Edge-preserving Texture Transfer Using the Laplacian Pyramid

by

Lars Doyle, **B.A., B.C.S.**

A thesis submitted to the
Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science
The School of Computer Science
Carleton University
Ottawa, Ontario
September, 2017

The undersigned hereby recommends to the
Faculty of Graduate and Postdoctoral Affairs
acceptance of the thesis

Edge-preserving Texture Transfer Using the Laplacian Pyramid

submitted by **Lars Doyle**, **B.A., B.C.S.**

in partial fulfillment of the requirements for the degree of

Master of Computer Science

Professor David Mould, Thesis Supervisor

Professor Oliver van Kaick, School of Computer Science

Professor WonSook Lee,
School of Electrical Engineering and Computer Science

Professor Robert Biddle, Chair,
School of Computer Science

Ottawa-Carleton Institute for Computer Science
The School of Computer Science
Carleton University
September, 2017

Abstract

Texture plays an important role in our perception of photographic images. It can be used to differentiate objects, draw the eye to salient regions, and convey information about a subject. However, post processing photographs in image editing software can be time consuming, often requiring advanced skills to achieve convincing effects. We introduce a multi-scale method, based on the Laplacian pyramid, to introduce auxiliary textures into photographic images. The final result retains the structural characteristics from the input, including edges, colour, and high-contrast existing texture, while enhancing the image with fine-scale details. In addition, we extend patch-based texture synthesis to include a guidance channel so that texture structures are aligned with an orientation field, obtained through the image structure tensor. Both the original structure tensor calculation and its subsequent smoothing are aided with geodesic knowledge so that our orientation fields are both edge preserving and smooth.

Acknowledgments

First and foremost, I would like to thank my thesis advisor, Dr. David Mould. He has had a huge influence on my approach to computer science research. His keen eye, insights, and advice have all been essential in the development of this work.

I would like to thank the thesis committee for reviewing my work and giving me valuable suggestions. Their advice has made this thesis stronger. I acknowledge Carleton University and the GIGL lab for their financial support. Thanks also goes out to my friends and colleagues in the GIGL lab.

We used many images from Flickr under a Creative Commons license. Thanks to the many talented photographers who provided the material.

Finally, I give special thanks to my wife Janis whose enormous encouragement and support has made this all possible.

Table of Contents

Abstract **iii**

Acknowledgments **iv**

Table of Contents **v**

List of Tables **viii**

List of Figures **ix**

1 Introduction **1**

1.1 Problem Statement 3

1.2 Contribution 3

1.3 Overview 4

2 Previous Work **5**

2.1 Introduction 5

2.2 Multi-scale Image Stylization 6

2.2.1 Image Pyramids 6

2.2.2 Pyramid-based Image Stylization 8

2.3 Example-based Texture Synthesis 11

2.3.1 Pixel-based 12

2.3.2 Patch-based 15

2.3.3 Optimization-based 18

2.3.4 Fast Nearest Neighbour Search 19

2.4 Texture Transfer 21

2.4.1 Photographic Images 21

2.4.2 Synthetic Images 26

2.5 Gradient Domain Image Editing 27
2.6 Edge-aware Filters 29

3 Algorithms **32**
3.1 Overview 32
3.2 Pyramid-based texture transfer 33
3.2.1 Laplacian Pyramid coefficient mixing 34
3.2.2 Gradient-based dynamic range compression 36
3.3 Synthesizing textures 40
3.3.1 The Primary Method 40
3.3.2 Texture Synthesis on a direction field 49
3.3.3 Orientation Field construction 51

4 Results and Discussion **57**
4.1 Results: Using a single texture 59
4.1.1 Image blending 63
4.2 Results: Using spatial masks 63
4.2.1 User-defined masks 63
4.2.2 Automated masks 66
4.3 Dynamic Range Compression 68
4.4 Texture synthesis 71
4.5 Parameters 73
4.5.1 Laplacian pyramids 73
4.5.2 Texture Synthesis 75
4.6 Comparison 77
4.6.1 Comparison: Pixel-based texture transfer 79
4.6.2 Comparison: Patch-based texture transfer 79
4.6.3 Comparison: Alpha blending 81
4.7 Efficiency 81
4.8 Limitations 84
4.8.1 Non-stationary textures 84
4.8.2 Colour 84
4.8.3 Perspective foreshortening 86
4.9 Summary 86

5 Conclusion and Future Work **88**
5.1 Conclusion 88
5.2 Future Work 90

List of References **92**

List of Tables

4.1 Texture examples 58
4.2 Timing Results (in seconds) 83

List of Figures

1.1 Double exposure image. Wanda Wulz. *Io + gatto*, 1932 2
1.2 Image blending effects. Bill Showalter. *Barn at the end of the line*, 2017 2
1.3 Left: original image, Right: result of our method. 4
2.1 Construction and collapse of a three level Laplacian pyramid 7
2.2 Texture continuum described by Kuri et al. [34]. From left to right: regular, near-regular, irregular, near-stochastic, stochastic. 12
2.3 Comparing synthesis order and neighbourhood search windows between Efros and Leung [21](Centre) and Wei and Levoy [61](Right) 13
2.4 Hertzmann et al.'s Image Analogies [30] applying texture transfer. Left: example texture. Centre: Target image. Right: Result 22
3.1 The pipeline of our texture transfer system 33
3.2 Left: Original image; Right: Sharpened image using smooth maximum function with $k = 0.25$ for 4 pyramid levels. 35
3.3 Laplacian coefficient mixing using $\alpha = 30$. Left: input image, Centre: texture image, Right: result image. 35
3.4 Adding Laplacian coefficients to an image. Top: The image signal shows saturated highlight regions. Bottom: Adding Laplacian coefficients to the image pushes intensity values out of range. 37
3.5 Region selection. From left to right: Original image, lower threshold shown in yellow, upper threshold shown in orange, final selected regions shown in individual colours. 38
3.6 Gradient remapping function comparing $\alpha = 0.1$ and $\alpha = 1.0$ in each figure. Left: $\beta = 0.2$, Right: $\beta = 0.5$. 39
3.7 Dynamic Range Compression. Left: Input image, Right: Compressed image. 40

3.8 SLIC super-pixel segmentation. Left: Original super-pixels, Centre: Guide image, Right: Guided super-pixels computed with Equation 3.5. 42
3.9 We dilate the current patch into the previously determined texture to produce an overlap region by means of a circular kernel. 44
3.10 Coherence candidates are chosen by forward shifting previously determined patches from their origins in the example texture. Left: Example texture, Right: Output texture. 44
3.11 Random candidates are sampled from exponentially decreasing windows around the best coherent candidate. 45
3.12 Three cases for patch overlaps. Left: Single region, Centre: Two or more regions, Right: Ring-shaped region. 46
3.13 Determining endpoints for overlap regions. 47
3.14 Min-cut through overlap region for all three overlap cases. Left and Centre: Source (yellow) and target (black) pixels are at opposite ends of the overlap region. Right: We disconnect the ring-shaped overlap region in order to provide end-points for the min-cut. Source and target pixels reside on either side of this division. 47
3.15 Texture synthesis results. Left: texture example, Right: synthesized texture. 48
3.16 Directional Texture Synthesis. Left: Example texture, Centre: Orientation field visualized with LIC [13], Right: Synthesized texture. . . . 50
3.17 Top row: Orientation fields for the texture example and input image, Bottom row: Rotations must be reversed to locate offset texture patches. 51
3.18 Computing weights for $k = 6$ orientations (top to bottom). Left: Orientation masks, Centre: Geodesic distance fields, Right: Orientation weights. 55
3.19 Image orientation field. Top left: Input image, Top right: Tensor coherence, Bottom left: Original orientation field, Bottom right: Smoothed orientation field. Orientation fields are visualized using LIC [13]. . . . 56
4.1 Input images are chosen to represent a variety of manufactured or natural subject matter. 59
4.2 Single texture results. From left to right: original images, texture examples, results, result details. Texture examples (top to bottom): *e, g, m,* and *k.* All results were generated with $\alpha = 30$. 61

4.3 Removing texture synthesis from the system. Top left: input image, Top right: static texture, Centre: static texture result, Bottom: result using texture synthesis. $\alpha = 50$. 62
4.4 Image blending examples. Primary image at top left: portrait, top right: cityscape, bottom left: landscape, bottom right: boat. 64
4.5 Results employing user-defined masks. Left column – top to bottom: texture examples, masks, input images. Top right: Texture example c ($\alpha = 40$), Centre right: Texture examples k and f ($\alpha = 40$ and 25), Bottom right: Texture example d ($\alpha = 40$). 65
4.6 Using textureness to guide spatial masks. Top left: input image, Top centre: textureness mask, Top right: example texture. $\alpha_1 = 10$, $\alpha_2 = 60$ 67
4.7 Using orientation coherence as a spatially guided mask. Top row – left to right: input image, mask, texture examples a and i, Bottom: result. 69
4.8 Results showing the effect of dynamic range compression. Top left: input image, Top right: texture example k, Centre: without compression, Bottom: with compression. Threshold parameters are set to $t_{high} = 245$ and $t_{low} = 200$ for highlights and $t_{high} = 10$ and $t_{low} = 45$ for shadows. We set $\alpha = 40$. 70
4.9 Texture synthesis results. From left to right: Square patches, SLIC patches, Square patches with rotation, SLIC patches with rotation. From top to bottom: We use texture examples *k, f,* and *g*. The orientation field is shown on the top. 72
4.10 Varying the smoothness parameter k in Equation 3.1. Top left: input image, Top right: texture example d, Bottom left: $k = 10$, Bottom centre: $k = 0.1$, Bottom right: $k = 0.05$. We set $\alpha = 40$. 74
4.11 Varying the number of Laplacian pyramid levels. Top left: input image, Top right: texture example b, Second from top: 1 level, Second from bottom: 5 levels, Bottom: 9 levels. Texture coefficients are shown on the left against a neutral background. 76
4.12 Varying the texture intensity parameter α: Top left: input image, Top centre: mask, Top right: texture example n, Bottom left: $\alpha = 10$, Bottom centre: $\alpha = 25$, Bottom right: $\alpha = 50$. 77

4.13 Varying the texture patch size. Top left: input image, Top centre: orientation field, Top right: texture example h, Bottom left: $S = 7$, Bottom centre: $S = 12$, Bottom right: $S = 30$ 78
4.14 Comparison with Hertzmann et al. [30]. Left: texture example, Centre: Hertzmann et al., Right: our result. 79
4.15 Comparison with Wang et al. [60]. Top left: input image, Top right: example texture, Bottom left: Wang et al., Bottom right: our result. . 80
4.16 Comparison with Fang et al. [23]. Top: texture example l, Centre left: input image, Centre right: result from Fang et al., Bottom left: input image, Bottom right: our result. 82
4.17 Comparison with alpha blending. Top left: input image, Top right: example texture, Bottom left: our method, Bottom right: alpha blending with 70% of the input image and 30% of the texture image blended into the luminousity channel of the input image. 83
4.18 Synthesizing non-stationary textures. Top: Primary method (Algorithm 2), Bottom: Using a direction field (Algorithm 3), Left: Patch origins shown as white dots, Right: Results. 85
4.19 Our method does not produce perspective effects. 87

Chapter 1

Introduction

Long before the availability of image editing software, photographers have sought to combine multiple images into a unified work of art. Often, the motivation was technical. For example, in the early days of photography, overexposed skies needed to be printed from a second negative due to issues with early emulsion sensitivity [56]. Artistic effects could also be produced – either within the camera or later in the darkroom. Figure 1.1 shows an image that the artist Wanda Wulz created by exposing two different subjects on the same film in the early 1930's. Currently, combining multiple images into a single result is much easier by means of digital image editing applications. As such, image blending effects are an ubiquitous sight in photography and advertising media.

Blending external textures into an existing image is of particular interest to many photographers [19]. This interest is reflected in the many online repositories that sell static texture images for the purpose of image editing. Often the intent is to create a nostalgic feeling, brought on by artificially weathering digital images. Figure 1.2 shows an image that has been altered to this effect. At other times, artists seek to show more obvious stylization effects – applying the look of tree bark to a person's skin would be one example. Additionally, textures can serve as visual replacement for tactile qualities in images. In this sense, they can give us an impression of what it would be like to touch the photographed surfaces.

Figure 1.1: Double exposure image. Wanda Wulz. *Io + gatto*, 1932

Figure 1.2: Image blending effects. Bill Showalter. *Barn at the end of the line*, 2017

1.1 Problem Statement

Simply blending two images together is not always the best approach when adding textures to a photograph. Image edges, contrast, and colours are typically affected through this process in unintended ways. Also, previous work in texture transfer starts with the assumption that content from the original image should be entirely replaced by content from the example texture. This may be the intended outcome in some applications but often we will want to keep the most salient features intact from the original image. Instead of replacing the image content entirely, we may want to enhance an existing image with new textures. Another issue, which is far more difficult to solve by manually manipulating images, is that of texture direction. It is often desirable that new textures should follow the orientation of structures in the original image. For example, if we add additional textures to a person's hair, we should expect this new texture to follow the grain of the hair.

1.2 Contribution

In this thesis, we propose a novel method to add external textures into an image. This texture is added in such a manner so that edges, colour and textures that are already present in the image are preserved. Our contributions are as follows:

- We employ the Laplacian pyramid to mix the coefficients from two image pyramids – an input image pyramid and a texture image pyramid. The final result retains the structural characteristics from the input while enhancing the image with fine-scale details taken from an external texture.

- Texture content is created with patch-based texture synthesis. We use an irregular tiling based on SLIC [1] superpixels as our primitive units of texture.

- We include a guidance channel in our texture synthesis method so that texture structures are aligned with an orientation field. We gather orientation information through the image structure tensor. Both the original structure tensor calculation and its subsequent smoothing are aided with geodesic knowledge so that our orientation fields are both edge preserving and smooth. The inclusion of an orientation channel allows us to control the output of a synthesized texture. However, orientation guidance limits our texture exemplars to stationary,

near stochastic textures to achieve good results.

1.3 Overview

We begin Chapter 2 with an exploration of previous work. This chapter will include a technical introduction to the Laplacian pyramid and a discussion on how other researchers have used it and alternate image decompositions to enable image editing. Also, in this chapter we review previous work in example-based texture synthesis with particular emphasis on Image Quilting [20] and Ashikhmin's [3] method. Additionally, we discuss how texture synthesis can be used to enhance the appearance of images in the related field of texture transfer.

After this background, we introduce the algorithms that we propose for our texture transfer application in Chapter 3. We present these algorithms in an intuitive order. Starting with the assumption that we have a ready texture image, we show how its fine-scale characteristics can be transferred into a photograph. Next, we present the basics of our primary texture synthesis algorithm. Finally, we show how its basic functionality can be extended to include orientation guidance.

We present the results of our method in Chapter 4. These include a discussion on masking effects and parameter settings that can be used to control the final results of our method. We also make comparisons with related research and discuss some of the limitations of our system. In our final chapter we summarize our work and point to the future with suggestions for further research.

Figure 1.3: Left: original image, Right: result of our method.

Chapter 2

Previous Work

2.1 Introduction

Image stylization, a subfield of computer graphics, aims to automatically transform photographs into stylized synthetic artworks. The resulting image either aims for a photorealistic interpretation, as seen in photographic style transfer [6, 52], or a non-photorealistic rendering (NPR), which draws inspiration from historical artworks. NPR practitioners have attempted a range of effects spanning the art history catalogue, from drawing [31] and painting stylizations [14] to large scale image abstractions [54]. A full review of these methods is beyond the scope of this survey and the reader is encouraged to consult the comprehensive book *Image and Video-Based Artistic Stylization* [50]. We limit our survey to techniques that specifically relate to image texture.

We situate our current research within a body of work on example-based texture synthesis and texture transfer. This chapter can be mostly divided into three sections. First, since we employ the Laplacian pyramid to transfer texture details between images, we include a technical background on its construction. With this knowledge in place, we expand our discussion to include image editing applications that it, and related pyramid-based data-structures, enable. We then turn our focus towards example-based texture synthesis. This section includes an examination of pixel and patch-based texture synthesis techniques that inform our current work. Finally, we show how researchers have applied texture synthesis methods to the goal of texture transfer – enhancing images with textures that are taken from an external source. Before concluding this chapter, we present two shorter backgrounds on gradient-domain image editing and edge-aware image filtering. As these techniques are used

throughout this thesis, the reader will likely benefit from an introduction to these topics.

2.2 Multi-scale Image Stylization

2.2.1 Image Pyramids

Image pyramids are useful for manipulating and analyzing images at multiple scales. The general framework has the input image low-pass filtered and then downsampled to half the original size in both the horizontal and vertical dimensions. This procedure is performed recursively, progressively removing high-frequency details, until some fixed stopping condition is reached or the image has been reduced to a single pixel in one of its dimensions. A Gaussian distribution is a popular choice for the low-pass filter and the resulting Gaussian pyramid is used extensively in texture synthesis as we will see later in this chapter.

Burt and Adelson originally introduced the Laplacian pyramid [11] for the purpose of image compression. Using small 5×5 separable linear filters, an image can be quickly decomposed into its pyramid representation. In practice, the 5×5 filter is not used directly for a two-dimensional convolution. Instead, an equivalent operation is to perform one-dimensional convolutions on the rows and columns successively. In order to produce an identical result to the original two-dimensional variant, this one-dimensional speedup requires that the outer product of the proposed 1×5 filter with its transpose is equal to the original two-dimensional filter.

After filtering the image rows, every second pixel is removed in the horizontal direction in a process known as decimation. The same filtering and decimation process in repeated in the vertical direction. Burt and Adelson propose filters of the form $[c\, b\, a\, b\, c]$ where the following conditions must be met:

$$\begin{aligned} a + 2b + 2c &= 1 \\ b &= \frac{1}{4} \\ c &= \frac{1}{4} - \frac{a}{2} \end{aligned} \tag{2.1}$$

Choosing $a = 0.4$ produces a *Gaussian-like* filter and the resulting Gaussian pyramid contains levels g_0, g_1, ... , g_{L-1} for an *L-level* pyramid. Using these proposed filters it can be shown that level g_l can be exactly reconstructed from level g_{l+1} by first padding g_{l+1} with zeros and then filtering the result with $4 \times [c\ b\ a\ b\ c]$. These downsampling and upsampling operations are labeled:

$$g_{l+1} = REDUCE(g_l)$$
$$g_{l-1} = EXPAND(g_l)$$

Using the above mentioned operations, a Laplacian pyramid, l_0, l_1, ... , l_{L-2}, is created with the difference images $l_l = g_l - EXPAND(g_{l+1})$ from the corresponding Gaussian pyramid. Level $L-1$ is then taken as a copy of the tiny top image g_{L-1}, which consists of the coarsest features of the original input. The coefficients in the lower Laplacian pyramid levels represent the details and edges at different spatial scales of the input, decomposed into approximately separate frequency bands. Finally, the original image can be reconstructed from the Laplacian pyramid from top to bottom such that $g_l = EXPAND(g_{l+1}) + l_l$ until the original image is reconstructed at g_0. Figure 2.1 illustrates this process.

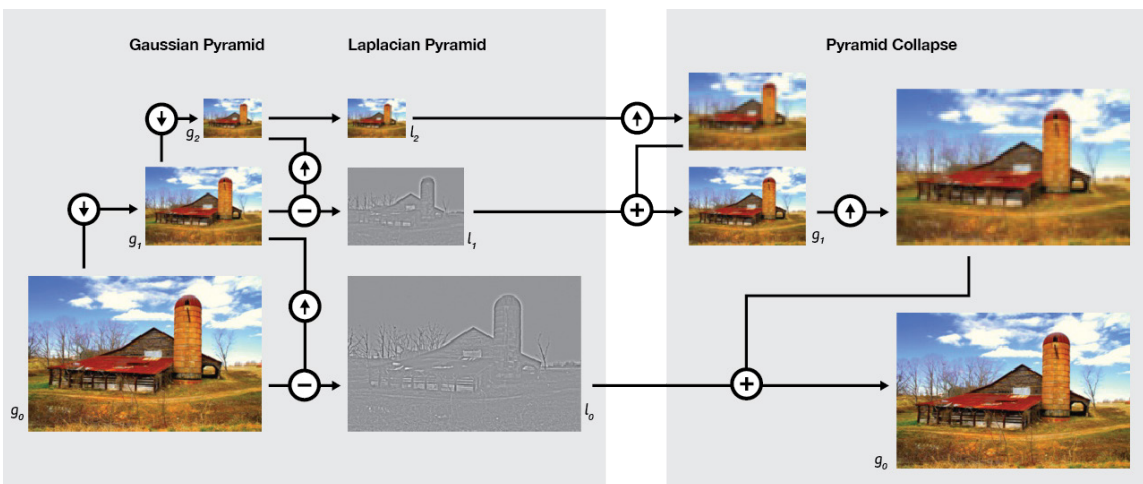


Figure 2.1: Construction and collapse of a three level Laplacian pyramid

As previously stated, the original purpose of the Laplacian pyramid was image compression. One can intuitively see how compression is possible if we consider the tiny image at the top of the Laplacian pyramid as an approximation of the full

resolution version. The remaining levels, which represent the missing details, consist mostly of small values centred around zero. They can be represented in fewer than the 24 bits typically used to encode image data. Since its introduction, the Laplacian pyramid has been repurposed to include multi-resolution image blending [12] and detail/tone manipulation [46], among other applications. Additionally, other image pyramid variations are possible by replacing the symmetric, linear low-pass filters used in the Gaussian/Laplacian pyramids with the edge preserving Bilateral filter [58], or directional Haar filters [27], among others. The downsampling step can also be omitted, as some authors have recommended, to avoid aliasing artifacts. In this situation we refer to an image stack as opposed to an image pyramid.

2.2.2 Pyramid-based Image Stylization

As we just saw, the coefficients in the Laplacian pyramid consist of the details and edges of an image as they are observed at multiple scales. These coefficients can be manipulated through scaling operations or combined with coefficients from multiple images to produce a variety of visual effects which are revealed in an output image when the pyramid is collapsed.

Burt and Adelson use the Laplacian pyramid to create photomosaics [12] – merging two or more images so that the seam is not visible. A naive approach simply blends the two images across a transition region, where alpha values are determined by spatial proximity to both images. This approach however, can result in one or more of following two problems. First, if the transition is too small then the seam between the images will be visible due to sharp jumps in image intensity. On the other hand, if the transition is too large then a double exposure (ghosting) effect can be seen where features of both images are visible. Consequently, the transition region should be no larger than the smallest image features while at the same time being no smaller than the largest image features. While this is not possible in a single scale image, the Laplacian pyramid provides the needed separation of features into different spatial scales. The method for image blending defines a boundary between the two images and then averages the Laplacian coefficients at a single pixel-wide boundary at each pyramid level. As the highest pyramid levels contain the image features at the largest scale, these features are adequately blended and provide a smooth transition region.

A similar decomposition, with a bilateral filter, can extract details from an image at multiple scales to produce a bilateral stack [6, 24]. However, in this case, the

edge preserving bilateral filter does not filter out the strongest edges, but instead maintains them so that they are present in the coarsest image at the top of the stack. This bilateral decomposition is often recommended for detail enhancement since aggressively scaling pyramid coefficients, produced by the linear filters we have just seen, can lead to haloing artifacts around strong edges.

