the exemplar stack is a tightly packed representation of the full family of all I multi-resolution pyramids existing for each translation of our array of texels.

Combining all texels gives a probability density function $p^l(\mathbf{x}) = \sum_{i=1}^{N} p_i^l(\mathbf{x})$ that represents the entire polyline at a given scale level. Figure 4.6 illustrates this representation at each level of resolution: starting at the finest resolution level $L = 8$ with $I = 2^L$ texels, the number of Gaussians (I) used in the mixture $p^l(\mathbf{x})$ is preserved down the Gaussian stack as each subsequent level is downsampled without being subsampled. At the coarsest level, all shape texels converge to a single unique Gaussian distribution that summarises the average shape of the entire polyline.

4.3.2 Nearest Neighbour Search

Nearest neighbour candidate sets are gathered for each shape texel in each of the exemplars. For a given shape texel T_i, a nearest neighbour is a texel T_j in some polyline who's neighbourhood is more similar to that of T_i's than any other texel in the polyline. In the case of texture synthesis, the nearest and second

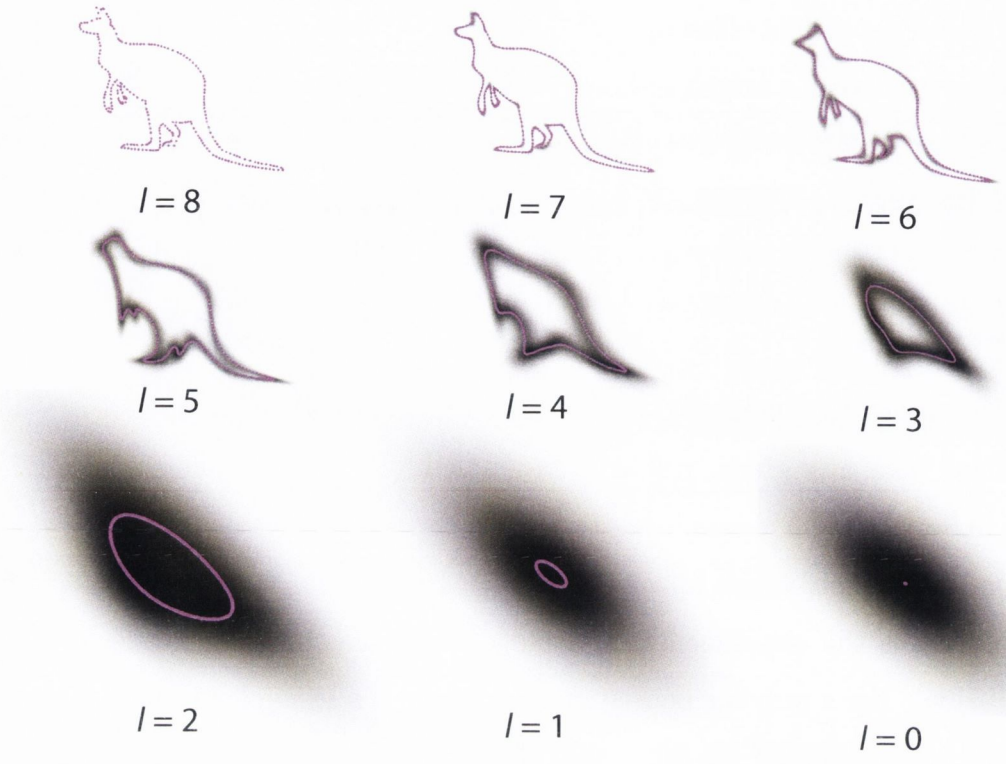


Figure 4.6: Multiscale representation of a shape as a p.d.f. $p^l(\mathbf{x}) = \sum_{i=1}^{N} p_i^l(\mathbf{x})$

nearest neighbours are found within the same exemplar polyline, with the single condition that each of the three texels in the polyline are at least five per cent away from each other in terms of their positions in the polyline's array. This strategy for gathering nearest neighbours to be used for texture synthesis is taken directly from the previous work shown in Section 2.2.6. In the hybrid case, for each texel in an exemplar, a single nearest neighbour texel is found for each exemplar. While the small overall number of shape texels in an average polyline makes a brute force search practical for most curves under 2048 texels, the patch match algorithm [BSFG09] can be used to find nearest neighbour correspondence along longer polylines.

The multi-scale neighbourhood comparison approach proposed for image hybridisation [RHDG10] is utilised. When comparing nearest neighbour locations

during pre-computation, neighbourhoods around a texel at the current level as well as all finer levels are gathered and the combined distance for all levels is used. This strategy safeguards the algorithm from carelessly breaking fine scale structures while at a coarser level. After running the nearest neighbour search on the entire exemplar set, each texel has an associated nearest neighbour list (NNL) that is used to accelerate synthesis.

4.4 Synthesis

Synthesis operates in a coarse-to-fine manner building a multi-resolution pyramid S of polylines. The first level S_0 is copied from a randomly chosen exemplar E_0^k for bootstrapping purposes. Each subsequent level is synthesised from the previous using three sequential operations, upscale, jitter and correction. We only add one key difference to this common framework, A fourth "solver" step is included. This new solver step operates in-between each of the original three stages. The synthesis process is illustrated in figure 4.7.

Synthesis texels $S_l[i]$ differ from exemplar texels $E_l^k[i]$ in only one regard. Whereas an exemplar texel would contain a list of pointers to multiple nearest neighbour exemplar texels (its NNL), synthesis texels contain only one pointer to a single nearest neighbour exemplar texel dereferenced by $*S_l[i]$. This single exemplar pointer is used to dynamically gather the synthesis texel's Nearest Neighbour Candidate Set $NNCS$ which is constantly changing through the synthesis process and must be regathered several times during the synthesis of each level.

Nearest Neighbour Candidate Set. Using the exemplar pointer $*S_l[i]$ for each of the synthesis texels and that exemplar texel's NNL in conjunction with the coherence principle [Ash01], A $NNCS$ is dynamically gathered for each synthesis texel, approximating its set of closest exemplar neighbourhood matches at any point during synthesis without the need for a slow nearest neighbour search

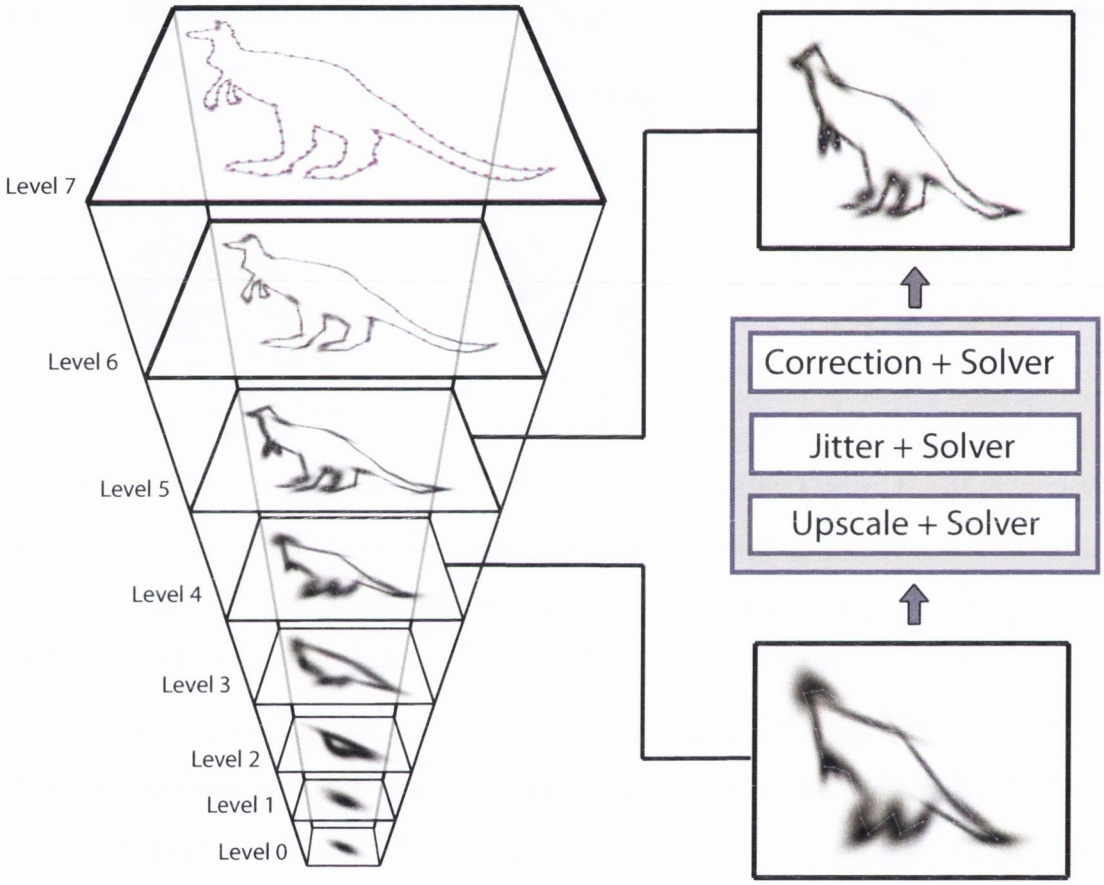


Figure 4.7: Synthesis pyramid. *Starting with a single shape texel at the bottom of the pyramid, each subsequent level contains twice the shape texels as the one below it.*

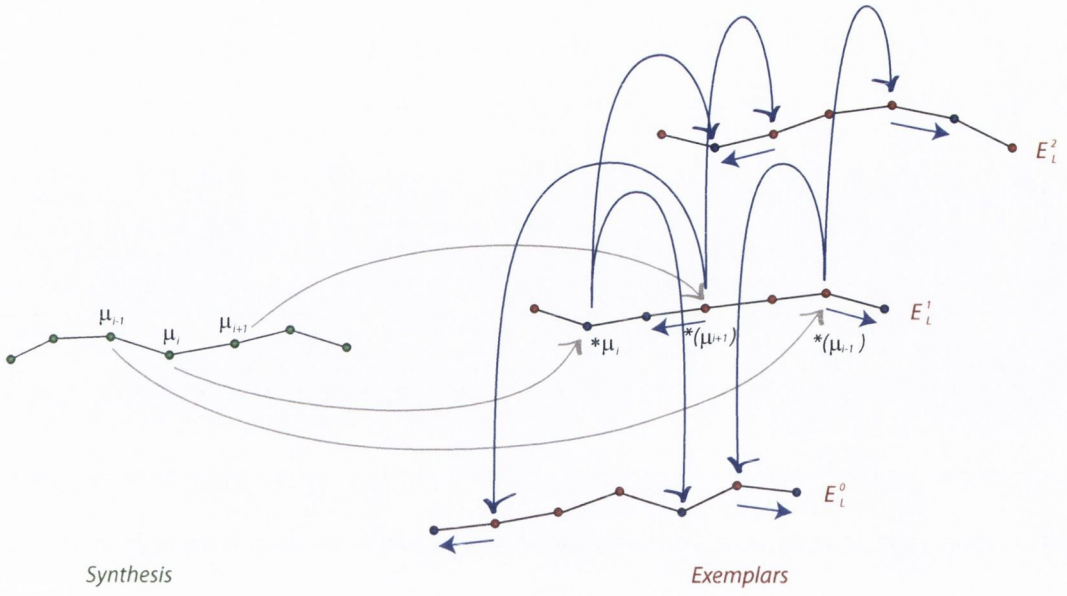


Figure 4.8: Nearest Neighbour Candidate Set. *This figure illustrates the collection procedure for a synthesis texel's $NNCS$. Grey arrows point from synthesis texels to their exemplar counterparts. Curved blue arrows point from an exemplar texel to its pre-computed nearest neighbour candidates in other exemplars (using its NNL). Straight blue arrows represent the level dependent offset h_l applied to links gathered from texels adjacent to $\boldsymbol{\mu}_i$ in order to find similar exemplar texels to $\boldsymbol{\mu}_i$ by the coherence principle. The new set of potential exemplar texels are shown in blue as opposed to the usual red.*

through all exemplar data. The coherence principle states that nearest neighbour selection is not always the best method for finding *perceptual* similarity; rather, maintaining congruency between neighbouring pixels plays a large role as well. As such, for each texel $S_l[i]$ its $NNCS$ is created by combining the exemplar link $*S_l[i]$ with its NNL. Following figure 4.8, this is illustrated by the gray line pointing to $*\boldsymbol{\mu}_i$ and the blue arrows pointing to nearest neighbour candidates in other exemplar polylines. Utilizing the coherence principle, more plausible candidates can be found for $S_l[i]$ by looking at the congruent neighbours ($*S_l[i-1]$ and $*S_l[i+1]$), along with their NNL also shown in the illustration as blue lines.

The only added complication when looking for more candidates through coherent texels, is that each exemplar texel gathered through a coherent neighbour is not a nearest neighbour to $S_l[i]$, rather the texels next to it. Again using the coherence principle, backtrack along the curve by h_l texels from these exemplar texels to the congruent neighbours that correspond to $S_l[i]$. This is shown in the illustration with straight blue lines which follow the curve. Note that the exemplar polylines are all at their finest level L, thus the level dependent offset $h_l = 1$.

With a clear idea of what a synthesis texel is and how its $NNCS$ is gathered, the entire synthesis process can be explained. This section first introduces the solver step due to its importance in each of the three original synthesis steps, and subsequently discusses those three steps: upscaling, jitter and correction.

4.4.1 Solver

During synthesis a texel's position is determined through its spatial relationships to the previous and next texels in the polyline. Ideally, the neighbourhood shape around a synthesis texel should mimic the neighbourhood shape around its exemplar counterpart as closely as possible. When two sequential texels in the polyline point to exemplar texels in different polylines, or non-sequential texels in the same polyline, discrepancies arise with respect to the spatial relationship between the two synthesis texels. In this case finding the optimal compromise between two competing exemplars leads to a "tug of war" scenario which the solver step simulates using standard gradient descent.

Specifically, for texel $S_l[i] = (\boldsymbol{\mu}_i, \Sigma_i)$ (ignoring l in the notation), relationship vectors $R_1 = \boldsymbol{\mu}_i - \boldsymbol{\mu}_{i-1}$ and $R_2 = \boldsymbol{\mu}_i - \boldsymbol{\mu}_{i+1}$ are found from each of its adjacent texels in the curve. Each relationship vector R_1 and R_2 has an exemplar counterpart $*R_1$ and $*R_2$. In the solver pass, all the relationship vectors (i.e. edges) are updated to mimic their exemplar counterparts as closely as possible.

The gathering of R and $*R$ is a similar process as the gathering of the $NNCS$

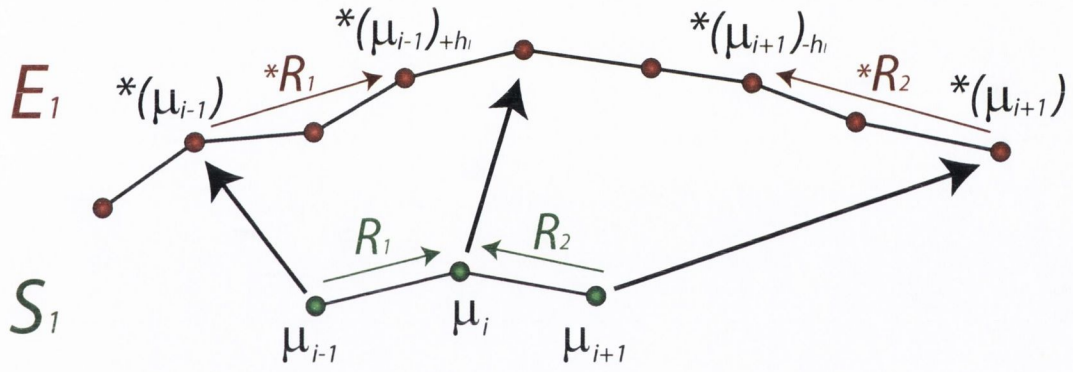


Figure 4.9: Solver. *Given a synthesis curve segment S_1 (green) and corresponding exemplar E_1 (red), we see how our solving algorithm computes synthesis and exemplar relationships. Note that this example is at level $l = 1$ and thus $h_l = 2$.*

using coherence. In this case however the NNL is ignored and the relationships between texels are found, not the texels themselves. Using figure 4.9 as a reference, exemplar relationship $*R_1$ is the exemplar counterpart of R_1 determined by taking the exemplar link from $\boldsymbol{\mu}_{i-1}$ to $*\boldsymbol{\mu}_{i-1}$ and then finding it's adjacent neighbourhood texel $*(\boldsymbol{\mu}_{i-1})_{+h_l}$ by backtracking along the curve using the level dependent offset h_l.

Now, armed with the exemplar relationships $*R_1 = *(\boldsymbol{\mu}_{i-1})_{+h_l} - *(\boldsymbol{\mu}_{i-1})$, and $*R_2 = *(\boldsymbol{\mu}_{i+1})_{-h_l} - *(\boldsymbol{\mu}_{i+1})$, subtract the synthesis relationships R from their respective exemplar relationships $*R$. The resulting vectors are used to offset $\boldsymbol{\mu}_i$. For each texel in the current synthesis level, a gradient vector $g[i]$ is computed with which to displace the texels positions by a small increment. Because the texels are exhibiting unequal push and pull on each other it is necessary to find the gradient of motion that will minimize the total push/pull energy of the system. This is an iterative optimization process where a new gradient is first computed for each texel and then texel positions are all updated by a small increment. Once the polyline shape is updated, a new energy is calculated for the entire system. This process repeats until the difference between the previous iteration's global energy and the current global energy is below a user specified accuracy threshold. The following

pseudo code 1 summarizes the process:

Algorithm 1 Pseudo code for solver pass.

repeat

array g initialized to 0

for $i = 0 \rightarrow N - 1$ **do**

Compute $R1$, $*R1$, $*R2$ and $R2$

g[i-1]+=R1-*R1

g[i]+=*R1-R1

g[i]+=*R2-R2

g[i+1]+=R2-*R2

end for

$\mu_i = \mu_i + \alpha\ g[i]$

until $e = \sum_{i=0}^{N-1} |g[i]|^2$ below a given threshold

We update each texel position by offsetting its current position $\boldsymbol{\mu}_i$ by gradient $g[i]$ multiplied by some user specified scaling coefficient α.

4.4.2 Upscale

The base of the synthesis pyramid S_0 is created by copying texel values directly from some exemplar where each synthesis texel stores a pointer $*S_0$ to the exemplar texel it is mimicking. All subsequent pyramid levels are created by upscaling the previous level. Given some polyline, the next finer polyline is created by splitting every edge and creating a new shape texel for each split. Therefore, the new polyline consists of vertices from the previous level's polyline (called parent texels), and all the new texels created by splitting edges (called child texels). Given the current level polyline S_l of size I, the next level polyline S_{l+1} will have size $2I$. As opposed to image upsampling which operates in a single pass, polyline upsampling operates in two passes. In the first pass the parent texels are duplicated from the previous level to the next. In the second pass the child texels are created using the

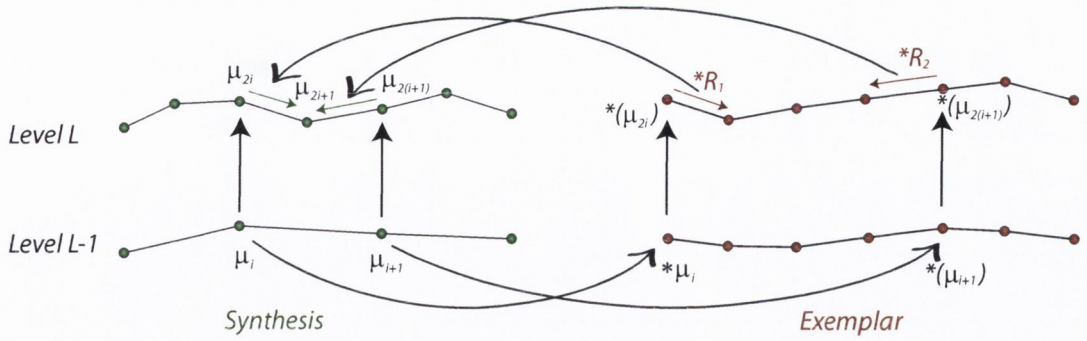


Figure 4.10: Upscale. *Upscaling is divided into two passes, first parent texels in level S_{L-1} are copied into S_L with relative position offsets given by the examplar. Next child texels are created using the two surrounding parent texels as a guide.*

parent texels as both a skeleton and an exemplar reference. The key distinction between the two texel classifications and the two upscaling operations is that a parent texel can be upscaled from a single synthesis texel and thus have a clear and direct exemplar archetype; child texels on the other hand require two synthesis parent texels and do not have their own clear exemplar counterpart, but must be created using their parents as a guide.

Parent Texels. For each texel i in the current synthesis level S_l, the exemplar index k and polyline texel index j are found for the exemplar texel pointer $*S_l[i]$. Although the parent texels are directly propagated from the current synthesis level to the next, their shape parameters $(\boldsymbol{\mu}_{2i}^{l+1}\Sigma_{2i}^{l+1})$ will change between levels. The new position are found by adding the relative change in the exemplar positions to the previous texel. Formally, using the positions $\boldsymbol{\mu}$ for each of the referenced texels $S_{l+1}[2i] = S_l[i] + [E_{l+1}^k[j] - E_l^k[j]]$, the new variance Σ is copied directly from the corresponding exemplar texel $E_{l+1}^k[j]$ and the exemplar pointer $*S_{l+1}[2i]$ is set to $E_{l+1}^k[j]$.

Child Texels. After the parents have been created, the new polyline has every even array element fully defined. A second pass fills in all the odd elements that correspond to edge splits and thus child texels. The position of a child texel is

determined by the positions of its already upsampled parents, and their exemplar counterpart's relationship to the corresponding split edge texel. This process is illustrated in figure 4.10 where each of the parent texels $\boldsymbol{\mu}_{2i}$ and $\boldsymbol{\mu}_{2(i+1)}$ contribute an estimate for $\boldsymbol{\mu}_{2i+1}$ based on their position, and the relationship $(*R)$ their exemplar link would have to its corresponding child texel based on coherence. The new texel position $\boldsymbol{\mu}_{2i+1}$ is determined by averaging the two parents' estimates.

Formally, $*R_1 = *(\boldsymbol{\mu}_{2i})_{+hl} - *(\boldsymbol{\mu}_{2i})$ so $\boldsymbol{\mu}_{2i+1} = \boldsymbol{\mu}_{2i} + *R_1$ gives the left hand parents estimate of where it's child should be. The right hand parents estimate is computed using the same process and the two estimates are averaged together to give the new child position $\boldsymbol{\mu}_{2i+1}$. This offers two exemplar texels that are both appropriate candidates to be the child texel's exemplar archetype. Of the two texels $*(\boldsymbol{\mu}_{2i})_{+hl}$ and $*(\boldsymbol{\mu}_{2i})_{-hl}$, one is randomly chosen and the variance Σ_{2i+1} is updated to be the new exemplar pointers variance.

Pseudo-code 2 formally describes the entire process. For the purposes of simplicity the curve is assumed to be closed.

Algorithm 2 Pseudo code for upscale pass.

for $i = 0 \rightarrow I - 1$ **do**

find exemplar index k and texel index j pointed to by $*S_l[i]$

$S_{l+1}[2i].\mu = S_l[i].\mu + E^k_{l+1}[j].\mu - E^k_l[j].\mu$

$S_{l+1}[2i].\Sigma = E^k_{l+1}[j].\Sigma$

$*S_{l+1}[2i] = E^k_{l+1}[j]$

end for

for $i = 0 \rightarrow I - 1$ **do**

compute $*R_1$ and $*R_2$

$S_{l+1}[2i+1].\mu = (S_{l+1}[i].\mu + *R_1 + S_{l+1}[i+2].\mu + *R_2)/2$

choose $*S_{l+1}[2i+1]$ randomly between $*(S_{l+1}[2i])_{+h_l}$ and $*(S_{l+1}[2(i+1)])_{-h_l}$

$S_{l+1}[2i+1].\Sigma = *(S_{l+1}[2i+1]).\Sigma$

end for

4.4.3 Jitter

Jitter is the mechanism through which variation is introduced to the synthesizing results. This paper experiments with spatial jitter [LH05] and structured jitter [RHDG10]. Spatial jitter operates along the curve and is restricted to some fraction of the length of the curve. Thus for texel $S_l[i]$, $*S_l[i]$ is udpated to some other exemplar texel within distance d along edge links from $*S_l[i]$. For hybridization, jitter is an adaptation of the correction step and will be explained next.

4.4.4 Correction

While jitter adds random new features to the output, it can also mutate the result further and further away from the input when jitter is allowed to run unchecked. Unbridled jitter will often result in the output no longer bearing a resemblance to the input exemplars. Correction is the inverse operation to jitter. Given the synthesis result, dissimilarity is reduced from the input by iteratively updating every local neighbourhood in the result to match some local neighbourhood in the input as best as possible. Each synthesis texel's exemplar pointer is updated to the best matching texel in its $NNCS$.

To begin correction, first gather the $NNCS$ for each synthesis texel $S_l[i]$. The synthesis texel's neighbourhood needs to be compared against each of the neighbourhoods in the $NNCS$ as discussed in section 4.2.5. Once a distance to each of the exemplar neighbourhoods has been calculated, the $NNCS$ is sorted from smallest distance to largest. Recall from the jitter section that structured jitter for hybridization is an adaptation of the correction method. Up until this point both correction and structured jitter have followed the same operations. The discussion now forks into two branches, first explaining how to complete this approach for jitter, and then for correction.

Jitter

Structure preserving jitter [RHDG10] adds variation to the synthesizing result, but does so within a user specified tolerance level. As such jitter will add new features to the output which are still statistically similar to the input within a given threshold. This new type of jitter drastically reduces the frequency of broken structured features during synthesis. Any nearest neighbour exemplar texel can be randomly chosen as the new $*S_l[i]$ if its neighbourhood distance is below the user supplied jitter value. Since the $NNCS$ has been sorted with respect to smallest distances, search the list for the first element that has a distance higher than the jitter value and randomly select any of the previous elements. A single solving pass is run over the entire synthesis level S^l after jitter.

Correction

In order to coerce local statistical similarity between the synthesis output and the exemplars, for each synthesised texel $S_l[i] = (\boldsymbol{\mu}_i^l, \Sigma_i^l)$ the correction approach updates $*S_l[i]$ to the first element in the sorted $NNCS$. This models the synthesis neighbourhood after its most similar exemplar neighbourhood so the solver step can morph the shape to more closely match the input exemplar. By iterating this process over each shape texel in the synthesis level S_l the statistical dissimilarity between the output and the input is reduced, curtailing the damage done by the jitter step. Once $*S_l[i]$ is updated to the best texel in the $NNCS$, the Gaussian shape for $S_l[i]$ is inherited from $*S_l[i]$. Correction is divided into two sub-passes, first correcting all even nodes, then all odd nodes in the polyline. A solving pass is invoked at the end of each correction sub-pass in order to reflect the udpated exemplar links into the polyline shape and the $NNCS$ is re-gathered for each synthesis texel. Two full correction passes are performed for each level.

4.5 Open vs. Closed Curves

Special consideration must be taken when dealing with either closed or open curves. Throughout most of this chapter synthesis has been described independent of curve type. Cases in which neighbours or neighbourhoods are sampled outside the scope of the array have been avoided. For the most part the concepts presented in this chapter are independent of curve type, however each case requires its own special considerations.

Closed Curves

Closed curves are cyclical. The neighbour for one end of the array is the element at the opposite end. Any texel referenced outside the range of the array should simply be modulated by the length of the array. This makes closed curves the simpler of the two cases.

Open Curves

Open curves are more complicated. Because they are not cyclical, they must be tiled spatially so that one end of each polyline segment connects seamlessly to the other end of the next segment. Unfortunately there is no input example information for how this connection should be done or what it should look like. As such, it becomes an issue of finding an ad hoc approach that looks correct. In our case we take the input polyline segment and average all shape texels together. We treat the axis of greatest variance as the axis that is tiled along. Using the variance along this axis we find the standard deviation and place a new polyline segment three standard deviations away from its neighbour.

During analysis the polyline is padded with tiles on either side so that it can be downsampled by one level. This process is iterated in order to build a full exemplar stack. However, the downsampled representation is only found for the

original polyline segment, using the tiled ends for neighbourhood texel sampling only. At each level a new tiling is created using the updated polyline segment.

During synthesis additional texture must be generated in order to produce extra texels at either end of the desired synthesised polyline that can be used as neighbours. Closed curves start with a single shape texel at the coarsest level. Open curves on the other hand start at the coarsest level with several texels tiled along a line, each texel is the coarsest level for a polyline segment. Any level that is corrected (correction operates at levels 2 and higher) requires a padding of 12 extra texels on either side of the polyline. These texels provide an idea of what the polyline would look like outside the range of synthesis. These ends of the texture are destroyed during correction because they do not have appropriate neighbourhoods to sample, thus after a new level is computed, these texels are trimmed off the ends. This is the 1D equivalent to the window trimming approach introduced in Parallel Controllable Texture Synthesis [LH05].

4.6 Symmetric Synthesis

For some examples (e.g. chess and celtic butterflies in Fig. 4.12), an additional scheme was used to encourage bilateral symmetry. Given a polyline stored as shape texels in an array, if the curve is bilaterally symmetric then half of the array will be a mirrored copy of the other half. After synthesising a full level, update the exemplar pointers of one mirrored half of the array with corresponding mirror texels in the other half and run an additional solver pass. This approach is done as a post-process for each synthesis level so that the core synthesis algorithm remains unchanged for symmetric examples.

For simplicity, symmetry is explicitly identified by the user in the exemplars as is the array index which begins one of the mirrored halves of the array. Note that if the polyline is perfectly symmetric then this results in exact symmetry in

the output. If the polyline is approximately symmetric then the results will have approximate symmetry.