Fattal et al. [24] show how multi-light image collections can be combined using a bilateral stack to enhance detail and contours in photographs. Many illustrators, especially in the context of scientific or technical drawings, tend to emphasize image contours when depicting a subject. This has the effect of reducing structural ambiguity and produces crisp, quickly understood images. Photography, in contrast, tends to only emphasize forms as they curve away from a light source by framing them in a shadow. Contours are generally lost when they face towards the illumination. From these observations we can see that by combining images, photographed with differing directional lighting, we can recreate an image of a subject framing both the details and larger structures from all directions. This, of course, requires a carefully controlled setup where the camera and subject remain motionless between exposures while a handheld flash provides directional lighting from a variety of positions (usually 3–5 in the author's examples). Using the bilateral stack, pyramid coefficients in this collection of images can be blended with a weighted average. In order to achieve the best effect, the weighting function adds preference to darker image regions (shadows) and high image gradients (details) from the source photographs.

Another application for image pyramids is style transfer between photographs. Here we start with two images: an input and an example photograph. The goal of style transfer is to manipulate the input image to look like the example image. Bae et al. [6] propose a two-scale bilateral stack to decompose an image into base and detail layers. The base layers of the input and example images are matched using standard histogram matching on the the image intensity levels. Manipulating the fine-scale image details and textures requires further processing to transfer the look of the example image. An activity map is estimated for both the input and the example image by smoothing the magnitude of a high-pass filtered image. Histogram matching the input activity map to the example activity map will produce a guide for scaling the coefficients of the input detail layer. At this stage, collapsing the two-layer decomposition may result in visual artifacts that can include gradient reversals, amplification of noise, and saturated highlights. A post-processing fix puts constraints

on the modified image gradient and requires reconstruction by solving a Poisson equation [47].

While the global style transfer of Bae et al. [6] gives good results for landscape photographs, it often produces poor results for portraits where unflattering blemishes can be enhanced. Typically, portrait photographers will edit their images using local methods. They apply one adjustment to the skin, another to the hair, and so on. This suggests a local approach to style transfer. Shih et al. [52] construct a dense correspondence between input and example images by first warping the example image, aided with face landmark templates, to the input image. This initial estimate is then further improved using SIFT Flow [39] to establish the final correspondence map. A multi-scale Laplacian stack is then used to transfer the local contrast at each scale between images using an activity map as a guide. Transferring the example base layer directly into the output stack before reconstruction serves to precisely incorporate the large scale tonal distribution of the example.

On a more subtle level Sunkavalli et al. [55] aim to harmonize the local contrast, texture, and noise of two images. Gradient-domain image compositing methods [47] can seamlessly merge two images improving on Burt and Adelson's method [12] by avoiding the long range coefficient blending at the top of the Laplacian pyramid. A disadvantage of gradient-domain methods, however, is that different source images often have different noise and texture characteristics. Image harmonization aims to match the statistics of two images, prior to blending, for a more unified look. Using pyramids constructed from oriented Haar filters, the authors histogram match pyramid coefficients between each corresponding level. This method greatly improves the similarity between images prior to merging. However, it is limited to stochastic textures and image noise. Larger scale texture structures – cracks in a painting, for example – cannot be reproduced. Nor can textures that are correlated across scales, as the method operates on each pyramid level independently.

Related to local image editing, local Laplacian filters (LLF) [46] provide state-of-the-art edge-aware filtering for detail manipulation and high dynamic range (HDR) image compression. Manipulating image details with contrast curves can produce good results locally, yet ideally, distant pixels should not be affected by contrast with an unrelated region. Building a Laplacian pyramid one coefficient at a time, LLFs filter an intermediate image with a contrast curve centred at an equivalent location in the corresponding Gaussian pyramid. A Laplacian pyramid is constructed from

this intermediate result and the requisite coefficient is copied to the output pyramid. The resulting Laplacian pyramid coefficients contain a complex mixture of local image adjustments centred at varying intensities. While this process suggests an $\mathcal{O}(N^2)$ complexity to perform filtering and pyramid construction at every coefficient in the output pyramid, it can be shown that only a subregion pyramid needs to be built for each coefficient, reducing complexity to $\mathcal{O}(NlogN)$. Further speedups are suggested by Aubrey et al. [5] who approximate a LLF by precomputing a subsample of Laplacian pyramids centred at different intensity values. Each final output coefficient is then interpolated between its closest representative pyramids. The authors also show that style transfer in the spirit of Bae et al. [6] can be accomplished with LLFs by replacing the contrast curves in the coefficient remapping function for a gradient histogram transfer function. Hao et al. [28] add spatially guided filters which are directed by an activity map to reduce overamplification of detail in low energy regions.

2.3 Example-based Texture Synthesis

Texture synthesis is an active area of research in both graphics and image processing with a large number of uses, including producing content for texture mapping in graphics and image completion in image processing. We define texture synthesis as producing a large (possibly unlimited) image from a small example image where the new image bears a close resemblance to the original.

Before describing some of the methods used for texture synthesis, we describe some of the characteristics of texture as well as the terminology that we use in this section and in the rest of the thesis. Kuri et al. [34] describe textures as existing on a continuum between regular and stochastic and we show examples from this continuum in Figure 2.2. Regular textures are rarely seen in nature. They are characterized by the periodic repetition of basic primitive units. Brick or floor tiles are an example of this class. At the other extreme, stochastic textures lack any repeating basic texture units. We are therefore unable to predict the colour of any given pixel given knowledge of a neighbour that is located only a short distance away. Sand is an example in this class. While individual sand grains can be thought of as the basic elements, they vary in colour and typically won't be individually observed in photographic images at a normal scale. Most textures fall between these two extremes. Closer

to the stochastic end of the continuum we find the large class of near-stochastic textures that regularly occur in nature. While these textures appear stochastic at the global level (they lack any periodic repetition), they appear to contain one or more repeating structures at a local scale. Another descriptive characteristic of texture is the spectrum between isotropic and anisotropic textures. An isotropic texture will contain the same statistical properties after rotation. While anisotropic textures, like hair, appear to be oriented in a particular direction.

Figure 2.2: Texture continuum described by Kuri et al. [34]. From left to right: regular, near-regular, irregular, near-stochastic, stochastic.

Early work in example-based texture synthesis by Heeger and Bergen [29] attempted to reshape random noise to appear like an input texture through multi-scale histogram matching using steerable pyramids. This approach achieved good results for stochastic textures but fell short when dealing with more structured inputs. As a consequence, researchers have turned to the concept of the Markov Random Field (MRF) to improve the representation of structures in synthesized textures. Instead of matching statistics globally, MRF-based methods consider that the appearance of each pixel is only dependent on a small set of neighbouring pixels. Another assumption requires that the input texture is stationary; it appears similar, at a given scale, independent of where it is observed. This section reviews pixel-based and patch-based texture synthesis methods that are built on the MRF model of texture.

2.3.1 Pixel-based

Following the MRF model of texture, pixel-based methods of texture synthesis produce an output texture one pixel at a time by only considering a small neighbourhood, centred at the current location of the synthesis. A series of methods following this approach were initiated by the work of Efros and Leung in their 1999 paper *Texture synthesis by non-parametric sampling* [21]. We describe this work here.

The algorithm is initialized by independently sampling a small image patch (3×3 in their example) from the example texture and copying it to the centre of the output texture. Spiraling outwards from this seed region, a texture is synthesized pixel by pixel in onion skin layers. At each stage of the algorithm, a feature vector is constructed using the RGB values of all previously determined pixels under a W pixel square window centred at the current location. Using this neighbourhood descriptor, the algorithm exhaustively searches the input texture for a set of candidate pixels whose distance under the L_2 norm is below a user defined threshold e from the true nearest neighbour (NN). The current pixel in the synthesis can then be sampled from a probability distribution or chosen randomly from this set and copied to the output. The central image in Figure 2.3 illustrates the neighbourhood matching scheme used in this process.

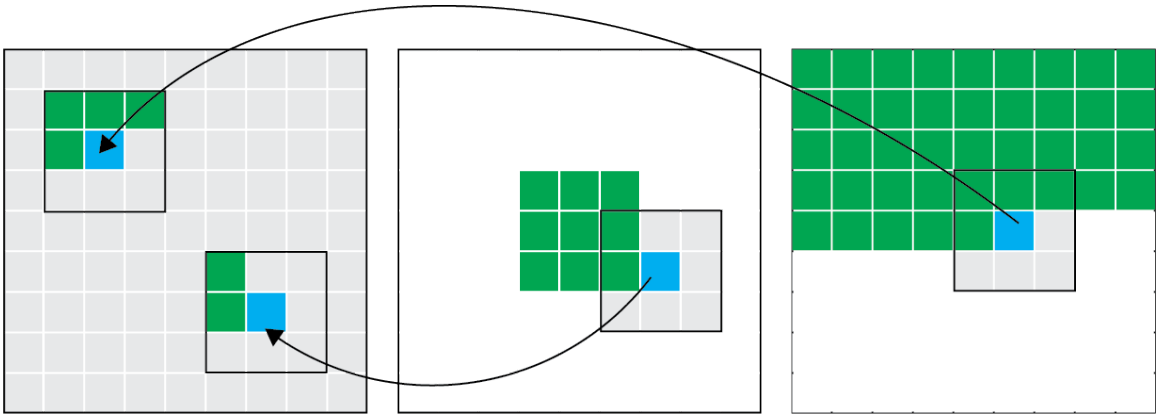


Figure 2.3: Comparing synthesis order and neighbourhood search windows between Efros and Leung [21](Centre) and Wei and Levoy [61](Right)

This algorithm produces good results for a wide variety of textures. It is also easy to use since the only parameters that need to be assigned are the neighbourhood window size W and the threshold level e. Unfortunately, this method can be quite slow. To ensure good results, the user must choose W to be the size of the largest structure element in the example that they wish to emulate. This can significantly slow the process, as the algorithm's complexity $\mathcal{O}(WN^2)$ depends on this parameter.

Wei and Levoy (WL) [61] address both time complexity and the scale of the individual texture elements in their extension of the basic method [21]. Using the variable size neighbourhoods, which spiral ordering produces, makes accelerating the NN search difficult. To address this issue, WL use a raster scan order to synthesize the

output texture from top to bottom and from left to right. This enables them to use a fixed, *L-shaped*, causal neighbourhood of previously determined pixels, located above and to the left of the current pixel in the synthesis process. The rightmost image in Figure 2.3 contrasts WL's neighbourhood sampling window with the one used by Efros et al.. As before, feature vectors are extracted from the RGB values, only now using a fixed neighbourhood, a partitioning tree NN search method [43] can be used to reduce the time complexity to $\mathcal{O}(WNlogN)$. The authors opt for *Tree Structured Vector Quantization* (TSVQ). Initialization of the output texture uses white random noise sampled from a histogram of the example texture RGB values. Borders are treated toroidally with the first few rows in the texture synthesis building on the random pixels taken from the bottom rows. After initialization, the algorithm is entirely deterministic and builds from the previously determined pixels.

Larger-scale texture structures are dealt with by incorporating a coarse-to-fine approach using the Gaussian Pyramid. The algorithm proceeds as described above, only now in addition to the *L-shaped* neighbourhood at each level g_l, feature vectors also incorporate the full square window centred at the current pixel in level g_{l+1} which have already been synthesized. This method enables large texture structures to be captured using only small windows in the coarse pyramid levels, propagating the results down into the finest resolution at the bottom of the pyramid.

Ashikhmin [3] improves on the WL algorithm in the context of a particular class of natural textures. He observes that many textures are composed of small, repeating irregular shapes whose familiarity to the viewer would cause any distortion to be unacceptable. Flowers, pebbles, and grass fall into this category. Growing textures pixel by pixel in an uncontrolled manner, as the WL algorithm does, can cause distortions to occur. In addition, objectionable smoothing occurs in some textures, which Ashikhmin attributes to independently finding the best matching pixel, using the L_2 norm, at each stage in the algorithm.

Rather than starting each NN search from scratch at each pixel, Ashikhmin observes that a set of good candidates have already been found by considering the origins of nearby synthesized pixels. An example will best illustrate this point. Assume that a pixel $p_{out}(x, y)$ is about to be synthesized. By noting that a previously determined pixel, two positions up and two positions to the left, originated in the example texture at position $p_{in}(35, 67)$ we can just offset this location (two to the right and two down) and find a candidate for p_{out} at $p_{in}(37, 69)$. Ashikhmin's method follows this principle

and constructs a candidate set by mapping synthesized pixels in an *L-shaped* neighbourhood to their example origins, forward shifted with the appropriate offsets. If a candidate pixel is not valid (offsetting would cause it to extend the image boundaries) then a random pixel is, instead, included as a candidate. From this candidate list, the best matched pixel is copied to the output and its origin is stored in an auxiliary array for subsequent queries.

This method encourages the verbatim copying of whole patches from the example texture which tend to grow downwards in the y direction, restarting as necessary when no good candidates are found. The seams between patch boundaries are hidden in the busy, irregular shapes of natural textures. However, when textures are too smooth or texture elements are larger than the copied patches, visual discontinuities can occur. In these cases WL's method will give better results.

Akl et al. [2] extend WL's algorithm by including structure tensor information to improve the synthesis of anisotropic textures. The original algorithm is modified to operate in two separate stages. In the first stage, an output structure tensor field is synthesized from the input texture's structure tensor field by replacing the L_2 norm of RGB colours with a tensor orientation distance. This output tensor field is then used to guide the synthesis of the final texture. Favourable results are shown in comparison with the original WL method for structured textures, yet Akl et al.'s method falls short of the patch-based methods that we describe next.

2.3.2 Patch-based

Ashikhmin's method [3] shows that local features in the input texture can be better preserved by keeping them together in the output. Indeed, his method is often seen as an intermediate step between pixel-based and the patch-based texture synthesis methods that are more explicitly formalized in this section. By sampling whole patches from the example and piecing them together like a puzzle in the output, patch-based texture synthesis methods can maintain the local structure of the example texture. From this basic premise, various strategies are used to synthesize textures that are also convincing on a global scale through combined efforts of patch sampling and hiding the seams that fall between adjacent patches.

Two early works in this sub-field are the *Chaos Mosaic* [63] and *Lapped Textures* [49]. Both of these systems operate by placing random patches from the example without considering how they relate to each other across their seams. This

strategy can work surprisingly well for many stochastic textures, but tends to break down for more structured inputs. Using an iterative method from the field of deterministic chaos, patches from an input texture can be stochastically rearranged in only a few iterations [63]. The Chaos Mosaic is initialized by tiling the entire input texture into the output to fill the desired dimensions. Then the chaos function is applied to rearrange small image patches. At this point, the seams between individual patches will be highly visible, revealing the underlying grid pattern. To improve on this initial state, widened seams between the individual patches are blacked out and refilled using the pixel-based method of Efros and Leung [21].

Lapped Textures [49] were created with the goal of applying texture directly to 3D meshes that lack a continuous parameterization over the plane. This interactive method starts with a user marking a mesh with direction vectors. The system then interpolates these vectors to produce a smooth direction field. The user will then cut one or more shapes out of the input texture to be used as basic texture units. For structured textures, the cut should fall on naturally occurring seams such as the grout between individual bricks. In the case of stochastic textures, an irregular *splotch* is used instead. Texture is synthesized directly on the mesh by randomly selecting an uncovered area and then pasting a texture patch. This process repeats until the mesh is completely covered. Each patch is rotated to align with the user-defined direction field before placement. As with Ashikhmin's method, textures with low-frequency gradients can cause visible artifacts; consequently, natural textures with high-frequency components are usually more successful. We borrow a few concepts from Lapped Textures for our patch-based texture synthesis method. We use irregular shaped super-pixels for texture patches, as computed by *simple linear iterative clustering (SLIC)* [1]. We also rotate whole texture patches to align with a direction field.

Efros and Freeman [20] deal with the issue of feature matching across patch seams in a similar manner to the pixel-based methods from the last section. Their *Image Quilting* method proceeds in a greedy fashion, placing patches one at a time in a raster scan order to fill in the output texture while minimizing the visual discontinuity between patches. The algorithm will be described here in some detail since many methods, including our own build upon this framework.

The algorithm is initialized by sampling, at random, a small patch from the example and placing it in the top left corner of the output texture. Then the algorithm

proceeds by repeating the following steps until the output texture is completely synthesized:

1. An overlap region is defined for each patch by expanding its border so that it overlaps the previously placed patches, above and to the left, by some amount (usually $\frac{1}{6}$ of the patch size). *Note: In the first row of patches there will only be an overlap on the left hand side.*

2. This overlap region defines the error surface between two adjacent patches and is calculated as the *Sum of Squared Differences (SSD)* over the corresponding pixels in the overlap.

3. The input texture is searched, and a set of candidate patches is constructed by considering patches whose total error, within the overlap region, falls below a parameter e of the best matching patch.

4. From this candidate set, the next patch is chosen at random and placed in the output texture.

While this method enables texture patches to match fairly well across the seams, it is still prone to leave visual artifacts where there is a mismatch between features. To rectify this issue, the initial seam is further improved by finding a min-cut across the error surface, using dynamic programming. This cut is performed twice, once in the vertical direction and again in the horizontal direction.

Also in 2001, Liang et al. [38] proposed a similar method, where the main differences entail accelerating the patch NN search with a kd-tree and PCA dimensionality reduction. They also rely on feathering across seam boundaries to hide artifacts, in place of a min-cut. However, since feathering can cause blurring and ghosting artifacts at strong features, most of the subsequent work on patch-based texture synthesis has implemented some variation of a min-cut.

Many researchers have proposed improvements to the basic Image Quilting method by focusing on two issues: breaking up the regular square grid, which tends to accentuate any imperfections in the output texture; and improving the boundary min-cut between adjacent patches. Kuri et al. [34] offset the horizontal and vertical grid by using interlocking hexagons. This simple change appears to hide imperfections in the same manner as the irregular shapes used in Lapped Textures. The authors also note that the dynamic programming method, used by Efros and Freeman, does

not give a true min-cut as the path is forced to move forward in each step, while better solutions could contain sideways or even backwards movement. They use Dijkstra's algorithm instead. In contrast, Long and Mould [40] improve on seam boundaries by observing that a min-cut will often not give the best visual result. Often, in order to achieve a lower global cost, a min-cut will make a short-cut across high error regions. This can cause highly visible artifacts to appear at patch seams. Inspired by path planning in robotics they instead penalize high cost edges to favour meandering, yet safer, routes that may nevertheless have a higher global cost. Nealen et al. [44] combine pixel and patch-based synthesis in much the same manner as Xu et al. [63] but limit the re-synthesis to high-error pixels in the overlap between patches.

Kwatra et al. [36] improve on these methods by placing large, layered, irregular patches from the input texture without organizing them on any predefined grid. Instead, the algorithm is initialized by placing the entire input texture into the output. Then the algorithm searches over all possible offsets of the input to find the next patch placement to extend the synthesized output. While this search process can appear too expensive for practical use, the authors show how it can be performed efficiently by first using a Fast Fourier Transform. After finding the best offset, an optimal seam is computed with graph cuts to determine which parts of the input to copy. The graph cut method also allows high-error cuts to be remembered between iterations so that they can be hidden by subsequent patch placements.

2.3.3 Optimization-based

While this final class of texture synthesis does not directly relate to our method, we mention it here in the interest of completeness. It will also help the reader follow the next section on texture transfer as some research builds upon this framework. Optimization-based texture synthesis departs from the greedy pixel and patch-based approaches that have previously been discussed. Instead, optimization-based methods aim to synthesize new textures by minimizing a global energy function and consequently avoid the pitfalls caused by greedy methods falling into bad parts of the search space.

Kwatra et al. [35] propose an algorithm that contains aspects of both pixel-based and patch-based methods. It is pixel-based because each pixel is synthesized individually but it also resembles patch-based methods because it uses a sparse grid of

overlapping patches in the output texture. In the spirit of the *Expectation Maximization (EM)* algorithm, a texture is synthesized iteratively in two phases. The E phase minimizes a quadratic energy function to obtain new colour values for the output texture. This energy function is defined globally as the SSD of each patch in the output texture with its nearest neighbour in the example texture. It can be minimized using least squares methods. Next, the M stage of the algorithm corrects blurring defects by finding new valid NN patches in the input. This iteration continues until convergence is reached or a desired number of iterations have been performed.

A similar method, proposed by Wexler et al., optimizes a constrained form of texture synthesis – the space-time completion of video [62]. In this framework, the input is composed of a video sequence, minus a region that needs to be filled. The output is the completed video in which the missing content has been replaced by plausible space-time textures from the input. Video texture synthesis follows a similar format to its image based equivalent, with the main difference being the added dimension of time. A point p is now represented as a tuple (x, y, t) and a patch is extended into the time dimension to form a volume. Compared to Kwatra et al. [35], Wexler et al. insist on tighter constraints for global coherence with the input. Using a dense sampling, the colour of each point p is now determined by all neighbouring patches that contain it. For example, using $5 \times 5 \times 5$ patches, a point will exist in 125 other patches. Each of these patches will be associated with a NN in the input. The colour of point p is then determined by averaging the colours of p as it is mapped to the input by all patches containing it.

2.3.4 Fast Nearest Neighbour Search

The texture synthesis methods described here all use some form of a NN search to associate patches in the output textures with their most similar patch in the input texture. Tree-based *approximate nearest neighbour (ANN)* search methods, such as k-d trees and TSVQ, reduce the search time to logarithmic complexity for a single image patch. However, this still doesn't provide the real time results that are necessary for many applications. Ashikhmin's method of using a coherent search, based on nearby previously determined pixels, has proven to be more useful in practise. Tong [59] expands on this idea by precomputing a set of *k-nearest-neighbours (k-NN)* for each pixel in the input texture so that at runtime texture synthesis has a ready set of pixel candidates to draw from. While this speeds up the synthesis stage, the analysis stage

can still be slow for large input textures.

Patchmatch [7] greatly improved image searching and demonstrates an order of magnitude speedup over k-d trees for ANN searches with equal mean error. It is particularly suited for the optimization-based class of texture synthesis where a nearest neighbour field (NNF) must be found at every iteration. The authors demonstrate that many image editing techniques such as image completion, image reshuffling, and image retargeting could be done in real time.

After initializing the NNF with either random pixels or some prior knowledge, the algorithm proceeds in two phases. First, a propagation phase checks pixels, above and to the left, to determine if forward shifting their origin in the source image will provide a better match. Next, a random search builds a set of possible candidates from exponentially decreasing windows around the best match found so far. This process proceeds in scanline order over all pixels in the image and then reverses itself to proceed from bottom to top, right to left. Usually 4 or 5 iterations of this back and forth refinement are required but the algorithm converges most quickly in the first few iterations, providing good enough matches for many applications.

The authors provide a formal analysis on the convergence properties of the algorithm, to which an interested reader can refer. However, for the purposes of this survey we provide a more intuitive explanation. While any randomly chosen candidate in the initialization process is unlikely to be a good match, it is also unlikely that, given a large enough field, no good matches will be present. The propagation phase of the algorithm serves to carry these good matches forward through the NNF when they are available. When they are not, the random search phase helps the algorithm escape from local minima. This back and forth, between propagation and random probing, leads to the fast convergence properties of PatchMatch.

The authors extend the basic algorithm to include searches over scales and rotations as well as storing k-NNs at each location [8]. Further extensions by Darabi et al. [18] add reflections and non-uniform scale to the NNF. This expanded version allowed them to synthesize convincing transitions between images for such effects as seamless blending and texture interpolation.

2.4 Texture Transfer

Drawing on results from texture synthesis, researchers in computer graphics have extended the methods seen in the previous section to transform the appearance of images and 3D models. The goal of texture transfer is to resynthesize a new image that maintains the appearance of a target image but includes many of the structural characteristics of an example texture image. This suggests a series of questions, as posed by Elad et. al. [22] in their work on style transfer, which we paraphrase here to address the more restricted problem of texture transfer:

- Which parts of the target image should be preserved or discarded?
- Should the colour pallet of the source texture or the target image be used? Or a combination of both?
- Should prominent edges be allowed to shift or scale?

And most importantly:

- How do you know if a result is successful?

There are no established answers to these questions, and as we will see, the goals of individual researchers can often guide their results in different directions. This section organizes texture transfer into two parts: photographic images and synthetic images. We focus our efforts on the former as it relates most directly to our work.

2.4.1 Photographic Images

Both Ashikhmin [3] and Efros et al. [20] have extended their texture synthesis methods to include applications in texture transfer and represent both a pixel-based and a patch-based approach. In its simplest case, we include an additional target image to serve as a guide. As a texture is synthesized, the neighbourhood matching scheme must also consider this extra constraint in its similarity metric. Ashikhmin [4] extends the upper *L-shaped* neighbourhood to include the full square neighbourhood around a to-be-synthesized pixel. Now, as before, the upper *L-shape* measures the difference between previously determined output pixels and candidates in the example texture with the exception that only high-frequency components are considered. The lower *L-shape* window measures the difference between the target image and the example

texture candidates. This combined approach ensures that the output image resembles both the target image and the example texture. The focus here remains on the speed of the coherence search. While many results could be unsatisfactory, a user of this system is able to experiment with different texture examples, which a slower system would discourage. Similarly, Efros et al. [20] also include a texture transfer component in their Image Quilting method. Updating the patch overlap error to include a luminosity difference component between patch candidates and target images allows a texture to be synthesized that bears a resemblance to the target image.

Figure 2.4: Hertzmann et al.'s Image Analogies [30] applying texture transfer. Left: example texture. Centre: Target image. Right: Result

Hertzmann et al. introduced a novel technique in their seminal paper *Image Analogies* [30]. This method reframes many image editing functions, from simple blurring filters to artistic NPR styles, texture transfer, and texture-by-numbers into a machine learning perspective. Learning consists of a single pair of images, A and A', which are the before and after stages of applying a filter to an image. Synthesis consists of making an analogy so that a target image B is to its synthesized output B' as the example A is to its filtered version A'. This technique requires that A and A' are precisely registered so that a pixel in A can be directly mapped to its filtered version in A' at the same spatial coordinates.

Image Analogies works using a multi-scale coarse-to-fine texture synthesis method

similar to Wei et al. [61]. Feature vectors are constructed for both the $A : A'$ pair and the $B : B'$ pair. They consist of colour intensity values, gradients, and other low level image features for both the upper *L-shaped* neighbourhood, which has already been determined in the current pyramid level, and the full square neighbourhood in the next coarser level. Features from B and B' are concatenated into a single vector – similarly for A and A'. The output image is then synthesized by finding the closest match in the $A : A'$ pair for the next pixel to be synthesized in the $B : B'$ pair. The resulting match is then copied from A' to B'. Their image search algorithm combines both Ashikhmin's coherent search with a tree-based ANN search, enhanced with PCA dimensionality reduction. The former is designed to produce coherent texture structures and the later to increase the search space and prevent poor seams from appearing between neighbouring coherent patches. The general Image Analogies approach extends to texture transfer by making A and A' the same texture image. While this intuitively breaks their analogy metaphor, from a technical perspective, it amounts to a process similar to Ashikhmin's method that we described above. The main differences entail the expanded search space and feature vectors.

Another application of Image analogies is *texture-by-numbers*. This technique enables texture to be applied to synthetic images, which include basic hand drawn maps that indicate the placement of different texture types. In this application, an example photo is chosen for A', which the user will annotate to produce the A image. The B image consists of another set of annotations, which a user designs. A novel result for B' then consists of the image content from A', but organized to the user's specifications in B.

Using texture synthesis to transfer textures between images will completely replace the image content in the target image with pixels from the example. This often destroys structure in regions where suitable content is not present in the example. Okura et al. present a method for selectively applying texture transfer to an image for scene manipulation in outdoor photography [45]. If you compare two scenes from the same viewpoint, yet at different times of year, it can be seen that some changes can be represented by a shift in colour (green grass becomes brown) while others (leaves absent from trees) are best represented as a texture change. Borrowing from Image Analogies, they use an example pair A and A' which contain identical scenes photographed at different times. An equivalent transformation is applied to B to hallucinate B' so so that it appears to have undergone the same transformation.

Analyzing the $A : A'$ pair can reveal which appearance transformations can be represented using only colour transfer [53] or if texture synthesis is required to achieve the desired results. A multi-scale optimization-based texture synthesis method is adapted from Wexler et al. [62] for this purpose. This method can produce non-trivial scene changes, learned from the example pair, to produce altered vegetation, appearance of snowbanks, and flooding rivers that appear convincing under casual observation.

Related to our work, texture transfer can synthesize new textures into an image that follow an orientation field. Using example-based brushstroke textures, Wang et al. employ a patch-based texture synthesis method to replace the luminosity channel of a target image [60]. User intervention is used to select a small number of brushstroke texture samples from a digitized version of a painting. Each texture patch is then rotated a fixed number of times so that it represents many possible orientations. The authors apply 24 rotations in the interval $[0, \pi]$ for their examples. A region-based direction field is constructed for the target image by first segmenting the image into similar colour regions using mean-shift [15]. Within each region, individual pixels are assigned the orientation of their closest site on the region's medial axis. Finally, orientations are smoothed with a Gaussian filter. From this setup, Wang et al. use an Image Quilting approach for texture synthesis. Using the average illumination value in the target image as guidance, they choose the appropriate texture swatch and select the rotated example that best matches the target direction field. Improvements are made to this method by subdividing the Image Quilting grid cells where non-uniform direction is found in the target direction field. This approach creates directional brush textures that replace the luminousity channel of the target image. However, due to the regular grid pattern and the quantized texture rotations, the results appear too repetitive and are often not visually appealing. In contrast, we use irregular shaped patches and orient them exactly to a direction field.

Lee et al. [37] extend Ashikhmin's [4] texture transfer method to synthesize oriented brushstrokes from painted example images. They use Kang's *Edge Tangent Flow* algorithm [33] to interpolate the image gradient and produce a smooth direction field that follows the image edges. Using the direction field as a guide, they add an extra term to Ashikhmin's similarity metric. An *I-shaped* neighbourhood is defined as a subset of the upper *L-shaped* neighbourhood and consists of a single-pixel-wide chain of pixels tangent to the orientation of the neighbourhood centre. The average colour of pixels in the previously determined texture along this line contributes to

the choice for the next synthesized pixel. As a result, textures appear to follow the direction field since pixels are similar in colour along this flow.

Fang et al. [23] provide photograph editing tools that add texture to objects as if they existed in 3D space. Using *shape-from-shading methods*, surface normals are extracted from an image. These normals are then used to cluster pixels into small patches with a similar orientation. A patch-based texture synthesis algorithm naturally follows; the authors use Kwatra's Graph Cut method [36]. Proceeding outward in a flood fill order from the centre pixel, foreshortening effects are created in each patch by mapping image pixels into texture coordinates. To find the texture coordinates of the next pixel they project it onto the tangent plane centred at the previous pixel. When this plane is rotated back into the image plane it provides the needed coordinates in the texture image. Additional effects are created by determining a height field of the input texture – also through shape-from-shading methods. The height field enables displacement mapping by translating the texture image in proportion to its height in the direction of the image surface normals. Texture orientation is determined manually by a user drawing an orientation field over the target image.

Super-resolution

Super-resolution is the process of creating a high-resolution image from low resolution inputs. Standard methods add new pixels between the existing ones and interpolate colour values from neighbouring locations. This often results in introducing artifacts or blurring edges. Synthesizing new high resolution data is one method for creating enhanced resolution images. Hertzman uses an Image Analogies [30] approach where a high resolution image is paired with a downsampled version to supply example high-resolution textures. However, this requires that the example image has similar texture structure to the target image. Freeman et al. present a one-pass algorithm [26] similar to Image Quilting [20] that makes use of a database of texture examples. This database is created by storing high-pass-filtered pairs of image patches at both low and high resolutions. The algorithm searches the database for similar texture patches that match both the low resolution target image and an overlap region of the corresponding high-resolution patch with the previously determined area. Their method produces results that are sharper than standard interpolation methods. However, image noise and jpeg artifacts can be erroneously treated as texture and enhanced in the output.

2.4.2 Synthetic Images

We have seen how texture synthesis can be used to improve or transform the look of photographic images. A related approach aims to add details to computer-generated images which often lack the textural richness of photography or human created artworks. Another application is terrain synthesis. Cruz et al. use patch-based texture synthesis to synthesize digital elevation models (DEM), from examples, guided by region labeling and low resolution DEM maps [17].

Returning to image enhancement, Benard et al. use a modified version of Image Analogies to stylize computer-generated animation from hand painted examples [9]. The output of rendered CG animation frames provide important information that can speed up NN searches as well as ensure that the coherence between frames is continuous. This information includes the offset of pixels between frames, as well as determining if it was occluded in a previous frame (or about to be occluded in the next frame). In a preprocessing stage, an artist will overpaint the synthetic images at certain keyframes, providing the paint texture that will be applied to the rest of the animation. Combined with the original CG output, this provides the needed example pair for Image Analogies. The algorithm then sweeps through the frame sequence back and forth in a coarse-to-fine manner updating the in-between frames until it converges. A parallel version of Patchmatch [7] is used for the nearest-neighbour search at each frame. Except for the first frame, at the coarsest resolution, the NNF can be initialized with offset pixel locations taken from previous frames or lower pyramid levels.

Related approaches, that use texture synthesis to improve computer-generated images, include Johnson et al. [32], who copy large image patches into a synthetic model, and Bonneel et al. [10], who use a modified texture-by-numbers approach. Both papers focus on outdoor scenes. Johnson et al. enhance moderately detailed CG models by first searching an image database for a set of images that match well on a set of scene descriptors. These images are co-segmented into regions that correspond to labels, such as water, forest, and mountain, which are derived from the input synthetic image. After colour and tone is transferred to the model, the images are aligned to the model at strong edges. Graph Cuts are used to cut seams between image patches, the gradients are transferred to the model, and finally the textured image is reconstructed with Poisson blending.

Bonneel et al. [10] propose a system that enables fast prototyping of outdoor

scenes. These can be quickly rendered so that a user can walk through 3D scenes at near-interactive rates. Using a set of basic geometry editing tools, a user will design a rough prototype of a scene. Directly rendering the simple geometry for guiding texture synthesis would lead to unnatural results. To improve this situation, a new guidance map is synthesized from a detailed annotation map based on an example image. This produces rich boundary transitions in the guidance map that can be used for a texture-by-numbers approach to synthesize natural image content. They use a reprojection-based solution for temporal coherence which reuses pixels from previous frames. Required new pixels are resynthesized with distortion and stitched to the current texture with Poisson blending.

2.5 Gradient Domain Image Editing

We have referenced Poisson blending [47] on a few occasions in the context of a general technique for reconstructing an image from its gradient field. We now provide some details on the theory and implementation of this method, which can be applied to numerous goals including image blending, HDR compression, local illumination changes, and texture flattening.

As a simple example, image content within a selected region can be removed by interpolating colour intensity values inwards from the selected boundaries. This process will result in a blurred image region that seamlessly blends with the boundary. A Laplace partial differential equation $\Delta I = 0$ accomplishes this by solving for I with Dirichlet boundary conditions and produces an image region where the gradient is as close as possible to zero. While this blurred image is of little use for many image editing applications, the process can be extended by including a vector field G to guide the interpolation while keeping the same Dirichlet boundary conditions. This vector field is often a gradient field extracted from another source or the modified gradient of the same image. The Laplacian now becomes a Poisson equation $\Delta I = div\, G$. This equation provides the necessary machinery to edit image regions in the gradient domain and then smoothly blend this transformed image with the surrounding content.

In practice, since we are working with digital images, we approximate the Laplacian ΔI and $div\, G$ with standard finite differences. Following Fattal et al. [25] we

compute the Laplacian operator as:

$$\Delta I(x,y) = I(x+1,y) + I(x-1,y) + I(x,y+1) + I(x,y-1) - 4I(x,y)$$

Using the forward difference, the image gradient is:

$$G_x = I(x+1,y) - I(x,y)$$
$$G_y = I(x,y+1) - I(x,y)$$

The divergence of the gradient is:

$$div\, G = G_x(x,y) - G_x(x-1,y) + G_y(x,y) - G_y(x,y-1)$$

using the backwards difference.

This combination of differences ensures that $div G$ is consistent with the Laplacian operator. Employing Neumann boundary conditions at image borders, we set gradient values $G(x,y) = 0$. Dirichlet boundary conditions are used at selection boundaries, setting $I(x,y)$ to the value of the background image.

Perez at al. show that seamless blending effects can be generated by taking the gradient field of an external source image as G. Solving the Poisson equation then serves to smoothly interpolate boundary colours from the target background image into this new content. The resulting seams between the two images will go largely unnoticed, provided that both images share common low-level statistics. A related variant of seamless blending involves constructing G from a combination of gradients taken from the source and target images. At each location in G, they select the gradient with the largest magnitude. Using this modification of the guidance vector field, the new image content will exhibit aspects of both images. We take a similar approach when adding image textures by means of the Laplacian pyramid. The main difference is that we use a smooth maximum function, while Perez et al. use the standard maximum.

Fattal et al. use Poisson blending to compress HDR images for display on LDR devices [25]. Based on the observation that large luminance variations in an image must give rise to large image gradients at some scale, the dynamic range of an image can be compressed by attenuating large gradient magnitudes. Using a multi-scale approach, large gradients are detected at various scales. They are then propagated

down to the highest resolution of the pyramid where they are attenuated. Solving the Poisson equation with Neumann boundary conditions reconstructs a compressed version of the original image. Perez et al. [47] generalize this approach to arbitrary region selections with Dirichlet boundary conditions while operating on a single scale. We directly implement Perez et al.'s version in our pipeline of algorithms.

2.6 Edge-aware Filters

Many image processing algorithms share the common goal of producing piecewise smooth, edge-sensitive outputs. The bilateral filter [58] combines spatial and colourspace weights so that pixels are averaged with neighbours that have a similar colour. A variant of the bilateral filter is the cross bilateral filter, also known as the joint bilateral filter [48]. In this version, colourspace weights are calculated from an external guide image. The consequence of this addition is that now edges in the guide are preserved rather than the original input edges.

Geodesic filters treat a 2D image as if it were a 3D surface. Distances are computed over this surface in much the same way as we would over mountainous terrain in the physical world. Criminisi et al. attempt to unify edge-aware image processing tasks such as segmentation, denoising, and image smoothing under a common framework by means of the Geodesic Distance Transform (GDT) [16]. Given a binary mask defined on an image, each pixel outside the mask is assigned a distance to its closest location in the mask. This distance is calculated by following the shortest path from a pixel back to its source while accumulating costs as it goes. Formally, the distance between any two pixels a and b is:

$$d(a,b) = \inf_{\Gamma \in P(a,b)} \int_0^{l(\Gamma)} C(\Gamma, s) ds \tag{2.2}$$

This distance is computed as the infimum of path costs among all possible paths connecting a and b, where each path is parametrized by its arclength $s \in [0, l(\Gamma)]$ and Γ is one such path. The function $C(\Gamma, s)$ is the infinitesimal cost of progressing on path Γ at s. It is given by:

$$C(\Gamma, s) = \sqrt{1 + \gamma^2 \Big(\nabla I(\Gamma(s)) \cdot \Gamma'(s) \Big)^2}, \tag{2.3}$$

where the dot-product here increases the distance cost as the image gradient

$\nabla I(\Gamma(s))$ becomes parallel to the path direction, given by its derivative $\Gamma^{'}(s)$. The parameter γ controls this geodesic influence.

Criminisi et al. provide an algorithm, based on the GDT, for edge-aware image smoothing. Colour values are quantized into k bins which become the masks from which the GDT is calculated. Using these k distance values at each pixel, a new colour value is assigned from the weighted average of the k quantized colours. The distance to weights calculation is computed as:

$$W_i(x) = e^{-\frac{D_i^2(x)}{\phi^2}}, \tag{2.4}$$

where $\phi > 0$ is a parameter that controls the spread of influence from each of the k colours. We extend this colour smoothing technique in our calculation of a smooth orientation field. However, in our application, we average structure tensors in place of colours.

Mould introduced the Cumulative Range Geodesic Filter for the purpose of texture-preserving image abstraction [42]. In place of Criminisi's GDT, a dedicated mask is constructed to assign the k most similar neighbours to a pixel's mask. These pixels are accumulated by using a front propagation algorithm that adds the most similar pixels one at a time. Pixel similarity is calculated with Equation 2.2 from Criminisi et al., only now the infinitesimal cost of progressing on a path is given by:

$$C(\Gamma, s) = |I(a) - I(\Gamma(s))| + \gamma|\nabla I(\Gamma(s)) \cdot \Gamma^{'}(s)| \tag{2.5}$$

Note, the spatial distance that is represented in the first term of Equation 2.3 has given way to the colour difference between a given pixel and its source at the mask centre. This change serves to enhance edge preserving properties in comparison with Criminisi's formulation. The result is a collection of irregular shaped masks that strongly avoid image edges. For the purpose of image abstraction, Mould averages the pixel colours within a mask. This smoothing operation has the effect of flattening nearby similar colours. Image texture is largely preserved because pixels with different colours will compute different masks even when they are spatially close. For our purposes, we use these geodesic masks to average gradient covariance matrices to compute the image structure tensor.

In this survey, we have introduced the most relevant work that has influenced our research. In the next chapter we will develop our own algorithms that extend from

this background.

Chapter 3

Algorithms

3.1 Overview

Our texture transfer system stylizes images through a non-linear mixing of Laplacian pyramid coefficients. These coefficients are taken from an input and a texture image. The texture image provides new details that we incorporate into the input image. We intend our image stylization effect to operate solely on fine-scale texture details and, as such, we require that the resulting output image maintains much of the large-scale features from the input, including tonal range and prominent edges. We also want the texture details to be modified so that they follow an orientation field derived from the input image. Having this control over the texture orientation will allow it to be more naturally incorporated into the final result.

Figure 3.1 shows a diagram illustrating the pipeline of our system. It is composed of two phases: a texture synthesis phase, where we generate texture content, and a texture transfer phase, where fine-scale features of this texture are merged with the input image by means of the Laplacian pyramid. We begin this section by describing our method for mixing coefficients in the Laplacian pyramid. We include a preprocessing stage where we compress extreme luminance values in order to minimize saturation in highlight and shadow regions. Next, we give details on our guided texture synthesis method. During texture analysis, we compute piecewise smooth orientation fields for both the input image and the texture example. These are later used to guide the synthesis of oriented textures.

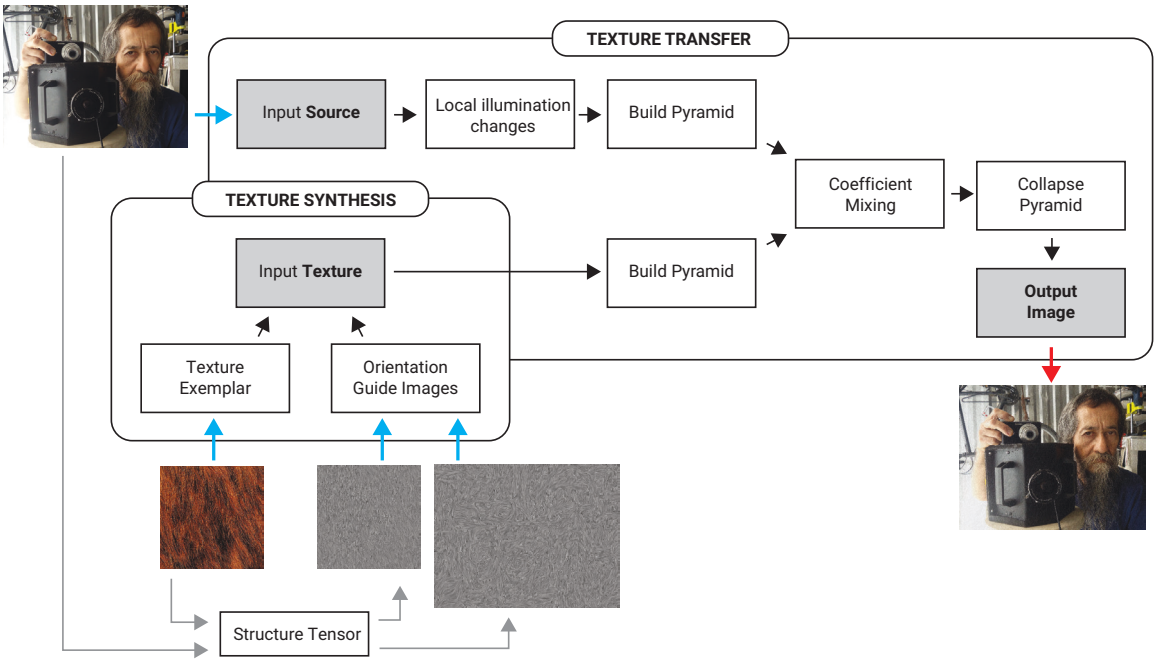


Figure 3.1: The pipeline of our texture transfer system

3.2 Pyramid-based texture transfer

In Chapter 2 we describe the basic construction of the Laplacian pyramid as it was introduced by Burt and Adelson [11]. We follow their methodology to construct Gaussian and Laplacian pyramids. Specifically, our implementation makes use of the pyramid construction tools available in the *opencv* library for C++. In this implementation, the a coefficient in the low-pass filter from Equation 2.1 is set to 0.36 to give a wide, Gaussian shaped, linear filter.

As noted in Chapter 2, many authors express concern that using the Laplacian pyramid for photographic manipulation can introduce halos around strong edges. We have not found this to be an issue in our work for two reasons. First, halo artifacts are typically caused by scaling coarser-level coefficients disproportionately compared with their finer-level correspondences. We only minimally scale the input image's Laplacian coefficients. Second, we restrict texture transfer to the fine-scale, lower pyramid levels by scaling down texture coefficients as we ascend its pyramid. By avoiding transferring strong step edges in the upper pyramid levels the associated halos that come with these image structures are not an issue in our work. For these reasons, the easy implementation and low $\mathcal{O}(N)$ complexity make the Laplacian pyramid appropriate for our use.

3.2.1 Laplacian Pyramid coefficient mixing

Given two Laplacian pyramids L_l^I and L_l^T, derived from an input and texture image, we will provide a result pyramid L_l^R that combines coefficients from both inputs. A simple strategy is to compare the absolute value between coefficients from both inputs and choose the maximum while retaining its sign. This is similar to the strategy used by Perez et al. [47] where they mix image gradients to blend images. However, this "choose the maximum" rule can cause image edges to become hidden if texture is added aggressively. Instead, we use a smooth maximum function that amplifies image details in regions where both input and texture coefficients have similar intensity. We compute each result coefficient $L_l^R(x, y)$ at level l as follows:

$$L_l^R(x, y) = sign \times \frac{1}{k} \times ln(exp(k \times |L_L^I(x, y)|) + exp(w_l \times k \times |L_l^T(x, y)|)), \quad (3.1)$$

where k is a parameter controlling the degree of smoothness in the maximum function and w_l is a weighting term that controls the intensity at each level of the texture pyramid. The variable *sign* can be 1 or -1 and refers to the sign of the coefficient with the larger absolute value.