4.7 Multi-Layer Synthesis

While the ability to synthesise textures or hybrids for a single open or closed curve allows the generation of many types of vector drawings, it is limited in terms of the visual complexity and the types of drawings that can be produced. Layers have been used by illustrators and cartoonist for years in order to build complex art by combining stacks of simple drawings. Inspired by this approach, we have extended our method for single curves to incorporate a bottom-up hierarchical graph based procedural approach for generating arrangements of single curves into a layered stack where curves at lower layers dictate which curves are synthesised at the higher layers.

Exemplar Graph

Inspired by inverse procedural modeling [BMAK10] and Multiscale Texture Synthesis [HRRG08], we have designed a graph of exemplars (Fig. 4.11) where each node contains all the exemplar information for a single open or closed curve, whether it be a single exemplar for texture synthesis, or multiple exemplars that will be hybridized. During synthesis each visited node becomes a new layer in the output.

The exemplar graph is itself a procedural model and traversed as follows. All exemplar graphs contain an empty root node from which the process starts. Each node (including the root) can have an arbitrary number of children and a value dictating how many of those children should be spawned. The graph is then recursively traversed in depth first order.

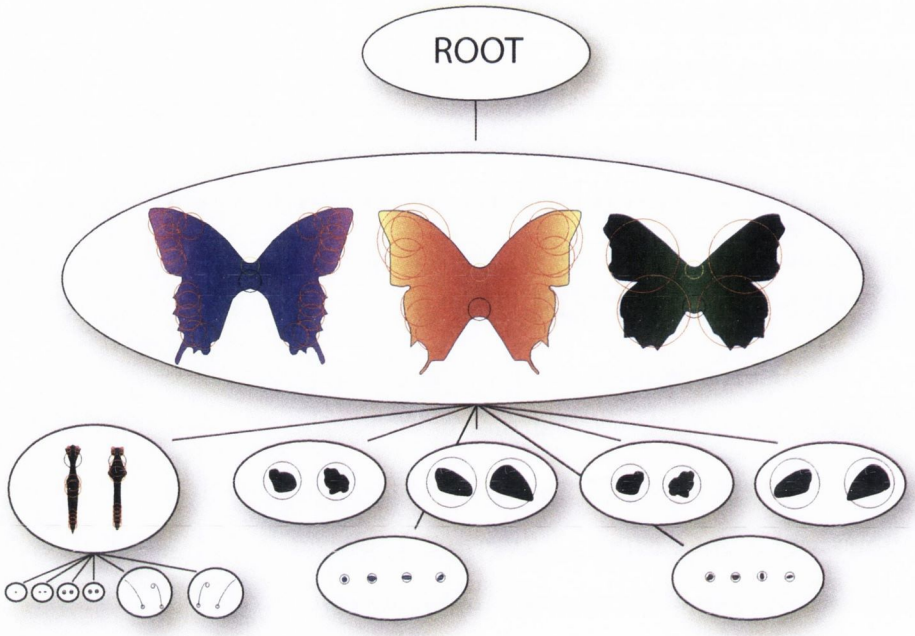


Figure 4.11: Exemplar Graph for teaser image 4.1. *This tree style graph defines a procedural model. Traversal goes from the root node to the leaf nodes in breath first order, each visited node generates a new curve layer in the synthesised output. The coloured circles shown superimposed over the exemplar curves are binding nodes which dictate which child curves should be synthesised in subsequent layers, and where they should be positioned with respect to the parent curve. This information is encoded as the circles colour.*

Binding Nodes

In addition to an array of shape texels defining the polyline, the exemplars can also contain binding nodes, these are tools that are used to link multiple layers together. The user can define binding nodes using circles. A binding node contains a position, radius, ID, number of spawnable children and a marker indicating whether it is an input or output node.

Consequently, after synthesising a parent layer, all the binding nodes are collected from the associated exemplar, and any output node then spawns one or

more children. Each exemplar has at most one input binding node, so a child graph node is found that has an exemplar with a matching input binding ID, thus spawning the next synthesis layer. The position of an output binding node in the parent layer determines the global position where the input binding node of the child layer should be placed. The child layer curve is then placed in relation to its input binding node.

Any shape texel within the radius of a binding node is considered to be a texel of influence on the binding nodes position. If no shape texels are within the radius then the closest shape texel is used. Texels of influence are only sampled from the finest level of the exemplar stack/synthesis pyramid. Thus, given a binding node and array of shape texels from the exemplar polyline, and an array of corresponding shape texels from the synthesis polyline, a new synthesis binding node position is found that mimics the binding-texel example relationship as best as possible. Again the gradient descent solver 4.4.1 can be used to find a solution, finding the position that minimises the overall relationship inconsistency of all texels of influence with relation to the binding node.

4.8 Results

Our results can be organised into three categories, single layer closed curves, single layer open curves, and multi-layer synthesis that combines multiple instances of the previous two categories across layers to build more complex shapes. Real time performance is achieved for most of the examples. It took 0.0278 seconds to generate the synthesis pyramid for one of the chess pieces with eight pyramid levels (128 texels at the finest level). This running time was averaged over 100 iterations of the entire synthesis algorithm. Larger, multi-layered examples also achieved real time performance.

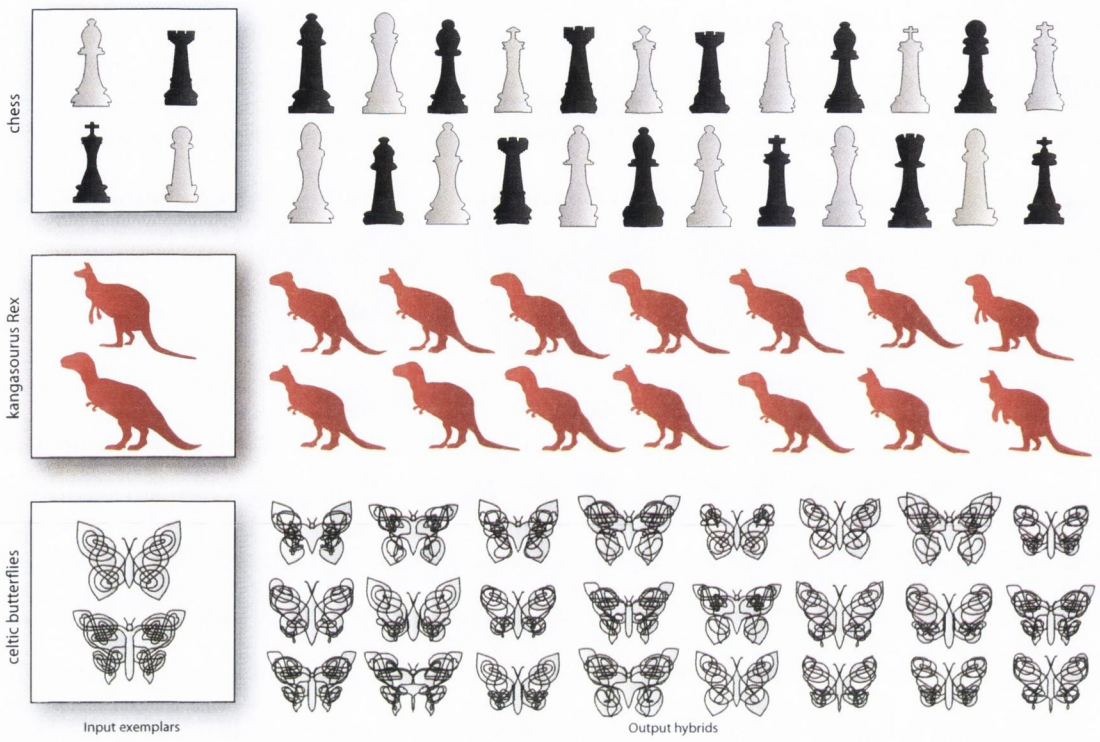


Figure 4.12: Closed Curve Hybrids. *A series of example inputs and the results generated by our method.*

Closed Curves

Synthesising silhouette and curve hybrids is currently an open problem in the vision and graphics fields. Three very different types of inputs have been chosen to illustrate the flexibility and quality with which the algorithm can hybridize closed curves (Fig. 4.12). The first example is a reproduction of the chess pieces presented in the previous Chapter 3. This result is presented first to offer a direct comparison between hybridisation of an image and its curve counterpart. Note that while these examples have been filled in with gradients for rendering (which is common for vector graphics), the underlying geometric representation is a closed polyline. The second example hybridizes a kangaroo with a tyrannosaurus-rex, two very different but anatomically similar shapes. The third and final result of this group shows that the algorithm can handle more complicated self-intersecting

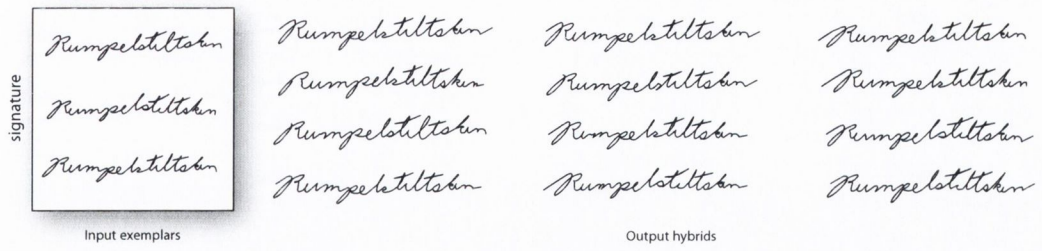


Figure 4.13: Open Curve Hybrids. *Three hand written instances of the signature 'Rumpelstiltskin' have been hybridized to create unique forgeries of the original signatures.*

curves. Note that while lines pass over each other, each butterfly is in fact a single closed curve. Although two shape texels may be very close to one another or even overlap spatially, topologically they can be very distant from each other along the polyline and only influence one another indirectly at the coarser levels. Despite the polyline having a non-visually intuitive connectivity, the overall shape of the butterflies are preserved and interesting new patterns occur.

Open Curves

This method is able to synthesise both hybrids as well as texture from one or more open curves. We chose to use cursive handwriting for both examples for two reasons. First, it's a highly complex and highly structured type of curve that naturally contains variation but also repetition. Second, the human visual system is highly trained for recognising handwriting and thus more able to identify possible artifacts in synthesis than it would for say a spiky curve or simple loops. The first example (Fig. 4.13) shows hybridisation on three open curves of the signature 'Rumpelstiltskin', effectively forging an infinite number of the given signatures. Notice that the personal style of the handwriting is preserved in each hybrid.

The second example (Fig. 4.14) shows that the curve synthesis method is suitable for *texture synthesis* as well as hybridisation. Given a small example curve,

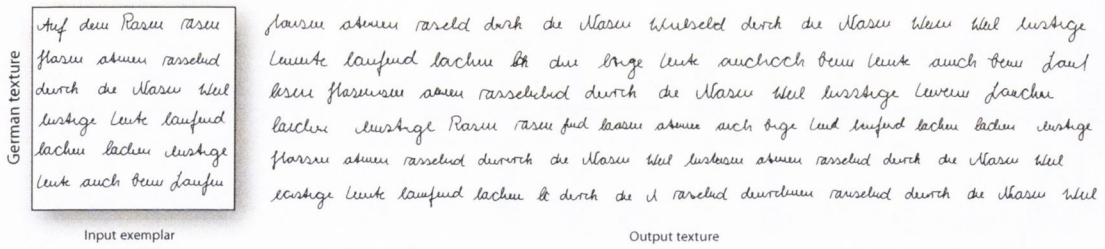


Figure 4.14: Open Curve Texture. *A German tongue twister represented as a single curve of hand written cursive. Segments of the curve between words are coloured white and the curve has been broken up into multiple lines in this figure for display purposes.*

an infinitely long curve sharing the same style and features can be produced. In this example the input is a single curve that communicates two common tongue twisters written in German. We chose a non English language assuming the reader will not understand it, and will judge this example solely on its visual appearance as a writing sample. While this example appears to contain multiple curves (one for each word), it is in fact a single curve. The linkages between words are coloured white in order to appear invisible on the white background. Several small artifacts may occur at the beginning or end of each word as the white linkage line passes over part of a character. Also, for formatting reasons, single curves have been manually split into multiple lines to fit on the page.

Multi-layer

When generating more complicated drawings, a single closed or open curve is rarely sufficient to fully represent the shape. Artists have utilised multiple layers to divide and conquer complicated drawings into an assemblage of simpler ones. Following the same process, we use an exemplar graph to procedurally assemble curve hybrids. Three different multi-layer examples are given in Figure 4.1 (where the corresponding exemplar graph is shown in Figure 4.11), and Figure 4.15. Notice

that the palm tree example consists of three parent/child levels in the exemplar node graph. The root node acts as a foundation and contains a single binding node determining where the palm tree will be positioned. Intuitively, the root spawns the tree trunk. Notice that each trunk has two binding nodes, an input binder at its base that ties into the root, and an output binder at the top that spawns the leaves. The final synthesis texel positions within the binding node radius will determine its final position during synthesis, which determines the position of the leaves. The exemplar graph lends itself naturally to defining flowers, vines, and other things with tree like structure. Due to our approach naturally lending itself to plants, we chose butterflies and pirate flags for our other examples to show the flexibility of the technique to handle a range of different exemplars. These results combine open and closed curves, some of which are bilaterally symmetric and some are not. Note that the approach does not limit the use of filter effects and gradients associated with rendering vector graphics. In this approach the popular .svg vector graphics file format is used for input and output. The filter effects associated with the input curve are preserved throughout the synthesis process, being updated and written to the output file. For fill effects such as gradients that are dependent on global positions, the gradient positions are updated according to the binding nodes, just like texels.

4.9 Discussion

We have presented a method for applying neighbourhood based texture synthesis to the curve domain. A polyline based representation for texture synthesis is introduced, and a geometric alternatives to several key components of the Parallel Controllable Texture Synthesis framework are proposed. This chapter presents a solution to several open problems in the area of geometry texture synthesis. While the results are primarily intended as a proof of concept for these ideas, they do of-

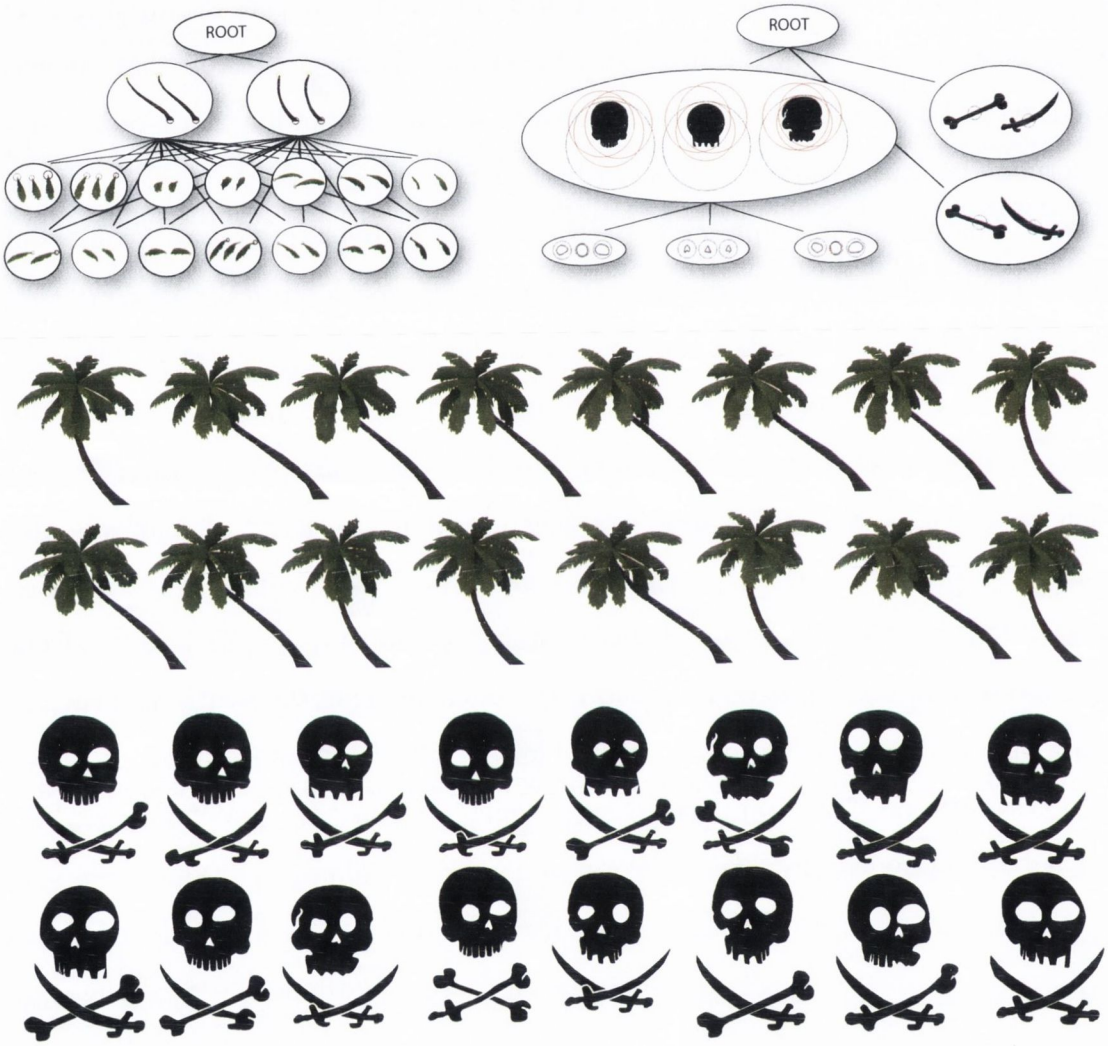


Figure 4.15: Multi-layer. *Exemplar graphs and synthesised results for two examples, palm trees (exemplar graph on top left, results in middle) and pirate flags (exemplar graph on top right, results on bottom).*

fer some limited usefulness to vector artists who need to generate variations simple drawings, clip-art or even wish to generate large populations of cartoon characters. We identify several limitations of the algorithm that weakens its practicality. Areas of future work are identified that can greatly expand the application of this approach.

4.9.1 Limitations

The neighbourhood descriptor, while representing shape, only represents shape that is immediately topologically connected by a curve. Thus, texels can be geodesically very distant from each other but very close in world space without influencing one another. This leads to self intersections when significant jitter is introduced. While in many examples this is not a problem, or in some cases might even be desirable, for many practical uses this is a limitation. Alternative neighbourhood sampling should be explored that gathers neighbourhoods based on world proximity rather than or in addition to topological connectivity.

Exemplar graphs have to be manually defined and binding nodes have to be manually placed within each exemplar. As the fundamental goal of any example based technique is to automate content creation, we consider the reliance on a manually defined procedure to be a limitation.

4.9.2 Future Work

While a broad spectrum of simple results have been presented showing the generality and flexibility of this new approach, several new directions can be explored for improving quality as well as broadening the types and level of sophistication of input shapes.

Improved Solver

For simplicity, we used a gradient descent solver to resolve inconsistencies in the solving pass. This can be improved upon by modifying a more advanced solving technique designed specifically for surfaces, such as PRIMO [BPGK06].

Smooth Curves

Rather than representing vertices connected by edges, our texel nodes could represent control points for smooth curves based on resolution independent mathematical functions [FS94]. While such a method will not increase the feature detail of an example being synthesised, it can replace the need for expensive fine scale feature information on smooth portions of our example.

N-Dimensional Geometry Synthesis

The shape texel, both as a general concept and as the mathematical tool presented, can be naturally extended to the third or higher dimension. This could allow for more arbitrary information in our shape texel, which could be used to store movement over time (animation), positions in 3D space (3D mesh synthesis), or colour/texture information.

Arbitrary Topology

Due to our multi-resolution simplification scheme, our graph topology is limited to continuous curves where each vertex can only define a single point on the curve. This reduces the types of vector graphics we can synthesise. In order to sidestep this problem we introduced exemplar graphs to connect several discrete synthesised curves across multiple layers, allowing us to synthesise tree type graphs. This, however, is a procedural solution that requires the artist to define the graph. This technique could become simpler and more powerful if more types of graph

topologies could be down-sampled.

4.10 Conclusion

In this chapter we introduce curve texture and hybrid synthesis. We propose several novel extensions to the work done in the previous chapter to re-purpose an image based pixel framework to a geometry based shape texel framework. At the core of our new idea is to use Gaussian mixture models as a shape representation. We explain approaches to downsample polylines, gather neighbourhoods and compare them. As we will see in the next chapter, these ideas presented for simple polyline curves can be extended to 3D meshes.

Chapter 5

Synthesising Geometry Textures and Hybrids

5.1 Introduction

Following the natural progression from the last chapter on 2D curves, we now focus on the problem of synthesising 3D surfaces in the form of mesh models. We build on the concepts introduced in Chapter 4 for synthesising geometric shape information as a texture or hybrid. The goal of this chapter is to extend the approach to work with 3D manifolds. Although work has been done on 3D texture synthesis (see Section 2.3.2), the techniques explored have been straight-forward extensions of traditional image synthesis into the third dimension through the use of voxel grids or 3D vector fields. Rather than synthesising the shape directly, these types of solutions rely on discretizing the euclidean space that the shape exists in and synthesising the space itself. This type of solution is wasteful from both a memory and running time standpoint as most of the discretised elements of space are empty and do not contribute to the shape of a surface. It is also an awkward approach as the shape is being indirectly synthesised through an intermediate re-parameterization, which has inherent limitations. In this chapter

we extend the concepts from the previous work on 2D curves so that 3D meshes can be synthesised directly with no need for re-parameterization. We incorporate our hybridisation approach in order to introduce the first solution to the problem of multiresolution mesh hybridisation that we are aware of. Our contributions in this chapter can be broken up into several categories:

- 3D Shape Texels, Description and Comparison
- Neighbourhoods
- Multiresolution Mesh Simplification
- Nearest Neighbour Search
- Synthesis
- Improved Solver
- Non-uniform Multiresolution Synthesis
- Image Geometry

5.2 3D Shape Representation

This section builds on the 2D shape representation presented in the previous Chapter 4.2. While many of the ideas from the last chapter extend conveniently to 3D meshes, others require a fresh look at the approach.

5.2.1 Geometric Input and Output

This work takes as input any 3D mesh of arbitrary genus with the requirement that it be a closed manifold and semi-regular. A 3D mesh is a graph with nodes (called vertices) that define a point in 3D space as well as other information such

as surface normal, colour, texture coordinates, etc. Vertices in the graph are connected together by edges. If a set of three vertices are all connected to each other by edges then they define a triangle. 3D meshes describe the surface of a 3D object through a collection of triangles, where triangles are used as the atomic unit of surface and are combined to create more complex surfaces. A closed manifold is just a 3D mesh where each edge contributes to exactly two triangles. Therefore, a closed manifold defines a smooth continuous surface with no holes or dangling geometry protruding from it. While semi-regular meshes are not strictly required to be closed manifolds, in practice many approaches that use semi-regular meshes, as well as methods for converting an arbitrary topology mesh into a semi-regular equivalent assume they are closed manifold. A semi-regular mesh can be thought of as a mesh of arbitrary non-regular topology (i.e. not all the vertices have valence six) that has been regularly subdivided some number of times as shown in figure 5.1.

As input the method accepts two or more semi-regular closed manifold 3D meshes. These meshes are not required to have the same base mesh topology or even the same genus. The output is always a single semi-regular closed manifold 3D mesh.

5.2.2 3D Shape Texel

Internal to the algorithm, mesh vertices are augmented with shape texel information as discussed in the previous chapter's Section 4.2.2. Using Gaussian distributions to represent discrete units of shape is convenient for our approach because it extends trivially to the third dimension. Each vertex thus has a 3D Gaussian function associated with it. This includes a mean $\boldsymbol{\mu}$ (i.e. (x,y,z) centre position, just like a vertex), and a covariance matrix (3×3) of a Gaussian probability distribution represented by Σ.

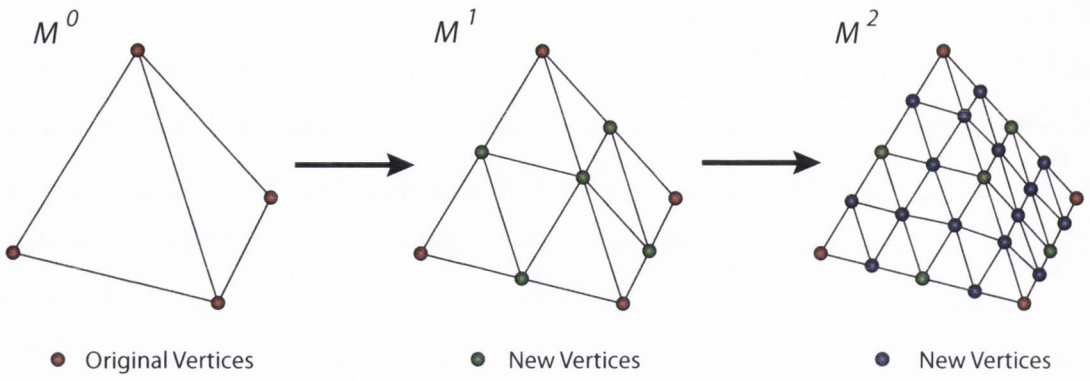


Figure 5.1: Multiresolution Topology. *Starting from this example base mesh M^0, M^1 is created by splitting the edges of of M^0 resulting in new vertices that subdivide each of the previous levels polygons into four new polygons. The highest resolution mesh M^L contains the vertices from M^0 along with the new vertices created through subdivision. This is known as a semi-regular mesh because the vertices in mesh M^L that come from M^0 retain their original valence (this is the irregular coarsest foundation), while all new vertices introduced through subdivision have a valence of six and form a regular triangular tiling.*

5.2.3 Neighbourhoods

A single Gaussian does not contain much information about the shape of a surface. Instead a neighbourhood of coherent vertices are grouped into a Gaussian mixture model. For a given vertex v, its neighbourhood consists of v as well as the vertices that are connected to it by an edge. This is illustrated in Figure 5.2 where we see v highlighted with a red sphere and the neighbours highlighted in other colours. Note that each neighbour vertex is assigned a different colour to illustrate its location in the neighbourhood.

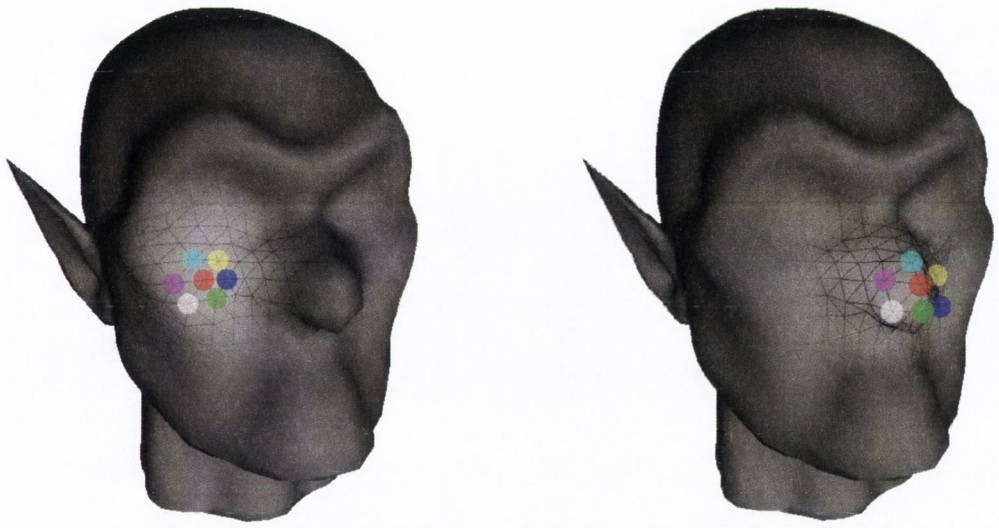


Figure 5.2: Neighbourhood. *The target vertex v is highlighted in red while each of the neighbours are highlighted in another colour.*

5.2.4 Averaging Neighbourhoods

Texel Computation at Coarser Scale.

Considering a vertex's ring of neighbours $\{\boldsymbol{\mu}^l_{v_s}, \Sigma^l_{v_s}\}_{s=0,\cdots,S}$ centred around vertex $(\boldsymbol{\mu}^l_v, \Sigma^l_v)$, and connected by an edge. The probability density function $\mathbf{x}$ of the entire neighbourhood consisting of the union of vertex v and its neighbours is modeled as a mixture of normal distributions:

$$p^l_v(\mathbf{x}) = \omega_v \times \mathcal{N}(\boldsymbol{\mu}^l_v, \Sigma^l_v) \cup \sum_{s=0}^{S} \omega_s \times \mathcal{N}(\boldsymbol{\mu}^l_{v_s}, \Sigma^l_{v_s})$$

where the weights ω_v and $\{\omega_s\}_{s=0,\cdots,S}$ control the importance of each mixture component. These weights are set as $\omega_v = 2/(S+2)$, and $\omega_s = 1/(S+2)$. Being modeled from a Gaussian distribution these weights sum to approximately 1.0.