For large values of k, equation 3.1 degrades to an ordinary maximum function. Smaller values of k, in contrast, enhance the amplification of the resulting coefficients in regions where texture and input images overlap. To illustrate this concept, Figure 3.2 shows the sharpening effect caused by merging an image with itself, i.e., $I = T$.

Rather than letting the example texture completely dictate the look of the resulting texturized image, we add a texture weighting term that adds parametric control. The weighting term from equation 3.1 is defined as:

$$w_l = \frac{\alpha}{\phi_l \times 2^l}, \quad (3.2)$$

where α is a user defined term that controls the intensity of the texture coefficients. Contrast will naturally vary between texture examples. To correct for this, we create a common frame of reference by including a normalization term ϕ_l that is robust to outliers. We set ϕ_l to p_{95}, where p_{95} is the 95th percentile of the Laplacian coefficients observed at level l in the texture image. Additionally, since we intend texture to be incorporated mostly into the finer-scale pyramid levels, we scale down the texture

Figure 3.2: Left: Original image; Right: Sharpened image using smooth maximum function with $k = 0.25$ for 4 pyramid levels.

coefficients exponentially by 2^l as we ascend its pyramid. Figure 3.3 demonstrates the result of this process.

Figure 3.3: Laplacian coefficient mixing using $\alpha = 30$. Left: input image, Centre: texture image, Right: result image.

The class of textures for which our method is designed will contain high-frequency content and lack large-scale intensity changes across the example. These textures will contain many non-zero coefficients in the lower Laplacian pyramid levels that quickly vanish as we ascend its pyramid. As we have already stated, these coefficients represent the fine-scale details that we wish to transfer into our input images. However, not all textures can be characterized in this way. Many textures will contain large-scale features where large coefficients persist into the upper levels of the Laplacian pyramid. These large coefficients can often cause halos to appear in their vicinity

when the output pyramid is collapsed. They will also introduce large-scale tonal variation into our stylized images that we do not intend. These issues are dealt with by exponentially scaling down the Laplacian coefficients in the texture image level by level. This step, which is defined in the 2^l term in the denominator of Equation 3.2, effectively removes much of the large-scale intensity changes in these textures. Consequently, they are now defined by their fine-scale features much like the textures for which our method is designed.

3.2.2 Gradient-based dynamic range compression

Since we have changed the Laplacian coefficients of our image pyramid, there is no reason why pixel values will stay within a limited dynamic range after we collapse the resulting Laplacian pyramid. Indeed, in extremely bright or dark regions, texture inclusion will cause pixel values to extend beyond the assigned $[0, 255]$ intensity value interval. Figure 3.4 illustrates this issue by means of a cross section extracted from a single-channel image. The bottom figure shows that as Laplacian coefficients are added to a signal – as we do when collapsing a Laplacian pyramid – the intensity values are pushed out of range.

A simple strategy to fix this problem is to clip invalid pixel values. However, this will only allow us to represent part of the new texture since many of its coefficients will be thrown away. Rescaling the original image is not an appealing option either since that will produce a flat, low-contrast result. Also, it is our intent to maintain the original large-scale dynamic range of the input image. Instead, we use the method of Perez et al. [47] for local dynamic range compression in order to selectively rescale highlight and shadow regions. By limiting the dynamic range compression method to selected regions, we can preserve the tonal integrity of most of the image while creating more space in the high and low end of the intensity range to introduce new textures. We now introduce our automatic method for selecting image regions to be compressed. While we present this method in the context of the high end of the intensity range, it should be noted that modifying the low end of the intensity range will be symmetric.

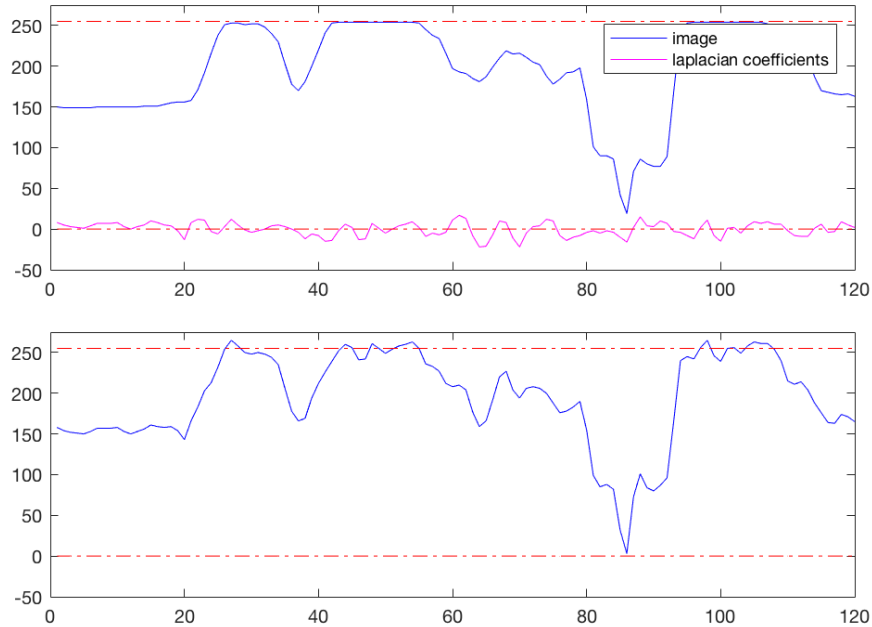


Figure 3.4: Adding Laplacian coefficients to an image. Top: The image signal shows saturated highlight regions. Bottom: Adding Laplacian coefficients to the image pushes intensity values out of range.

Region Selection

As threshold parameters are often difficult to set, we use a hysteresis thresholding scheme to select bright image regions for dynamic range compression. Given a user defined pair of threshold values, t_{high} and t_{low}, we wish to select contiguous regions in the image where image intensity values exceed t_{low} while also containing pixels which exceed t_{high}. The intent of using two threshold values is twofold: the upper threshold marks the true highlights that we want to compress while the lower threshold marks a wider region around the selected highlight where we interpolate the image intensity. In our examples we set t_{high} to 245 and t_{low} to 210 for bright regions. For dark regions, we set t_{high} to 10 and t_{low} to 45.

Algorithm 1 provides the pseudo-code for our region selection method. Working in the luminosity channel of CIELAB colour space we first threshold the input image at t_{low}. Next, we cluster these thresholded pixels into isolated regions by using a connected components algorithm. These regions will be candidates for dynamic range compression provided that their pixel count exceeds a user specified amount ϵ. We set ϵ to 20 in all our examples to allow small spot highlights to exist. We then verify

Algorithm 1: Select Regions

input : luminosity image I, threshold parameters t_{high} and t_{low}, minimum region size ϵ
output: selected regions Ω

$I_{low} \leftarrow$ threshold I at t_{low}
$\Omega \leftarrow \emptyset$
$\Omega' = \{\omega | \omega$ is a connected component of $I_{low}, |\omega| \geq \epsilon\}$
for *each pixel* x *in* I **do**
  if $I(x) > t_{high}$ *and* $I(x)$ *is not marked* **then**
    denote Ω'_i as the connected component in Ω' which contains x
    mark all pixels in I corresponding to positions in Ω'_i
    $\Omega \leftarrow \Omega \cup \Omega'_i$
  end
end
return Ω

that candidate regions enclose true highlights by insisting that they contain pixels whose intensity exceeds t_{high}. Figure 3.5 provides a visualization of this selection process. Pixels marked at the lower threshold are shown in yellow and pixels marked at the upper threshold are shown in orange. In this example, three regions are finally selected. They are individually coloured at the far right.

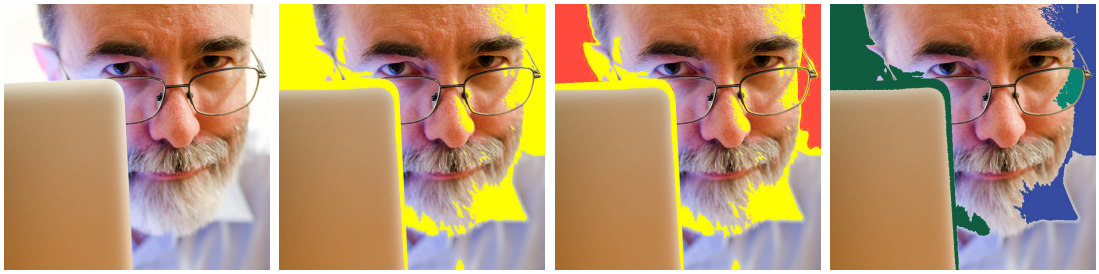


Figure 3.5: Region selection. From left to right: Original image, lower threshold shown in yellow, upper threshold shown in orange, final selected regions shown in individual colours.

Dynamic Range Compression

We now have a set of highlight-containing regions Ω where we can apply dynamic range compression. We follow the method described by Perez et al. and modify the

original image gradient to guide Poisson blending. As we stated in Chapter 2, we can compress the dynamic range of an image by selectively scaling down large image gradients. Working in the log domain, we apply the following remapping function to the image gradient:

$$G' = sign(G) \times \alpha^{\beta} \times |G|^{1-\beta}, \tag{3.3}$$

where α is a parameter that controls the range of the output gradient field G' and β controls the degree in which large gradients are compressed. Figure 3.6 plots this function using various values of α and β. Small gradient values are amplified to maintain local contrast while large gradients are compressed in order to reduce large intensity jumps. In our examples, we maintain Perez et al.'s preferences, and set α to 0.2 times the average gradient magnitude within the selected region, and β to 0.2.

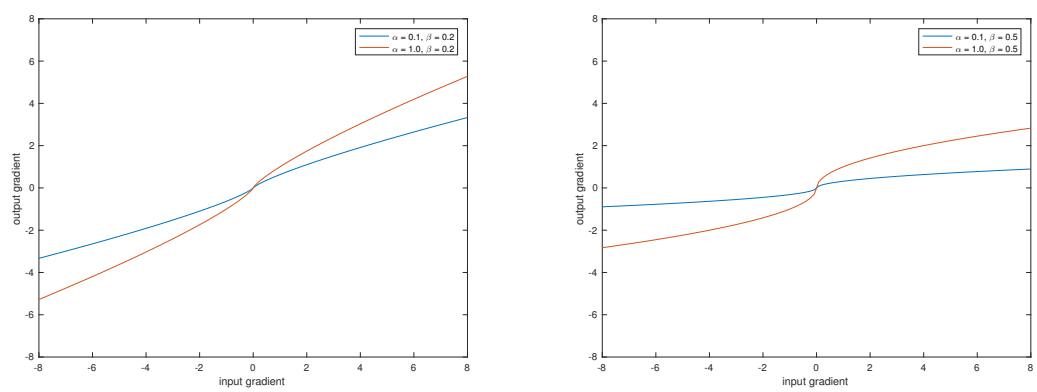


Figure 3.6: Gradient remapping function comparing $\alpha = 0.1$ and $\alpha = 1.0$ in each figure. Left: $\beta = 0.2$, Right: $\beta = 0.5$.

Using G' as a guidance vector field, we reconstruct I using Poisson blending. We find that it is also necessary to set pixels at the selection boundary to t_{low} to avoid undesirable colour interpolation at region boundaries. Figure 3.7 shows the result of this process. Notice that to the sides of the subject's face the intensity values have been decreased, yet the rest of the image remains unchanged from the original. Now we have a larger range of pixel values that can be used for incorporating texture detail.

Figure 3.7: Dynamic Range Compression. Left: Input image, Right: Compressed image.

3.3 Synthesizing textures

We now have a method for incorporating textures from a secondary texture image into an input image by means of the Laplacian pyramid. In this section we describe our method for synthesizing these auxiliary textures. We base our texture synthesis method on the tradition of Patch-based systems that extend from Image Quilting. We begin by describing our primary texture synthesis method. This is a general method that synthesizes textures, from a given texture example, without an external guide. With this method in place, we will show how it can be extended to include orientation guidance. This guide allows us to control the orientation of texture structures in the output. Finally, we describe how we build and modify a structure tensor field so that it describes the texture and edge orientations of the input image.

3.3.1 The Primary Method

We have already seen Image Quilting and the subsequent methods that aim to improve upon its original construct. We continue in this tradition and establish our primary texture synthesis method in this framework. One of the issues with the original Image Quilting algorithm is the regular grid upon which texture patches are placed. This repeating pattern can make any imperfections in the synthesized texture easily noticeable, since, once the patch seams become apparent, we are drawn to look for more defects at each step of the appropriate grid interval. Using irregular shaped

texture patches is one strategy to hide imperfections and it has already been explored through the use of interlocking hexagons [34] and the varied size and positioned patches used in GraphCut textures [36]. We also take our inspiration from Lapped Textures [49] where they paste layers of irregular shaped texture *splotches* over a mesh. By avoiding placing patches in predicable ways, poorly matched seams can go unnoticed for many texture classes. For the purpose of 2D texture synthesis, we divide the output image plane into a tiling of irregular shaped patches by means of applying *Simple Linear Iterative Clustering* (SLIC) [1] to the input image. Besides providing an irregular tiling, SLIC also aligns patch boundaries with image edges. The benefit of this property is that poorly matched seams in the synthesized texture can be camouflaged amongst the image edges in the final result.

Our algorithm proceeds in the fashion of Patch-based texture synthesis, choosing one texture patch at a time to place in the output texture. Each patch is chosen to minimize the error in an overlap region between neighbouring patches. Final seams are then cut through the overlap region using a multiple-source Dijkstra's algorithm to establish a minimum cost path.

Simple Linear Iterative Clustering

Achanta et al. introduced SLIC [1], an efficient k-means clustering algorithm to produce small, compact *super-pixels* which align well with image edges. It uses a combined colour and spatial distance metric where a single parameter m is used to control the tradeoff between the two terms. Distance is calculated as:

$$D = d_{lab} + \frac{m}{S} d_{xy}, \tag{3.4}$$

where d_{lab} and d_{xy} are the L_2 norms of colour and spatial differences. The parameter S is the spatial bounds used to search around cluster centres as well as the step size used to initiate cluster centre seeds. Penalizing spatial distances with high values of m produces fairly compact shapes. In our case, where we desire more irregular shapes, we favour clustering based on colour similarity and hence use smaller values of m. We set m to 50 in our examples. However, in regions of uniform colour – a blue sky for example – colour does not provide any information to the distance calculation and super-pixels are formed using spatial distance alone and therefore tend to be uniform in shape and size. This is undesirable, since it is precisely these

regions where any imperfections in our synthesized textures will be most noticeable when they are introduced into the input image. We mitigate this issue by augmenting SLIC's distance metric to include a third term which uses colour information from an additional guidance image. Our distance metric now becomes:

$$D = \lambda d_{guide} + (1 - \lambda) d_{lab} + \frac{m}{S} d_{xy}, \tag{3.5}$$

where d_{guide} is the L_2 norm between pixels and their cluster centres in the guide. The parameter λ controls the influence of this auxilary influence. Typically, we set it to a small value ($\lambda = 0.2$) so its effect is only felt when d_{lab} is close to zero. Figure 3.8 shows the result of our super-pixel segmentation with and without the use of this guide image. We use a noise field that we produce using Photoshop's *clouds* filter to guide clustering. Alternately, any image that exhibits irregular colour variations at the scale of the desired super-pixels will suffice.

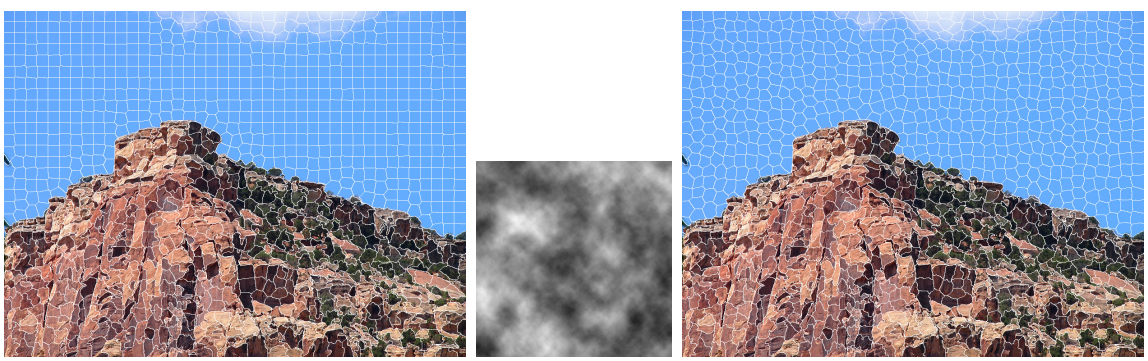


Figure 3.8: SLIC super-pixel segmentation. Left: Original super-pixels, Centre: Guide image, Right: Guided super-pixels computed with Equation 3.5.

Texture Synthesis

We synthesize our output texture T one patch at a time. As our irregular tiling does not provide a natural raster scan ordering, we instead proceed outwards from the centre in a *breath-first* order derived from the super-pixel graph $G = (V, E)$. The vertex set V is comprised of our computed super-pixels. We assign an edge $e(u, v)$ between all adjacent super-pixels u and v.

Algorithm 2 formalizes our patch-based texture synthesis method. We segment I, as described above, to obtain a set of irregular shaped patches P which we proceed

Algorithm 2: Texture Synthesis

input : example texture E, input image I
output: output texture T

$P \leftarrow$ SLIC segmentation of I
Initialize P_0 with a random patch sample from E
copy texture pixels from E to T as they are assigned to P_0
for *each remaining patch* P_i **do**
  $L_i \leftarrow$ compute overlap of P_i into previously determined regions of T
  $C_{coh} = \{c|c$ is a coherence candidate for P_i from $E\}$
  $C_{best} \leftarrow$ BestMatch(T, C_{coh}, L_i, P_i)
  $C = \{c|c$ is a random candidate for P_i from $E\} \cup C_{best}$
  $C_{best} \leftarrow$ BestMatch(T, C, L_i, P_i)
  $L'_i \leftarrow$ MinCut(T, C_{best}, L_i, P_i)
  $C_{best} \leftarrow C_{best} \cup L'_i$
  copy C_{best} to T
  store location of C_{best} for future coherence queries
end
return T

to fill with texture samples. After initializing our output texture T with a random texture patch taken from our example texture E, we proceed as follows: First, an overlap region is defined by dilating our current patch P_i into the previously determined regions of T. We achieve this by passing a circular kernel around the perimeter of P_i. Figure 3.9 shows that any pixel that is not part of P_i and has been previously determined in T is marked as the current overlap region L_i. The overlap region is then used to evaluate texture candidate patches C through the *BestMatch(·)* routine. This routine computes the *sum of squared differences* under the overlap region between candidate patches and the output texture. It returns the best matched candidate.

Patch Search

Our patch search method is inspired by Ashikhmin's pixel-based coherence search [3] as well as the random search method used in PatchMatch [7]. Given a patch P_i that we are about to synthesize, we first check to see if we can extend any of the previously determined texture patches from P_i's neighbours in our super-pixel graph. Let $\vec{v}_j$ be the vector pointing from the centre of the jth neighbour of P_i to the centre of P_i.

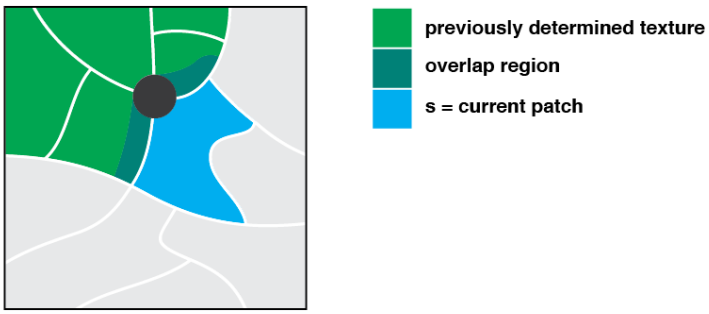


Figure 3.9: We dilate the current patch into the previously determined texture to produce an overlap region by means of a circular kernel.

Figure 3.10 shows how we build our coherence candidate set C_{coh} by forward shifting each neighbour j by $\vec{v}_j$ from its origin in E. These candidates are then evaluated to determine C_{best}.

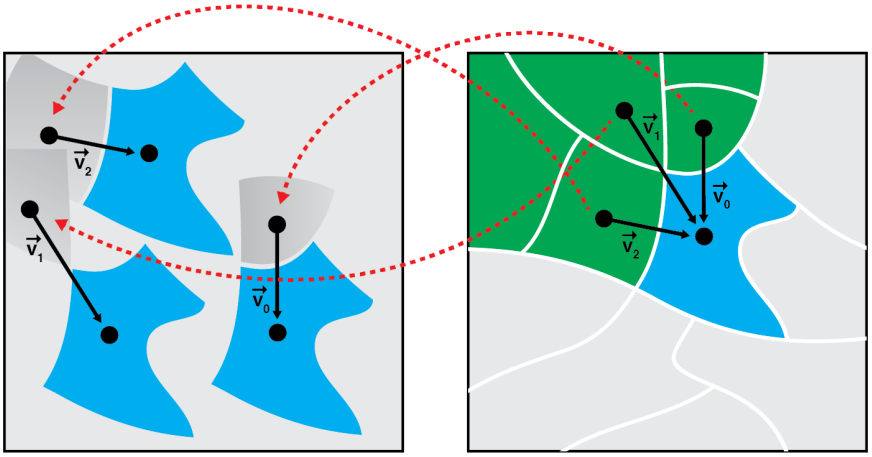


Figure 3.10: Coherence candidates are chosen by forward shifting previously determined patches from their origins in the example texture. Left: Example texture, Right: Output texture.