At a coarser level, v^l is approximated by a simple normal distribution $\tilde{v}^l \equiv \mathcal{N}(\boldsymbol{\mu}^{l-1}_v, \Sigma^{l-1}_v)$ such that

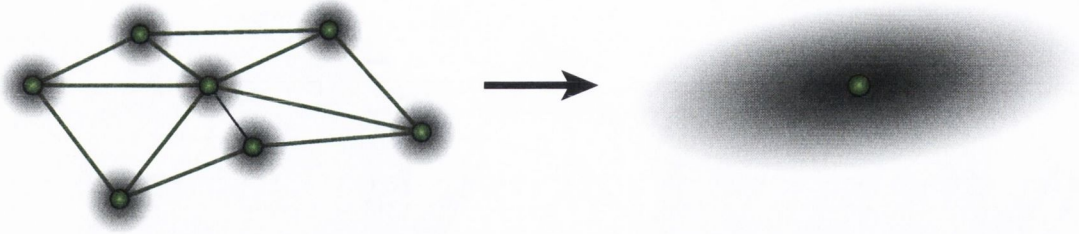


Figure 5.3: Merging Texels. *Left: Texel* $(\boldsymbol{\mu}_v^L, \Sigma_v^L)$ *has six neighbours that are combined in the mixture* $p_v^{(L)}(\mathbf{x})$. *Right: all seven texels are merged into one texel* $(\boldsymbol{\mu}_v^{L-1}, \Sigma_v^{L-1})$ *at the next level simplifying the local pdf to* $\tilde{p}_v^L(\mathbf{x})$.

$$\begin{cases} \boldsymbol{\mu}_v^{l-1} = \int \mathbf{x}\ p_v^l(\mathbf{x})\ d\mathbf{x} \\ \Sigma_{v,xyz}^{l-1} = \int (x-\mu_x)(y-\mu_y)(z-\mu_z)\ p_v^l(\mathbf{x})\ d\mathbf{x} \end{cases}$$

giving

$$\begin{cases} \boldsymbol{\mu}_v^{l-1} = \omega_v \boldsymbol{\mu}_v + \sum_{s=0}^{S} \omega_s\ \boldsymbol{\mu}_{v_s}^l \\ \\ \Sigma_{v,xyz}^{l-1} = \omega_v \left[\Sigma_{v,xyz}^l\right] + \sum_{s=0}^{S} \omega_s \left[\Sigma_{s,xyz}^l + (\mu_{v_s,x}^l - \mu_{v,x}^{l-1})(\mu_{v_s,y}^l - \mu_{v,y}^{l-1})(\mu_{v_s,z}^l - \mu_{v,z}^{l-1})\right] \end{cases}$$

with $(\mu_{v,x}^l, \mu_{v,y}^l, \mu_{v,z}^l)$ indicating the three coordinates of vector $\boldsymbol{\mu}_v^l$ and $\Sigma_{v,xyz}^l$ being the associated covariance. $(\boldsymbol{\mu}_v^{l-1}, \Sigma_v^{l-1})$ defines a coarser texel at the level $l-1$. As an illustration, figure 5.3 shows $p_v^L(\mathbf{x})$ and the corresponding $\tilde{p}_v^L(\mathbf{x})$.

5.2.5 Comparing Neighbourhoods

Comparing Two Gaussian PDFs

This chapter follows the previous in terms of the approach taken to compare two Gaussian PDFs, as can be reviewed in section 4.2.5. To summarize the approach,

the closed form solution for KL distance is defined as

$$M(A,B) = (\boldsymbol{\mu}_A - \boldsymbol{\mu}_B)^T \Sigma_B^{-} 1 (\boldsymbol{\mu}_A - \boldsymbol{\mu}_B). \tag{5.1}$$

$$KL(A,B) = \frac{1}{2}\{\log \frac{|\Sigma_B|}{|\Sigma_A|} + (Tr(\Sigma_B^{-} 1 \Sigma_A) - 3) + M(A,B)\}. \tag{5.2}$$

The KL distance 5.2 is comprised of the Mahalanobis distance 5.1 along with terms that more accurately model the distance between PDF variance. As such, the Mahalanobis distance can be used as a faster albeit less accurate alternative to the full KL distance. The KL distance measures relative information entropy between PDFs and as such is not symmetric and thus not a true metric. It can be easily made symmetric by switching A and B and averaging the two results together. Note that equation 5.1 differs slightly from the one introduced in the previous chapter. Because μ and Σ are now three dimensional values rather than two, we subtract a constant equal to the new dimensionality from the trace term.

Comparing Gaussian Mixture Models

While there is no closed form solution for comparing two Gaussian mixture models using the KL divergence measure, this is not a setback for our approach as the neighbourhoods themselves contain correspondence information between pairs of Gaussian PDFs. That is, much like the corresponding pixels in two 5×5 neighbourhoods are compared with each other independent of the other 24 pixels in the neighbourhood, two individual PDFs in our Gaussian mixture Model neighbourhood are compared with each other directly independent of the rest of the PDFs in the mixture. This reduces the comparison problem from a full comparison of two Gaussian Mixture Models down to a summation of independent comparisons between pairs of Gaussian distributions, for which the above closed form solution is appropriate.

In previous image based approaches there has always been a clear notion of neighbour correspondence. When comparing two 5×5 neighbourhoods the two pixels in the corresponding element of each array are compared. When dealing with geometric neighbourhoods there is no direct correspondence between the vertices. Images have an intrinsic concept of direction, i.e., top, bottom, left, right. This problem was introduced in the last chapter on polyline synthesis, however for 1D functions there are only two possible neighbourhood orientations to choose from. By extending geometry to the third dimension, neighbourhood orientation ceases to be a trivial problem.

No matter how many neighbour vertices are connected to the target vertex, combined they topologically form a ring around the target vertex. While there is no intrinsic direction for neighbourhoods with respect to the surface of the model, relative directions between the neighbourhoods themselves can be determined. In our case we have chosen to give the neighbours clockwise order around their centre vertex with respect to the surface of the manifold. This creates the restriction that two neighbourhoods can only be compared with each other if they have the same number of neighbours. A vertex can only look for nearest neighbours among vertices with the same degree.

This leaves one unknown: the best orientation between the two neighbourhoods A and B. Although neighbours are ordered in a clockwise ring around their centre vertex, there is no clear beginning or end of the ring. The "first" vertex in the ring is chosen arbitrarily. Thus when comparing neighbourhood B to neighbourhood A, every possible rotation of B must be compared with A. If the two neighbour lists have S elements then the best orientation of B that minimises its distance from A is found. This gives a minimum error value and an array offset i as shown by

$$distance = \min i\{\Sigma_{i=0}^{S}\{\Sigma_{s=0}^{S} KL(A[j]B[(i+s) \mod S])\}\}.$$

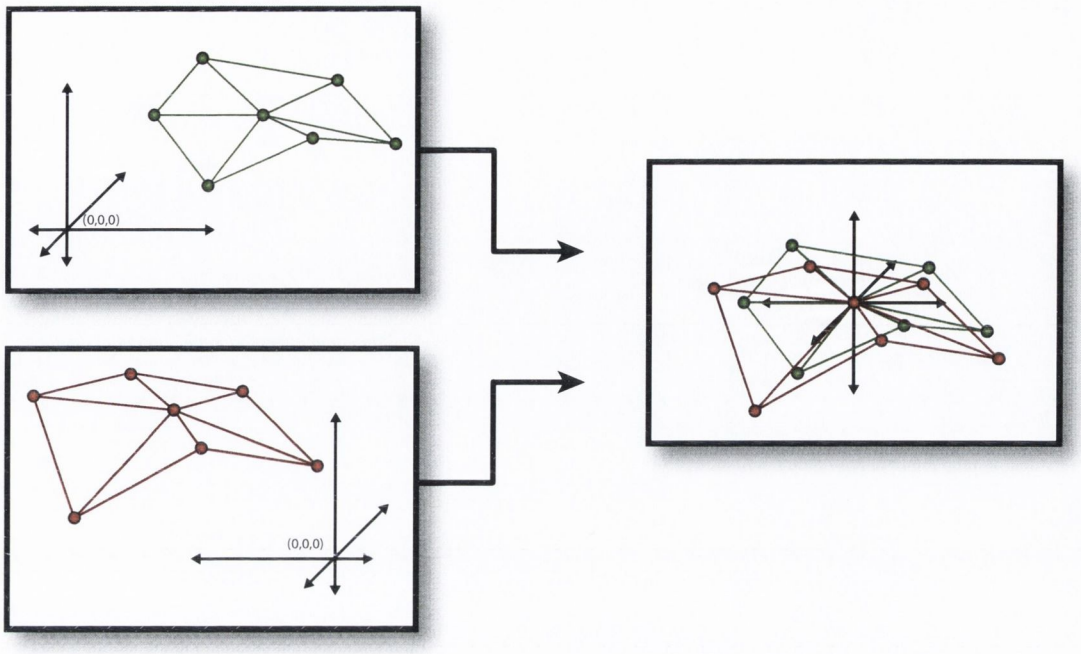


Figure 5.4: Centering Neighbourhoods *The neighbourhood distance measured shape dissimilarity between two neighbourhoods. As such both neighbourhoods must first be centred to the origin by removing the mean $\boldsymbol{\mu}$ of the centre vertex from each of the neighbourhood vertces (left). Once both neighbourhoods have been centred around the origin (right), they can be compared.*

Because this operation tests the difference in neighbourhood shape, the difference between centre neighbourhood positions must be removed as it will contribute to the mean distance in the KL metric and overpower the actual distance between neighbourhood shapes. This can be easily done by treating the mean $\boldsymbol{\mu}$ of the centre vertex as the origin of the entire neighbourhoods local coordinate frame. This is illustrated in figure 5.4 where each neighbourhood is translated so that it is centred around the origin.

5.3 Analysis

Before synthesis can be performed, the exemplar data must first be analysed and a Gaussian pyramid is constructed for each of the exemplar inputs. Once the multiresolution representation is created, each vertex constructs a nearest neighbourhood list to the most similar shaped vertices in the other exemplars. Both of these operations are done in an off-line analysis phase that is only performed once, the results are stored in order to improve synthesis speed.

5.3.1 Multiresolution

In the previous literature as well as the previous chapters of this thesis, the input exemplars are converted into Gaussian stacks [LH05]. Unfortunately, the creation of a stack requires that the input be completely regular, while there are approaches for mapping a genus zero mesh onto a regular plane, the result is not appropriate for neighbourhood based texture synthesis because it creates a severely anisotropic remeshing that is not deterministic to the shape of the input mesh. In other words, two identical local regions in otherwise different meshes will produce severely different remeshed geometry and the local neighbourhoods will not correspond based on the KL shape distance. As such the Gaussian stack is abandoned and replaced by the earlier Gaussian pyramid [WL02]. The inputs are meshes that are both closed manifold and semi-regular 5.2.1. The reasoning behind this becomes clear when constructing the Gaussian pyramid. In the image framework, a Gaussian pyramid is constructed by taking 2×2 neighbourhoods of pixels and averaging them together into a single new pixel at the coarser scale. Thus each coarser level of the Gaussian pyramid has been downsampled to a quarter of the resolution of the previous level.

A semi-regular mesh shares the same multiresolution properties as the image pyramid where groups of four polygons are combined into one polygon at the

coarser scale as was shown in figure 5.1. Thus each coarser level is downsampled to a quarter of the resolution of its previous level.

Semi-regular meshes are beneficial for compression, storage, transmission, level-of-detail control and rendering. Because of their widespread usefulness, the topic of remeshing an irregular mesh into a semi-regular form has been studied and several approaches exist. While semi-regular remeshing is a separate problem from the one presented in this chapter, it is of foundational importance as the resulting semi-regular mesh greatly influences the quality of synthesis. In this project we have implemented a simple but effective remeshing scheme that we think is very appropriate for neighbourhood based synthesis. As this is not a central part of the mesh synthesis problem and we do not claim our semi-regular remesher as a contribution of this thesis, we discuss the details in the remeshing section C.1 we have included as an appendix.

Once the inputs have been remeshed into a semi-regular form, all that is left to complete the Gaussian pyramid is to find the correct Gaussian distribution function for each shape texel. The finest level of the pyramid is initialised with Gaussian functions of spherical variance where the mean value is set to the vertex's position ($\mu = x, y, z$, $\Sigma = I$). The coarser levels are found by iteratively moving down the pyramid. Note that a semi-regular mesh at a finer level contains all the vertices of the next coarser level in addition to all the new vertices created by splitting the edges from the coarser level. When finding the new Gaussian function for a vertex at the coarser level, the same vertex is taken from the finer level and its neighbourhood is gathered using the approach presented in the neighbourhoods Section 5.2.3. This neighbourhood is then averaged together using the approach discussed in the Averaging Neighbourhoods Section 5.2.4, the resulting Gaussian is assigned to the coarser level shape texel.

Figure 5.5 shows two Gaussian pyramids with the same base mesh topology where the mesh topology was created using our semi-regular remeshing approach

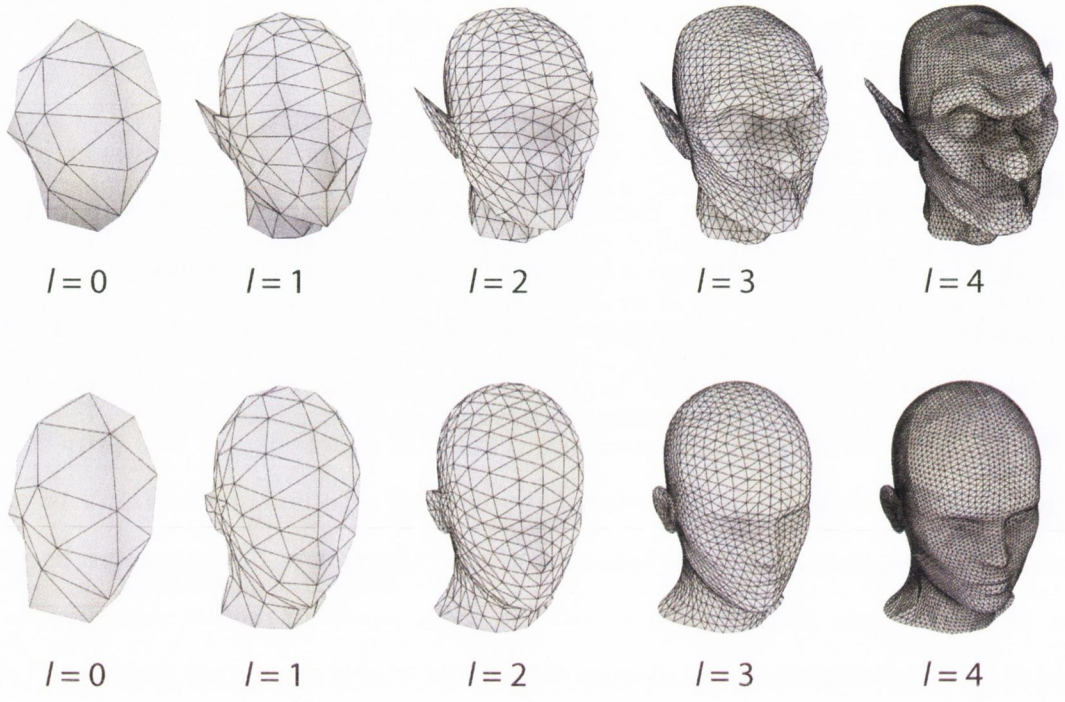


Figure 5.5: Gaussian Pyramids *Two Gaussian pyramids created using our remeshing approach and multiresolution neighbourhood averaging. The top pyramid is an ogre's head while the bottom image is human.*

and the vertex positions are the means of the Gaussian functions assigned to each vertex.

5.3.2 Nearest Neighbour Search

Continuing the trend of nearest neighbour candidate set gathering. Each vertex in each of the exemplars maintains a list of its most similar vertices in the other exemplars. A brute force search is performed with a simple but very effective time saving augmentation. The normal of each vertex represents the average face orientation for all the faces in that vertex's neighbourhood. Before doing an expensive comparison between two neighbourhoods, the dot product is computed for the two normals and the neighbourhood comparison is skipped if the result is negative -

i.e. the two average neighbourhood orientations are more than 90 degrees apart from each other.

We continue using the multiscale neighbourhood approach introduced in Section 3.3.2 for the results in this chapter. However when comparing nearest neighbours we found that the need for multiscale neighbourhoods is greatly diminished by the replacement of stacks with pyramids. This makes sense intuitively. In the case of stacks, at the coarser levels there are many coherent regions of very similar neighbourhoods due to blurring. The philosophy behind multiscale neighbourhoods is that at the coarser levels coherent unique structured features at the finer scales are blurred into nearly the same features. Because the choices made at the coarse scale effects what is produced at the fine scales, this can mess things up down the road. It is important to look forward during synthesis when there are very similar coarse features that can lead to very dissimilar fine features.

In the case of a multiresolution pyramid, all the coherent regions of similar neighbourhoods at the coarse scales of the stack are downsampled into only one single neighbourhood to sample from. This simplifies the problem making the need for multiscale neighbourhoods somewhat moot as each coarse level neighbourhood should now be very different from the others, not requiring the extra fine level information in order to distinguish similarity across sales. As such the multiscale neighbourhood is only really necessary for a Gaussian stack based approach as used in Chapters 3 and 4. We chose to continue using multiscale neighbourhoods however, not because we believe it is necessary in order to produce useful results, but because it can only help and not harm. As the multiscale neighbourhood search is done as precomputation, we decided that performance was not a justification against using multiscale neighbourhoods.

5.4 Synthesis

The general synthesis framework for 3D meshes is unchanged from the previous Chapter 4. Synthesis operates in a coarse-to-fine manner, building a multiresolution pyramid of discrete levels. Each level other than the first (which is copied directly from the input for bootstrapping purposes), is synthesised from the previous level using three sequential operations, upscale, jitter and correction with a fourth solver step being run after each. The contributions in this section relate to the specific challenges introduced for each of the four synthesis steps when increasing the dimensionality to 3D meshes.

Synthesis vertices are similar to exemplar vertices but with one key difference. A synthesis vertex links to one specific exemplar vertex (selected using nearest neighbour) and this link is noted using $*$. It is the goal of the synthesis vertex to imitate this exemplar vertex as closely as possible. In general, the process of hybridisation at any given level can be thought of as follows.

Starting from the base mesh, detail is then added to the synthesising model through an iterative refinement process discussed in Chapter 4 Section 4.4. While Chapter 4 deals with the case of 2D curves and not 3D meshes, and differs in terms of multiresolution strategy, the overall refinement approach follows directly from this earlier work and is best summarized by figure 4.7. Because the new solver step piggybacks on each of the original three steps, it will be discussed first. Following the explanation of the solver, we will discuss how upscale, jitter and correction are modified to work with 3D meshes.

5.4.1 Solving

In this section the solver step introduced for polylines 4.4.1 is further developed, making it appropriate for the synthesis of 3D meshes. During synthesis the absolute positions of exemplar vertices have no direct bearing on the absolute positions

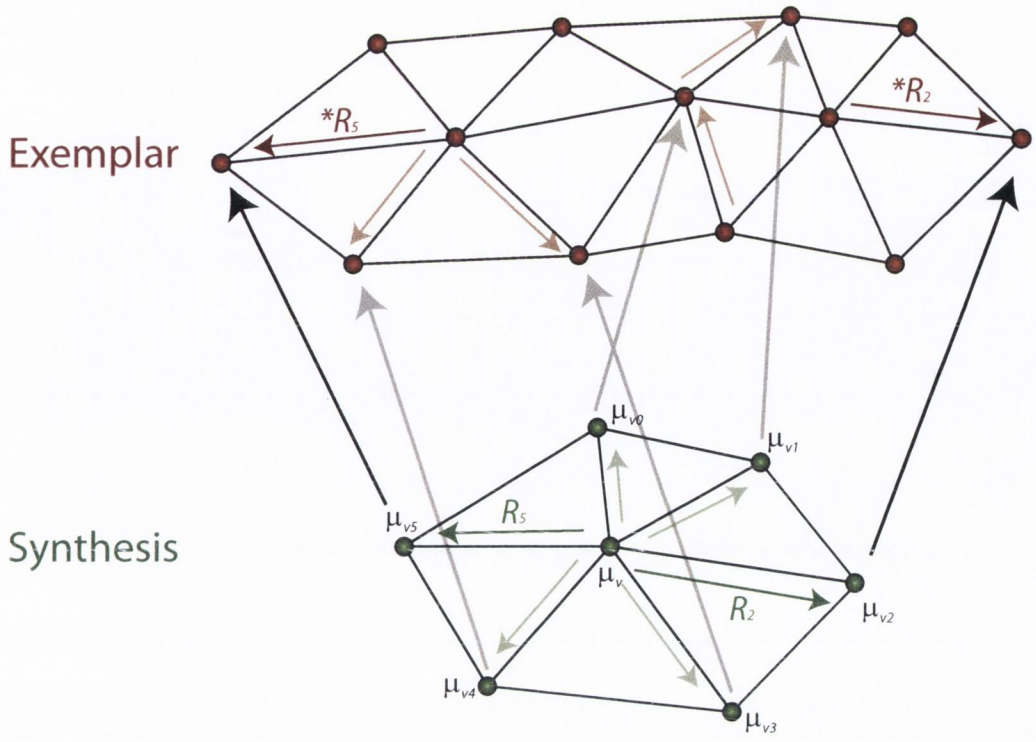


Figure 5.6: Solver. *Given a synthesis vertex $S_l[v]$ (green) and corresponding exemplar E_l (red), we see how our solving algorithm computes synthesis and exemplar relationships. Notice that we show Relationships R_5 and R_2 with the rest faded out as they are algorithmically redundant and visually clutter-up the illustration.*

of synthesis vertices. Instead the position of a vertex is determined by the spatial relationship to its ring of neighbours. Ideally, the spatial relationship between a synthesis vertex and each of its neighbours should reflect the spatial relationship of its exemplar pointer and that vertex's neighbours as closely as possible. Because neighbouring synthesis vertices are going to have non-neighbouring exemplar pointers, in order for each synthesis neighbourhood to mimic its exemplar counterpart as closely as possible, compromises on each edge relationship will have to be made between the two conflicting examples. This is the objective of the solving step and the reason why it is crucial in the synthesis process.

For each vertex v in the current synthesis level S_l a gradient vector $g[v]$ is computed and used to displace the vertex's position by a small increment. Because the vertices are exhibiting unequal push and pull on each other, it is necessary to find the gradient of motion that will minimise the total push/pull energy of the system. This is an iterative optimisation process where first a new gradient is computed for each vertex, then vertex positions are updated along their gradients by a small increment and a new energy value is calculated for the system. This process repeats until the difference between the previous iteration's global energy and the current global energy is below a user specified accuracy threshold. Specifically, for vertex $S_l[v] = (\mu_v, \Sigma_v)$ (ignoring l in the notation) we iterate through all of its ring neighbour relationships as shown in figure 5.6. Relationship R_s is defined as $R_s = \boldsymbol{\mu}_{vs} - \boldsymbol{\mu}_v$ and the exemplar counterpart $*R_s$ is likewise determined as $*R_s = *\boldsymbol{\mu}_{vs} - *(\boldsymbol{\mu}_{vs})_v$ where the term $*(\boldsymbol{\mu}_{vs})_v$ is described in paragraph 5.4.1.

By subtracting the exemplar relationship from the corresponding synthesis relationship, we get the relative relationship. In the previous chapter we used this vector as both the direction and magnitude of force applied to our vertices in order to minimise difference in edge shape between the synthesised output and the exemplar input.

When dealing with 3D shapes we discovered that this approach was too crude to preserve smooth surfaces when edges in a neighbourhood do not have roughly the same length. Very small changes in shape can lead to large changes in surface smoothness which is very noticeable due to drastic changes in lighting at those points which really emphasizes the otherwise subtle error.

The solver step does not take into account differences in edge length when deciding a new position that will minimise edge inconsistency. As a consequence, longer edges have more voting power than shorter edges when deciding where their communal vertex should be moved to. The core goal of the solver step is to reduce the difference in edge shape between the synthesis result and the example. Edge

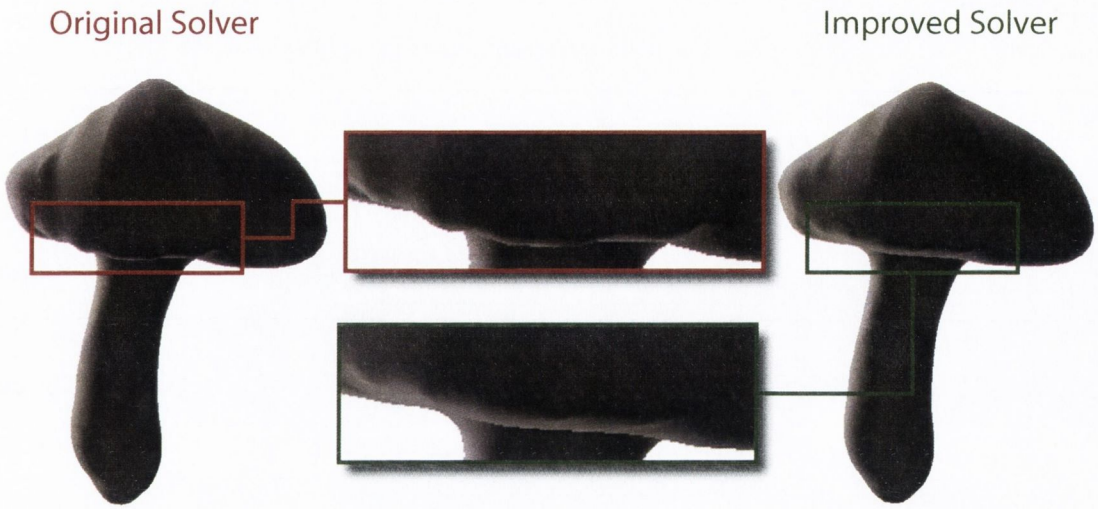


Figure 5.7: Original vs. Improved Solver. *The difference in solver performance is shown through a side by side comparison. Two mushrooms have been synthesised with correction turned off in order to maximize randomness and the same random number seed value is used to jitter both results. Mushrooms were chosen for this example because the exemplars are very smooth with no creases or jagged edges. Therefore, solver quality can be judged by the smoothness seen in the final result. While both solvers produce the same mushroom shape in general, the improved solver reduces the global error in edge shape resulting in a smoother surface with fewer and less severe artifacts.*

shape is determined by edge length and edge direction. Thus our improved solver minimises the difference in these values between the exemplar and result.

The improved solver still uses the difference in synthesis and exemplar relationship to determine the direction of motion. However, in our improved approach this vector is normalised and scaled according to the difference in angle and length between the synthesis and exemplar relationships. In order to calculate this scaling factor, push power and angular power are calculated as follows

$$push[s] = \frac{||*R_s|| - (\frac{R_s}{||*R_s||} \cdot \frac{*R_s}{||*R_s||}) \times ||*R_s||}{||*R_s||},$$

$$angular[s] = |\frac{\arccos(\hat{R}_s \cdot *\hat{R}_s)}{\pi}|.$$

These two power values dictate how much force each edge exerts on its vertices. The push value is determined by projecting the synthesis edge onto the exemplar edge and finding the relative change in edge length with the exemplar. Imagine that the edges are springs, if the projected synthesis edge is the same length as the exemplar edge, then push contributes 0 units of force. If the projected synthesis edge is twice the length as the exemplar edge, then push exerts 1 unit of force. If the projected synthesis edge has no length then it also projects 1 unit of force. This scales the force this synthesis edge exerts on its vertex by its length relative to its ideal exemplar length which it attempts to mimic. Likewise, the angular force is simply the angular difference between the synthesis and exemplar edges scaled so that an angle difference of 0 contributes 0 units of force and an angle difference of 90 degrees contributes 1 unit of force. Both forces scale linearly as length and angle changes. The gradient vector is the summation of all relative relationships for a vertex v normalised and scaled by the push and angular forces

$$g[v] = \sum_{s=0}^{S} \widehat{R_s - *R_s} \times (push[s] \times (1 - ratio) + angular[s] \times (ratio)).$$

The ratio parameter interpolates the weight of importance between correct edge lengths vs. correct edge angles. We used ratio = 0.75 as we found that favouring angle over distance produced smoother results. A side by side comparison of the original solver vs. the improved solver is given in figure 5.7. In this example correction was turned off to maximize randomness and the same seed value was used for jittering both results. The new solver does not drastically change the resulting shape, rather it produces new neighbourhoods that more closely resemble their exemplar counterparts. The mushrooms were chosen in this example because they are a very smooth surface thus solver quality can be judged by the degree of smoothness in the result.

We update each texel position by offsetting its current position $\boldsymbol{\mu}_v$ by gradient $g[v]$ multiplied by some user specified scaling coefficient α as illustrated by the following pseudo code.

Algorithm 3 Pseudo code for improved solver pass.

repeat

array g initialised to 0

for $v = 0 \rightarrow V - 1$ **do**

for $s = 0 \rightarrow S - 1$ **do**

Compute R_s and $*R_s$

$angle = acos(dot(normalize(*R_s), normalize(R_s)))/pi$

$projDistance = dot(*R_s \times (1.0/*R_s.distance), R_s \times (1.0/*R_s.distance) \times *R_s.distance)$

$push = abs((*R_s.distance - projDistance)/*R_s.distance)$

$R = normalize(R_s - *R_s)$

$R = R \times (angle \times ratio + push \times (1 - ratio))$

$g[v_s] += R$

$g[v] += -R$

end for

end for

$\mu_v = \mu_v + \alpha\ g[v]$

until $e = \sum_{v=0}^{V-1} |g[v]|^2$ below a given threshold

Finding $*(\boldsymbol{\mu}_{vs})_v$

Finding $*(\boldsymbol{\mu}_{vs})_v$ is an important operation that is used in all four synthesis steps. Recall that each vertex has the same degree as any exemplar vertex it links to. Also recall that each vertex contains a clockwise ordered list of pointers to its ring of neighbours and an offset value i for each exemplar link that brings the two lists into directional alignment. If vertex v_s and $*v_s$ have similar neighbourhoods, then it follows from the coherence property [Ash01; BSFG09] that their neighbours $(v_s)_s$

and $*(v_s)_s$ would also be similar. We use the term $(\boldsymbol{\mu}_{vs})_v$ to refer to vertex v as a neighbour of v_s where the v is a notational placeholder for whatever neighbour index points back to our vertex v. $*(\boldsymbol{\mu}_{vs})_v$ is the exemplar equivalent to this back-link and is found by the synthesis back-link index added to i.

This process is illustrated in figure 5.6. While the mathematical notation is fairly convoluted, the concept is simple. From some vertex v we move to its neighbour, move from the neighbour to the neighbours exemplar link, then follow the exemplar equivalent of the back link to an exemplar vertex similar to v that is not actually pointed to by $*v$. This is necessary when we are creating v in the upscaling process as no $*v$ exists for us to use as a blueprint. This operation is also necessary for jitter and correction in order to find more potential nearest neighbours for v to compare against [LH05; RHDG10].

5.4.2 Upscaling

From any level in synthesis, upscaling is the mechanism through which higher resolution details are added to the model. Each upscaling iteration creates a new level in our synthesis pyramid which consists of the union of the vertices in the last level (known as parent vertices) with the new vertices created by splitting edges (known as child vertices). Upscaling involves the creation of the next synthesis level of vertices (as well as their edges), and finding new positions for each of the vertices. We begin this process by upsampling the parent vertices. We then use the parent vertices as reference points to find the children vertices.

Starting from some pyramid level l of synthesis vertices S_l where $*S_l[v]$ dereferences the corresponding exemplar vertex we are modeling a synthesised vertex after. Note that $(*\boldsymbol{\mu}_v^l, *\Sigma_v^l)$ is the long hand notation for our exemplar vertex $*S_l[v]$. For each parent vertex $S_l[v]$, a copy is created at the next finer level in our synthesis pyramid as follows

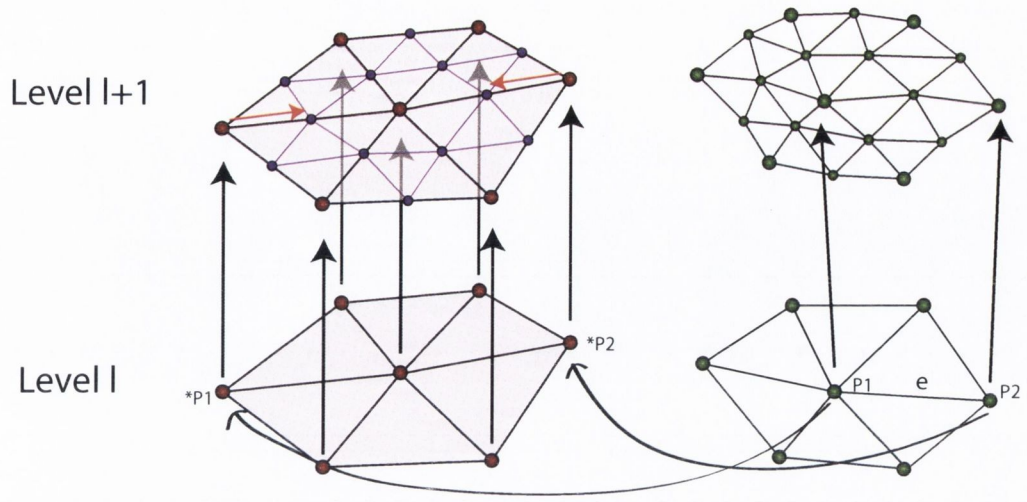


Figure 5.8: Upscaling.

$$S_{l+1}[v] = (\boldsymbol{\mu}_v^l + (*\boldsymbol{\mu}_v^{l+1} - *\boldsymbol{\mu}_v^l), *\Sigma_v^{l+1})$$

where $*\boldsymbol{\mu}_v^{l+1}$ is the finer level copy of $*\boldsymbol{\mu}_v^l$. This propagates the current levels vertices into the next finer level. The remainder of our finer level vertices are found by splitting the edges between vertices at the current level.

Thus, given an edge e consisting of two parent vertices $P1$ and $P2$, we find the child vertex c resulting from splitting the edge. Because c is being created, it has no $*c$ to use as an example of what the new vertex's neighbourhood should look like, and thus where its position should be. We use the same process as the solver (paragraph 5.4.1) to find a nearest neighbour exemplar link. Having created a new vertex c, the first step is to update the neighbour list for $P1$ so that $P2$ is replaced with c and likewise for $P2$'s neighbour list. Now when we find $*(\boldsymbol{\mu}_{cs})_c$ where c is our vertex and s is one of our two parents, we are given two equally valid choices for $*c$ as well as a relative position for c with respect to each parent that we can average together for a final position. This process is illustrated in figure 5.8. Note our child vertices are connected so that each polygon is split into four new ones.

We have included pseudo-code formally describing the entire process.

Algorithm 4 Pseudo code for upscale pass.

given current level S_l which contains vertex and edge lists

for $v = 0 \rightarrow V - 1$ **do**

$S_{l+1}[v].\mu = S_l[v].\mu + *S_{l+1}[v].\mu - *S_l[v].\mu$

$S_{l+1}[v].\Sigma = *S_{l+1}[v].Sigma$

end for

for $e = 0 \rightarrow E - 1$ **do**

compute $*R_1$ and $*R_2$

$S_{l+1}[V + e].\mu = (S_{l+1}[P1].\mu + *R_1 + S_{l+1}[P2].\mu + *R_2)/2$

choose $*S_{l+1}[V + e]$ randomly between $*(S_{l+1}[P1])_c$ and $*(S_{l+1}[P2])_c$

$S_{l+1}[V + e].\Sigma = *(S_{l+1}[V + e]).\Sigma$

end for

5.4.3 Jitter/Correction

Jitter is the mechanism through which randomness is added to the synthesis process so that unique results can be produced. Jitter semi-randomly recombines features from the different inputs into a single new output. Unfortunately this process can be dangerous and lead to outputs comprising features that are no longer statistically similar to those of the inputs. Correction is the inverse problem; given a randomised synthesis result, it reduces the overall dissimilarity between the output and the inputs.

Much like the previous subsection on upscaling, this subsection extends the correction/jitter approach from the previous chapter to work for 3D meshes. Following the structure preserving jitter and correction strategies used in the previous texture/hybrids literature [LH05; RHDG10], for each vertex $S^l[v]$ a list of nearest neighbour exemplar candidates NNL is gathered. This list consists of the exemplar vertex $*S^l[v]$ along with all of the similar exemplar vertices $*(\boldsymbol{\mu}_{vs})_v$ as v for

each of the S neighbours as described in paragraph 5.4.1. NNL consists of this list of vertices in conjunction with their pre-computed nearest neighbour links.

The neighbourhood n_v is gathered for synthesis vertex $S^l[v]$ as described in Section 5.2.3 and compared against each of the neighbourhoods gathered for the nearest neighbour vertices in NNL. Each comparison yields a distance value between neighbourhoods that is used to sort the NNL from smallest to largest distance. At this point the jitter and correction step take different directions in how they make use of the sorted list NNL.

Jitter

While the goal of structure preserving jitter is to introduce randomness, it does so in a semi-random manner in which local feature shapes (i.e. neighbourhoods) can only randomize within a given threshold. In practical terms this means that any nearest neighbour exemplar vertex can be randomly chosen to be the new $*S^l[v]$ if its neighbourhood distance compared against n_v is below the user supplied jitter value. Since the nearest neighbour list NNL has been sorted with respect to smallest distances, the list is searched for the first element that has a distance higher than the jitter value and one of the previous elements in the array is selected randomly. A single solving pass is run over the entire synthesis level S^l after jitter.

Correction

The correction scheme is based on coherence [Ash01], which states that nearest neighbour selection is not always the best method for finding perceptual similarity. In fact, coherence, or the relationship between neighbouring pixels, plays a large role in structure and perceptual similarity. As such the distance values for nearest neighbours are scaled by a bias factor if they cause the synthesis vertex to jump its exemplar pointer to a different exemplar. This favors coherent patch formation.

The correction approach [LH05] updates $*S^l[v]$ to the first element in NNL.

This forces the synthesis neighbourhood n_v to conform its shape to its most similar exemplar neighbourhood. By iterating this process over every vertex in synthesis level S^l, the global statistical similarity between the output and the input is improved.

Rather than correcting every vertex in a single pass, we follow the previous literature and divide our correction into multiple (four) sub-passes, performing a solving pass at the end of each sub-pass. This is complicated by the fact that the data is no longer completely regular. For polylines, two sub-passes corrected first the even vertices and then the odd ones in the list. Images follow the same strategy but with four sub-passes necessary with the extra dimension. We no longer have a notion of even/odd ordering. Similar to the strategy introduced by Wei and Levoy [WL01], we choose the vertices for each of the four subpasses by randomly re-ordering the list of vertices for a given pyramid level and dividing it into four sub-lists. We apply this approach to images and show that synthesis quality is not noticeably effected by a random sampling of vertices for each sub-pass.

5.4.4 Symmetric Synthesis

For some examples (e.g. Marine Life and Heads in Fig. 5.20 and Fig. 5.21 respectively), an additional scheme was used to encourage bilateral symmetry. Models were manually placed in their local coordinate frame so that their natural symmetry plane aligned with the origin of some axis. Thus for each vertex with a positive value along that axis, a mirrored point exists on the negative side of that axis. A mirror link is found to be the vertex that is spatially closest to that mirrored point. This extra searching step is necessary to account for approximate as well as perfectly symmetric inputs.

After synthesising a full level, update the exemplar pointers of one mirrored half of the model with corresponding mirror texels in the other half and run an additional solver pass. This approach is done as a post-process for each synthe-

sis level so that the core synthesis algorithm remains unchanged for symmetric examples.

5.5 Non-Uniform Multiscale

Images are stored in memory as a regular uniform 2D grid. Therefore it has always been sensible for texture synthesis approaches to synthesise an entire sample up to a single uniform resolution. However, it would be naive and impractical to continue this practice into the geometry domain. As a result we propose non-uniform upsampling.

The key idea behind non-uniform upsampling is that some textures vary in detail across different coherent regions. This is sometimes true for images and often true for 3D models. Figure 5.9 shows an image with non-uniform detail. On the left we see a duck swimming through a calm pond. While this image has uniform resolution it has non-uniform image details. The high resolution of pixels is necessary to capture the details of the duck while the visual detail of the water could be captured with a much lower pixel density. On the left we see what a non-uniform sampling would look like for images where the size of the pixels reflect the actual scale of image details.

Although non-uniform texture synthesis is attractive for images from a running time standpoint, ultimately it is standard practice to store and display images as a uniform regular sampling and would be awkward to do otherwise. Polygonal meshes, on the other hand, have no such storage and rendering restrictions. They are far more flexible topologically and thus more appropriate for non-uniform sampling. Given the substantial cost of doing geometry synthesis relative to image synthesis, it not only becomes attractive to do non-uniform sampling but also extremely practical.

Figure 5.10 illustrates the relevance of non-uniform sampling for 3D meshes.

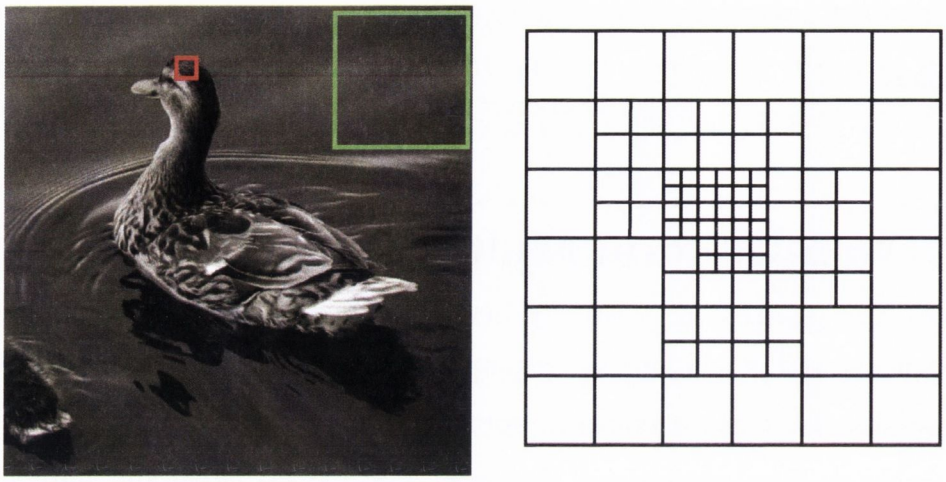


Figure 5.9: Non-Uniform Sampling. *Left: an example of non-uniform detail in a uniformly sampled image. The red box shows a very high detail region that requires the full resolution of pixels. The green box on the other hand shows a low detail region of the image that requires fewer than the full resolution of pixels. Right: This is an illustration of what a non-uniform sampling would look like, where high detail regions of an image would be given a higher pixel resolution than others.*

Notice how this model of a human face varies drastically with respect to the scale of its various features such as eyes, nose, mouth, ears and the shape of the skull. While the shape of the skull is rather coarse and can be properly modeled with very little geometry, high detail features like the nose and ears require far more geometry to approximate the same surface area.

5.5.1 Non-Uniform Neighbourhood Sampling

At border regions between two scales, neighbourhood gathering will sample pixels that are coarser than the pixel of interest. Figure 5.11 illustrates the problem encountered when the centre of a neighbourhood is at a higher resolution than the pixels next to it. Pixels are sampled at a scale that was never actually synthesised.

Figure 5.10: Non-Uniform Sampling. *Notice how the amount of geometry for a given amount of surface area varies greatly across this model. Highly detailed features such as the ears or lips require more geometry than coarser features such as the cheeks or skull.*

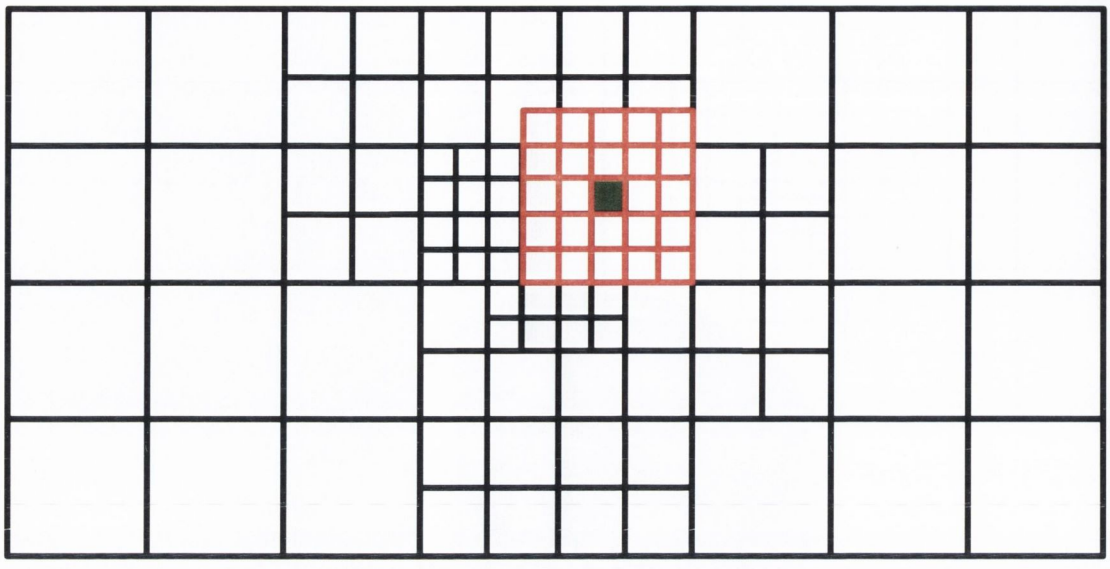


Figure 5.11: Non-Uniform Neighbourhood Sampling. *When sampling the neighbourhood of a pixel next to a scale border, a higher resolution pixel than actually exists in the data needs to be sampled. By doing a blur pass over the coarser scaled pixels, an approximation of the higher resolution image is produced and can be sampled from.*

These higher resolution pixels are approximated by blurring the coarser pixels. The same concept applies to geometry. The only difference is that vertex data is interpolated along mesh edges.

5.5.2 Vertex Activity

Non-uniform upsampling introduces a number of new technical challenges to the standard uniform upsampling framework. The distinction between a vertex being "active" (red) and "inactive" (green) is now important throughout the synthesis framework. During the Gaussian Pyramid creation process, only active vertices have their neighbourhoods gathered and simplified. During the nearest neighbour search only active vertices are eligible neighbours. As such during synthesis, only

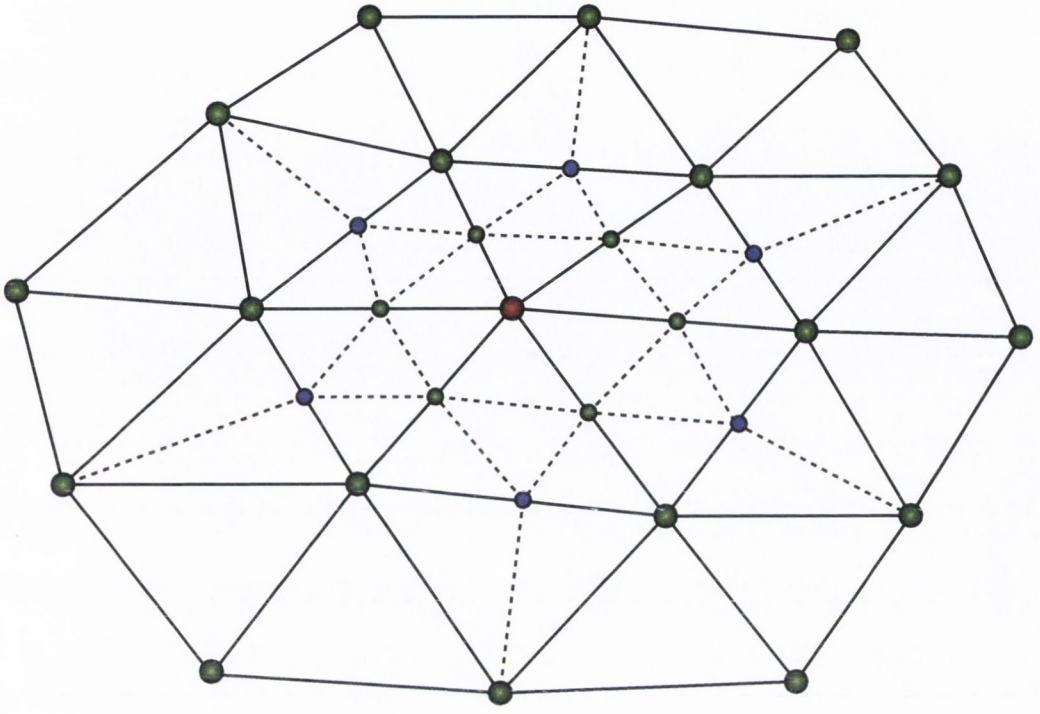


Figure 5.12: Non-Uniform Upsampling. *In the case that only a subset of our vertices remain active for upsampling to the next pyramid level (coloured red), then we only split the edges that contain an active parent (shown as smaller green vertices), and we create dummy vertices (blue) for rendering purposes.*

active vertices are eligible for upscale, jitter and correction.

Non-Uniform Upscaling

The exemplar pyramids can have non-uniform resolution where vertices at a given level are denoted as either active or inactive based on whether their edges can continue to be split to create finer detail neighbourhoods.

In the case that only a subset of the vertices in synthesis level S^l point to exemplar vertices that are still active, then the mesh has to be upscaled adaptively as shown in figure 5.12. In this example a single vertex is active (shown in red)

and the rest are inactive (green). When splitting edges there are three cases:

- Both parents are active in which case the edge is split and marked as active.
- One parent is active in which case the edge is split and marked as inactive.
- Both parents are inactive in which case the edge is not split.

It is important to distinguish a new child vertex as active or inactive because active vertices are subject to jitter and correction while inactive vertices are not. In our example we only see the second and third case.

Notice that the blue child vertices do not follow any of the three cases. The blue vertices are a product of an unofficial fourth case (unofficial because it is only necessary for rendering purposes and not for synthesis). In this case two of the three edges comprising a polygon have been split, thus the third edge must also be split (blue) in order to up-sample one polygon into four. Although this edge is split, it is not a new child as far as the algorithm is concerned and only exists as a dummy vertex for rendering purposes. The positions of the blue vertices are just the centre of the edge. New edges are created as shown by the dashed lines.

Note that each edge creates the border between two polygons. When the fourth case is triggered for a polygon and the third edge is split, this must be updated into the bordering polygon as well. If this polygon has no other split edges then it can be split into two polygons as seen in fig. 5.12. If this polygon already has two split edges then this new split completes the third edge for one-to-four polygon up-sampling. However, if this polygon already has only one split edge, then the new edge split will cause two of the three edges to be split, triggering the fourth case again for this neighbour polygon as well, and the process will need to be repeated. This is illustrated in figure 5.13, where an inactive node in the previous illustration 5.12 has been changed to active, causing two polygons that had previously been split into two to now be split into four. Two polygons that had previously been untouched are now split into two.

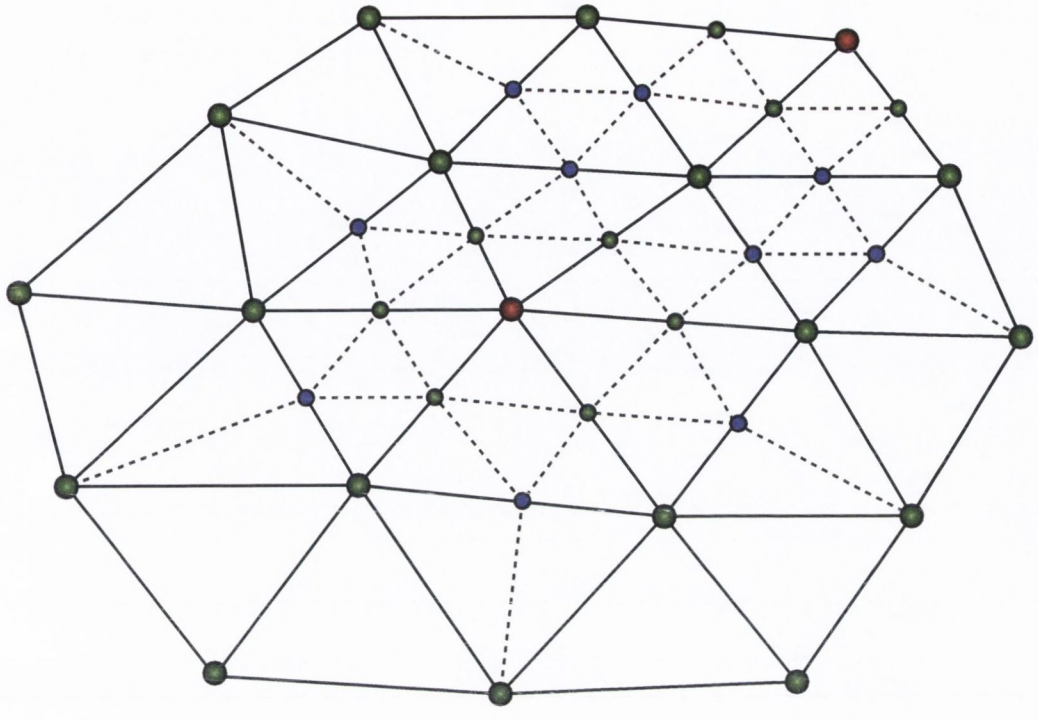


Figure 5.13: Non-Uniform Upsampling - Fourth Case. *The simple example from figure 5.12 has been updated with another active vertex in order to illustrate the fourth case for non-uniform upsampling. Polygons that were split into two in the previous illustration are now split into four and a new split edge is introduced into a previously unmodified polygon.*

A practical example of non-uniform upscaling is shown in figure 5.14 for one of the exemplar models. The utility and necessity of this approach is shown through the added flexibility of the algorithm to increase mesh resolution only where finer detail features exist such as the ears, eyes, nose and mouth. Leaving most of the surface of the mesh at a much lower resolution.

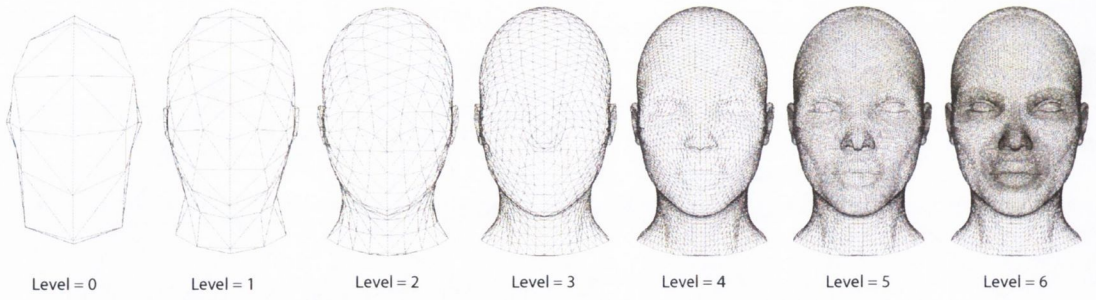


Figure 5.14: Non-Uniform Exemplar Pyramid. *Notice that level four is the final level for uniform upscaling. Levels five and six rely on non-uniform upscaling in order to add detail to the eyes, ears, nose and mouth.*

5.6 Image Geometry

As a transition between the previous work on image hybrids and the current work on geometry hybrids, image geometry uses the latter to achieve the former. Image geometry validates that the geometry synthesis ideas presented in this chapter achieves comparable synthesis quality to the established pixel based image texture synthesis framework.

Because this chapter extends neighbourhood based texture synthesis into a new type of multimedia data, it is difficult to judge the quality of synthesis compared to the previous work. The goal of image geometry is to use the geometry synthesis framework as an indirect image synthesiser. The results of which can be compared against the ground truth hybridisation images from Chapter 3. This gives the reader the ability to validate results from this chapter with standard image based synthesis in an apples to apples comparison.

Given multiple images as input, the goal is to hybridize them into a new single image. First images are converted into a regular mesh where each pixel is represented by a vertex and adjacent pixels are given edge links between them as shown in figure 5.15. In this case the topology of the resulting mesh is both regular and

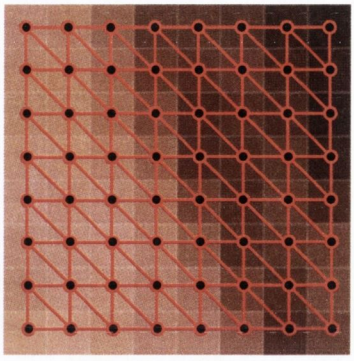


Figure 5.15: Image Geometry Mesh. *This figure illustrates the mesh topology used for image geometry and how each vertex corresponds to a pixel in the original image.*

cyclical where vertices at border pixels are connected with edges to border vertices on the other side of the image. Cyclical connectivity is necessary so that mesh sampling imitates having the same image tiled on an infinite plane as was proposed in Parallel Controllable Texture Synthesis [LH05].

We have divided the image geometry test into two different types, one for colour images and one for grey-scale relief maps. Each type is useful for measuring the success of the geometry synthesis method with respect to image synthesis. For colour images each vertex encodes the RGB value of its corresponding pixel as XYZ coordinate. The resulting geometric transformation is then used as input for the algorithm described in this chapter. For gray-scale relief images, the colour values are averaged together into an intensity I and the XY coordinates of the image pixel combined with their I value is encoded as the XYZ coordinate for each vertex. Gaussian pyramids for these two geometry encodings are shown in figure 5.16.

Still looking at figure 5.16, notice that colour images produce geometry with no real perceivable structure, in fact vertex placement appears random and chaotic to the eye. As such, this is an important test for the geometry approach as it should

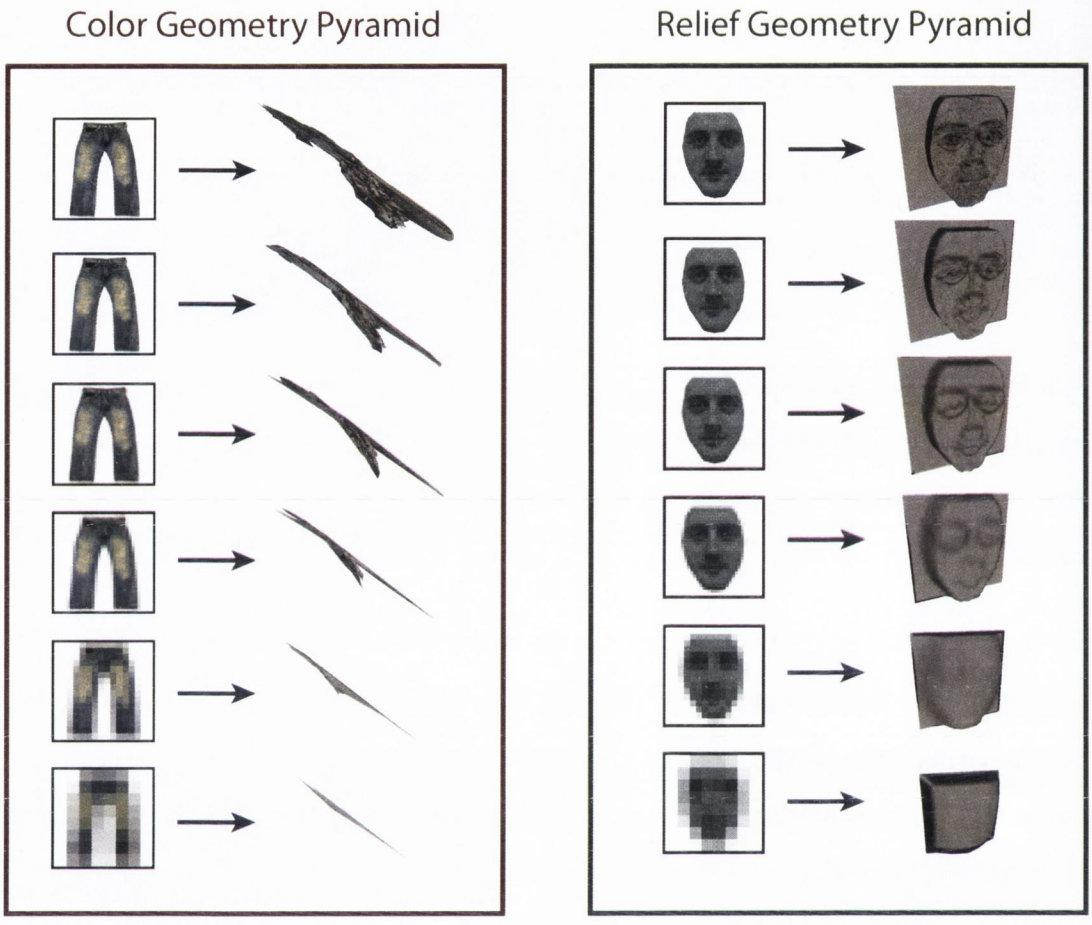


Figure 5.16: Gaussian image pyramids with corresponding geometry pyramid counterparts. *Gaussian pyramids are shown for colour images (left) and relief images (right). The Gaussian image pyramid is shown on the left and its conversion to geometry is shown to the right. The reasoning behind these two tests is revealed by these two Gaussian pyramids. Colour images produce a geometric representation that has no bearing on the actual shape of the model, this is an important test to show how our approach deals with seemingly chaotic shapes with no perceivable structure or smooth gradients. The relief test is important because it serves as a missing link between the colour test and regular geometry whereby an image is still being synthesised, but that image does form a 3D model that has a meaningful structure and smooth gradients which our algorithm was designed for.*

theoretically be able to handle any shape regardless of whether its smooth, self intersecting, concave, convex or has any other shape property. Relief images produce geometry with a somewhat smooth 3D shape that is visually meaningful like the pure geometry examples the method was designed for. Keep in mind that the fundamental goal of any type of neighbourhood based texture synthesis approach is to reproduce the input neighbourhood statistics into the output as closely as possible. That is, it should minimise the distance from each neighbourhood in the output with its closest neighbourhood in the input. The colour image test validates that the geometry approach achieves this goal just as well as its image based archetype. The relief test when compared against the colour test is meant to show that the geometry approach performs uniformly well over a vast range of shape properties (i.e. it does not just work well for the smooth volumetric shapes it was designed for). Flexibility is the mark of a good example based synthesis approach.