However, if we only use the coherent search method we will run into a problem: what do you do when forward shifting a patch origin causes a candidate to extend beyond the bounds of the texture example? Ashikhmin introduces a random candidate under these circumstances. However, randomly restarting a texture patch when no coherence candidates are available can cause discontinuities in the synthesized output. PatchMatch, instead, supplements each step of the ANN field algorithm with a small supply of random candidates chosen from exponentially decreasing windows centred

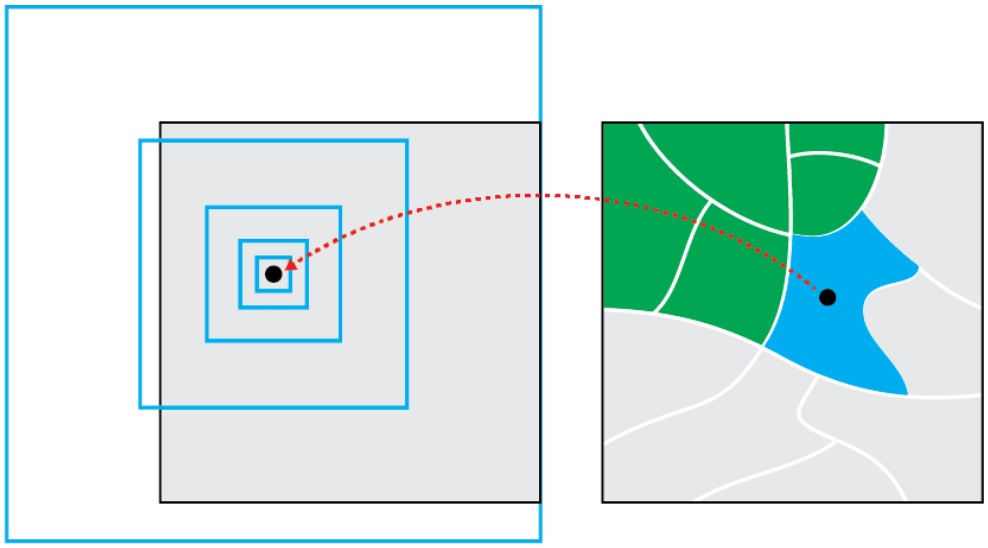


Figure 3.11: Random candidates are sampled from exponentially decreasing windows around the best coherent candidate.

at C_{best}. These candidates are then propagated into the field when they improve upon the current result. We also include this step in our search method and restate their sampling method below. It is illustrated in Figure 3.11. Our version differs only in that irregular shaped patches are sampled and evaluation is based on a patch overlap region rather than a full square neighbourhood. The ith random candidate patch C^i_{rand} is sampled at:

$$C^i_{rand} = C_{best} + w\frac{1}{2^j}\mathbf{R}_j, \tag{3.6}$$

where $\mathbf{R}_j$ is sampled uniformly at random from $[-1,1]\times[-1,1]$ and w is the maximum search radius defined as the distance from C_{best} to its most distant boundary in E. We sample candidates at $j = 0, 1, 2, \ldots$ until the search radius around C_{best} is reduced to a single pixel. At each j we sample n candidates, where setting n to 3 provides a good balance between expanding the search space and computational efficiency. It should also be observed that sampling is densest in the region closest to C_{best}. This allows small offsets to locally refine C_{best} while random probing into distant regions enables the search to escape local minima. Then we evaluate this new set of candidates to determine if a better choice can be found for P_i. Following from Ashikhmin, we perform our texture synthesis in a single pass which we find produces adequate results for texture transfer. In contrast, PatchMatch operates iteratively and refines ANN field candidates though each pass of the algorithm.

Min-cut

Now that we have chosen an appropriate texture sample for P_i, we improve seam boundaries between P_i and its neighbours in T by computing a *min-cut* through the error surface defined on the overlap region. The error at each pixel is computed as the squared difference between corresponding pixels in T and the overlap region L_i belonging to P_i. There are three cases that can occur for our patch overlap regions, which can be seen in Figure 3.12. **Case 1:** The most common case occurs when the current patch overlap intersects with the previously determined texture in T in a single location. **Case 2:** Alternately, the overlap can intersect the output texture in two or more locations. **Case 3:** Finally, the current patch can be surrounded on all sides by the output texture if it is that last patch to be placed in a local area. The first two cases can be treated alike by computing a min-cut through each overlap region independently. However, the *ring-shaped* overlap in the third case will need to be cut to produce distinct source and target pixels.

Figure 3.12: Three cases for patch overlaps.
Left: Single region, Centre: Two or more regions, Right: Ring-shaped region.

Before we can compute a shortest path through the error surface we must first detect source and target pixels. These end-points occur at opposite sides where the overlap region intersects with the output texture. Consider a pixel p in the overlap region. If one of its eight neighbours n_j is situated in P_i and one of n_{j+1} or n_{j-1} (accounting for circular arithmetic) resides in the unsynthesized region of T, then p will belong to a set of end-points. In Figure 3.13, n_5 is contained in P_i and n_4 belongs to the unsynthesized region. Therefore, we count p as an end-point. It can also be seen that n_3 meets the requirements and belongs to the set of selected end-points. This process will give us two sets of pixels at both sides of the overlap region where it

intersects the previously determined output texture. Additionally, we expand these sets by dilating them with a circular kernel along the boundary between the patch overlap and the *as-yet-to-be* determined texture. The two end-point sets are then arbitrarily labeled source and target and coloured yellow or black in Figure 3.14, respectively.

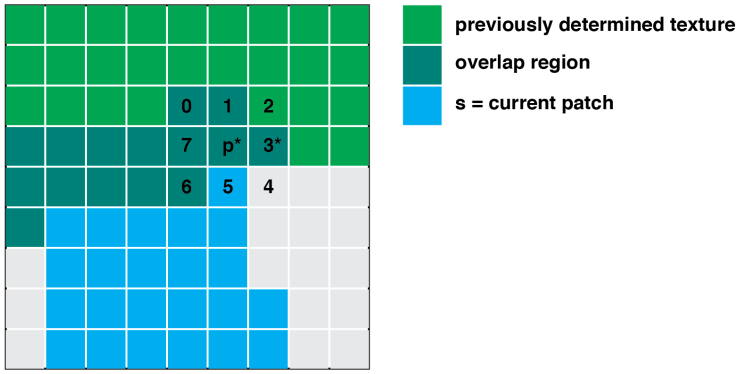


Figure 3.13: Determining endpoints for overlap regions.

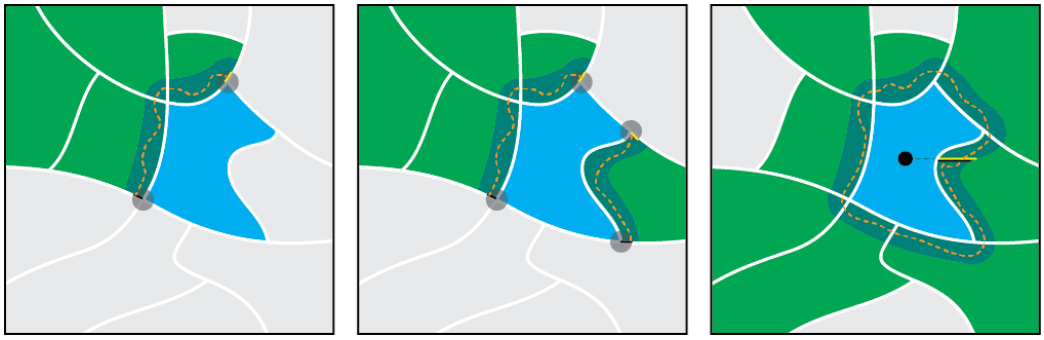


Figure 3.14: Min-cut through overlap region for all three overlap cases.
Left and Centre: Source (yellow) and target (black) pixels are at opposite ends of the overlap region. Right: We disconnect the ring-shaped overlap region in order to provide end-points for the min-cut. Source and target pixels reside on either side of this division.

We are now ready to compute a shortest path through the patch overlap region. We define a weighted graph on L_i and assign individual pixels to the vertex set. Then, examining our error surface, we assign $max(error(u), error(v))$ as the weight between adjacent vertices u and v. Our final initialization steps assign the path distances of all source pixels to zero and prevents edges from connecting them.

After this setup, we use Dijkstra's algorithm to compute the shortest path to all of the remaining pixels in the graph. By examining the distance costs for pixels in the target set, we can determine the shortest path between source and target through the overlap region. The precise path can also be reconstructed by tracing back, through the chain of predecessors, to a source pixel. We then expand P_i into the overlap region by propagating its front up to this dividing path. Finally, we copy P_i into T. A result of our primary texture synthesis method is shown in Figure 3.15.

A small modification must be made to handle the third overlap case where the overlap region forms a ring around the current patch. We separate pixels on either side of a horizontal line that is extended from the centre of P_i. We assign pixels above the line to the source set and pixels below the line to the target set. Edges are prevented from directly connecting source and target pixels in order to avoid the trivial path. As we now require that paths join up on both sides of this horizontal division, we must run Dijktra's individually for each target pixel. We find a shortest path for each target, initiated from a single source pixel that resides opposite the target. The rightmost image of Figure 3.14 illustrates this setup.

Figure 3.15: Texture synthesis results. Left: texture example, Right: synthesized texture.

We have now described the individual components of Algorithm 2. To briefly summarize, we proceed to place the best matching candidate patch at each location until the output texture is complete. After each patch placement, a min-cut is computed to produce an optimal seam between adjacent patches. In the next section we show how guidance from a direction field can be used to extend the primary method.

3.3.2 Texture Synthesis on a direction field

We intended to synthesize textures so that local structures are oriented similarly to corresponding locations in the input image. This orientation can be described through the image structure tensor and Section 3.3.3 provides information on its construction. Currently, we will take for granted that we have orientation fields for both our input image and example texture. These fields provide, at each position, a vector describing local structure orientation. We can now extend Algorithm 2 to include directional guidance. We present our directional texture synthesis method, which is formalized in Algorithm 3. The essential difference between these two methods is that now we rotate texture patches to orient structures in the example texture to structures in the input image. We denote our example texture orientation field as Υ_x^E which specifies the orientation of pixel x in the example texture. Similarly, Υ_x^I denotes the orientation field of our input image.

Algorithm 3: Directional Texture Synthesis

input : example texture E, input image I, orientation fields Υ^E and Υ^I
output: output texture T

$P \leftarrow$ SLIC segmentation of I
Initialize P_0 with a random rotated patch sample from E
copy texture pixels from E to T as they are assigned to P_0
for *each remaining patch* P_i **do**
  $L_i \leftarrow$ compute overlap of P_i into previously determined regions of T
  $C_{coh} = \{c|c$ is a coherence candidate for P_i from $E\}$
  $C_{best} \leftarrow$ BestRotatedMatch(T, C_{coh}, L_i, P_i, Υ^E, Υ^I)
  $C = \{c|c$ is a random candidate for P_i from $E\} \cup C_{best}$
  $C_{best} \leftarrow$ BestRotatedMatch(T, C, L_i, P_i, Υ^E, Υ^I)
  $L_i' \leftarrow$ MinCut(T, C_{best}, L_i, P_i)
  $C_{best} \leftarrow C_{best} \cup L_i'$
  copy C_{best} to T
  store location of C_{best} for future coherence queries
  store rotation of C_{best} for future coherence queries
end
return T

We evaluate patch candidates with our *BestRotatedMatch(·)* routine. This routine operates as the *BestMatch(·)* routine from Algorithm 2 except that now the texture image is rotated about the candidate patch centre, to align Υ_{x1}^E with Υ_{x2}^I, prior to

evaluation. Here $x1$ and $x2$ are patch centres in the texture example and input images, respectively. We determine the rotation amount as:

$$\theta = \theta_I - \theta_E, \tag{3.7}$$

where θ_I is the orientation of a patch centre in Υ_x^I and θ_E is the orientation of a patch centre in Υ_x^E. Once a texture sample is chosen for P_i, its appropriate rotation θ is applied before it is copied to T. Additionally, θ is stored for future use as it will be needed for our modified coherence search method. In Figure 3.16 we show the example texture on the left synthesized to resemble the orientation field in the centre. The orientation field is visualized using *Line Integral Convolution* (LIC) [13].

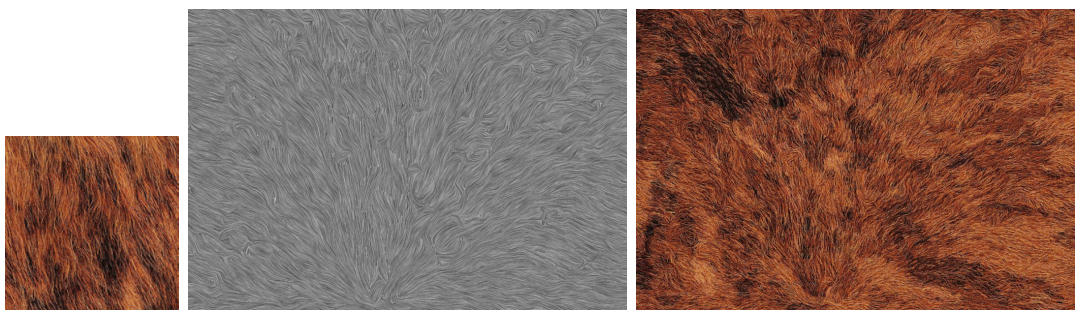


Figure 3.16: Directional Texture Synthesis. Left: Example texture, Centre: Orientation field visualized with LIC [13], Right: Synthesized texture.

Rotated Patch Search

Our coherence search method, as illustrated in Figure 3.10, must now consider rotations in order to locate candidate texture patches in E. Suppose that the jth neighbour of P_i was rotated by θ before it was copied to E and $\vec{v}_j$ indicates the offset of P_i from j. If we are to extend a texture patch from j's location in E then we must also consider how j is oriented in T. Figure 3.17 illustrates a simplified case where two uniform orientation fields for the texture example and input image are shown in the top row. Applying the rotation matrix $\mathbf{R}_{-\theta}$ to $\vec{v}_j$ allows us to correctly locate the offset patch in E. Our PatchMatch-based random search method remains unchanged from Algorithm 2.

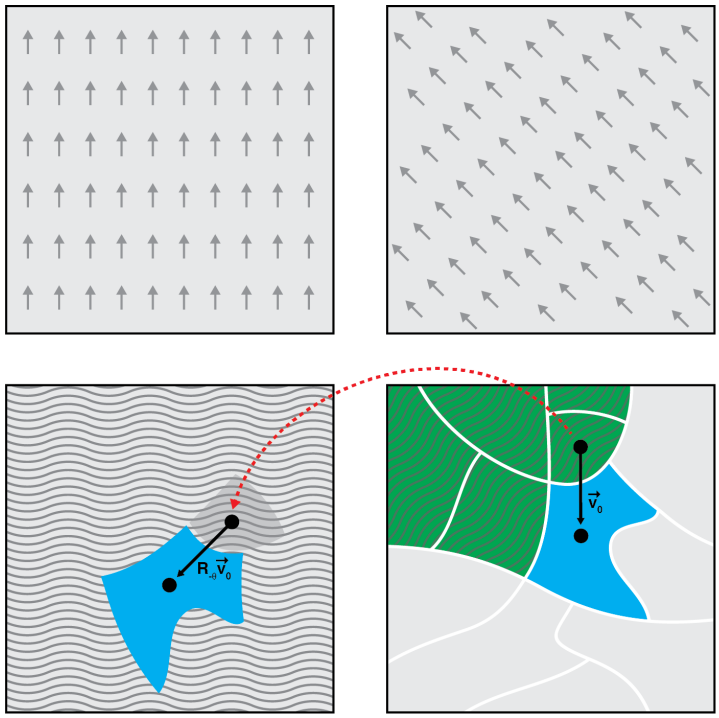


Figure 3.17: Top row: Orientation fields for the texture example and input image, Bottom row: Rotations must be reversed to locate offset texture patches.

3.3.3 Orientation Field construction

We now describe how our orientation fields are constructed. The first order image gradient G provides an initial approximation of local structure orientation in an image. However, this is often considered a poor choice for a couple reasons. First, a gradient vector is unable to distinguish between edge and corner structures. Second, it is desirable to consider neighbour information when describing local structure in order to reduce noise and propagate information into the space between the dominant structures. However, averaging gradient vectors is not possible. Consider a thin line in an image. On one side of this line you may have a positive gradient while the gradient will be negative on the other side. Averaging across this line will cause these values to cancel out. A better option is the second order structure tensor that is constructed by means of spatial smoothing the outer product of gradient vectors.

Formally, we define the structure tensor M as:

$$M := \sum_{i=0}^{N-1} w_i \begin{pmatrix} G_x^2(i) & G_x(i)G_y(i) \\ G_x(i)G_y(i) & G_y^2(i) \end{pmatrix}, \tag{3.8}$$

where G_x and G_y are partial derivatives in the x and y directions of an image. A weighted sum is computed from the outer product of G over a neighbourhood, where w_i provide the weighing factors. While Gaussian weights are often used, we consider smoothing across edges to be undesirable since orientation information can spread into unrelated image regions. A bilateral weighting scheme can be used but ideal range weights are often difficult to set as they can vary spatially between image regions. We use neighbourhood masks provided by the cumulative range geodesic filter instead. This gives us a unique mask for each pixel location, composed of the n most similar pixels at each location. Using this mask we set w_i to $\frac{1}{n}$.

The structure tensor M is a symmetric positive semi-definite matrix. Its eigendecomposition results in two non-negative eigenvalues λ_1 and λ_2 which are presented in order of decreasing magnitude. The two corresponding orthogonal eigenvectors $\vec{e}_1$ and $\vec{e}_2$ are oriented towards and parallel to the smoothed image gradient. We determine the local structure orientation θ at each location x as:

$$\theta(x) = tan^{-1}\left(\frac{\vec{e}_{1y}(x)}{\vec{e}_{1x}(x)}\right) \tag{3.9}$$

and assign $\theta(x)$ to the appropriate orientation field, Υ_x^E or Υ_x^I, to be used in Algorithm 3.

In addition to computing the orientation of a structure tensor it is necessary measure its coherence. Tensors whose eigen-decomposition results in one dominant eigenvector, $\lambda_1 \gg \lambda_2$, provide a good indication of local structure orientation. On the other hand, in cases where λ_1 and λ_2 are similar in strength, structure orientation becomes unreliable. We compute this coherence measure as:

$$C(x) = \frac{\lambda_1(x) - \lambda_2(x)}{\lambda_1(x) + \lambda_2(x) + K}, \tag{3.10}$$

where K is a constant that we introduce to prevent division by zero and lower the

coherence measure of weak structures. This coherence measure will be used to compute orientation distance when we discus tensor field smoothing next. Additionally, we can use spatial masks derived from coherence information to transfer texture unevenly amongst image regions.

Tensor field smoothing

The resulting structure tensor field will, at this point, contain detailed information on the local structure of our input image. However, for our patch-based texture synthesis method, we can not create oriented textures at a scale smaller than our designated patch size. The solution to this problem is tensor field smoothing.

Criminisi et al. [16] present a method for edge-aware colour smoothing in images by means of a geodesic distance transform. Similarly, we would like our tensor fields to exhibit similar smoothness properties, only now, we include orientation information to geodesic distances. In areas of similar colour and structure orientation we would like to see smooth orientation transitions. However, when we cross edges, either in colour space or orientation space, we would like to see discontinuities in our direction fields.

We adapt Criminisi et al.'s smoothing method as follows: We quantize structure tensors into k orientation bins between 0 and π. Orientations θ and $\pi + \theta$ are considered equivalent for our purposes since they provide the same information for orienting texture patches. Each bin is then represented by its weighted average, where tensors are weighted by their coherence measure. The pixels from these bins then become the masks from which we compute a geodesic distance transform. We use Dijkstra's algorithm to integrate distances from each pixel to its closest source in each of the k masks. Distances between any two pixels are computed as:

$$D = w_{xy} d_{xy} + w_{lab} d_{lab} + w_{or} d_{or}, \tag{3.11}$$

where colour and spatial distances are measured as in Equation 3.4. Distance terms d_{lab} and d_{or} are brought into a common frame of reference by dividing them by the 95th percentile of observed distance values in the appropriate domain. The weight terms w_{lab} and w_{or} are supplied by the user to indicate preferences. In all of our results we set w_{xy}, w_{lab}, and w_{or} to 1, 3, and 5, respectively. Following Akl et al. [2],

we compute the orientation distance d_{or} in Equation 3.11 is as:

$$d_{or}(M_1, M_2) = |sin(\theta_1 - \theta_2)| \times min(C_1, C_2), \tag{3.12}$$

With these measures we can now compute the distance between any two pixels in an image. We produce k distance fields, by means of Dijktra's algorithm, so that each pixel contains its geodesic distance to its closest location in each of the k masks. These distances will be used to smooth our tensor field. We accomplish this by computing a new tensor at each location as a weighted average of our k representative orientations. We compute the weights for each orientation as:

$$W_i(x) = e^{-\frac{D_i^2(x)}{\phi^2}}, \tag{3.13}$$

with ϕ supplied as a user-defined parameter that controls the spread of influence from each of k quantized orientations. In our examples we set ϕ to 40. This provides a good balance between extremely smooth or detailed orientation fields. In Figure 3.18 we can see the transition from orientation masks to distance fields to weights for $k = 6$ quantized directions. On the left, the pixels that are quantized into one of the six orientation bins are shown in black. These pixels form the masks from which distance calculations are referenced. In the centre, we show the geodesic distance transform. The grey level of each pixel is shown as its distance from its closest site in the mask region. Finally, on the right we show the orientation weights that are calculated with Equation 3.13 using the pixel distances. These visualizations are derived from the image shown in the top left of Figure 3.19.

Also, in the bottom row of Figure 3.19 we show visualizations of the orientation fields using *Line Integral Convolution* (LIC) [13]. The original orientation field is presented on the left. On the right we see how the smoothing operation transforms it. In the top right, tensor coherence is shown for the original orientation field as it appears before smoothing. Coherence ranges in the interval $[0, 1]$ and is visualized equivalently from black to white.

Smooth orientation fields are computed for both the example texture and the input image and are supplied as Υ^E, and Υ^I for inputs into Algorithm 3. As we explained in Section 3.3.2, these orientation fields are used to guide the texture synthesis process so that local orientation in the output texture is aligned with local orientation in

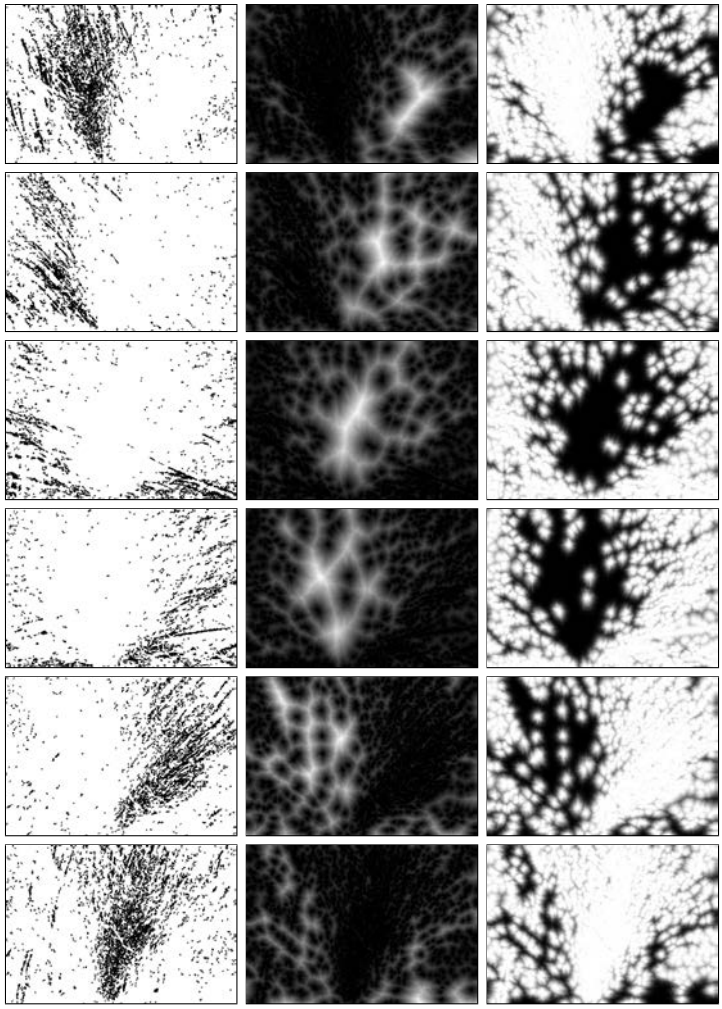


Figure 3.18: Computing weights for $k = 6$ orientations (top to bottom). Left: Orientation masks, Centre: Geodesic distance fields, Right: Orientation weights.