A full analysis of the results and comparison with previous work is presented in the results Section 5.7

5.7 Results

The results for this chapter are organised into three different categories. First we present Image Geometry in order to validate that the quality of this approach is comparable to image based methods. We then present pure geometry hybrids as well as a comparison between uniform and non-uniform multiscale.

5.7.1 Image Geometry Results

The resulting hybrids are presented in figures 5.17 and 5.18. Not only is the quality of the results comparable to the image based approach, but the geometric approach has the added benefit that it does not need to first recolour the input images in

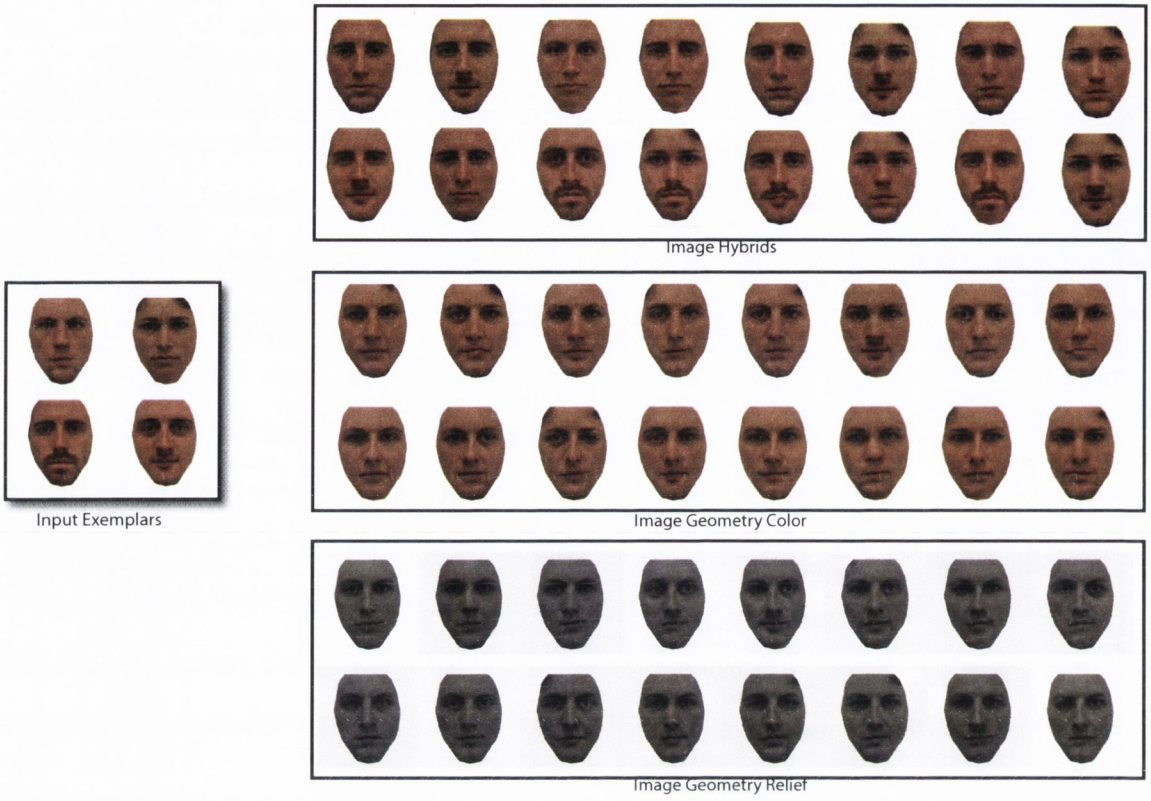


Figure 5.17: Faces. *The top set of images were taken directly from the image hybrids results for comparison purposes. The middle set of images were generated using the coloured image geometry approach and the bottom set of images were generated using the relief image geometry approach.*

order to make them colour consistent with each other. Whereas the image based approach treated pixel colours as absolute values and found the euclidean distance between the same element in two neighbourhoods. In the geometric algorithm both neighbourhoods are first centred around their middle vertex so relative changes in colour from the centre are compared. This leads to better nearest neighbour matching for two very differently coloured images. During synthesis we avoid the apparent seams between two coherent patches taken from two different exemplars. This Frankenstein artifact is smoothed out because absolute colour values are no longer being copied over from the exemplar, instead the neighbour pixels already

synthesised in the previous level are updated with the relative colour changes. This gives an interpolation between two colour inconsistent patches. The solver step also goes a long way to providing colour consistency by optimizing all the relative colour differences between adjacent vertices to mimic those found in the exemplars, two vertices from different exemplars will move through colour space until the relative gradient between the two reflects the gradient seen in the two examples. This naturally pushes two different coloured neighbourhoods together into a mutual middle colour.

This is a double edged sword as the same mechanism that gives implicit colour correction also causes the artifacts seen in some of the resulting images where the border region between the image and its white background has been blended such that sharp edges are not always preserved. This is caused when jitter picks exemplar vertices with poor correspondence to each other at visually sharp edges. The solver step tries to reconcile the conflicting edge directions and ends up smoothly blending what should be a sharp edge. This is the reason why the white background becomes a slightly darker and non-uniform off-white in some of the results.

Note that the image geometry results have a recurring artifact that the image hybrid results do not. This is caused by the loss of the Gaussian stack. At the coarser level images have become very low resolution and a lot of information has been lost. As such, if two image objects have even a slight translation or size difference from each other, then at the coarse level pixels will be matched as nearest neighbours that do not align with each other at the finer scales (this shows why Gaussian stacks and multiresolution descriptors are important for hybridisation). In the faces example 5.17, the white border is preserved as it is the same for each exemplar, however internally facial features have slight translation offsets from each other, making coarse level nearest neighbours inaccurate. This artifact is especially noticeable in the jeans example due to the difference in silhouette for each image.

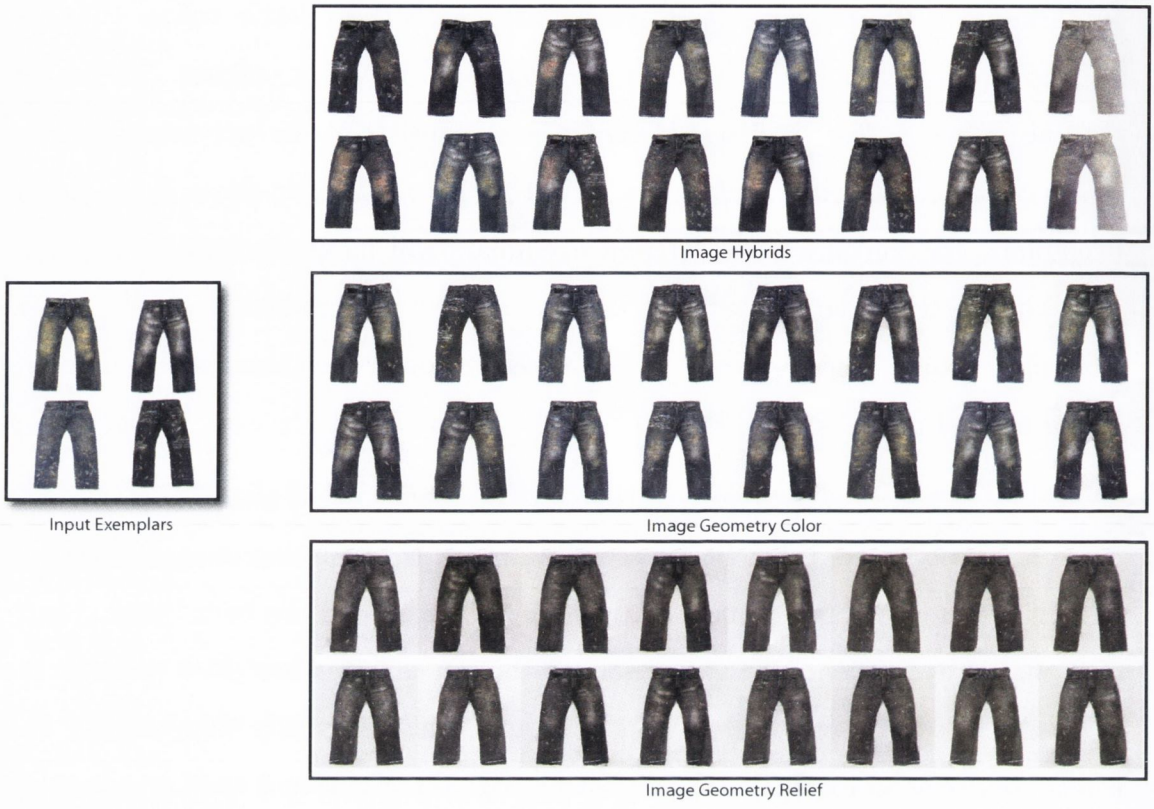


Figure 5.18: Jeans. *The top set of images were taken directly from the image hybrids results for comparison purposes. The middle set of images were generated using the coloured image geometry approach and the bottom set of images were generated using the relief image geometry approach.*

Besides the geometry specific modifications such as solving and Gaussian pyramids that will naturally lead to slightly different results, we think that in general the quality of resulting hybrids are roughly equivalent between all three approaches, image hybrids, image geometry colour hybrids and image geometry relief hybrids. The side by side results are shown in figures 5.17 and 5.18. Coloured image geometry gives reasonable assurance that this new mesh specific neighbourhood based texture synthesis framework performs comparatively to the previous image based approach. By synthesising a seemingly chaotic shape we believe we are stress testing the approach with a worst possible case example for our approach.

Relief image geometry is an important additional example as it shows that synthesis performs equally well on chaotic noisy meshes as it does on smooth manifolds. The difference between these two mesh representations are shown in figure 5.16. The mark of a good neighbourhood based texture synthesis approach is that it performs uniformly well give any set of neighbourhoods. Colour and relief image geometry show roughly equivalent image results for very different underlying mesh representations.

5.7.2 Geometry Results

The first example (figure 5.19) illustrates a simple hybridisation of four mushroom models. Each of the mushroom examples share the same Gaussian Pyramid topology and are synthesised with no symmetry restrictions. This is a good example for observing the approaches ability to hybridize smooth regions without introducing crease artifacts. Because all the input exemplars are smooth, it follows that all the output hybrids should also be smooth.

The marine life example shown in figure 5.20 uses four models, a great white shark, dolphin, hammerhead shark and manatee. This is an interesting example as it introduces two new features, bilateral symmetry and examples with different Gaussian Pyramid topologies. That is, the mesh for the base of each example pyramid is topologically different from the others. When synthesising a new hybrid, the base mesh is chosen from an exemplar at random. Despite different topologies between examples, the four animals clearly hybridize together.

Non-Uniform Multiscale Geometry Results

The heads example in figure 5.21 shows how results can be improved with non-uniform upsampling by offering a side by side comparison with uniform vs. non-uniform multiscale synthesis of the same two exemplar models. In the lower set of results, highly detailed facial features such as the ears, nose, mouth and eyes are

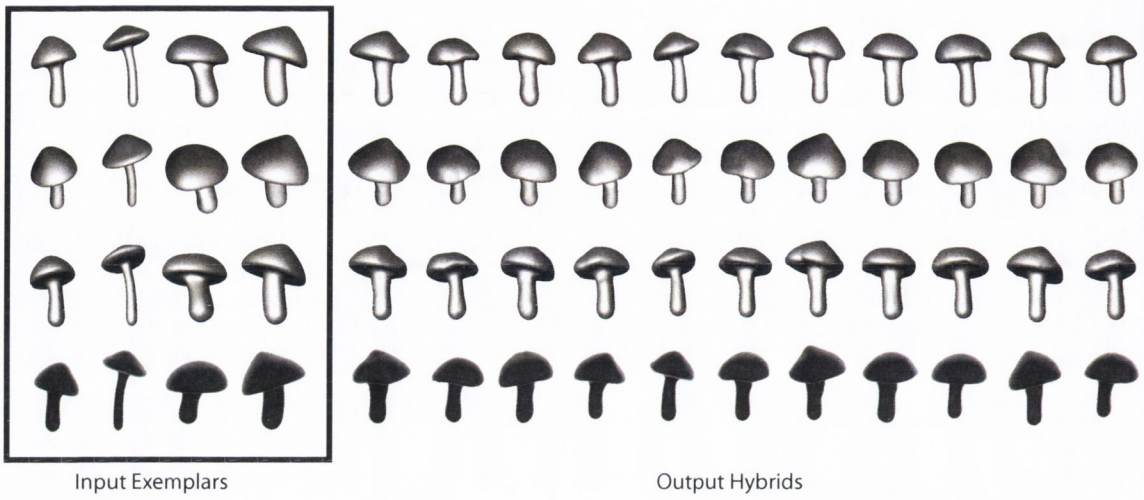


Figure 5.19: Mushrooms. *Due to the smoothness of the input exemplars, it is easy to observe that hybridisation properly transitions between two different exemplar regions. Notice that the output exemplars share the overall smoothness of the input and no sharp edge artifacts are added at borders between two patches from different exemplars, this is largely credited to our new solver approach.*

upscaled to a higher resolution than the rest of the head. Note that in this example both exemplars use the same base mesh and thus have the same exemplar mesh topology up until the finest uniformly upsampled level shown as the top set of results. While both non-uniform resolution examples are upscaled by another two levels, the two exemplar pyramid meshes now diverge from each other topologically as different sets of polygons are further upsampled for each exemplar. The main limitation seen with this extension is the case where a border seam arises between two patches taken from different exemplars at different scale levels. The dummy vertices appear turbulent in the final result because there are no exemplar links for dummy vertices, so they are passed over by the solver which is meant to fix this type of artifact.

Running times vary due to mesh resolution. The marine example averaged 36.7 seconds over 50 sets while the high resolution heads took 3.6 minutes aver-

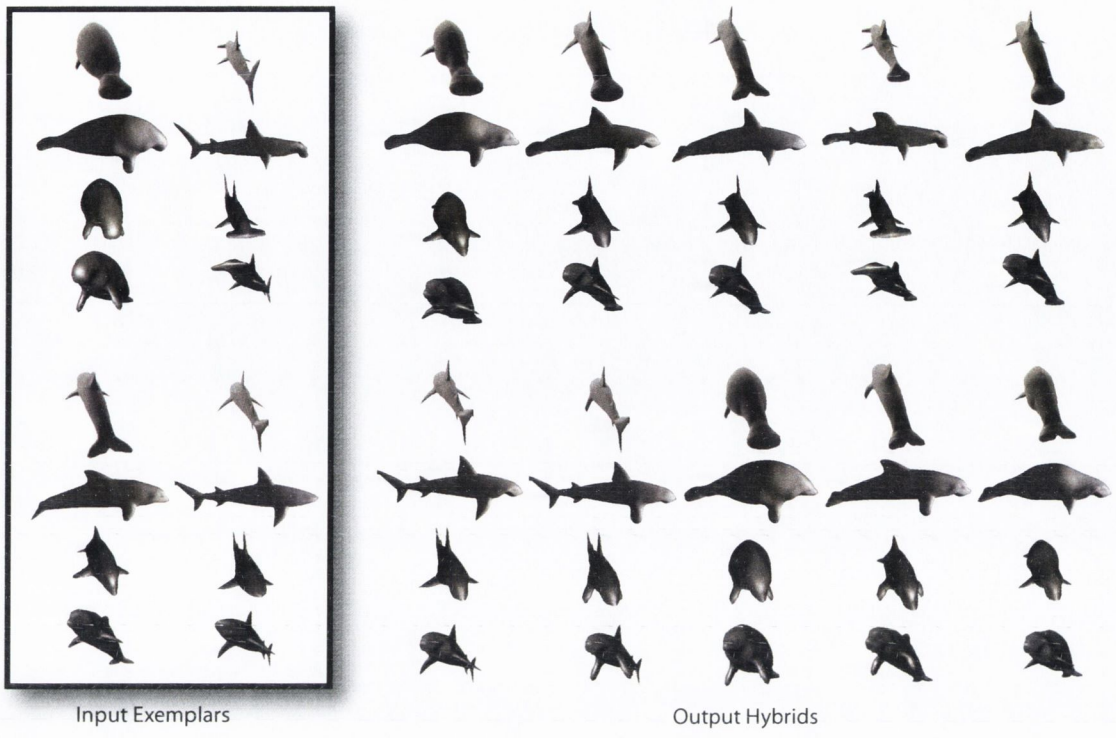


Figure 5.20: Marine Life. *This example introduces two new features to geometry synthesis, bilateral symmetry and exemplars with different base meshes.*

aged over 50 sets. These times can be improved through code optimisation and parallelisation.

5.8 Conclusion

In this chapter we have extended our hybrid and 2D polyline synthesis concepts from Chapters 3 and 4 to 3D meshes and we have introduced non-uniform neighbourhood based texture synthesis and shown why it is a practical extension for 3D models. Our image geometry experiments offer a strong validation of the approach by visually comparing it to its pixel based predecessor. Geometry hybridisation opens more doors than it closes and could lead to interesting and useful future work.

Figure 5.21: Heads. *The top set of exemplars and hybrids are all sampled uniformly up to a level where the majority of the polygons visually represent the underlying shape and further upsampling would just lead to wasted resources. The bottom set of images takes the uniformly upsampled exemplar pyramids from the top set and continues to upsample highly detailed features non-uniformly so that the eyes, nose, lips and ears recieve extra resolution.*

5.8.1 Limitations and Future Work

Gaussian Stacks

Having to abandon the Gaussian stack multiresolution data structure is a setback for this approach. Future work could focus on a new remeshing approach that yields a full regular model without creating non-deterministic stretched neighbourhoods. Another interesting approach would be a hybridisation between the Gaussian stack and pyramid representations. Because the meshes are semiregular and thus have large coherent regular regions, it should be possible to experiment with sets of smaller stacks organised together as a pyramid.

Improved Solver

While this chapter improved the quality of solving for 3D meshes. There are other approaches that could potentially be even better. The PRIMO approach [BPGK06] was designed to give smooth manifolds a rigid but flexible structure so that forces can be applied to parts of the model in a physical simulation and the model will deform as a whole as though it was made from a flexible material. Treating our models as flexible shells and treating neighbourhood inconsistency as force on that shell is similar to what we are already doing and PRIMO has been shown to tackle that problem very well.

Discrete element hybridisation and synthesis

There has been a lot of recent attention given to the synthesis of arrangements and combinations of discrete elements of 3D objects. These approaches complement the ideas presented in this chapter nicely. Once hybrids are created for individual elements, these other approaches can take over in order to build more complex models by arranging these elements together. A recently published technique created independently and concurrently to our work [MWT11] extends the work

done on discrete element texture synthesis (see Section 2.3.3) to 3D objects. In their future work they mention the need for an approach that synthesises the actual discrete elements themselves, as we do in this chapter.

Improved Non-Uniform Multiscale

The non-uniform multiscale approach appears to cause non smooth artifacts by stretching and bending edges in undesirable ways when there is a border region between two coherent patches that come from different exemplars at different scale levels. We believe this is caused by the dummy vertices. Because they do not have an exemplar counterpart, there is no way to solve for them. We do the simplest most naive alternative and leave them fixed in space, allowing the result of the model to solve around them. While this works to some extent, solving can not be performed on dummy vertices or other vertices connected to dummy vertices. This results in strips of vertices along border regions that are not being solved for shape. While there is no exemplar information available for these edges, they could still be solved in such a way that favors surface smoothness.

Chapter 6

Conclusion

6.1 Summary and Final Words

This thesis has introduced the idea of hybridisation as a sub-problem of texture synthesis and offered a strategy to adapt a previous texture synthesis algorithm for hybridisation in Chapter 3.

The other main focus of this thesis has been to expand the scope of neighbourhood based texture synthesis to also encompass geometry based multimedia data. We introduced several key new concepts in Chapter 4 starting from the image based texture synthesis framework and establishing the parallel algorithmic components for geometry synthesis. The keystone of which is the shape texel that provides a mechanism through which shapes can be segmented into discrete units and averaged/compared just like their image based counterpart the pixel. These ideas first presented for 2D polylines were then extended to 3D meshes in Chapter 5 with several mesh specific ideas being introduced, such as the improved solver as well as non-uniform multiscale.

Our work has followed a course progressing from standard texture synthesis towards 3D mesh hybridisation. Along the way we have tackled several problems

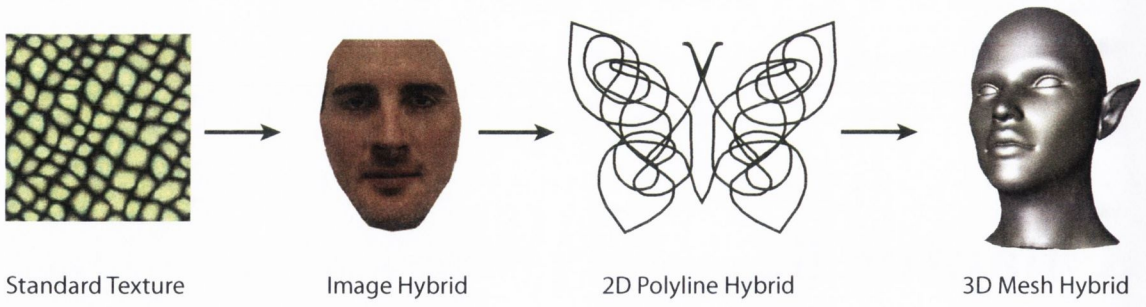


Figure 6.1: Research Progression. *Starting from standard texture synthesis, we first modify the algorithm for hybridisation, identifying that hybridisation is a subproblem of texture synthesis. Armed with a new hybridisation approach, next we extended texture synthesis once again increasing its scope to geometry in the form of 2D polylines. Finally we expand on the concepts that make geometry synthesis possible, in order to handle 3D meshes.*

in steady progression towards our final goal as shown in figure 6.1.

6.2 Future Directions

There are several interesting and useful directions in which this work can be extended. We showed how geometric multimedia data can be incorporated into a neighbourhood based texture synthesis framework. This is only a first step. Theoretically any multimedia data type could be synthesised as a texture/hybrid.

Animation

I think the natural progression of this line of research is animation. Much the way shape texels easily extended from 2D to 3D, they can also extend from 3D to 4D to make shape/time texels. This extra time information can be applied directly to 3D meshes in order to achieve surface animations such as a flag waving in the

wind, or ocean waves. In addition to simple surface animations, we believe that our approach could be readily adapted for skeletal animations. Skeletons form a natural tree style graph of vertices connected by edges. The neighbourhood for a vertex would then be gathered and compared across time (in both directions) as well as down the tree (same concept as multiscale descriptors). Using our improved solver with only the angular component, inconsistencies could be easily fixed and any user placed constraints, such as hands or feet needing to be in certain positions at certain times could also be forced through the solver. Because synthesis would be correcting neighbourhoods that look forward and backward across time, the overall result will naturally conform to these fixed constraints.

Example Based Audio Texture and Hybrid Synthesis

While early work has been done to synthesise continuous audio textures from small sample clips (as discussed in the audio section of the background 2.4), audio texture synthesis is still relatively primitive with the state of the art by O'Regan and Kokaram [OK07] following the synthesis procedure made popular by Efros and Leung [EL99]. O'Regan made a powerful discovery that the dual tree complex wavelet transform produced very descriptive coefficients for audio clips. By re-ordering these coarse clips using a Markov chain and then utilising the inverse complex wavelet transform, they reconstruct the new full frequency waveform. The dual tree complex wavelet transform seems to audibly blend two sound clips together, which is a very powerful tool moving forward with audio texture synthesis. While plenty of research has been done from a signal processing standpoint for audio textures, comparatively little work has been done from the point of texture synthesis and this is reflected by the primitive techniques being used. It would be an exciting new direction to incorporate audio into modern texture and hybrid synthesis frameworks. At the moment no work has been done with respect to globally varying audio textures or highly structured textures such as human speech. Likewise, nobody has looked into symmetry in audio textures in order to find a

hierarchy of patterns for synthesising complex and varying audio textures such as musical compositions. This is currently an unexplored area of audio processing using texture synthesis.

Mixed Multimedia

Given the ability to synthesise multiple multimedia data types using a single common texture synthesis framework, it would be feasible to mix those data types in order to produce even more complex and customized results. An example related to this thesis would be geometry data mixed with images in the form of textured models. The geometry and texture maps could be synthesised in tandem using a single synthesis pyramid where the texels now contain colour and shape information. Alternatively, some levels of the pyramid could still contain texels which are exclusively colour or shape based so that first a shape could be synthesised independent of colour and then used as a guide for synthesising colour on top of it.

Similarly if animation and audio were synthesised using this framework, they would also be prime candidates for mixing. If audio were to be recorded during a motion capture session, then example based animations would have corresponding sound information such as footsteps and breathing patterns. Animation synthesis could then direct corresponding sounds.

Ultimately the holy grail for mixed multimedia and example based synthesis in general would be a complete synthesised world including coloured 3D objects animating and generating sounds all unique and all created by the same algorithm using only examples.