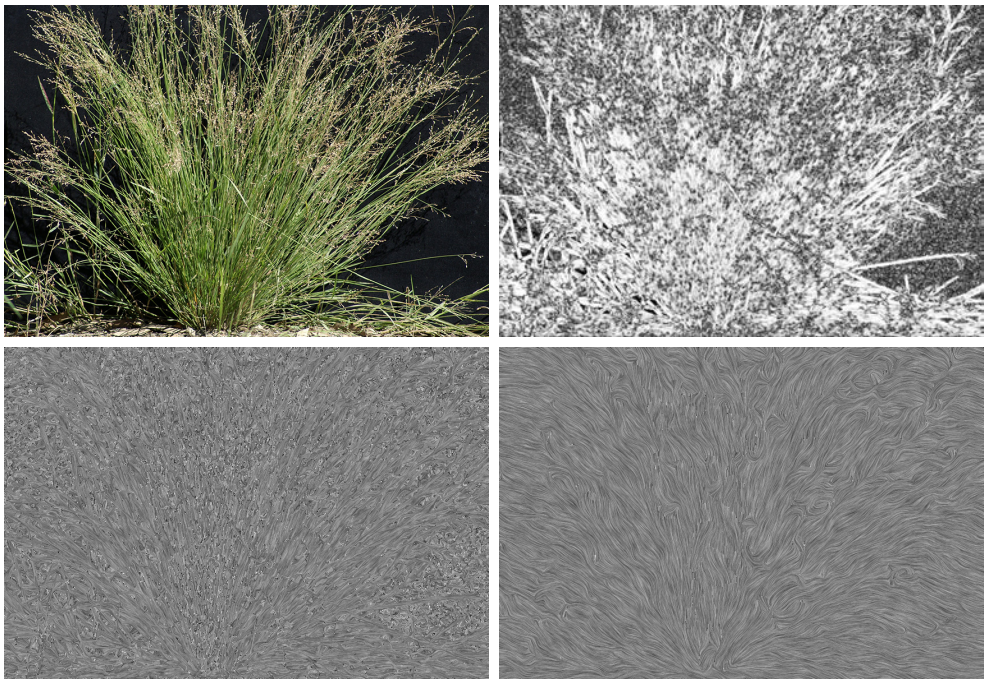


Figure 3.19: Image orientation field. Top left: Input image, Top right: Tensor coherence, Bottom left: Original orientation field, Bottom right: Smoothed orientation field. Orientation fields are visualized using LIC [13].

the input image. The purpose of this alignment is to create a greater harmonization between added textures and the original image. When the synthesized texture is transferred to the input, by means of Laplacian pyramid coefficient mixing, we should see new image textures that are oriented to the original edges and textures of the input. The next chapter will show the results of the algorithms that were presented here and provide discussion on its success and limitations.

Chapter 4

Results and Discussion

In this chapter we present the results of our texture transfer method. Using a range of subject matter, from portraits to still lifes and landscapes, we show how the following visual effects can be created. By adding large-scale textures to an image, we can blend the appearance of two images. If we incorporate high-frequency stochastic textures into an image, we can increase the visual richness of otherwise flat image regions. The appearance of film grain in classic photography once provided similar characteristics. Other, highly stylized effects are created by coercing textures to follow an orientation field. In this case, painterly effects can be exhibited, where textures follow edge and texture orientations, similar to an artist's brush.

Additionally, in this chapter we demonstrate how user-controlled parameters can alter our texture-enhanced results. Previous work has also attempted to transfer textures between images and we compare our results against some of these methods. Finally, we show some of the limitations of our system which include the types of textures that can be synthesized and their boundedness to the fronto-parallel plane.

Figure 4.1 and Table 4.1 show the input images and texture examples that we used in this chapter. We categorize texture examples as stochastic, isotropic, anisotropic, or non-stationary. Texture is synthesized using either the primary or directional texture synthesis method. We refer to these two methods as Algorithm 2 and 3, respectively. Unless otherwise indicated, stochastic and isotropic textures are synthesized using Algorithm 2 while anisotropic textures are synthesized with Algorithm 3. Non-stationary textures can be synthesized with either method and for each particular example we indicate which we use.

Table 4.1: Texture examples

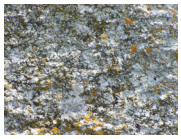
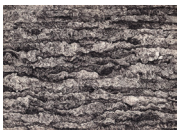
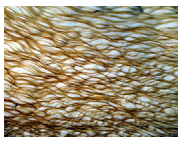
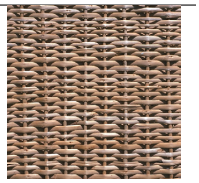
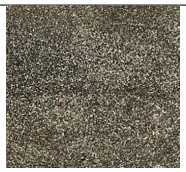
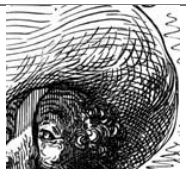
Category		**Textures**							
isotropic	a		b		c		d	
anisotropic	e		f		g		h	
anisotropic	i		j		k		l	
stochastic	m							
non-stationary	n		o					

Figure 4.1: Input images are chosen to represent a variety of manufactured or natural subject matter.

4.1 Results: Using a single texture

We present some results of our method in Figure 4.2 where the resulting images all have a single texture applied uniformly across the image plane. Later, in Section 4.2 we will discuss how masking effects can be used to vary the output at different spatial locations.

At the beginning of Section 2.4 we suggested some questions that should be considered while evaluating a texture transfer method. We are now in a position to reflect further on these questions in light of the results presented in Figure 4.2. Notice that at a casual glance, our altered images do not appear drastically different from the input images. This is caused by three reasons. First, we limit our method to only change the Laplacian coefficients of the luminosity channel. The colour pallette of the input image is thus retained. Second, our method maintains the dominant edges from the input. Finally, because our method limits coefficient mixing to a fixed number of pyramid levels, large-scale tonal variation is maintained from the input.

We believe that all of the above characteristics are desirable in the context of this work. Texture should be added where it is not already present. Where there is significant texture or edges in the input image, the effect should be minimal. In this sense, our texture transfer method is purely additive. Textures can be added to an image but not removed. In flat or lightly textured regions, the smooth maximum

function from Equation 3.1 favours Laplacian coefficients from the example texture. In highly textured regions, or at strong edges, the Laplacian coefficients from the input image are favoured. Observe the man's face in the uppermost location of Figure 4.2. The synthesized texture follows the contours in his face and is seen clearly. Yet, prominent features, such as the eyes and facial creases, are still just as visible as they are in the unaltered version. A similar effect can be seen in the bowl of cherries. The transferred texture can be seen on the fruit, but against the high contrast lines of the background, the new texture does not show through. In another example, the foreground skyscrapers in the third image are shown crisply against the stochastic texture. However, further back in the scene, where atmosphere envelopes the buildings, the added texture is more prominent. This effect creates an illusion of greater depth in the image as foreground and background are further separated.

In the second image from the top, which features a cluster of cranberries, we can clearly see the effect of the directional texture synthesis. Here, the texture seems to flow around the berries and leaves depicted in the image. Applying texture in this manner enhances the illusion of movement. Many NPR painting styles have sought a similar goal in emulating highly visible and energetic brushwork – often mentioning the painter Vincent van Gogh as a popular inspiration [22, 60].

Figure 4.3 removes texture synthesis from the system and demonstrates incorporating a static texture image into our input photograph. The type of texture shown here is difficult to synthesize, owing to its non-stationary features. Instead, images such as this are usually manually created by an artist. In the centre image we see this static texture added to the photograph using our Laplacian pyramid texture transfer method. Notice how the large features cross over the image edges and visually separate the photograph content from the new texture. The result at the bottom uses a texture synthesized from a small sample taken from the exemplar. We have lost the large scratches that help characterize this texture in this result. However, we also see a benefit in using a synthesized texture. Now the features from the texture example do not typically cross image edges since they are oriented parallel with them. This results in a more unified appearance between added texture and original content.

Figure 4.2: Single texture results. From left to right: original images, texture examples, results, result details. Texture examples (top to bottom): *e, g, m,* and *k*. All results were generated with $\alpha = 30$.

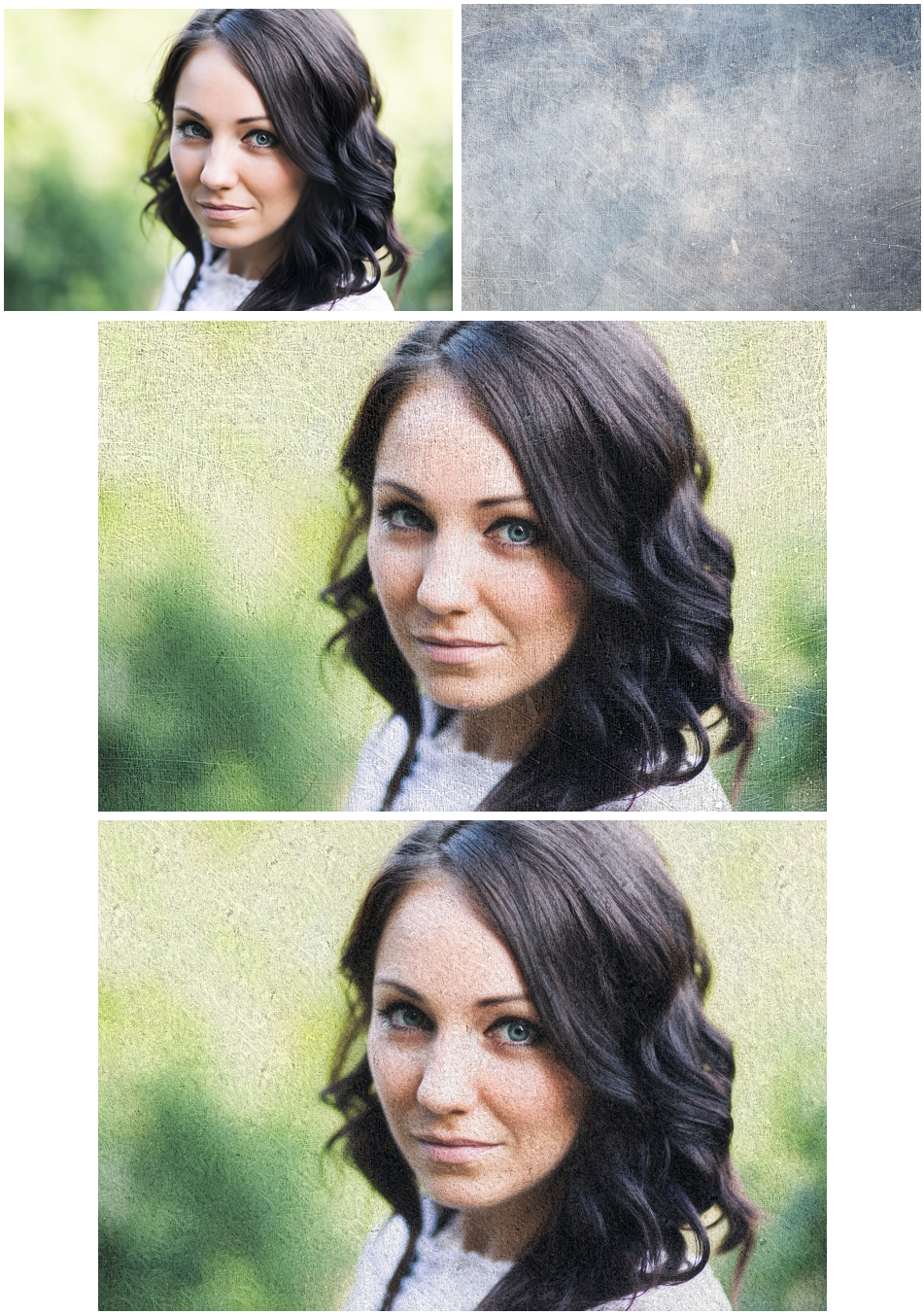


Figure 4.3: Removing texture synthesis from the system. Top left: input image, Top right: static texture, Centre: static texture result, Bottom: result using texture synthesis. $\alpha = 50$.

4.1.1 Image blending

Our method of using the Laplacian pyramid for transferring new textures into a photograph can also be used for image blending. We simply replace the auxiliary texture image for a secondary photograph. The method works as before, only now we don't apply the weighting function from Equation 3.2 to the Laplacian coefficients in the second image. Instead, we keep the coefficient values of the second image unchanged so that its visual qualities are not altered.

Figure 4.4 illustrates the result of image blending where the role of primary and secondary images alternates between left and right examples. One can see that these operations are not equivalent. For example, the outcome of adding the landscape to the boat is not the same as the outcome of the reverse. There are two reasons for this. First, as we only transfer Laplacian pyramid coefficients into the luminousity channel of the primary image, the colour palette remains unchanged. Second, the new image is built on the large scale tonal features that reside at top of the Laplacian pyramid in the primary image. Any changes that are made to the lower detail layers will be realized in relation to this starting position in the result. In both the left and right images, details are clear and well defined for both primary and secondary inputs.

4.2 Results: Using spatial masks

4.2.1 User-defined masks

The results in Figure 4.2 demonstrate the characteristics of our texture transfer method. However, in many cases, results can be more interesting if we give the user some control as to how textures are added to an image. A typical use-case involves a digital artist selecting targeted regions with an image editing application. This allows texture to be added independently to different image locations: one texture is applied to the foreground while another is applied to the background, for example. We use the GrabCut [51] semi-automated segmentation method to define a collection of masked regions on our input images. Manual work for this process usually demands 30–60 seconds of user time per image. It involves using a graphical user interface to label foreground and background pixels with a mouse. GrabCut then uses these designated pixels to guide the final segmentation.

Figure 4.4: Image blending examples. Primary image at top left: portrait, top right: cityscape, bottom left: landscape, bottom right: boat.

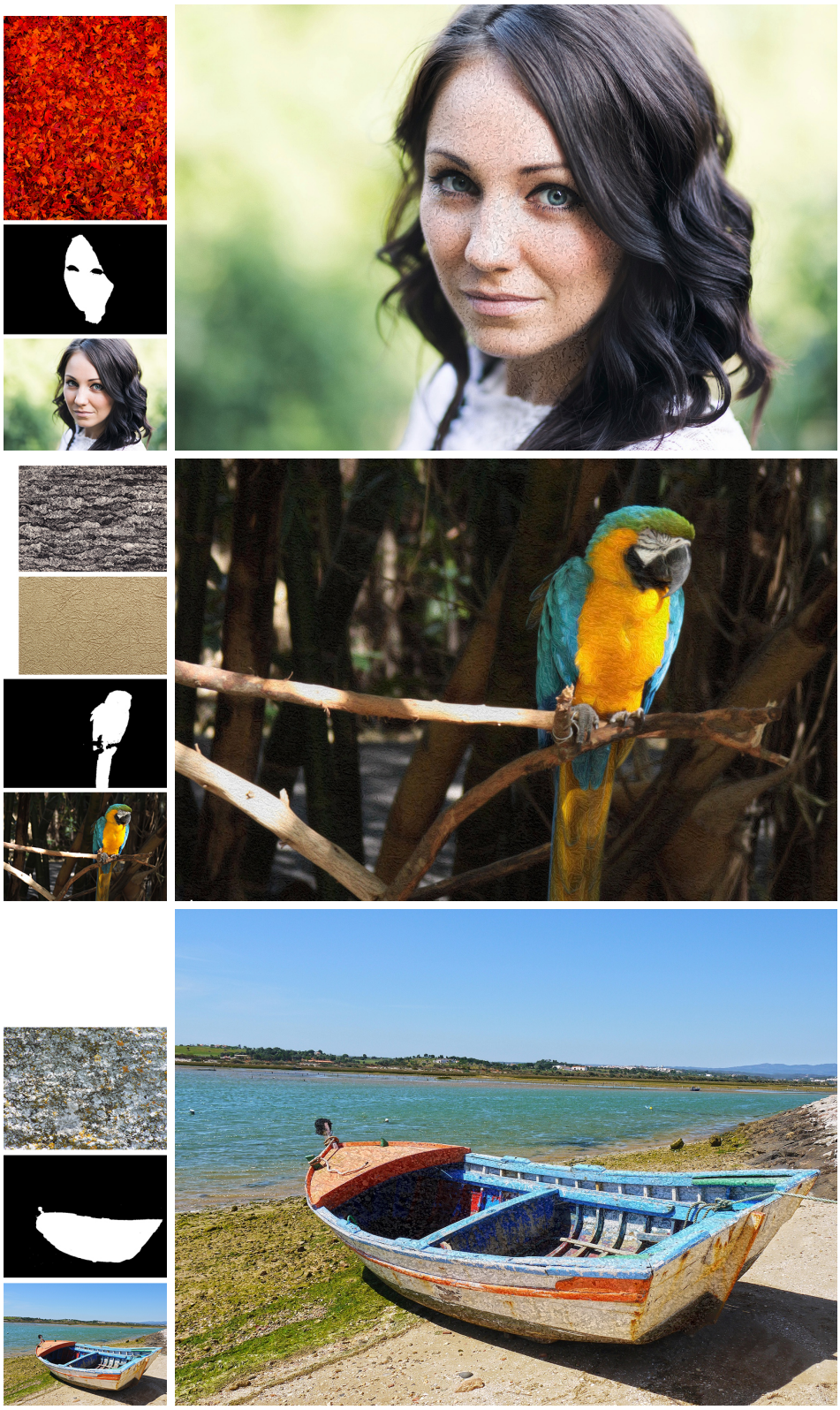


Figure 4.5: Results employing user-defined masks. Left column – top to bottom: texture examples, masks, input images. Top right: Texture example c ($\alpha = 40$), Centre right: Texture examples k and f ($\alpha = 40$ and 25), Bottom right: Texture example d ($\alpha = 40$).

Figure 4.5 shows some examples where we add texture to targeted regions of an image. Adding high frequency textures to part of an image can be used as a general technique to bring emphasis to a particular subject. This effect can be seen in the image of the boat. Adding external textures increases the perceptual salience of the boat as well as increasing its weathered appearance. A more obvious stylization effect is shown in the leaf texture that is added to the woman's face. The image of the parrot contains two added textures. One in the foreground and one in the background. Using two textures in the manner often helps to differential objects. We can also vary the texture intensity for each region independently. The background texture is applied more subtly to the forested area while the foreground texture is more obvious on the parrot. Section 4.5.1 will provide further discussion on this texture intensity parameter.

4.2.2 Automated masks

We can also apply automated methods to produce spatially guided maps that control how texture is added. Notice in the portrait image of Figure 4.2 how the texture is highly visible in the flat region at the bottom left, almost to the point of distraction. It seems natural to want to decrease the intensity of texture in low activity regions since it can appear out of proportion to the rest of the image. This can be done by spatially rescaling the Laplacian pyramid coefficients of the texture image. Using an activity map, that measures the local *textureness* [6] of a given pixel, we reduce the magnitude of Laplacian coefficients where textureness is low. Specifically, we follow the method of Bae et al. [6] and cross-bilateral filter the magnitude of a high-pass filtered version of the input image. The original input image is used to guide the cross-bilateral filter. The output is a piecewise smooth map where unrelated adjacent regions have minimal influence on each other. After taking a square root and normalizing the output, we obtain a textureness measure of each pixel in the interval $m = [0, 1]$. Formally, textureness is defined at each pixel as:

$$m(x, y) = \frac{m'(x, y)}{\max m'(x, y)}, \tag{4.1}$$

where the intermediate result, $m'(x, y)$, is obtained as:

$$m'(x, y) = \sqrt{B(x, y)}, \tag{4.2}$$

and the image $B(x, y)$ has been filtered as we described above. The square root operator is used to compress the intermediate result and achieve a more even distribution of textureness values. A visualization of this map is shown on the left of Figure 4.6. We can now modify the texture intensity term α, from Equation 3.2, to include both minimum and maximum texture intensity terms α_1 and α_2. The modified equation now becomes:

$$w_l(x, y) = \frac{\alpha_1 + m(x, y) \times (\alpha_2 - \alpha_1)}{\phi_l \times 2^l} \tag{4.3}$$

Notice in Figure 4.6 that we can still observe the texture in the lower left corner of the image. However, in contrast to the result from Figure 4.2, its intensity is not distracting.

Figure 4.6: Using textureness to guide spatial masks. Top left: input image, Top centre: textureness mask, Top right: example texture. $\alpha_1 = 10$, $\alpha_2 = 60$

We can also obtain spatial guidance by using the orientation coherence map that we obtained through Equation 3.10. The coherence of a given pixel provides a good

predictor for how well directional texture synthesis will represent the underlying image content. We can use this information to blend isotropic and anisotropic textures into image regions where coherence is low or high, respectively. Isotropic textures are synthesized using Algorithm 2 and anisotropic textures are synthesized using Algorithm 3. However, using this coherence measure directly is not an option since it will often contain a large amount of noise. Instead, we first apply a cross bilateral filter, as we did for the textureness map. The resulting coherence map contains entries in the interval $m = [0, 1]$. It is used to obtain a new texture Laplacian pyramid where individual coefficients are calculated as:

$$L_l^T(x, y) = m(x, y) \times L_l^{aniso}(x, y) + (1 - m(x, y)) \times L_l^{iso}(x, y), \tag{4.4}$$

where L_l^{aniso} and L_l^{iso} are Laplacian pyramids for the anisotropic and isotropic texture images. This new Laplacian pyramid will be used as the input for coefficient mixing in Equation 3.1.

Notice in Figure 4.7 how the two textures blend together. In the flat regions of the skin, an isotropic bark texture is present. Then as we look towards the jawline, the texture smoothly transitions to an anisotropic texture following the contours of the face. Texture in the jewellery is also mostly taken from the anisotropic source where it reinforces both linear and circular structures.

4.3 Dynamic Range Compression

In Figure 4.8 we illustrate the effect caused by applying dynamic range compression to an input image prior to texture transfer. We outlined the steps taken to compress highlight and shadow areas in Section 3.2.2. These steps involve choosing appropriate threshold parameters and interpolating luminousity levels across a compressed gradient field by means of Poisson blending. Notice that on the left, where we do not apply the dynamic range compression technique, the added texture is not well represented in the bright image regions. This is because the positive Laplacian coefficients from the texture image are clipped as they are pushed out of range. On the right, we compress these bright regions. Now the texture is visible even as it crosses into the previously saturated highlights.

Figure 4.7: Using orientation coherence as a spatially guided mask. Top row – left to right: input image, mask, texture examples a and i, Bottom: result.

Figure 4.8: Results showing the effect of dynamic range compression. Top left: input image, Top right: texture example k, Centre: without compression, Bottom: with compression. Threshold parameters are set to $t_{high} = 245$ and $t_{low} = 200$ for highlights and $t_{high} = 10$ and $t_{low} = 45$ for shadows. We set $\alpha = 40$.

4.4 Texture synthesis

The intent of our primary and directional texture synthesis methods, as we present them in Algorithms 2 and 3, are to provide high frequency content for texture transfer. While we do not claim to provide a general standalone texture synthesis method, it is worth looking at this aspect of our system in isolation. Figure 4.9 shows some of the synthesized textures that we can produce. The two columns on the left were synthesized with Algorithm 2 while the final two columns use Algorithm 3, guided by the orientation field shown at the top. Within each grouping, the left-most column compares square image patches against our SLIC-based patches on the right.

Our algorithms perform well on textures that contain busy stochastic elements. For example, the results in the central row, which use texture example f, demonstrate a successful outcome with no visible artifacts. However, problems arise when more clearly defined structures are present in the example texture. Ashikhmin [3] observes similar limitations where verbatim pixel copying ends and restarts. Since we borrow their coherent search mechanism in our work, we also expect to see similar behaviour. These restart artifacts are most apparent in the bottom example of Figure 4.9. On the left, the texture is synthesized using square patches. Consequently, a clear vertical seam can be seen where texture sampling runs off the edge of the example texture requiring a restart. To the right, we see the same texture as it is synthesized with SLIC-based patches. The same problem occurs, only now, the irregular SLIC grid helps to hide poor seam boundaries. This illustrates the advantage of using our irregular SLIC-based patches.

The two right-most examples for this bottom texture provide a similar comparison. Patch boundaries are difficult to align perfectly when texture rotations are used in Algorithm 3. However, if we compare the outcome between using a square grid and the irregular SLIC-based grid, it is clear that imperfections are more noticeable in the former case.

While we point out the noticeable flaws in our texture synthesis methods, we must also restate that our intent for producing textures is to transfer Laplacian pyramid coefficients into an input photograph. Once this texture transfer is complete, many of the flaws that we see here are lost among other image details.

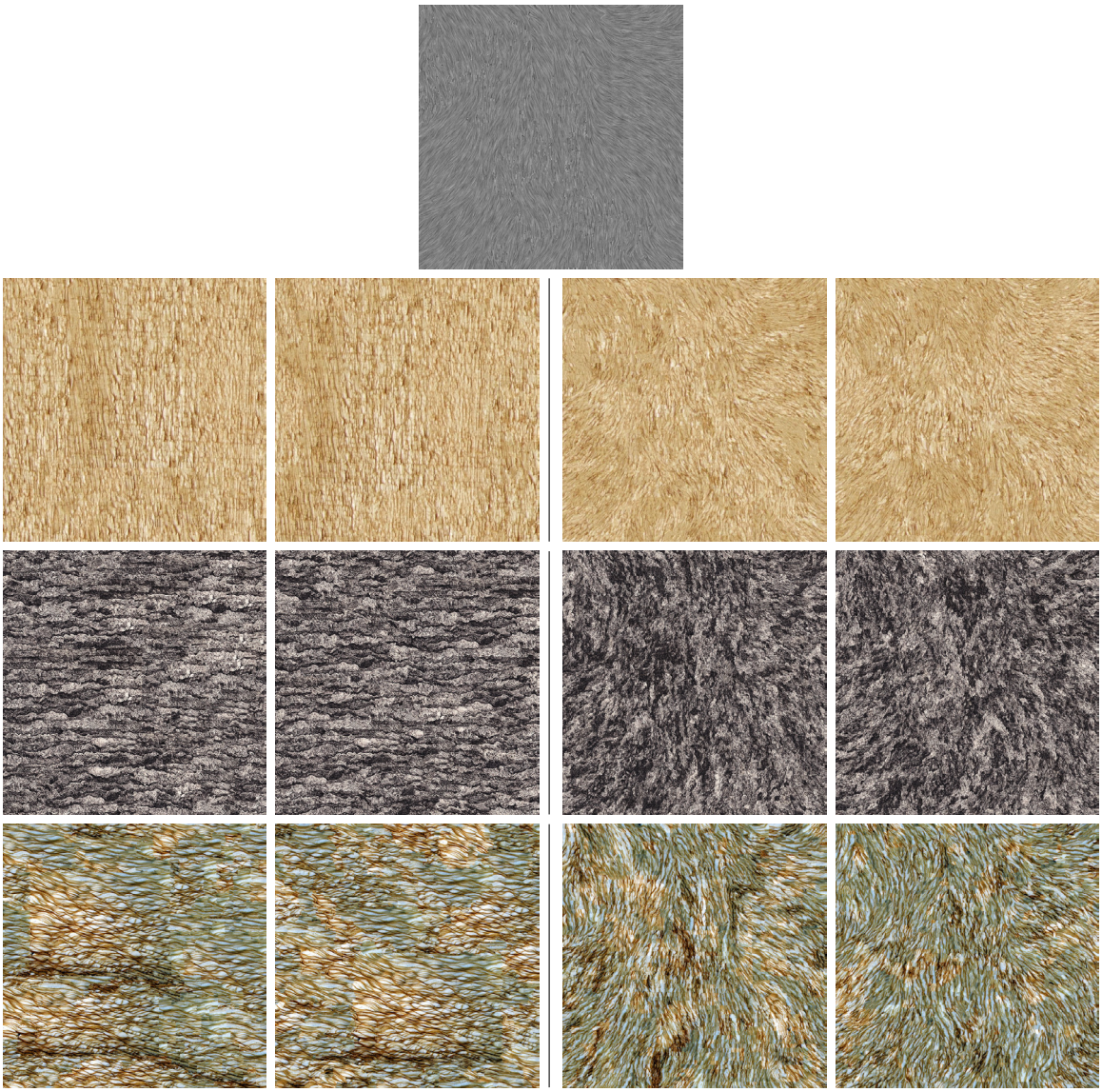


Figure 4.9: Texture synthesis results. From left to right: Square patches, SLIC patches, Square patches with rotation, SLIC patches with rotation. From top to bottom: We use texture examples *k, f,* and *g.* The orientation field is shown on the top.

4.5 Parameters

Our method employs several parameters where the user can control the output of the system. Some of these are set to default values, as indicated in Chapter 3, while others are intended for direct manipulation. We discuss user-controlled parameter settings in this section.

When texture is transferred into an input image, by means of its Laplacian pyramid coefficients, there are a group of parameters that can control the appearance of the output. For instance, we can control the degree that Laplacian pyramid coefficients are scaled in the smooth maximum function. Also within the Laplacian pyramid, we can choose the number of levels where texture coefficients are incorporated. We expect that applying textures to higher levels will exhibit more of the structural effects from the example texture. Limiting the texture transfer to a few levels at the bottom of the pyramid will likely produce textures that are more defined by surface characteristics. Most importantly, however, the texture transfer effect is controlled by a single parameter α that scales up and down the intensity of the newly incorporated textures.

The quality of our directional texture synthesis method is effected largely by the size of the atomic texture patch units. There will be a balance that must be found between choosing a patch size that is large enough to represent the structures from the example texture while small enough to synthesize convincingly smooth textures on an orientation field.

4.5.1 Laplacian pyramids

Smooth maximum

As we add new textures to an image we can inadvertently decrease the salience of image edges. While edge strength does not actually decrease with the addition of texture, its relative strength, in relation to newly textured regions, is decreased. Our smooth maximum function, which we present in Equation 3.1, helps to mitigate this issue. If the parameter k is set to a small number, less than 1 for example, then edge contrast will be increased. Figure 4.10 shows the effect of varying k at levels 1, 0.1, and 0.05. As k is decreased, image contrast is increased where input and texture Laplacian pyramid coefficients are similar in intensity.

Figure 4.10: Varying the smoothness parameter k in Equation 3.1. Top left: input image, Top right: texture example d, Bottom left: $k = 10$, Bottom centre: $k = 0.1$, Bottom right: $k = 0.05$. We set $\alpha = 40$.

Number of pyramid levels

A simple method of controlling the output of our texture transfer system is to vary the number of levels used in the Laplacian pyramids. The first level of the Laplacian pyramid contains the finest image details. Then as we ascend the pyramid, larger-scale features are captured. The second from the top image of Figure 4.11 shows a result produced while limiting Laplacian coefficient mixing to a single level. Here one can see that texture appears very much on the surface of the image. In contrast, the bottom two images of Figure 4.11 allow Laplacian coefficients to be mixed up to the 5th and 9th pyramid level, respectively. We now see more of the structural aspects of the example texture in the output.

Texture intensity

The parameter α in Equation 3.2 provides a predictable means to control the intensity of the added textures. Figure 4.12 illustrates different texturizing effects at $\alpha = 10$, 25, and 50. The lowest setting introduces texture subtly, to the extent that it is barely perceptible. At the highest setting, the added texture is emerging against the high contrast image details at the front of the car. This has probably pushed it too far but the preferred outcome is, of course, up to the user.

4.5.2 Texture Synthesis

Patch size

As with other patch-based texture synthesis methods, we recommend using a patch size at the scale of the largest features that we wish to synthesize. Since we are using SLIC super-pixels, patches will be uniform neither in shape nor in size. However, an average size (S^2) can be determined from the lattice step S used to seed super-pixel centres during initialization. In the context of directional texture synthesis, we have another aspect to consider. If patch sizes are too large then the illusion of texture following an orientation field is lost. At the bottom-right of Figure 4.13 we see a texture that has been synthesized with large patches to follow the orientation field shown at the bottom-right. In this example, using texture patches produced with $S = 30$, disconnected regions are seen at patch boundaries where orientation changes

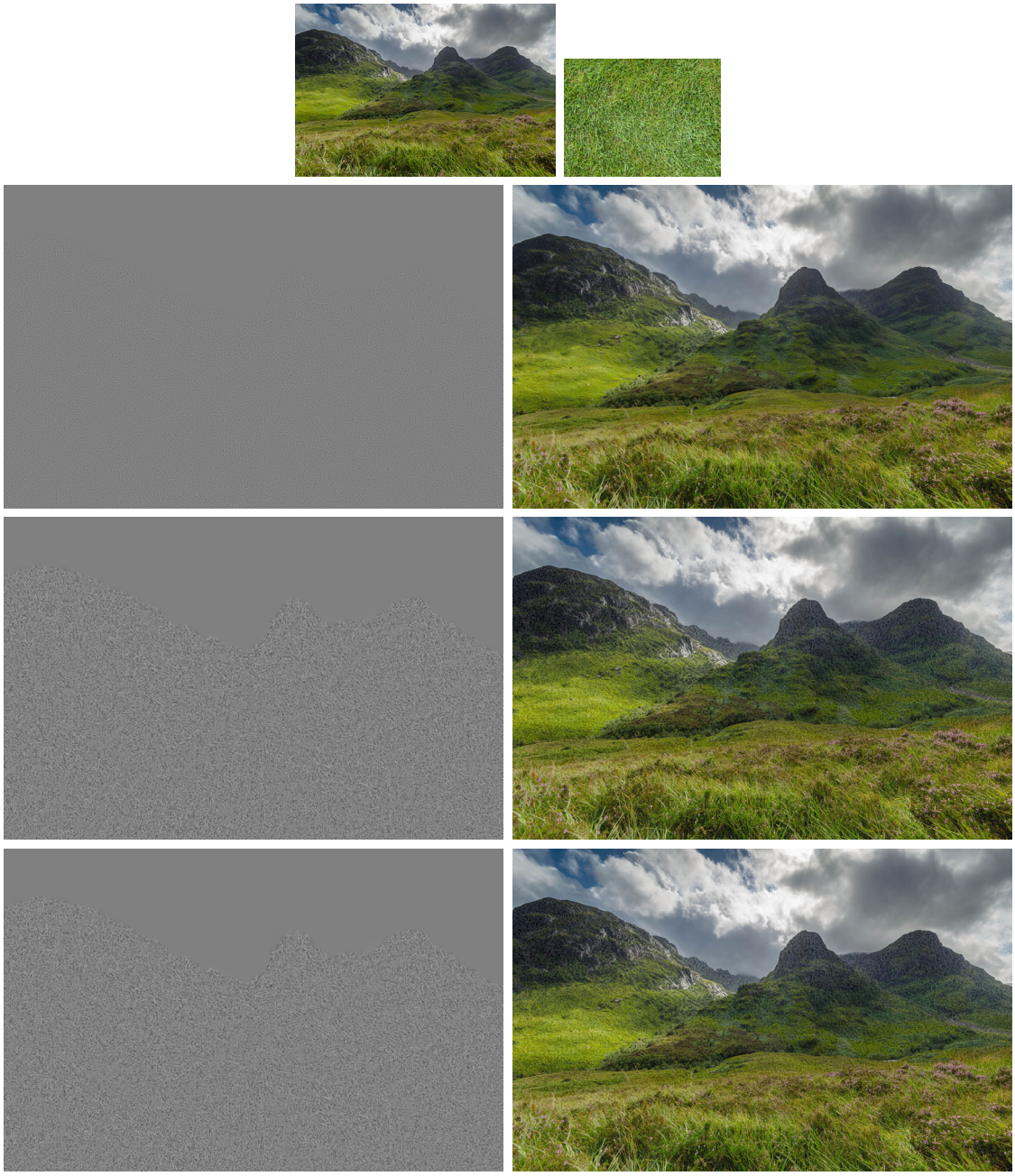


Figure 4.11: Varying the number of Laplacian pyramid levels. Top left: input image, Top right: texture example b, Second from top: 1 level, Second from bottom: 5 levels, Bottom: 9 levels. Texture coefficients are shown on the left against a neutral background.

Figure 4.12: Varying the texture intensity parameter α: Top left: input image, Top centre: mask, Top right: texture example n, Bottom left: $\alpha = 10$, Bottom centre: $\alpha = 25$, Bottom right: $\alpha = 50$.

rapidly. In principle, no patch size larger than a single pixel can produce smoothly flowing textures. However, the results shown in the bottom-left and bottom-centre of this figure exhibit reasonably smooth direction changes while using step sizes of 7 and 12, respectively.

4.6 Comparison

We now compare our texture transfer method against a few approaches that we consider to be most related to our own. While many of the goals in the presented work may be different than ours, we show how our method performs with similar, if not identical, inputs. One of the main techniques that causes our method to stand out from the others is the use of the Laplacian pyramid for adding textures. All of the other texture transfer systems in this section replace the input image content entirely with newly synthesized pixels – either across all colour channels or limited to the luminousity channel. This total replacement of the input pixels creates more pronounced appearance changes in the output. Our method maintains more of the characteristics from the input image, including large-scale tonal values and strong edges. Similar to work on texture synthesis, research on texture transfer has followed a pixel-based or patch-based approach. We show comparisons with both.

Figure 4.13: Varying the texture patch size. Top left: input image, Top centre: orientation field, Top right: texture example h, Bottom left: $S = 7$, Bottom centre: $S = 12$, Bottom right: $S = 30$

4.6.1 Comparison: Pixel-based texture transfer

Figure 4.14 compares our work against that of Hertzmann et al. [30]. In contrast to their work, we do not synthesize a new image entirely. Instead, we add textures into the finer-scale levels of the Laplacian pyramid allowing the original large-scale tonal values to remain unchanged. This difference suggests that the texture in our result was imposed on top of a photograph, while Hertzmann et al.'s result was created with the texture. Hertzmann et al.'s method does not apply any transformations to the example texture. As such, it is necessary that the example texture contains all the information that will be reproduced in the output. This information includes adequate luminousity levels and texture orientations to represent the input image.

Figure 4.14: Comparison with Hertzmann et al. [30]. Left: texture example, Centre: Hertzmann et al., Right: our result.

4.6.2 Comparison: Patch-based texture transfer

Our texture transfer system uses patch-based texture synthesis. In this section we compare our method against others that copy larger texture patches into an input image to produce a stylized effect.

A direct comparison with Wang et al.'s method is shown in Figure 4.15. We sampled a texture swatch from the same van Gogh painting, *Starry Night above the Rhone*, that is used in the comparison. Wang et al. produce an orientation field

that is based on the shape of large segmented image regions. This leads to large regions that are textured with repetitive brush strokes in a single direction. While our result shows a similar uniform texture direction in the sky region, the remainder of the image exhibits a greater variety of texture directions. Our local approach to smoothly gathering orientation information leads to a more realistic painting style – one that is informed by edge and texture information.

Figure 4.15: Comparison with Wang et al. [60]. Top left: input image, Top right: example texture, Bottom left: Wang et al., Bottom right: our result.

The Textureshop system of Fang et al. [23] goes further than we do in attempting to realistically apply texture to 3D object surfaces. In our work, we focus on automatically aligning textures with feature orientations. This is only a minimal concern in Textureshop. They instead allow a user to manually indicate texture orientation. Even then, orientation is on a much larger scale than we have locally defined it. The bulk of their efforts are, instead, focused on distorting the example texture to indicate the correct perspective of the target object. These differences make a direct comparison difficult. In the example, shown at the top of Figure 4.16, it appears that the lion sculpture is composed of the texture substance. In our approach, which can be

seen below, it appears that the texture has been etched into the surface of the stone.

4.6.3 Comparison: Alpha blending

As a final comparison, we compare our method against simple alpha blending. Alpha blending computes a new image R from a linear combination of the pixels from two inputs. In our case this will be an input image I and a texture image T. The blended result is computed as: $R(x, y) = (\alpha)I(x, y) + (1 - \alpha)T(x, y)$. One can see in the right image of Figure 4.17 that this naive calculation of new pixel values does not produce a good representation of either the input image or the texture. Furthermore the image contrast has been dulled and edges are no longer clear. Compare this result against our method on the left. In this example we use the same input and texture images. Our result has crisp edges, no colour distortion, and the new texture does not interfere with the input's original high texture areas.

4.7 Efficiency

Due to the subjective nature of image stylization effects, it is often necessary for users to take a trial and error approach before they are satisfied with an outcome. This requires a system fast enough so that different variations of inputs and parameters can be experimented with in a reasonable amount of time. Our directional texture synthesis method provides the main bottleneck in our texture transfer system. This is mainly due to repeatedly running Dijktra's algorithm to compute geodesic distance fields and determine the min-cut between texture patches. On the other hand, texture transfer, through the Laplacian pyramid, is a fast linear operation. In this scenario, once a texture has been synthesized to the dimensions and orientations of an input image, a user is able to experiment with various parameter and masking effects with fewer delays. Table 4.2 shows example running times that highlight the individual stages of our algorithms on two input images. Both examples use the same 640×425 example texture.

We have not attempted to optimize our CPU implementation in any way but we believe that some aspects could be improved with some effort. A common acceleration technique would be to implement parts of it on the GPU. Alternately, Table 4.2

Figure 4.16: Comparison with Fang et al. [23]. Top: texture example l, Centre left: input image, Centre right: result from Fang et al., Bottom left: input image, Bottom right: our result.

Figure 4.17: Comparison with alpha blending. Top left: input image, Top right: example texture, Bottom left: our method, Bottom right: alpha blending with 70% of the input image and 30% of the texture image blended into the luminousity channel of the input image.

Table 4.2: Timing Results (in seconds)

Operation	500×333	1700×1133
Directional texture synthesis ($S = 12$)	15.34	100.20
- Input subroutine: Geodesic Mask (12 pixels)	1.10	11.86
- Texture subroutine: Geodesic Mask (12 pixels)	1.91	1.75
- Input subroutine: Orientation smoothing (6 orientations)	3.34	54.39
- Texture subroutine: Orientation smoothing (6 orientations)	6.14	5.71
Poisson reconstruction (2 regions)	0.24	3.73
Laplacian pyramid coefficient mixing	0.03	0.38
Total	15.61	104.31

shows that the most computationally demanding phase of our system is smoothing our orientation fields. This requires several computations of an exact distance field over the image using Dijktra's algorithm, each one requiring $\mathcal{O}(NlogN)$ complexity. Criminisi [16] suggests that Fast Marching Methods [57] can be used to find an approximate distance transform in linear time. Approximating the distance transform in this manner would significantly improve our overall running time, although potentially affecting quality.

4.8 Limitations

4.8.1 Non-stationary textures

Our directional texture synthesis method has difficulty with non-stationary textures. Figure 4.12 shows how the results produced with Algorithm 2 sample uniformly from the example texture. However, when we add patch rotations in Algorithm 3, samples from the example texture tend to be drawn from regions of similar appearance.

A likely cause for this difference is the fact that Algorithm 2 copies larger verbatim image regions from the example texture. This allows it to push into regions with different image characteristics. In contrast to Algorithm 2, patches that are rotated in Algorithm 3 are extremely unlikely to have a zero error in the overlap region as features will not align perfectly. We believe that this inability to find exact overlap matches skews Algorithm 3 to sample from a random candidate at a greater frequency compared to Algorithm 2. The result is smaller contiguous texture chunks.

For the purpose of texture transfer, the inability to synthesize non-stationary textures is not necessarily a problem since it is often our intent to transfer high frequency texture characteristics. In fact, the example from the bottom of this figure is used as the texture input in Figure 4.12. Additionally, synthesizing large scale changes in image appearance in an uncontrolled manner could lead to unforeseen outcomes in the final texturized image.

4.8.2 Colour

Our method is unable to transfer colour information from the texture example to the input image. As a consequence, we restrict Laplacian pyramid coefficient mixing

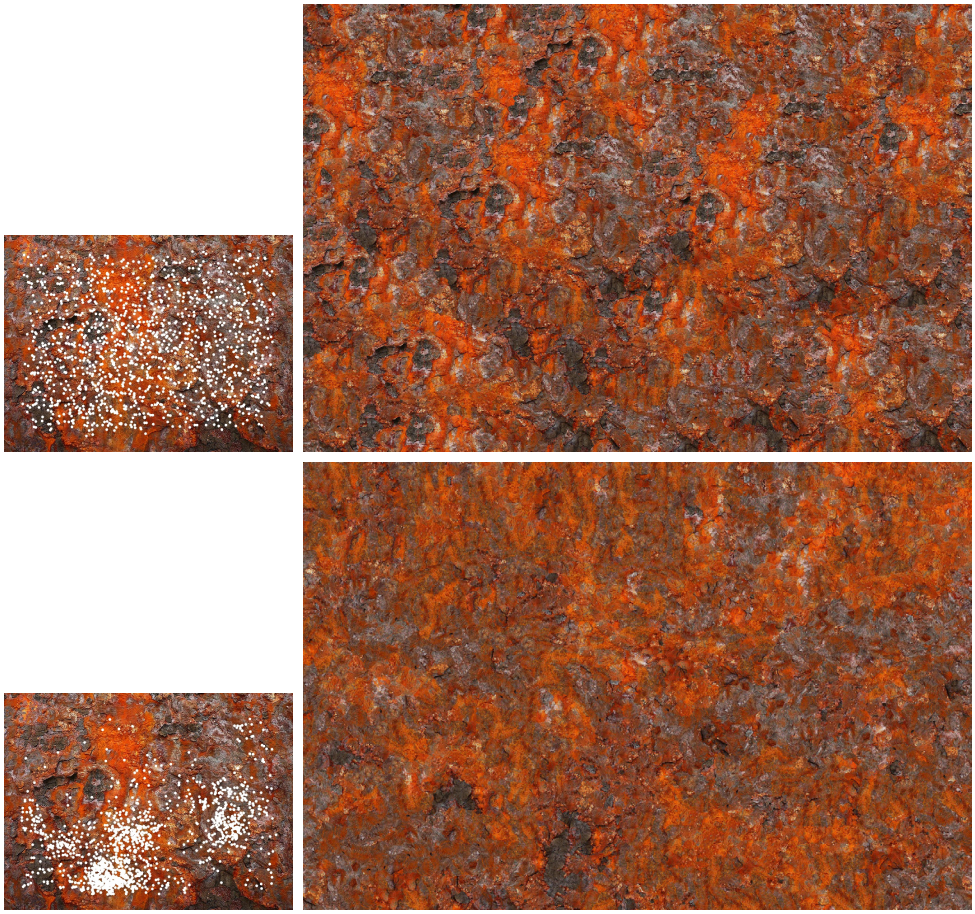


Figure 4.18: Synthesizing non-stationary textures. Top: Primary method (Algorithm 2), Bottom: Using a direction field (Algorithm 3), Left: Patch origins shown as white dots, Right: Results.

to the luminousity channel of CIELAB colour space. The reason for this limitation can be seen by considering the nature of the Laplacian pyramid coefficients. These coefficients do not represent colour values directly. Instead, they represent differences between an image and its low-pass filtered version at each level of the Gaussian pyramid. If we move coefficients from a texture image pyramid to an input image pyramid we now change their point of reference. This works for luminousity coefficients since we want textures to build on the large-scale features of upper pyramid levels. However, colour hue differences do not translate in the same way.