Appendix A

Image Hybrids

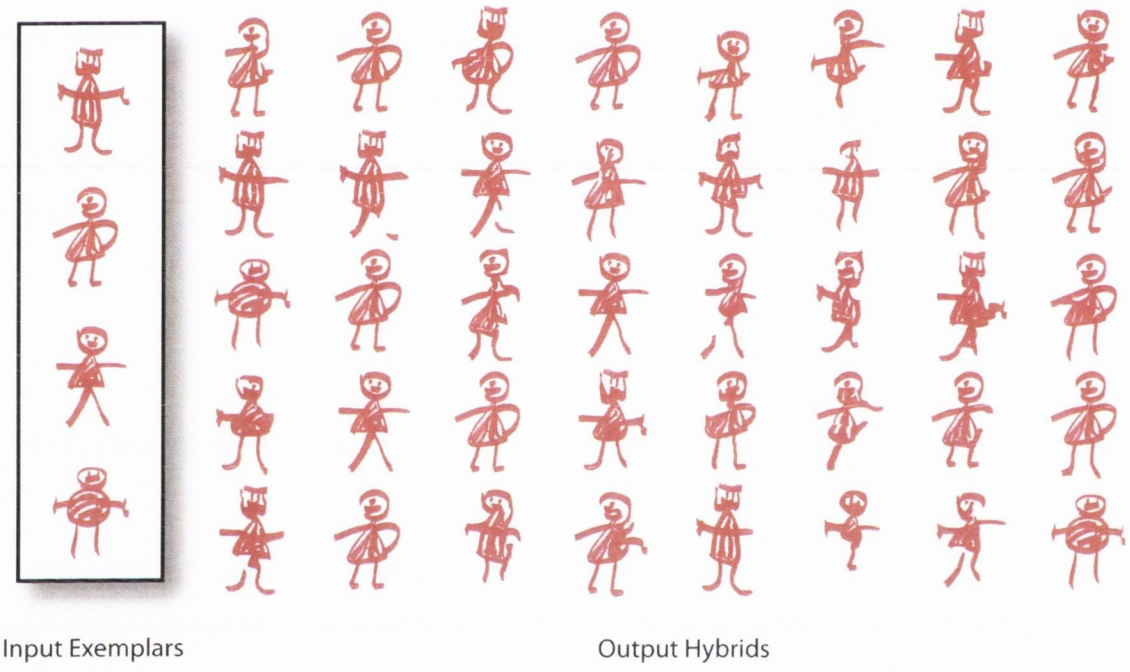


Figure A.1: Supplemental Doodle Hybridisation Results. *This example illustrates that human generated drawings can be hybridised while preserving artistic style. These 40 results were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

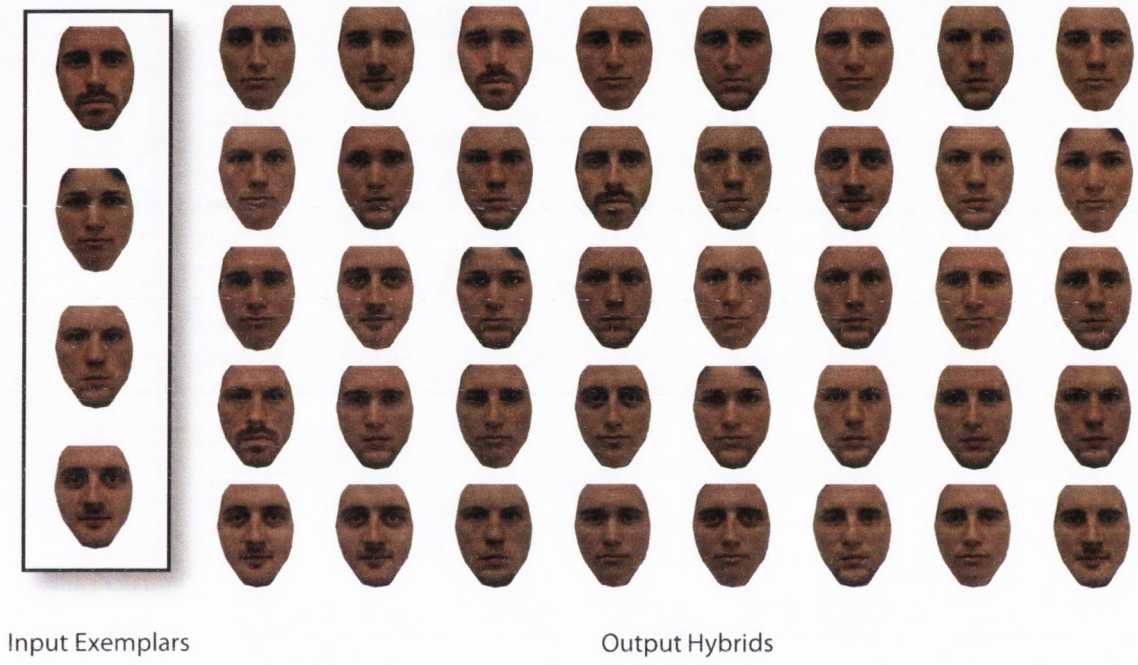


Figure A.2: Supplemental Face Hybridisation Results. *Human faces are an especially challenging hybridisation example due to the complex structure and aptitude of the human eye to recognize flaws in a face. These 40 results were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

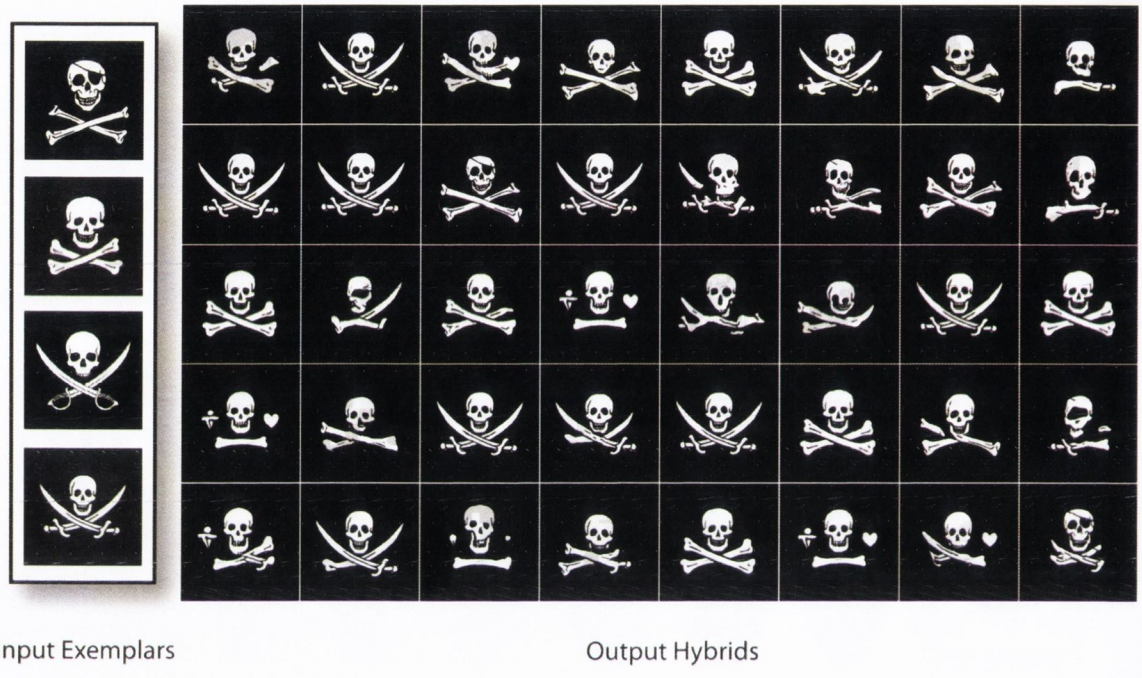


Figure A.3: Supplemental Pirate Hybridisation Results. *In the pirate example we have highly structured and recognisable symbols. Notice the interesting structure drift between the bones and swords. Also note that not all of the skulls have jaw bones and/or teeth. Having large but interchangeable variations in structure leads to very interesting synthesis results. These 40 results were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

Figure A.4: Supplemental Dude Hybridisation Results. *The Dude by Andy Warhol is an interesting example because the faces are very similar but have different expressions and are captured from different angles with different hair styles. Notice that the amount of jitter appropriate for faces is not enough to allow variation in hair styles. These 40 results were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

Figure A.5: Supplemental Jean Hybridisation Results. *Jeans made an excellent real world example and validate the standardization method. Notice that each pair of jeans has small colour differences and larger intensity differences which do not cause problems in the final synthesised result. These 40 results were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

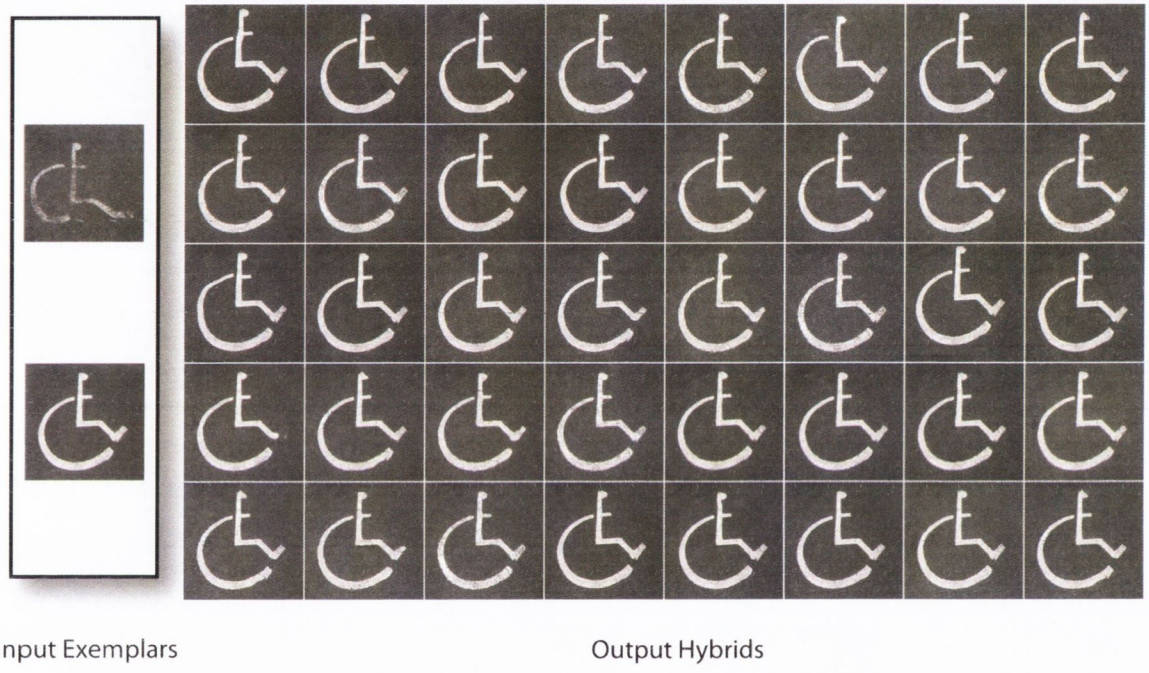


Figure A.6: Supplemental Road Sign Hybridisation Results. *This example illustrates that image hybridisation can be used to generate weathered and eroded variations of real world objects. These 40 results were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

Appendix B

Curve Hybrids

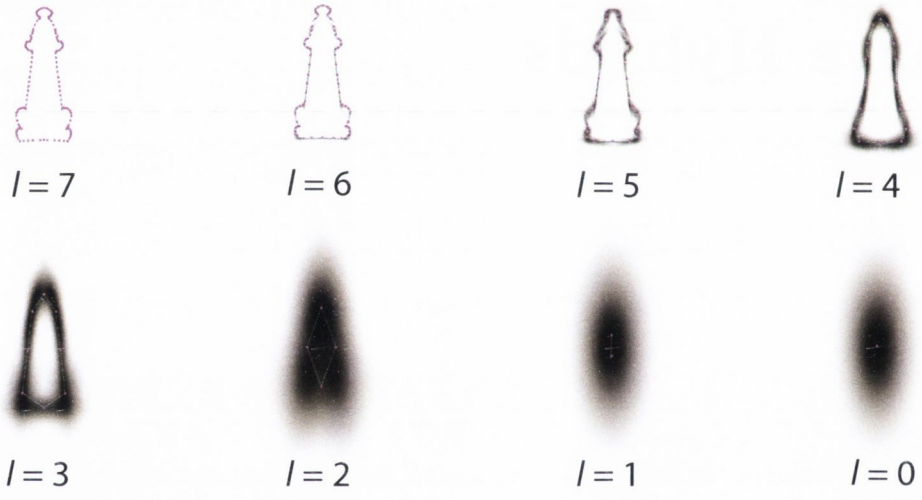


Figure B.1: Chess Curve Synthesis Pyramid. *This figure shows each level of a synthesis pyramid for a chess piece shaped curve. Pink dots illustrate shape texel positions while the surrounding dark cloud is the probability distribution function given by the shape texels. Lines between shape texels illustrate the synthesised curve.*

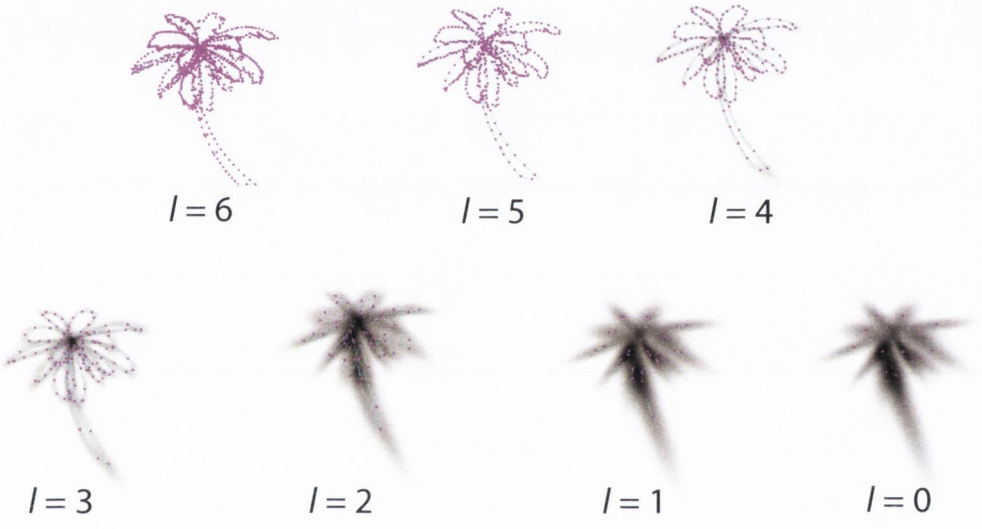


Figure B.2: Palm Tree Curve Synthesis Pyramid. *This figure shows each level of a synthesis pyramid for a palm tree exemplar graph of curves. Pink dots illustrate shape texel positions while the surrounding dark cloud is the probability distribution function given by the shape texels. Lines between shape texels illustrate the synthesised curve. At level zero each pink dot represents a separate closed curve to be synthesised.*

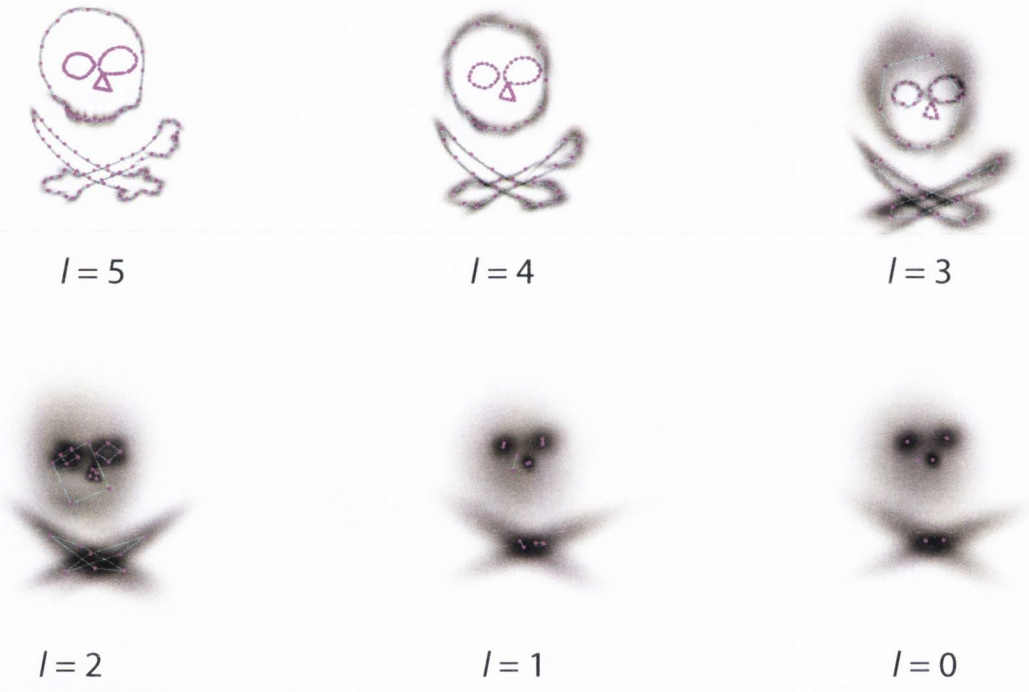


Figure B.3: Pirate Flag Curve Synthesis Pyramid. *This figure shows each level of a synthesis pyramid for a pirate flag exemplar graph of curves. Pink dots illustrate shape texel positions while the surrounding dark cloud is the probability distribution function given by the shape texels. Lines between shape texels illustrate the synthesised curve. At level zero each pink dot represents a separate closed curve to be synthesised.*

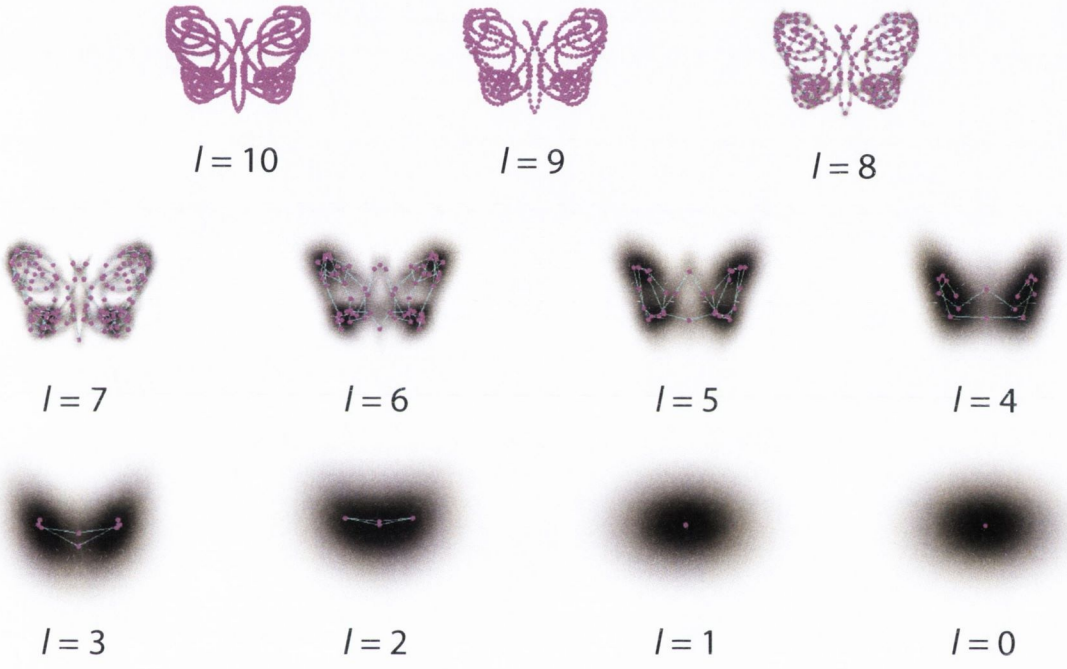


Figure B.4: Butterfly Curve Synthesis Pyramid. *This figure shows each level of a synthesis pyramid for a self-intersecting butterfly shaped curve. Pink dots illustrate shape texel positions while the surrounding dark cloud is the probability distribution function given by the shape texels. Lines between shape texels illustrate the synthesised curve.*

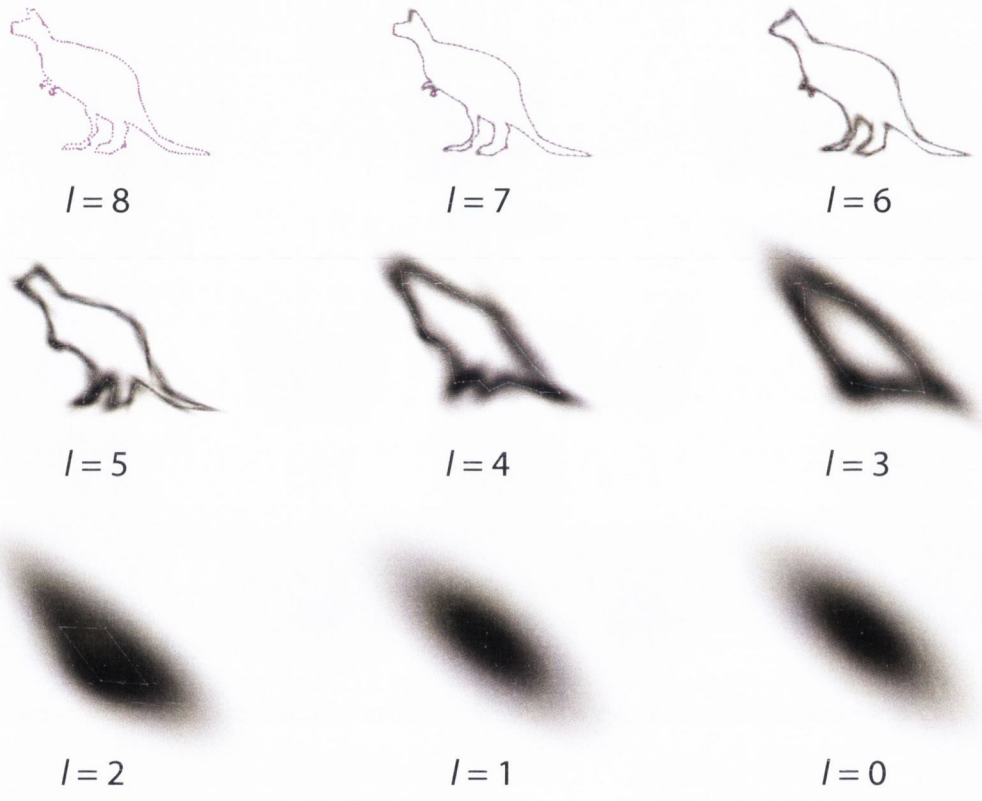


Figure B.5: Kangasaurus Curve Synthesis Pyramid. *This figure shows each level of a synthesis pyramid for a kangasaurus shaped curve. Pink dots illustrate shape texel positions while the surrounding dark cloud is the probability distribution function given by the shape texels. Lines between shape texels illustrate the synthesised curve.*

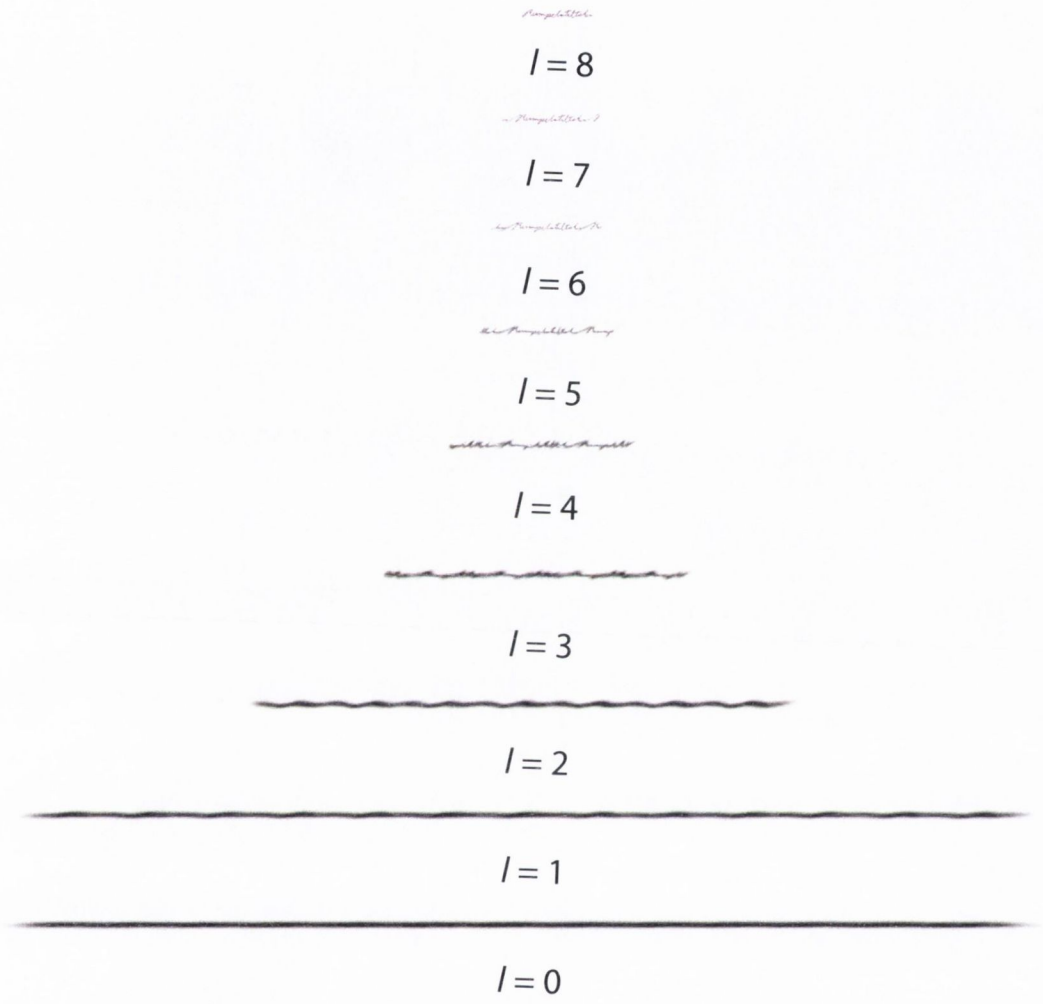


Figure B.6: Rumpelstiltskin Curve Synthesis Pyramid. *This figure shows each level of a synthesis pyramid for an open curve which spells the name Rumpelstiltskin. Pink dots illustrate shape texel positions while the surrounding dark cloud is the probability distribution function given by the shape texels. Lines between shape texels illustrate the synthesised curve. Note that at coarser levels, additional curve information is supplied on each end so that full neighbourhoods can be sampled.*

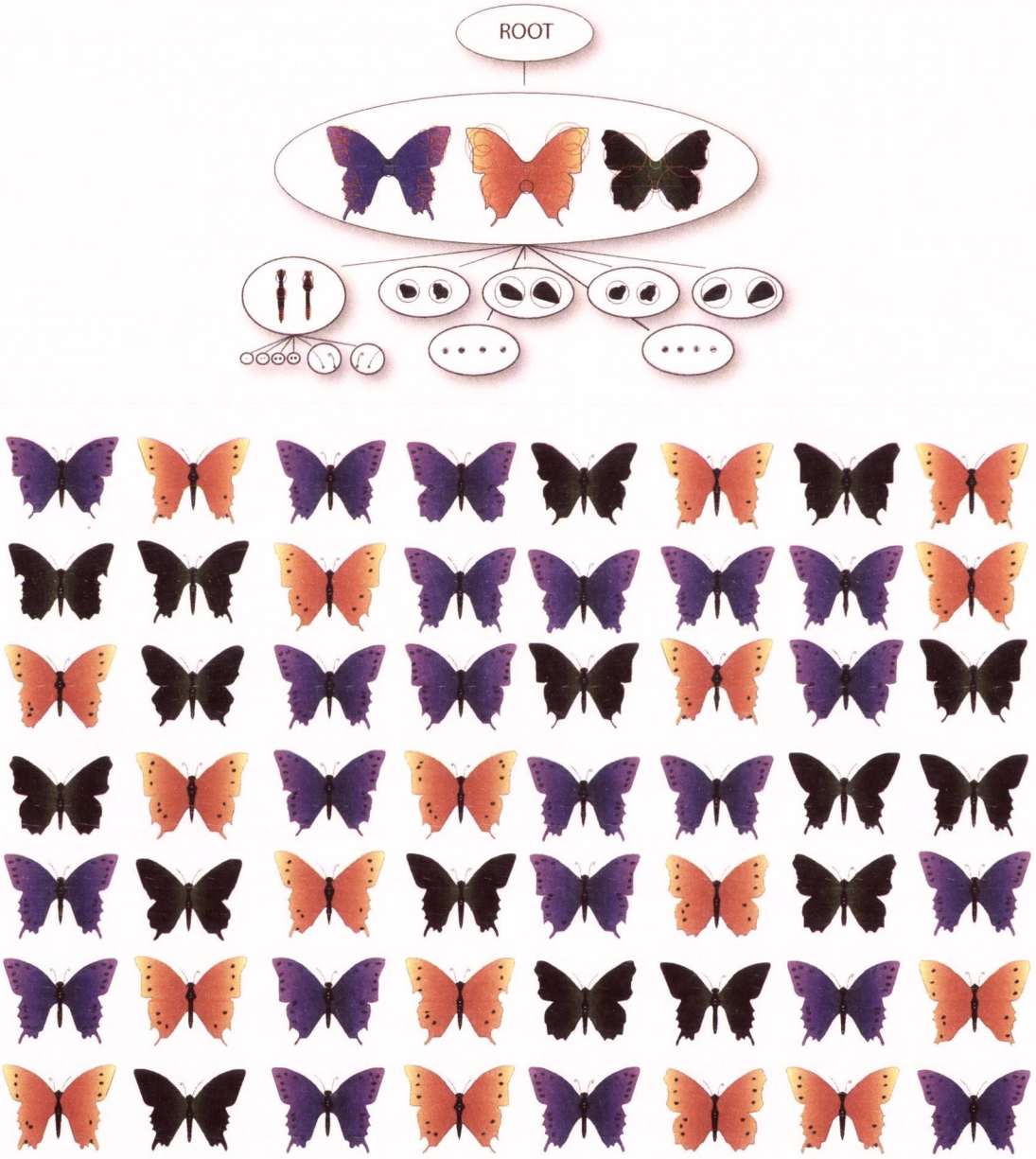


Figure B.7: Supplemental Butterfly Graph Results. *These 56 multi-layered butterflies were generated at random from the corresponding exemplar graph (top). These results were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

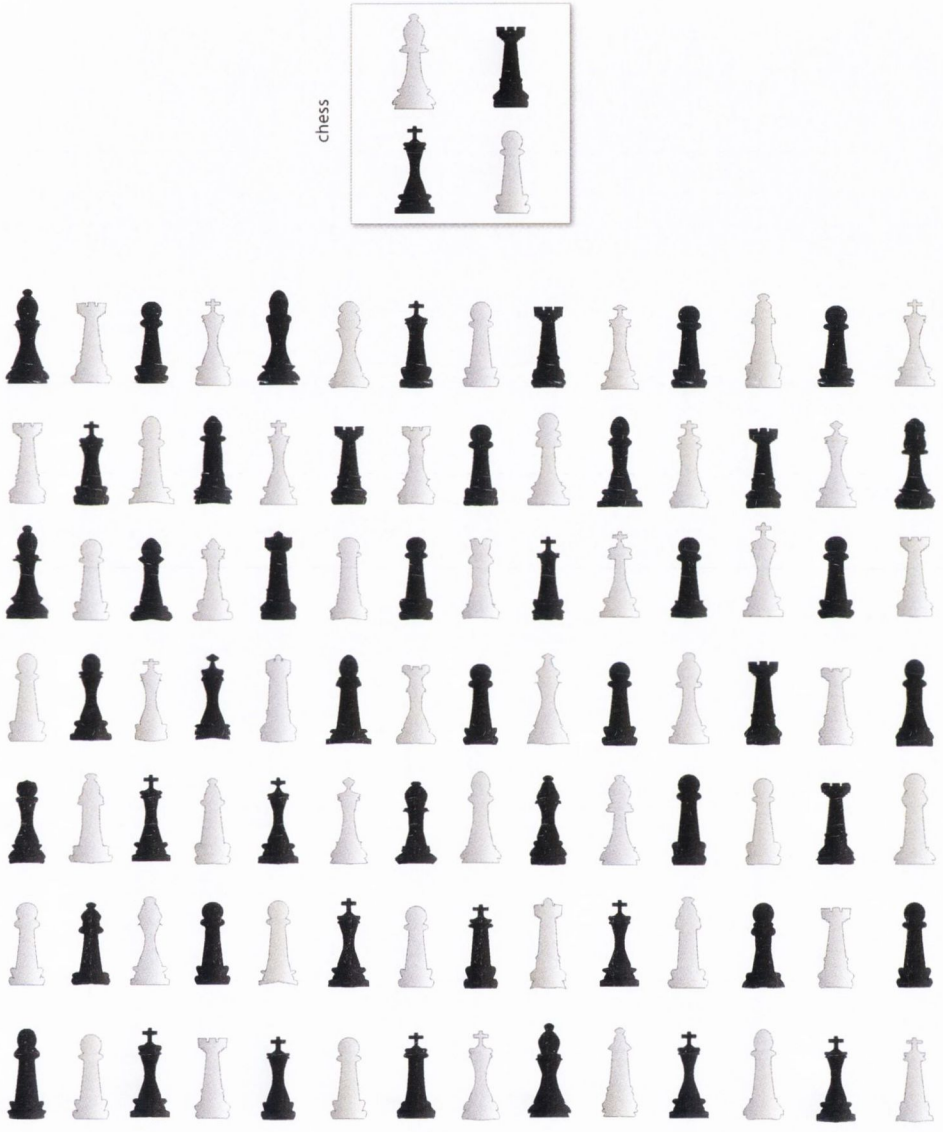


Figure B.8: Supplemental Chess Results. *These 98 results were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