Another issue will be noise. If we allow our smooth maximum function from Equation 3.1 to operate independently at each location, pyramid level and colour channel, without correlating these pieces of information, then results could be unpredictable.

4.8.3 Perspective foreshortening

Currently, our texture synthesis method only produces fronto-parallel textures. Figure 4.19 illustrates that the illusion of a texture actually belonging to an object can be lost when it doesn't adhere to the correct object perspective. In the sphere example that we see here, the frequency of the texture features should increase as we move towards the object boundaries. This is due to the foreshortening effect caused when the object surface curves away from the line of sight. Textureshop produced plausible perspective effects by warping texture patches to follow the 3D object surface. We can use this as inspiration for our own method, to provide greater realism in our texture enhanced images. However, this would require 3D information of the scene that would have to be estimated with shape from shading methods.

4.9 Summary

In our approach, we introduce new coefficients into the Laplacian pyramid to add additional texture content to an image. This method allows synthesized textures, with orientation guidance, to be combined with the original image structures. The result retains, or enhances, the original image features, while adding new details that appear to belong to the original image. This seemingly natural inclusion is partly due to aligning the structure tensor orientation between the input image and new texture examples. Another reason is that texture information is mostly incorporated into

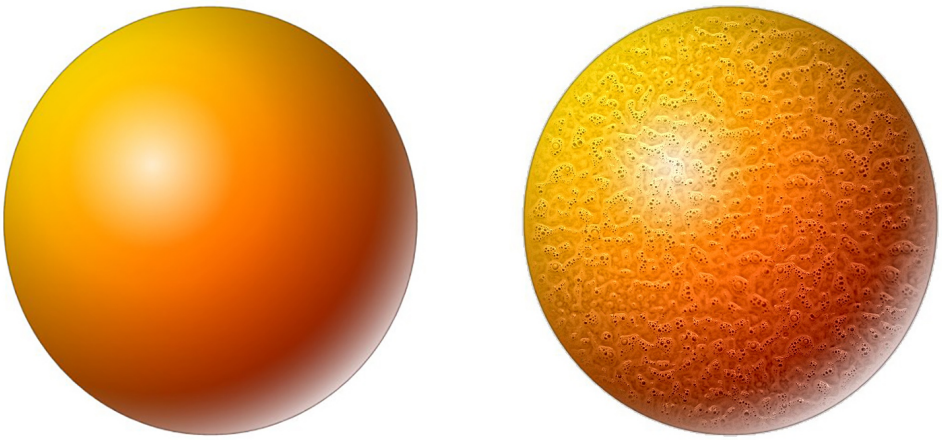


Figure 4.19: Our method does not produce perspective effects.

high-frequency bands of the Laplacian pyramid. Therefore, new texture coefficients are built on top of a base that contains the large-scale image features from the original photograph.

In this chapter we present the results garnered from our system and make comparisons to previous work. Unlike naive image blending, our method allows both the added textures and the original image content to be clearly represented in the output. Additionally, the most salient features from the input photograph are retained. These features include colour values, tonal range, strong edges, and highly textured regions. In contrast, other texture transfer methods destructively replace original image features for new content.

We discussed parameter settings that can alter the look of the output image. Specifically, we bring attention to the texture intensity tuning parameter α that allows the intensity of new textures to be added in a predictable fashion. Some of the limitations that we point out include difficulty synthesizing structured and non-stationary textures on an orientation field. These are larger problems in texture synthesis and have not been adequately addressed in the current work. Nevertheless, the oriented textures that we do synthesize are often good enough to be incorporated into our input images without drawing undue attention to imperfections.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we have addressed two distinct image processing techniques and combined them to stylize photographs with added textures. The first technique uses the Laplacian pyramid to mix coefficients between an input image and a synthesized texture. Using our method, we can produce image stylization effects that augment images with new textures while preserving strong edges and original textures. Additionally, we draw on previous work in patch-based texture synthesis and develop a method for synthesizing textures to follow an orientation field.

In previous work on pyramid-based image stylization it has often been reported that image pyramids, produced with linear filters, are unsuitable for many stylization effects. This is indeed the case when image contrast is enhanced through large coefficient scaling. The resulting halo artifacts are objectionable in photography applications. In our research, we show that the Laplacian pyramid can still be used to stylize images if coefficient scaling is kept minimal, especially in the upper levels of the pyramid.

Our research provides the following three contributions:

- We use the Laplacian pyramid to stylize an input image with fine-scale details from an example texture.
- We adapt patch-based texture synthesis to use SLIC [1] superpixels as primitive texture units.
- We develop a method to synthesize textures that follow an orientation guidance channel.

We use a smooth maximum function to introduce new texture coefficients into the Laplacian pyramid of an image. This function differs from the ordinary maximum in that it amplifies similar values. This amplification helps to preserve edges against a background of newly added texture. Our method builds upon the original large-scale features of the input image by replacing coefficients in the high-frequency bands of the Laplacian pyramid. Adding texture in this fashion enhances the illusion that textures naturally belong to the original image. Previous methods have sought to replace image content with new textures entirely.

We contribute to patch-based texture synthesis methods in two aspects. First, we use irregular SLIC-based patches as our atomic units of texture. This can benefit the synthesis process by hiding seam artifacts in unpredictable locations. In contrast, using a square grid of texture patches will place seams at regularly occurring intervals and therefore enabling visual patterns of seam artifacts to emerge. In Figure 4.9 we compared texture synthesis results between square and SLIC-based patches. Our irregular shaped patches show fewer artifacts in difficult to synthesize textures. Another reason to use SLIC-based patches is that when synthesizing textures for texture transfer it is desirable that patch boundaries align with image edges whenever possible. This adds another opportunity to hide seam imperfections since the Laplacian coefficients from the texture image are unlikely to be transferred to the output in these locations.

Our other contribution to texture synthesis is orienting texture patches with a smooth structure tensor field. We adapt Criminisi's geodesic colour smoothing method to operate on structure tensors. Using geodesic distances enables us to limit orientation averaging in an edge preserving manner while considering colour and orientation-space distances.

In this thesis, we demonstrated our method from a variety of perspectives. We experimented with different masking effects, demonstrated image blending, and highlighted individual aspects of the texture transfer system including dynamic range compression and texture synthesis. We found it especially interesting to guide the transition between isotropic and anisotropic textures through orientation coherence in Figure 4.7. We also compared our method against previous work in texture transfer. This allowed us to try out our method with paintbrush marks through a comparison with Wang et al. [60] in Figure 4.15. In a comparison with Fang et al. [23] we showed a result with a near regular texture. This can be seen in Figure 4.16. While our

method performs best on natural or near-stochastic textures, it was interesting to see that seam artifacts are not objectionable when viewed in the context of the final stylized image result.

Our system uses various parameters to control the result of texture transfer. Whenever possible we find default values for these parameters that work well with all images. In other instances we allow the user to control the output through a simple set of parametric controls. Most importantly, texture intensity can be scaled up and down through a single parameter in a predictable manner.

5.2 Future Work

While we have successfully designed a method to transfer example textures into an image, we see several avenues to extend our work. Drawing inspiration from Textureshop, we would like to enhance the realism of our added textures by considering 3D aspects. Using photometric methods from computer vision we can estimate the surface orientation of objects in the input image. We could use this information to adjust the magnitude and spatial frequency of texture features to create perspective effects.

Another direction of inquiry would be to improve the orientation continuity of our patch-based texture synthesis. Currently, the continuity of our synthesized texture is limited by the size of individual texture patches. Since any particular patch can only contain a single orientation, there will often be a discontinuity between orientations at patch seams. We saw this patchwork of orientations exaggerated in Figure 4.13 when texture patches become much larger than the changes in their underlying orientation field. Lu et al. [41] have employed patch warping techniques to automatically construct decorative patterns that follow curved lines. A similar approach could be taken by us so that patch orientations are better aligned at their seams.

Lastly, we suggest that demands on the user could be minimized with improvements to the GrabCut semi-automatic segmentation system. In our implementation, pixel differences are based solely on colour information. We believe that augmenting colour information with texture descriptors could lead to lower human involvement in our application. Since we ultimately wish to alter image textures, considering pixel similarity in a similar domain seems convincing. After all, any improvements that lower the demands on a user are greatly appreciated. This frees up time for other

higher-level tasks such as experimenting with different textures that can be used to enhance digital artworks.

List of References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk, "SLIC superpixels compared to state-of-the-art superpixel methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274–2282, Nov. 2012. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2012.120

[2] A. Akl, C. Yaacoub, M. Donias, J.-P. Da Costa, and C. Germain, "Texture synthesis using the structure tensor," *IEEE Transactions on Image Processing*, pp. 4082–4095, Nov. 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01267069

[3] M. Ashikhmin, "Synthesizing natural textures," in *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, ser. I3D '01. New York, NY, USA: ACM, 2001, pp. 217–226. [Online]. Available: http://doi.acm.org/10.1145/364338.364405

[4] ——, "Fast texture transfer," *IEEE Computer Graphics and Applications*, vol. 23, no. 4, pp. 38–43, Jul. 2003. [Online]. Available: http://dx.doi.org/10.1109/MCG.2003.1210863

[5] M. Aubry, S. Paris, S. W. Hasinoff, J. Kautz, and F. Durand, "Fast local laplacian filters: Theory and applications," *ACM Transactions on Graphics*, vol. 33, no. 5, pp. 167:1–167:14, Sep. 2014. [Online]. Available: http://doi.acm.org/10.1145/2629645

[6] S. Bae, S. Paris, and F. Durand, "Two-scale tone management for photographic look," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 637–645, Jul. 2006. [Online]. Available: http://doi.acm.org/10.1145/1141911.1141935

[7] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman, "Patchmatch: a randomized correspondence algorithm for structural image editing," *ACM Transactions on Graphics*, vol. 28, no. 3, p. 24, 2009.

[8] C. Barnes, E. Shechtman, D. B. Goldman, and A. Finkelstein, "The generalized patchmatch correspondence algorithm," in *Proceedings of the 11th European Conference on Computer Vision Conference on Computer Vision: Part III*, ser. ECCV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 29–43. [Online]. Available: http://dl.acm.org/citation.cfm?id=1927006.1927010

[9] P. Bénard, F. Cole, M. Kass, I. Mordatch, J. Hegarty, M. S. Senn, K. Fleischer, D. Pesare, and K. Breeden, "Stylizing animation by example," *ACM Transactions on Graphics*, vol. 32, no. 4, pp. 119:1–119:12, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2461912.2461929

[10] N. Bonneel, M. van de Panne, S. Lefebvre, and G. Drettakis, "Proxy-guided texture synthesis for rendering natural scenes," INRIA, INRIA Research Report RR-7298, May 2010. [Online]. Available: http://www-sop.inria.fr/reves/Basilic/2010/BVLD10

[11] P. Burt and E. Adelson, "The laplacian pyramid as a compact image code," *IEEE Transactions on Communications*, vol. 31, no. 4, pp. 532–540, Apr. 1983.

[12] P. J. Burt and E. H. Adelson, "A multiresolution spline with application to image mosaics," *ACM Transactions on Graphics*, vol. 2, no. 4, pp. 217–236, Oct. 1983. [Online]. Available: http://doi.acm.org/10.1145/245.247

[13] B. Cabral and L. C. Leedom, "Imaging vector fields using line integral convolution," in *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '93. New York, NY, USA: ACM, 1993, pp. 263–270. [Online]. Available: http://doi.acm.org/10.1145/166117.166151

[14] J. P. Collomosse and P. M. Hall, "Genetic paint: A search for salient paintings," in *Proceedings of the 3rd European Conference on Applications of Evolutionary Computing*, ser. EC'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 437–447. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-32003-6_44

[15] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, May 2002. [Online]. Available: http://dx.doi.org/10.1109/34.1000236

[16] A. Criminisi, T. Sharp, C. Rother, and P. P'erez, "Geodesic image and video editing," *ACM Transactions on Graphics*, vol. 29, no. 5, pp. 134:1–134:15, Nov. 2010. [Online]. Available: http://doi.acm.org/10.1145/1857907.1857910

[17] L. Cruz, L. Velho, E. Galin, A. Peytavie, and E. Guerin, "Patch-based terrain synthesis," in *Proceedings of the 10th International Conference on Computer Graphics Theory and Applications*, ser. GRAPP 2015. Portugal: SCITEPRESS - Science and Technology Publications, Lda, 2015, pp. 189–194. [Online]. Available: https://doi.org/10.5220/0005360201890194

[18] S. Darabi, E. Shechtman, C. Barnes, D. B. Goldman, and P. Sen, "Image melding: Combining inconsistent images using patch-based synthesis," *ACM Transactions on Graphics*, vol. 31, no. 4, pp. 82:1–82:10, Jul. 2012. [Online]. Available: http://doi.acm.org/10.1145/2185520.2185578

[19] Digital Photography Review. (2011) Software technique: Creating and adding textures. [Online]. Available: https://www.dpreview.com/articles/7269948481/software-technique-creating-and-adding-textures

[20] A. A. Efros and W. T. Freeman, "Image quilting for texture synthesis and transfer," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 341–346. [Online]. Available: http://doi.acm.org/10.1145/383259.383296

[21] A. A. Efros and T. K. Leung, "Texture synthesis by non-parametric sampling," in *Proceedings of the International Conference on Computer Vision*, ser. ICCV '99, vol. 2. Washington, DC, USA: IEEE Computer Society, 1999, pp. 1033–. [Online]. Available: http://dl.acm.org/citation.cfm?id=850924.851569

[22] M. Elad and P. Milanfar, "Style transfer via texture synthesis," *IEEE Transactions on Image Processing*, vol. 26, no. 5, pp. 2338–2351, May 2017.

[23] H. Fang and J. C. Hart, "Textureshop: texture synthesis as a photograph editing tool," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 354–359, Aug. 2004. [Online]. Available: http://doi.acm.org/10.1145/1015706.1015728

[24] R. Fattal, M. Agrawala, and S. Rusinkiewicz, "Multiscale shape and detail enhancement from multi-light image collections," in *ACM SIGGRAPH 2007 Papers*, ser. SIGGRAPH '07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1275808.1276441

[25] R. Fattal, D. Lischinski, and M. Werman, "Gradient domain high dynamic range compression," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 249–256, Jul. 2002. [Online]. Available: http://doi.acm.org/10.1145/566654.566573

[26] W. T. Freeman, T. R. Jones, and E. C. Pasztor, "Example-based super-resolution," *IEEE Computer Graphics and Applications*, vol. 22, no. 2, pp. 56–65, Mar. 2002. [Online]. Available: http://dx.doi.org/10.1109/38.988747

[27] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2008.

[28] S. Hao, D. Pan, Y. Guo, R. Hong, and M. Wang, "Image detail enhancement with spatially guided filters," *Signal Processing*, vol. 120, no. C, pp. 789–796, Mar. 2016. [Online]. Available: https://doi.org/10.1016/j.sigpro.2015.02.017

[29] D. J. Heeger and J. R. Bergen, "Pyramid-based texture analysis/synthesis," in *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '95. New York, NY, USA: ACM, 1995, pp. 229–238. [Online]. Available: http://doi.acm.org/10.1145/218380.218446

[30] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 327–340. [Online]. Available: http://doi.acm.org/10.1145/383259.383295

[31] A. Hertzmann and D. Zorin, "Illustrating smooth surfaces," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive*

Techniques, ser. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 517–526. [Online]. Available: http://dx.doi.org/10.1145/344779.345074

[32] M. K. Johnson, K. Dale, S. Avidan, H. Pfister, W. T. Freeman, and W. Matusik, "Cg2real: Improving the realism of computer generated images using a large collection of photographs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 9, pp. 1273–1285, Sep. 2011. [Online]. Available: http://dx.doi.org/10.1109/TVCG.2010.233

[33] H. Kang, S. Lee, and C. K. Chui, "Coherent line drawing," in *Proceedings of the 5th International Symposium on Non-photorealistic Animation and Rendering*, ser. NPAR '07. New York, NY, USA: ACM, 2007, pp. 43–50. [Online]. Available: http://doi.acm.org/10.1145/1274871.1274878

[34] D. Kuri, E. Root, and H. Theisel, "Hexagonal image quilting for texture synthesis," in *22nd International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*. Václav Skala-UNION Agency, 2014, pp. 67–76.

[35] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra, "Texture optimization for example-based synthesis," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 795–802, Jul. 2005. [Online]. Available: http://doi.acm.org/10.1145/1073204.1073263

[36] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick, "Graphcut textures: Image and video synthesis using graph cuts," in *ACM SIGGRAPH 2003 Papers*, ser. SIGGRAPH '03. New York, NY, USA: ACM, 2003, pp. 277–286. [Online]. Available: http://doi.acm.org/10.1145/1201775.882264

[37] H. Lee, S. Seo, and K. Yoon, "Extended papers from npar 2010: Directional texture transfer with edge enhancement," *Comput. Graph.*, vol. 35, no. 1, pp. 81–91, Feb. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.cag.2010.11.008

[38] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, and H.-Y. Shum, "Real-time texture synthesis by patch-based sampling," *ACM Transactions on Graphics*, vol. 20, no. 3, pp. 127–150, Jul. 2001. [Online]. Available: http://doi.acm.org/10.1145/501786.501787

[39] C. Liu, J. Yuen, and A. Torralba, "Sift flow: Dense correspondence across scenes and its applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 5, pp. 978–994, May 2011.

[40] J. Long and D. Mould, "Improved image quilting," in *Proceedings of Graphics Interface 2007*, ser. GI '07. New York, NY, USA: ACM, 2007, pp. 257–264. [Online]. Available: http://doi.acm.org/10.1145/1268517.1268559

[41] J. Lu, C. Barnes, C. Wan, P. Asente, R. Mech, and A. Finkelstein, "Decobrush: Drawing structured decorative patterns by example," *ACM Transactions on Graphics*, vol. 33, no. 4, pp. 90:1–90:9, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2601097.2601190

[42] D. Mould, "Image and video abstraction using cumulative range geodesic filtering," *Computers and Graphics*, vol. 37, no. 5, pp. 413–430, Aug 2013.

[43] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2227–2240, Nov 2014.

[44] A. Nealen and M. Alexa, "Hybrid texture synthesis," in *Proceedings of the 14th Eurographics Workshop on Rendering*, ser. EGRW '03. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, pp. 97–105. [Online]. Available: http://dl.acm.org/citation.cfm?id=882404.882419

[45] F. Okura, K. Vanhoey, A. Bousseau, A. A. Efros, and G. Drettakis, "Unifying color and texture transfer for predictive appearance manipulation," *Computer Graphics Forum*, vol. 34, no. 4, pp. 53–63, Jul. 2015. [Online]. Available: http://dx.doi.org/10.1111/cgf.12678

[46] S. Paris, S. W. Hasinoff, and J. Kautz, "Local laplacian filters: Edge-aware image processing with a laplacian pyramid," in *ACM SIGGRAPH 2011 Papers*, ser. SIGGRAPH '11. New York, NY, USA: ACM, 2011, pp. 68:1–68:12. [Online]. Available: http://doi.acm.org/10.1145/1964921.1964963

[47] P. Pérez, M. Gangnet, and A. Blake, "Poisson image editing," in *ACM Transactions on Graphics*, vol. 22, no. 3. ACM, 2003, pp. 313–318.

[48] G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. Hoppe, and K. Toyama, "Digital photography with flash and no-flash image pairs," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 664–672, Aug. 2004. [Online]. Available: http://doi.acm.org/10.1145/1015706.1015777

[49] E. Praun, A. Finkelstein, and H. Hoppe, "Lapped textures," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 465–470. [Online]. Available: http://dx.doi.org/10.1145/344779.344987

[50] P. Rosin and J. Collomosse, *Image and Video-Based Artistic Stylization*. London, New York: Springer, 2013.

[51] C. Rother, V. Kolmogorov, and A. Blake, "Grabcut: Interactive foreground extraction using iterated graph cuts," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 309–314, Aug. 2004. [Online]. Available: http://doi.acm.org/10.1145/1015706.1015720

[52] Y. Shih, S. Paris, C. Barnes, W. T. Freeman, and F. Durand, "Style transfer for headshot portraits," *ACM Transactions on Graphics*, vol. 33, no. 4, pp. 148:1–148:14, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2601097.2601137

[53] Y. Shih, S. Paris, F. Durand, and W. T. Freeman, "Data-driven hallucination of different times of day from a single outdoor photo," *ACM Transactions on Graphics*, vol. 32, no. 6, pp. 200:1–200:11, Nov. 2013. [Online]. Available: http://doi.acm.org/10.1145/2508363.2508419

[54] Y.-Z. Song, P. L. Rosin, P. M. Hall, and J. Collomosse, "Arty shapes," in *Proceedings of the Fourth Eurographics Conference on Computational Aesthetics in Graphics, Visualization and Imaging*, ser. Computational Aesthetics'08. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2008, pp. 65–72. [Online]. Available: http://dx.doi.org/10.2312/COMPAESTH/COMPAESTH08/065-072

[55] K. Sunkavalli, M. K. Johnson, W. Matusik, and H. Pfister, "Multi-scale image harmonization," *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 125:1–125:10, Jul. 2010. [Online]. Available: http://doi.acm.org/10.1145/1778765.1778862

[56] The Metropolitan Museum of Art. (2012) First major exhibition devoted to history of manipulated photography before digital age opens at metropolitan museum october 11. [Online]. Available: http://www.metmuseum.org/press/exhibitions/2012/faking-it

[57] P. J. Toivanen, "New geodesic distance transforms for gray-scale images," *Pattern Recognition Letters*, vol. 17, no. 5, pp. 437–450, May 1996. [Online]. Available: http://dx.doi.org/10.1016/0167-8655(96)00010-4

[58] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Computer Vision*. IEEE, Jan. 1998, pp. 839–846.

[59] X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo, and H.-Y. Shum, "Synthesis of bidirectional texture functions on arbitrary surfaces," in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '02. New York, NY, USA: ACM, 2002, pp. 665–672. [Online]. Available: http://doi.acm.org/10.1145/566570.566634

[60] B. Wang, W. Wang, H. Yang, and J. Sun, "Efficient example-based painting and synthesis of 2d directional texture," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 3, pp. 266–277, May 2004.

[61] L.-Y. Wei and M. Levoy, "Fast texture synthesis using tree-structured vector quantization," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 479–488.

[62] Y. Wexler, E. Shechtman, and M. Irani, "Space-time completion of video," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 3, pp. 463–476, Mar. 2007. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2007.60

[63] Y.-Q. Xu, B. Guo, and H. Shum, "Chaos mosaic: Fast and memory efficient texture synthesis," Tech. Rep. MSR-TR-2000-32, Microsoft Research, Tech. Rep., April 2000.