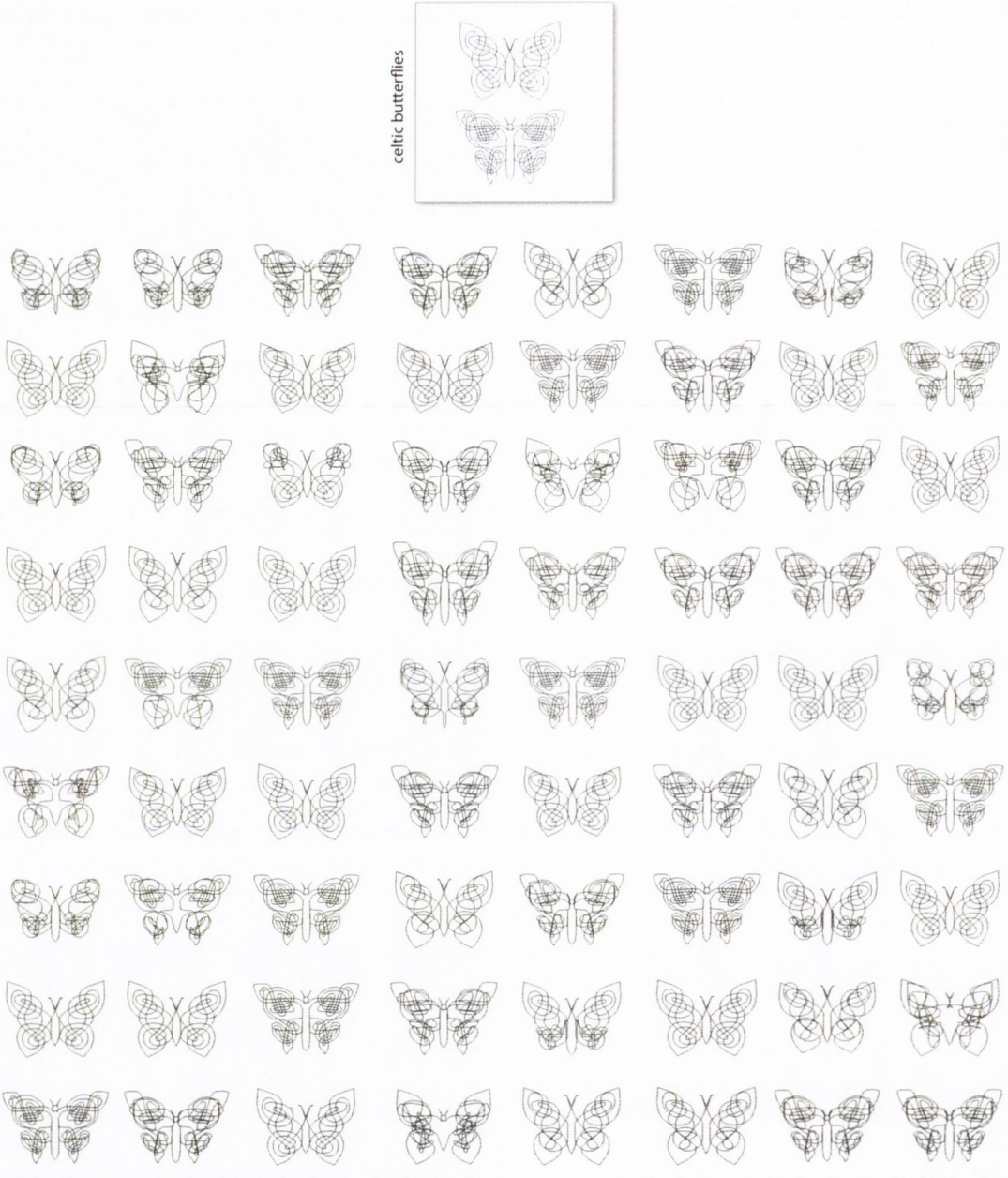


Figure B.9: Supplemental Celtic Butterfly Results. *These 72 results were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

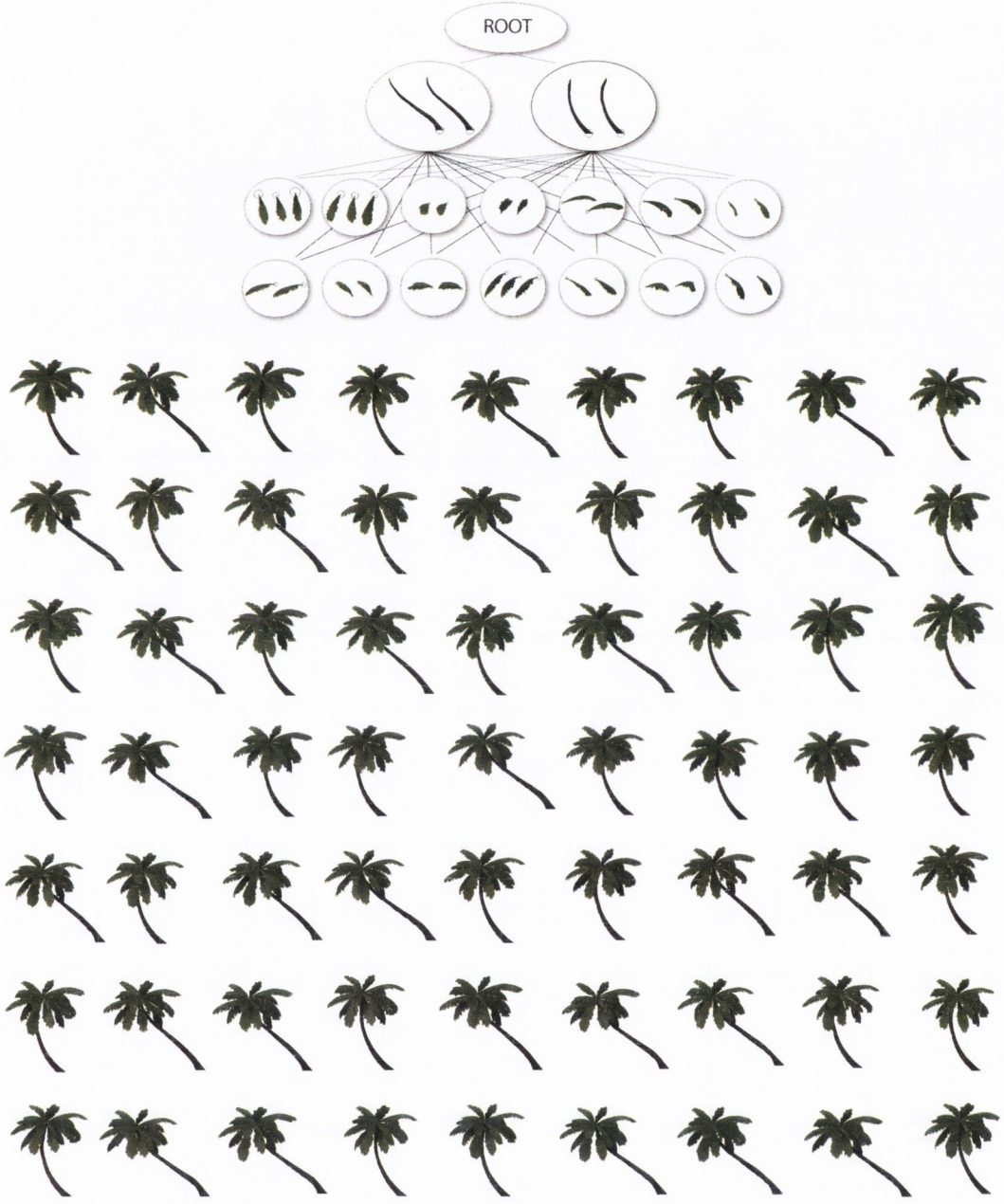


Figure B.10: Supplemental Palm Tree Graph Results. *These 63 multi-layered palm trees were generated at random from the corresponding exemplar graph (top). These results were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

Figure B.11: Supplemental Pirate Flag Graph Results. *These 98 multi-layered pirate flags were generated at random from the corresponding exemplar graph (top). These results were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

Appendix C

Mesh Hybrids

C.1 Semi-Regular Remeshing

C.1.1 Related Works

Traditionally remeshing has been a topic of considerable study in computer graphics. There are many approaches for acquiring 3D shape data. Either through laser scanners, photometric vision techniques, procedural methods, or most commonly human generated art. Regardless of which approach is used, we rarely find that the input shape is in a format that is convenient for compression, storage, transmission, level-of-detail control, or rendering. In our case we find arbitrary meshes unsuitable for multilevel simplification, which is necessary for the geometry hybrid approach.

Regular Remeshing

While semi-regular meshes have advantages for many graphics applications, a completely regular remeshing would yield the greatest benefits for compression, storage, transmission, level-of-detail control and even rendering. In our case a completely regular remeshing would be attractive as it could be processed into

a Gaussian Stack [LH05], instead of a Gaussian Pyramid. Unfortunately regular remeshing is not appropriate for hybridisation because it creates a severely anisotropic remeshing that is not deterministic to the shape of the input mesh. In other words, two identical local regions in otherwise different meshes will produce severely different remeshed geometry and the local neighbourhoods will not correspond based on our hybridisation shape descriptor.

Isotropy vs. Anisotropy

Geometry hybridisation requires further consideration about the nature of remeshing beyond the topological connections. Because the approach compares local shape neighbourhoods, it is important that given two similarly shaped meshes, the remeshing scheme produces either an isotropic remeshing where all polygon edge lengths and internal angles are roughly equal, or if any anisotropy is permitted, then it is guaranteed to be the same for similar shapes between the two remeshings. This requirement is the primary reason why the regular remeshing approach described above is unsuitable for geometry hybridisation. In their pursuit of a completely regular remapping of an arbitrary mesh, their sampling becomes extremely uneven in largely unpredictable ways, resulting in very high and non geometrically deterministic anisotropy. At the moment a Semi-Regular remeshing approach is necessary.

Semi-Regular Remeshing

An early approach for remeshing an arbitrary mesh into a semi-regular representation used Voronoi diagrams in conjunction with harmonic maps [ME95]. This approach is attractive due to its connection to the mathematical foundation of wavelets. Another strategy [LSS*98] uses hierarchical simplification in a fine-to-coarse manner to build a semi-regular representation from an arbitrary input.

A novel approach [KVLpS99] finds semi-regular remeshing through the process

of shrink wrapping, whereby they approximate a plastic membrane with a semi-regular triangle mesh and simulate shrinking it around the original mesh through an optimisation of two forces. One is an attraction force between the membrane and the original model, and the other is a relaxation force that reduces distortion. Another approach leverages recent work in texture atlas map creation [PS03] to transform an arbitrary mesh into a semi-regular mapping of image patches. Building on the idea of using a parameterization mapping, a recent approach [Gus07] builds a semi-regular mesh by first breaking an arbitrary mesh up into charts, and then finding a smooth parameterization with respect to the differential structure built on the base domain.

Two concurrent approaches start with a coarse base mesh and iteratively subdivide the geometry [GVSS00; LMH00]. After splitting an edge the new vertex position is found by casting a ray along the surface normal into the target mesh. While our approach is not based on ray casting and surface displacement, we adopt the overall subdivision strategy, and spirit behind these approaches.

C.1.2 Method

Our approach takes a manifold in the form of an input mesh M. It then generates a multi-resolution mesh pyramid starting from some coarsest mesh M^0 by iteratively splitting all edges and subdividing each polygon into four new polygons as Illustrated in figure C.1. This process is known as upsampling. The resulting semi-regular mesh pyramid mimics the shape of the input mesh M at each level l of resolution from level 0 up to some level L.

Having described the input and output, as well as the topological nature of the mesh pyramid, we tackle the real heart of the problem. How does one first create the unregular base mesh M^0 So that it effectively creates a coarsest approximation of the shape given by the input mesh M? Second, once we have a base mesh M^0, how do we determine where new vertices created through subdivision should be

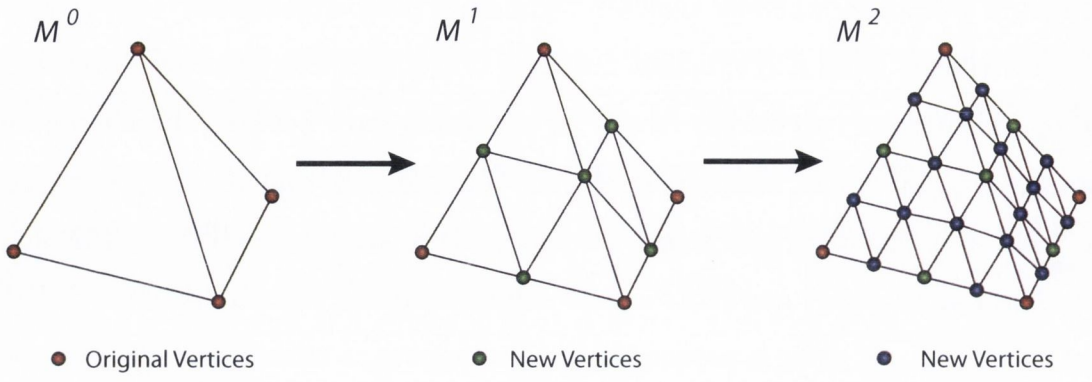


Figure C.1: Multiresolution Topology. *Starting from this example base mesh M^0, M^1 is created by splitting the edges of M^0 resulting in new vertices, which subdivide each of the previous levels polygons into four new polygons. The highest resolution mesh M^L contains the vertices from M^0 along with the new vertices created through subdivision. This is known as a semi-regular mesh because the vertices in mesh M^L that come from M^0 retain their original valence (this is the irregular coarsest foundation), while all new vertices introduced through subdivision have a valence of six and form a regular triangular tiling.*

placed? The rest of this section is organised into two subsections that tackle each of the questions just asked.

C.1.2.1 Creating Base Mesh

Before the geometry can be subdivided, it is first necessary to find some arbitrary polyhedral mesh that approximates the fundamental shape of the input mesh M using the least amount of geometry possible. It is necessary for this mesh to be as low polygon as possible because it acts as our coarsest starting point for synthesis. We can not hybridize features that are coarser than this mesh.

A base mesh is needed with the same genus as the target shape. In the case of genus 0 meshes we found that a tetrahedron, was generally sufficient to reproduce the shape. However a tetrahedron will naturally lead to a highly anisotropic

remeshing given a target with long skinny protrusions, such as the ears on the Stanford bunny model. In order to achieve a more isotropic remeshing it would be desirable to start with a base mesh that more closely approximates the shape.

We offer no new ideas on finding this base mesh as several appropriate methods already exist [WHDS04; Hop96]. For simplicity we hand modeled all of the base meshes seen in this thesis.

C.1.2.2 Subdivision

Our strategy follows Normal Meshes [GVSS00] in spirit. Given a base mesh, we split all the edges and find a new vertex for each edge that best approximates the geodesic middle point along the input mesh who's shape we are trying to reproduce. In the Normal Meshes approach, Each polygon is subdivided into four new polygons and the newly created vertices are ray casted along their normal direction into the original mesh. Their positions are updated to the ray/mesh intersection points. This is a simple yet powerful technique. It's easy to implement, memory efficient and produces a very even remeshing. Normal Meshes does have its limitations though. it requires that the base mesh closely approximates the full resolution shape so that rays are intersected with relatively smooth geometry and normal meshes generally produces poor remeshing if the ray intersects with two polygons along its path. The original Normal Meshes paper offers a secondary strategy for fixing artifacts cased by the simplicity of the method, however this secondary strategy is complicated and just patches over the core problem - that a ray cast based remeshing does not understand the full shape of the original mesh and in facts only looks at a very small sliver of the mesh shape information available.

Our approach was devised by analyzing why Normal Meshes works well in most cases and why it fails in others. We believe that Normal Meshes is a powerful technique because it evenly splits an edge between two vertices into two equally

lengthed edges. By only updating the split edge vertex along its normal, this even edge length is mostly preserved. This approximates the geodesic midpoint on the original mesh with respect to the two vertices defining the original split edge. This approximation is in fact very good in the case that the original mesh is relatively smooth between the two vertices defining the original edge split.

We believe that Normal Meshes fails in some cases because it only approximates finding the geodesic middle point between the edges parent vertices and for complex geometry this can lead to poor approximations. As such Our approach mimics the basic strategy where each polygon is subdivided into four and positions for the new vertices are found on the original mesh. The difference is that we find this point by traversing along the surface of the mesh itself to get a true geodesic middle, rather than trying to approximate the geodesic middle by doing a raycast intersection test. This avoids the danger of not finding an intersection, finding more than one intersection and finding intersections which introduce folds in the remeshed geometry that are not present in the original mesh.

While our approach does not benefit from the memory savings made by the normal meshes approach, it does produce better results and is more flexible and forgiving with the shapes it can remesh.

This approach operates in two passes. In the first pass each edge in the current remesh level is split. A new position is found on the original mesh for the split edge vertex and a list of other potential position candidates is stored for each new vertex. The second pass is similar to the improved solver introduced in Chapter 5. Better positions are found for each vertex from their list of possible positions stored from the first pass. The positions of these vertices are updated through an iterative optimisation scheme.

Figure C.2: Region Coding. *Left: distance along the surface of the mesh to each of the base vertices. Middle: Voronoi regions around each base vertex. Right: marked in red, geodesic equal distance points between edge (black line) endpoints.*

First Pass

The first pass splits each edge in the current remeshed model, producing the next level of the remeshed model. The positions for the newly created vertices are initially set to be the midpoint of the split edge. A copy of this model is stored for later and will be referred to as E_r. For each split edge, a geodesic border is found on the original mesh between the two end points of the original edge before it was split. A geodesic border is a continuous chain of vertices along the surface of the original mesh which have the same distance along the surface of the original mesh to each of the edge vertices. This is illustrated in figure C.2 which shows the Voronoi cells on the ogre mesh around six vertices comprising a tetrahedral base mesh. On the left the colours correspond to the closest geodesic distance to one of these six vertices. The colours on the middle image show the Voronoi cells around each of those base vertices. The image on the right highlights in red the chain of geodesic border vertices which are equally distant to each of the edge vertices along the surface of the mesh. The edge to be split is illustrated with a black line. Once this list of border vertices is found, the newly created split edge vertex is initially set to the position of the border vertex with the lowest distance to each

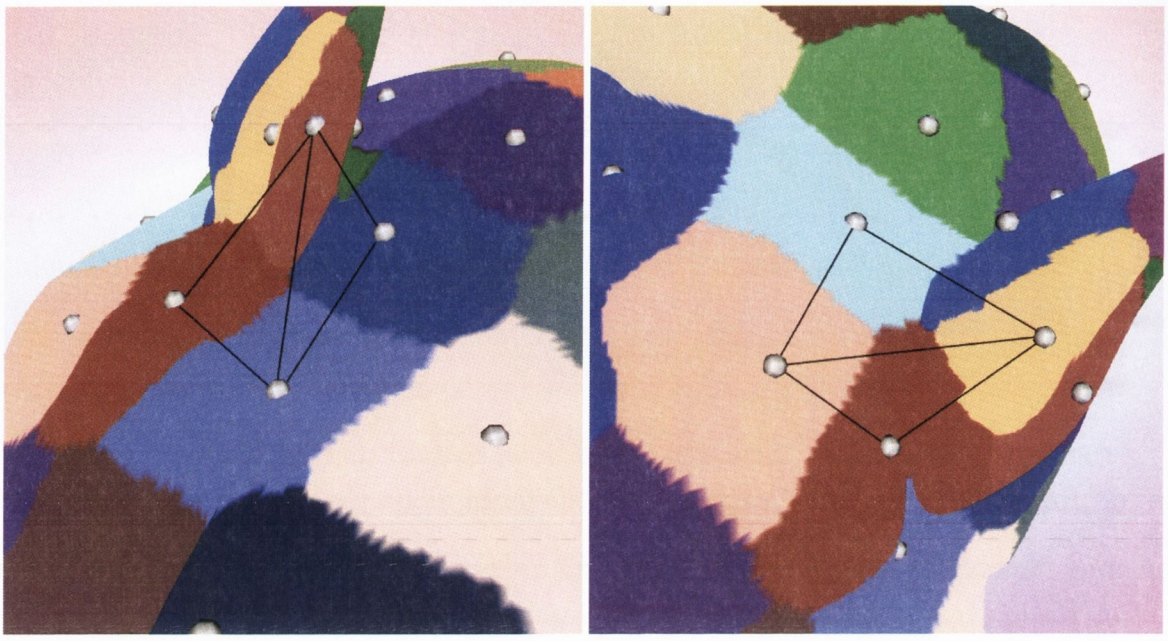


Figure C.3: Special Case. *If no border vertices exist between the Voronoi cells of an edges endpoints, this means there is a special case and further testing is necessary.*

of the edge ends resulting in a new mesh shape we call S_r. The list of its border points are stored for each split edge vertex.

In the special case that the Voronoi mapping does not yield border vertices for a particular edge (as seen in figure C.3), or yields too few border vertices based on some user set minimum, an alternative strategy is invoked. In this case a new geodesic distance map is created for the two edge vertices only, and the rest of the vertices of the remeshed model are left out. This is a slow operation but always yields lots of border candidates.

Second Pass

In order to produce a good remeshing we have two strategies. The first is to split each edge along its geodesic border, such that the split edge vertices are always equally distance along the surface of the mesh to each of the original edge vertices.

The second strategy is to maintain the edge distances on the remeshed model S_r to reflect those edge distances from the initial upscaled model E_r generated earlier. The reasoning behind this is that the four new polygons created by splitting the edges of the parent polygon will have the same shape properties as the parent polygon, which will lead to a very uniform spread of remeshed polygons.

As such the second pass mimics the improved solver presented in Section 5.4.1. In each solving iteration, each newly created vertex's position is updated so that the shape of the finer resolution polygons it contributes to more closely resembles the parent polygon they are upsampled from. For each vertex this is done by testing each of the adjacent positions in the border chain as well as the current one and moving to that which minimizes the total squared difference in edge lengths between the edges around the vertex in the example model E_r and the model being solved S_r. This process is repeated until the rate of change of the global energy falls below a certain threshold. Note that this process uses the same energy for calculation for distance as the improved solver, but does not use the angular power term. Also, whereas in the improved solver, the vertex would be moved along the optimal direction to minimize the total energy of the system, in this case the vertex moves to neighbouring vertices in the original mesh, and only to vertices in a continuous chain along the geodesic border determined in the first pass.

C.1.3 Results

The results of this approach are presented in figure C.4. In general we find this approach to work reasonably well for our purposes. The results appear to maintain the key strengths found in the Normal Meshes approach while avoiding the weaknesses.

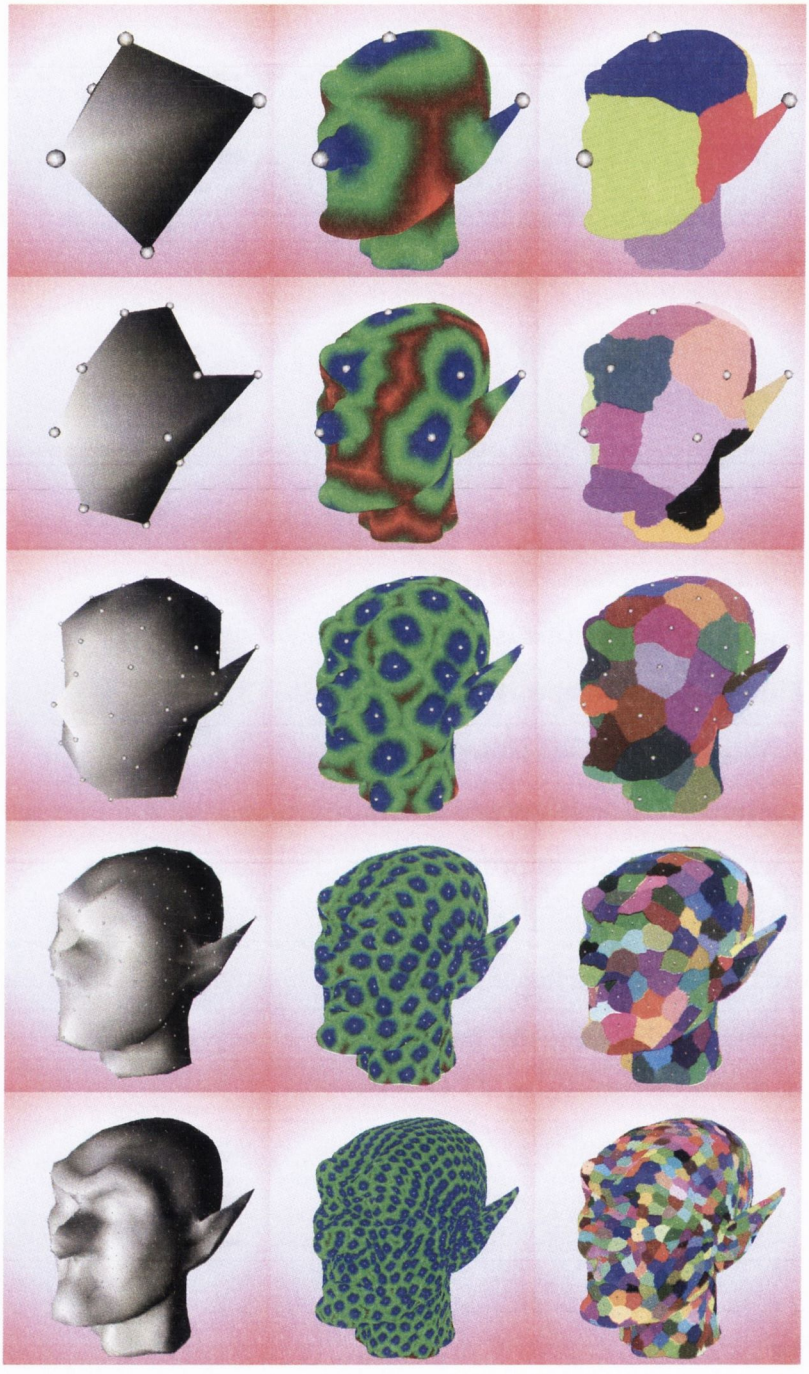


Figure C.4: Results. *Starting from a tetrahedral as the base geometry, the mesh is up-sampled four times. The left column shows the remeshed model, the middle column shows distances between vertices and the right column shows Voronoi regions around each of the vertices.*

C.2 Supplemental Results

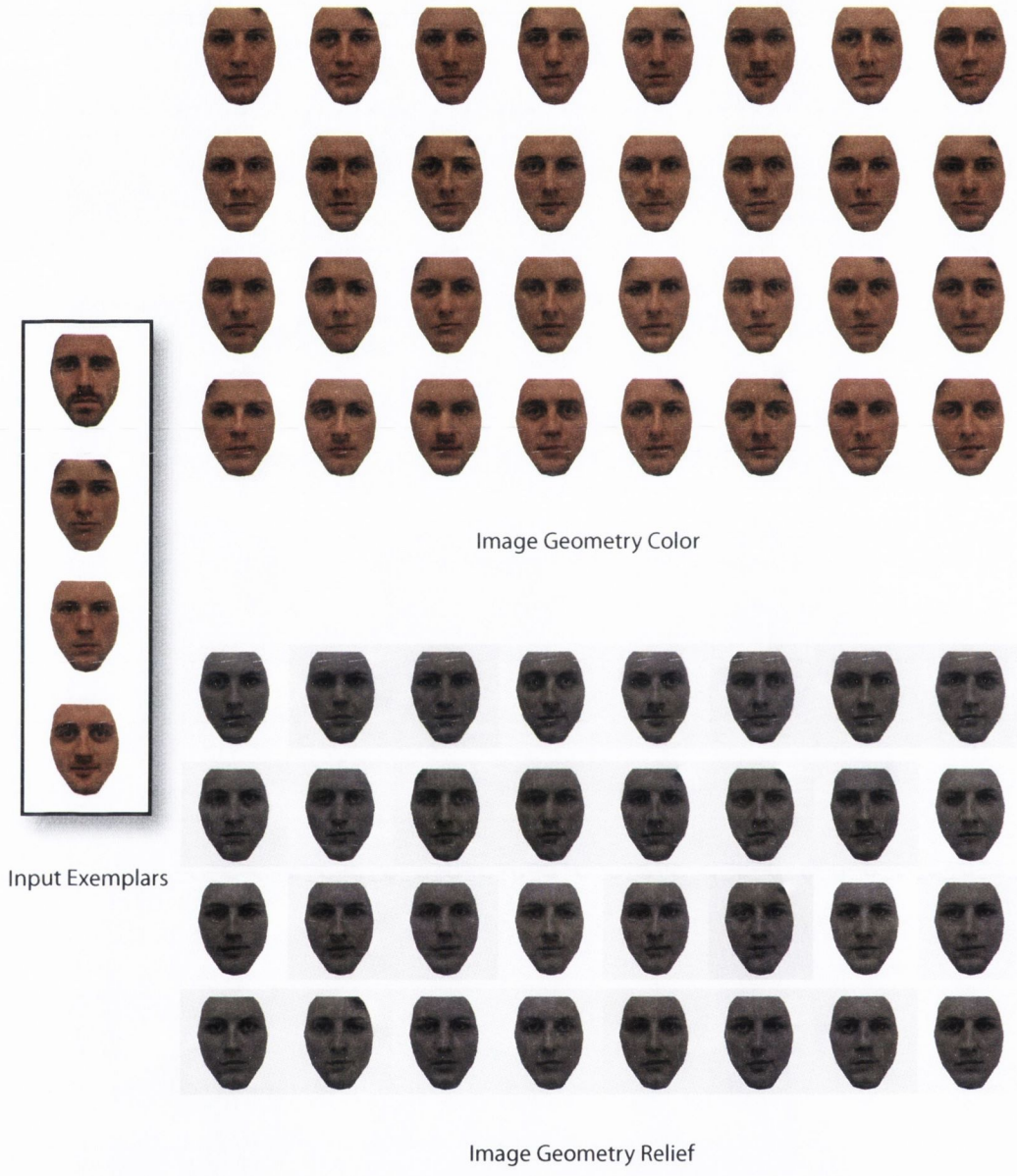


Figure C.5: Supplemental Faces Results. *The top set of images were taken directly from the image geometry technique using the colour approach. The bottom set of images were generated using the relief approach. These 32 results for each approach were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

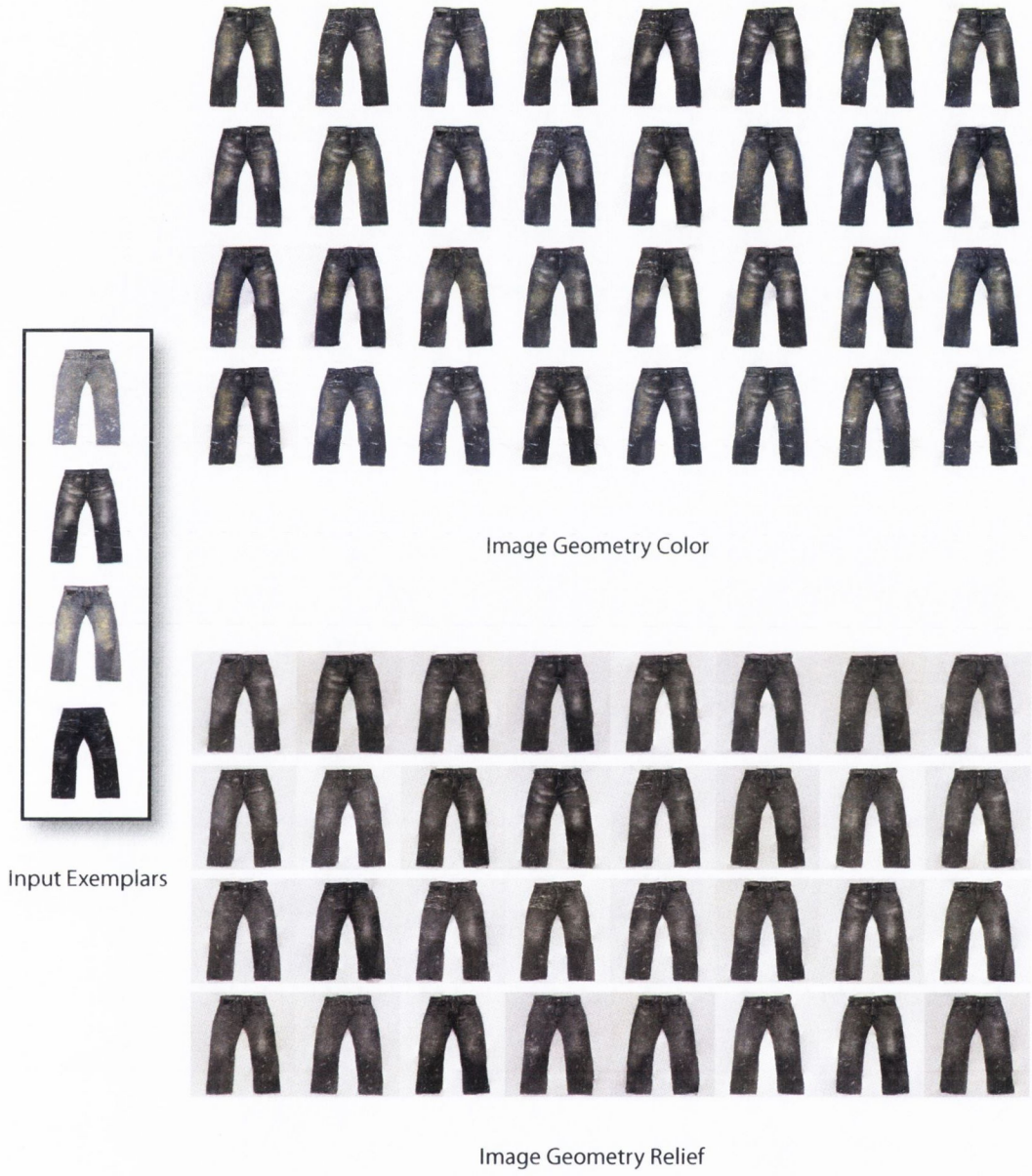


Figure C.6: Supplemental Jeans Results. *The top set of images were taken directly from the image geometry technique using the colour approach. The bottom set of images were generated using the relief approach. These 32 results for each approach were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

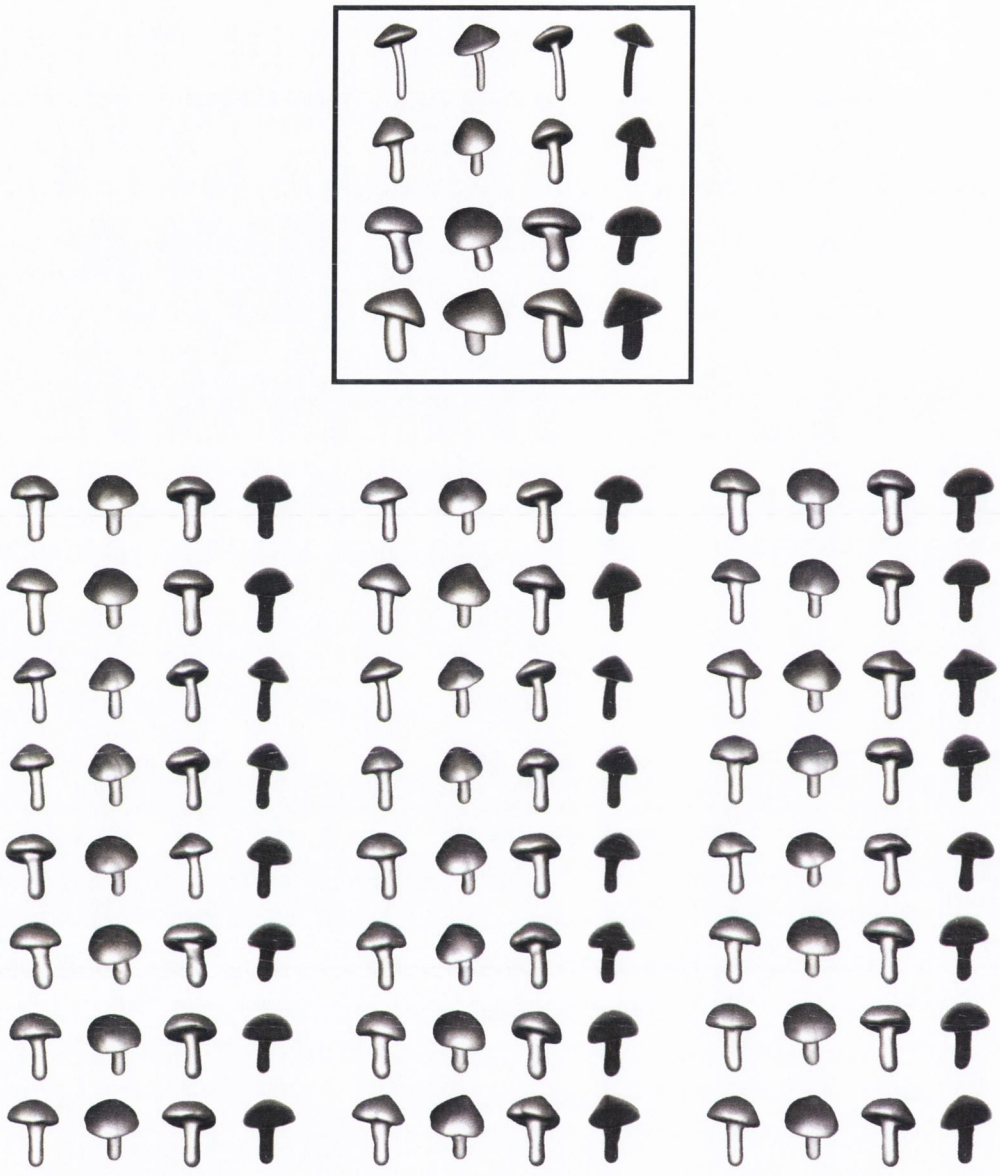


Figure C.7: Supplemental Mushroom Results. *The four input exemplars are given in the top box, the other models are synthesised. Notice that the output exemplars share the overall smoothness of the input and no sharp edge artifacts are added at borders between two patches from different exemplars. These 24 results were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

Figure C.8: Supplemental Marine Results. *This example uses bilateral symmetry and exemplars with different base meshes. The four input exemplars are given in the top box, the other models are synthesised. These 21 results were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

Figure C.9: Supplemental Head Results. *Given a human and ogre mesh, widely varying hybrids are produced. This example utilizes bilateral symmetry. The input exemplars are given in the top left box, the other heads are synthesised. These 27 results were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

Figure C.10: Supplemental High Resolution Head Results. *This example expands on the previous by non-uniformly upsampling the exemplar pyramids. These 25 results were not hand-picked, they are representative of an average set of outputs one could expect for this example.*

Bibliography

[AFY04] Alon I. D., Fishbach A., Yeshurun Y.: Spectral sound gap filling, 2004.

[AL05] Ares Lagae Olivier Dumont P. D.: Geometry synthesis by example. In *Proceedings of the International Conference on Shape Modeling and Applications 2005* (2005), pp. 176 – 185.

[APK04] A. C., P. P., K. T.: Region filling and object removal by exemplar-based image inpainting. In *IP* (2004), pp. 1200–1212.

[Ash01] Ashikhmin M.: Synthesizing natural textures. In *SI3D* (2001), pp. 217–226.

[BaBGF10] Barnes C., andDan B Goldman E. S., Finkelstein A.: The generalized PatchMatch correspondence algorithm. In *European Conference on Computer Vision* (Sept. 2010).

[BBT*06] Barla P., Breslav S., Thollot J., Sillion F., Markosian L.: Stroke pattern analysis and synthesis. In *Computer Graphics Forum (Proc. of Eurographics 2006)* (2006), vol. 25.

[BFK03] Brendan N. J., Frey B. J., Kannan A.: Epitomic analysis of appearance and shape. In *In ICCV* (2003), pp. 34–41.

[BJEYLW01] Bar-Joseph Z., El-Yaniv R., Lischinski D., Werman M.: Texture mixing and texture movie synthesis using statistical learning. *IEEE TVCG 7*, 2 (2001), 120–135.

[BMAK10] Beneš B., Měch R., Aliaga D. G., Kristof P.: Inverse procedural modeling by automatic generation of l-systems. *Computer Graphics Forum 29*, 3 (2010), 665–674.

[BP04] Bhat P. Ingram S. T. G.: Geometric texture synthesis by example. In *Eurographics* (2004), pp. 41–44.

[BPGK06] Botsch M., Pauly M., Gross M., Kobbelt L.: Primo: Coupled prisms for intuitive surface modeling. In *In SGP06: Proceedings of the Fourth Eurographics Symposium on Geometry Processing* (2006), pp. 11–20.

[BSFG09] Barnes C., Shechtman E., Finkelstein A., Goldman D. B.: PatchMatch: A randomized correspondence algorithm for structural image editing. In *SIGGRAPH* (2009), p. 3.

[BWW10] Bokeloh M., Wand M., Wand M.: A connection between partial symmetry and inverse procedural modeling. In *SIGGRAPH* (2010).

[CK10] Chaudhuri S., Koltun V.: Data-driven suggestions for creativity support in 3d modeling. *ACM Trans. Graph. 29*, 6 (Dec. 2010), 183:1–183:10.

[CKGK] Chaudhuri S., Kalogerakis E., Guibas L., Koltun V.: Probabilistic reasoning for assembly-based 3D modeling. *ACM Transactions on Graphics (Proc. SIGGRAPH) 30.*

[COLS98] Cohen-Or D., Levin D., Solomivici A.: Three dimensional distance field metamorphosis. *Trans. on Graphics 17*, 2 (1998), 116–141.

[cZLW00] chun Zhu S., Liu X. W., Wu Y. N.: Exploring texture enesmbles by efficient markov chain monte carlo — toward a "trichromacy" theory of texture. In *IEEE Trans. on Pattern Analysis and Machine Intelligence* (2000), pp. 554–569.

[cZM06] chun Zhu S., Mumford D.: A stochastic grammar of images. In *Foundations and Trends in Computer Graphics and Vision* (2006), Now Publishers, pp. 259–362.

[DDEY*02] Dubnov S., Dubnov S., El-Yaniv R., Lischinski D., Werman M.: Synthesizing sound textures through wavelet tree learning. In *Computer Graphics and Applications, IEEE* (2002), pp. 38–48.

[DeB97] DeBonet J.: Multiresolution sampling procedure for analysis and synthesis of texture images. In *SIGGRAPH* (1997), pp. 361–368.

[DI03] Drori I. Cohen-Or D. Y. H.: Fragment-based image completion. In *SIGGRAPH* (2003), pp. 303–312.

[DTT08] Davies R., Twining C., Taylor C.: *Statistical Models of Shape. Optimisation and Evaluation.* 2008.

[DZ06] Dischler J.-M., Zara F.: Real-time structured texture synthesis and editing using image-mesh analogies. *The Visual Computer 22*, 9 (2006), 926–935.

[EF01] Efros A., Freeman W.: Image quilting for texture synthesis and transfer. In *SIGGRAPH* (August 2001), pp. 341–346.

[EL99] Efros A., Leung T.: Texture synthesis by non-parametric sampling. In *ICCV* (September 1999), pp. 1033–1038.

[FKS*04] Funkhouser T., Kazhdan M., Shilane P., Min P., Kiefer W., Tal A., Rusinkiewicz S., Dobkin D.: Modeling by example. *ACM Transactions on Graphics 23* (August 2004), 652–663.

[FRS*12] Fisher M., Ritchie D., Savva M., Funkhouser T., Hanrahan P.: Example-based synthesis of 3d object arrangements. In *ACM SIGGRAPH Asia 2012 papers* (2012), SIGGRAPH Asia '12.

[FS94] Finkelstein A., Salesin D. H.: Multiresolution curves. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), ACM Press, pp. 261–268.

[GB00] Guo B. Shum H. X. Y.: Chaos mosaic: Fast and memory efficient texture synthesis. In *Tech. Rep. MSR-TR-2000-32, Microsoft Research* (2000).

[gDCV] gomes J., Dorsa L., Costa B., Velho L.: *Warping and Morphing of Graphical Objects.*

[GGG03] Goldberger J., Gordon S., Greenspan H.: An efficient image similarity measure based on approximations of kl-divergence between two gaussian mixtures. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on* (2003), vol. 1, pp. 487–493.

[GK05] Gallagher C., Kokaram A.: Nonparametric wavelet based texture synthesis. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on* (2005), pp. 462–465.

[GSMCO09] Gal R., Sorkine O., Mitra N. J., Cohen-Or D.: iwires: An analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics (Siggraph) 28*, 3 (2009), #33, 1–10.

[Gus07] Guskov I.: Manifold-based approach to semi-regular remeshing. *Graphical Models 69* (2007), 1–18.

[GVSS00] Guskov I., Vidimce K., Sweldens W., Schrder P.: Normal meshes. In *SIGGRAPH* (2000), pp. 95–102.

[HB95] Heeger D. J., Bergen J. R.: Pyramid-based texture analysis/synthesis. In *SIGGRAPH* (1995), pp. 229–238.

[HH09] Haindl M., Hatka M.: Near-regular texture synthesis. In *Computer Analysis of Images and Patterns* (2009), pp. 1138–1145.

[HJO*01] Hertzmann A., Jacobs C. E., Oliver N., Curless B., Salesin D. H.: Image analogies. In *SIGGRAPH* (2001), pp. 327–340.

[HOCS02] Hertzmann A., Oliver N., Curless B., Seitz S. M.: Curve analogies. In *Proceedings of the 13th Eurographics workshop on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2002), Eurographics Association, pp. 233–246.

[Hop96] Hoppe H.: Progressive meshes. In *SIGGRAPH* (1996), pp. 99–108.

[HP01] Hoskinson R., Pai D.: Manipulation and resynthesis with natural grains. In *Proceedings of the International Computer Music Conference ICMC01* (2001).

[HRRG08] Han C., Risser E., Ramamoorthi R., Grinspun E.: Multiscale texture synthesis. In *SIGGRAPH* (2008), p. 51.

[Hug92] HUGHES J. F.: Scheduled fourier volume morphing. *Computer Graphics(Proc. SIGGRAPH 92)* (1992), 43–46.

[HV10] HELÉN M., VIRTANEN T.: Audio query by example using similarity measures between probability density functions of features. *EURASIP J. Audio Speech Music Process. 2010* (Jan. 2010), 1:1–1:9.

[HWK94] HE T., WANG S., KAUFMAN A.: Wavelet-based volume morphing. *Proc. IEEE Visualization '94* (1994), 85–92.

[IMIM08] IJIRI T., ME(CH R., IGARASHI T., MILLER G.: G.: An example-based procedural system for element arrangement. *Comput. Graph. Forum* (2008).

[JKH04] JEBARA T., KONDOR R., HOWARD A.: Probability product kernels. *Journal of Machine Learning Research, Special Topic on Learning Theory*, 5 (July 2004), 819–844.

[Jul81] JULESZ B.: Textons, the elements of texture perception and their interactions. *Nature 290* (1981), 91–97.

[KCKK12] KALOGERAKIS E., CHAUDHURI S., KOLLER D., KOLTUN V.: A Probabilistic Model of Component-Based Shape Synthesis. *ACM Transactions on Graphics 31*, 4 (2012).

[KEBK05] KWATRA V., ESSA I., BOBICK A. F., KWATRA N.: Texture optimization for example-based synthesis. In *SIGGRAPH* (2005), pp. 795–802.

[KFCO*07] KOPF J., FU C.-W., COHEN-OR D., DEUSSEN O., LISCHINSKI D., WONG T.-T.: Solid texture synthesis from 2d exemplars. In *SIGGRAPH* (2007), p. 2.

[kLmHGM05] K. Lai Y., m. Hu S., Gu D. X., Martin R. R.: R.r.: Geometric texture synthesis and transfer via geometry images. In *In: ACM Symposium on Solid and Physical Modeling* (2005), pp. 15–26.

[KP02] Kim J., Pellacini F.: Jigsaw image mosaics. In *ACM Transactions on Graphics* (2002), pp. 657–664.

[KSE*03] Kwatra V., Schodl A., Essa I., Turk G., Bobick A.: Graphcut textures: Image and video synthesis using graph cuts. In *SIGGRAPH* (2003), pp. 277–286.

[KVLpS99] Kobbelt L. P., Vorsatz J., Labsik U., peter Seidel H.: A shrink wrapping approach to remeshing polygonal surfaces. pp. 119–130.

[LC87] Lorensen W. E., Cline H. E.: Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics(Proc. SIGGRAPH 87)* (1987), 163–169.

[LDSS99] Lee A. W. F., Dobkin D., Sweldens W., Schrder P.: Multiresolution mesh morphing. In *PROCEEDINGS OF SIGGRAPH 99* (1999), pp. 343–350.

[LH05] Lefebvre S., Hoppe H.: Parallel controllable texture synthesis. In *SIGGRAPH* (2005), pp. 777–786.

[LH06] Lefebvre S., Hoppe H.: Appearance-space texture synthesis. In *SIGGRAPH* (2006), pp. 541–548.

[LL03] Liu Y., Lin W.-C.: Deformable texture: the irregular-regular-irregular cycle. In *The Third International Workshop on Texture Analysis and Synthesis* (2003).

[LLH04] Liu Y., Lin W.-C., Hays J.: Near-regular texture analysis and manipulation. In *SIGGRAPH* (2004), pp. 368–376.

[LLX*01] Liang L., Liu C., Xu Y., Guo B., Shum H.: *Real-Time Texture Synthesis By Patch-Based Sampling.* Tech. Rep. MSR-TR-2001-40, Microsoft Research, 2001.

[LMH00] Lee A., Moreton H., Hoppe H.: Displaced subdivision surfaces. In *SIGGRAPH* (2000), pp. 85–94.

[LSS*98] Lee A. W. F., Sweldens W., Schroeder P., Cowsar L., Dobkin D.: Maps: Multiresolution adaptive parameterization of surfaces. In *SIGGRAPH* (1998), pp. 95–104.

[LT02] Liu Y., Tsin Y.: The promise and the perils of near-regular texture. *INTERNATIONAL JOURNAL OF COMPUTER VISION 62* (2002), 1–2.

[LV98] Lazarus F., Verroust A.: Three-dimensional metamorphosis: a survey. *The Visual Computer 14*, 8/9 (1998), 373–389.

[LWX*09] Liu Y., Wang J., Xue S., Tong X., Kang S. B., Guo B.: Texture splicing. *Computer Graphics Forum 28*, 7 (2009), 1907–1915.

[LWZ04] Lu L., Wenyin L., Zhang H.-J.: Audio textures, 2004.

[mDMLG02] m. Dischler J., Maritaud K., Lvy B., Ghazanfarpour D.: Texture particles. *Computer Graphics Forum 21* (2002), 401–410.

[ME95] Matthias Eck Tony DeRose T. D. H. H. M. L. W. S.: Multiresolution analysis of arbitrary meshes. pp. 173–182.

[MGP06] Mitra N. J., Guibas L., Pauly M.: Partial and approximate symmetry detection for 3d geometry. In *ACM Transactions on Graphics* (2006), vol. 25, pp. 560–568.

[MGVC00] M. B., G. S., V. C., C. B.: Image inpaiting. In *SIGGRAPH* (2000), pp. 417–424.

[MP97] Moghaddam B., Pentland A.: Probabilistic visual learning for object representation. *PAMI 19*, 7 (1997).

[MPK09] Mohammed U., Prince S. J., Kautz J.: Visio-lization: generating novel facial images. In *SIGGRAPH* (2009), p. 57.

[MWT11] Ma C., Wei L.-Y., Tong X.: Discrete element textures. *ACM Trans. Graph. 30*, 4 (Aug. 2011), 62:1–62:10.

[MZD05] Matusik W., Zwicker M., Durand F.: Texture design using a simplicial complex of morphable textures. In *SIGGRAPH* (2005), pp. 787–794.

[New07] Newsam S.: Hierarchical texture motifs. In *SPIE International Symposium on Electronic ImagingImage Processing: Algorithms and Systems VI* (2007).

[NM04] Newsam S., Manjunath B. S.: Normalized texture motifs and their application to statistical object modeling. In *IEEE International Conference on Computer Vision and Pattern Recognition: Workshop on Perceptual Organization in Computer vision* (2004).

[NP96] Navarro R., Portilla J.: Robust method for texture synthesis-by-analysis based on a multiscale gabor scheme. In *In Proc. of the SPIE* (1996), pp. 86–97.

[OK07] ORegan D., Kokaram A.: Multi-resolution sound texture synthesis using the dual-tree complex wavelet transform. In *In Proceedings of the 15th European Signal Processing Conference* (2007).

[Per85] Perlin K.: An image synthesizer. In *SIGGRAPH* (1985), pp. 287–296.

[PKD07] Pitié F., Kokaram A. C., Dahyot R.: Automated colour grading using colour distribution transfer. *CVIU 107*, 1-2 (2007), 123–137.

[PLM98] Paget R., Longstaff I. D., Member S.: Texture synthesis via a noncausal nonparametric multiscale markov random field. In *Image Processing, IEEE Transactions on* (1998).

[PMW*08] Pauly M., Mitra N. J., Wallner J., Pottmann H., Guibas L.: Discovering structural regularity in 3D geometry. *ACM Transactions on Graphics 27*, 3 (2008), #43, 1–11.

[PNN*96] Portilla J., Navarro R., Navarro R., Nestares O., Tebernero A.: Texture synthesis-by-analysis based on a multiscale earlyvision model, 1996.

[PS00] Portilla J., Simoncelli E.: A parametric texture model based on joint statistics of complex wavelet coefficients. *IJCV 40*, 1 (2000), 49–70.

[PS03] Pedro Sander Zoe Wood S. G. J. S. H. H.: Multi-chart geometry images. In *Proceedings Symposium on Geometry Processing* (2003), pp. 146–155.

[RB07] Ramanarayanan G., Bala K.: Constrained texture synthesis via energy minimization. *IEEE TVCG 13*, 1 (2007), 167–178.

[RCOL09] Rosenberger A., Cohen-Or D., Lischinski D.: Layered shape synthesis: automatic generation of control maps for non-stationary textures. In *SIGGRAPH Asia* (2009), pp. 1–9.

[RHDG10] Risser E., Han C., Dahyot R., Grinspun E.: Synthesizing structured image hybrids. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2010) 29*, 4 (2010), 85:1–85:6.

[SaP95] Saint-arnaud N., Popat K.: Analysis and synthesis of sound textures. In *in Readings in Computational Auditory Scene Analysis* (1995), pp. 125–131.

[TZL*02] Tong X., Zhang J., Liu L., Wang X., Guo B., Shum H.-Y.: Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH* (2002), pp. 665–672.

[WHDS04] Wood Z., Hoppe H., Desbrun M., Shroeder P.: Removing excess topology from isosurfaces. *ACM Transactions on Graphics 23* (2004), 190–208.

[WHZ*08] Wei L.-Y., Han J., Zhou K., Bao H., Guo B., Shum H.-Y.: Inverse texture synthesis. In *SIGGRAPH* (2008), p. 52.

[WK91] Witkin A., Kass M.: Reaction-diffusion textures. In *SIGGRAPH* (1991), pp. 299–308.

[WL00] Wei L., Levoy M.: Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH* (July 2000), pp. 355–360.

[WL01] Wei L., Levoy M.: Texture synthesis over arbitrary manifold surfaces. In *SIGGRAPH* (August 2001), pp. 355–360.

[WL02] Wei L., Levoy M.: *Order-Independent Texture Synthesis.* Tech. Rep. TR-2002-01, Stanford University CS Dept., 2002.

[Wor96] Worley S. P.: A cellular texture basis function. In *SIGGRAPH* (1996), pp. 291–294.

[WSBD00] Wood Z. J., Schröder P., Breen D., Desbrun M.: Semi-regular mesh extraction from volumes. In *Proceedings of the conference on Visualization '00* (Los Alamitos, CA, USA, 2000), VIS '00, IEEE Computer Society Press, pp. 275–282.

[WY04] Wu Q., Yu Y.: Feature matching and deformation for texture synthesis. In *SIGGRAPH* (2004), pp. 364–367.

[WYZG09] Wang L., Yu Y., Zhou K., Guo B.: Example-based hair geometry synthesis. In *SIGGRAPH* (2009).

[WZ02] Wang Y., Zhu S.: A generative method for textured motion: Analysis and synthesis. In *Proc. European Conf. on Computer Vision* (Copenhagen, June 2002).

[XZCOC] Xu K., Zhang H., Cohen-Or D., Chen B.: Fit and diverse: Set evolution for inspiring 3d shape galleries.

[YW07] Yonatan Wexler Eli Shechtman M. I.: Space-time completion of video. In *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2007), pp. 463–476.

[ZCFT09] Zhu L., Chen Y., Freeman W., Torralba A.: Nonparametric bayesian texture learning and synthesis. In *Advances in Neural Information Processing Systems* (2009).

[ZFCO*11] Zheng Y., Fu H., Cohen-Or D., Au O. K.-C., Tai C.-L.: Component-wise controllers for structure-preserving shape manipulation. In *Computer Graphics Forum (In Proc. of Eurographics 2011)* (2011), vol. 30, p. to appear.

[ZG02] Zelinka S., Garland M.: Towards real-time texture synthesis with the jump map. In *EGWR* (2002), pp. 99–104.

[ZGWX05] Zhu S. C., Guo C. E., Wang Y., Xu Z.: What are textons? *International Journal of Computer Vision 62*, 1 (2005), 121–143.

[ZHW*06] Zhou K., Huang X., Wang X., Tong Y., Desbrun M., Guo B., Shum H.-Y.: Mesh quilting for geometric texture synthesis.

In *ACM Transactions on Graphics (TOG) archive* (2006), vol. 25, pp. 690 – 697.

[ZM98] Zhu S. C., Mumford D.: Filters, random fields and maximum entropy (FRAME)–towards a unified theory for texture modeling. *IJCV 27*, 2 (1998), 107–126.

[ZRDM01] Z. B.-J., R. E.-Y., D. L., M. W.: Texture mixing and texture movie synthesis using statistical learning. *IEEE TVCG 7* (2001), 120–135.

[ZS04] Zelinka S. G. M.: Jump map-based interactive texture synthesis. In *ACM Transactions on Graphics 23, 4* (2004), pp. 930–962.