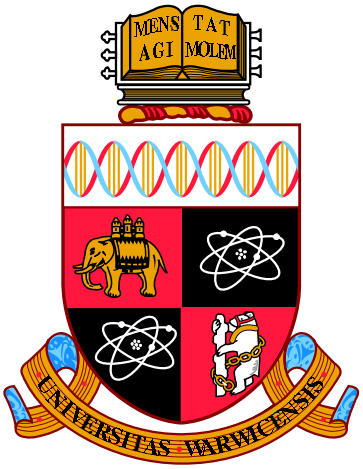


High Quality Texture Synthesis

by

Martin Kolář

Thesis

Submitted to the University of Warwick

for the degree of

Doctor of Philosophy

Warwick Manufacturing Group

September 2016

THE UNIVERSITY OF
WARWICK

Contents

List of figures and illustrations

List of Tables

Acknowledgments

I would like to thank Professor Alan Chalmers and Kurt Debattista for their patience and support throughout my time at Warwick. I'm grateful to Johnson Tiles for providing the CASE studentship which funded my research.

Great thanks go to my fellow students Ratnajit, Jon, Josh, Amar, Pinar, Tim, Ali, Rossella, Elmedin, Stratos, Debmalya, Tom, and Carlo for lending a hand with the challenging little things.

Finally, I'd like to thank my family and friends for being there and bearing with me while I've been so busy.

Declarations

This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree. The work presented (including data generated and data analysis) was carried out by the author.

Abstract

Texture synthesis is a core process in Computer Graphics and design. It is used extensively in a wide range of applications, including computer games, virtual environments, manufacturing, and rendering. This thesis investigates a novel approach to texture synthesis in order to significantly improve speed, memory requirements, and quality. An analysis of texture properties is created, to enable the gathering a representative dataset, and a qualitative evaluation of texture synthesis algorithms.

A new algorithm to make non-repeating texture synthesis on-the-fly possible is developed, tested, and evaluated. This parallel patch-based method allows repeatable sampling without cache, without creating visually noticeable repetitions, as confirmed by a perceptive objective study on quality.

In order to quantify the quality of existing algorithms and to facilitate further development in the field, desired texture properties are classified and analysed, and a minimal set of textures is created according to these properties to allow subjective evaluation of texture synthesis algorithms. This dataset is then used in a user study which evaluates the quality of texture synthesis algorithms. For the first time in the field of texture synthesis, statistically significant findings quantify the quality of selected repeatable algorithms, and make it possible to evaluate new improved methods.

Finally, In an effort to make these findings applicable in the British tile manufacturing industry, the developed texture synthesis technology is made available to Johnson Tiles.

Chapter 1

Introduction

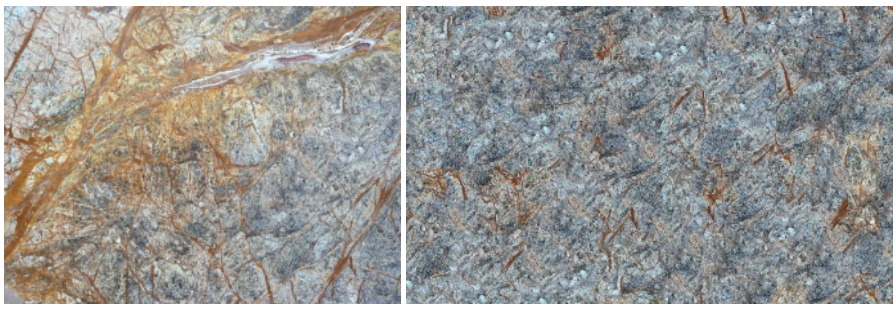
Example-based texture synthesis is a key component of computer graphics, design, and computer vision. It is used extensively in a wide range of applications, including computer games, inpainting, virtual environments, manufacturing, rendering, and editing.

In computer graphics, creating textures from a small example is a valuable tool to replicate visually pleasing patterns in new forms. For example, textures may be rendered into new environments, onto virtual objects, or over sections of an image to be hidden. In design, the manual task of creating large canvases from small examples can be greatly improved by using algorithmic texture synthesis methods. In other fields, such as computer vision, texture synthesis algorithms are used to create training images for large scale classifiers when real data is too sparse or difficult to gather [Yang et al., 2015, Hafemann et al., 2014, Sonka et al., 2014, Pietikäinen, 2000].

In the tile manufacturing industry, natural texture samples are collected, photographed, and manually combined to create canvases for profiling and printing onto ceramic tiles. This process is very labour-intensive, and requires an expert several days to create satisfactory results (figure 1.1).

1.1 Current State of Texture Synthesis

Texture synthesis algorithms are typically devised to increase perceived texture quality, rendering speed, scalability, and to lower memory requirements. In order to fulfil real-



(a) exemplar photograph (b) manually patched texture

Figure 1.1: Marble Texture for Tile Manufacturing

time rendering needs, parallel texture synthesis methods have been devised. However, none of these allow repeatable online synthesis without visually noticeable artefacts, making them impractical for some advanced applications, such as ray-tracing. Furthermore, the quality of non-parallel texture synthesis algorithms is deeply dependent on parameters, and such methods do not work well for arbitrary textures. Figure 1.2 shows a visual comparison of the best available texture synthesis algorithms. Note that all of these contain visible flaws, not present in the input (a). For example, methods (b) and (c) produce outputs by combining patches of the exemplar, and produce visible seams within texels, such as the cut through the green smartie in the top middle of figure 1.2(b). Methods (d) and (e) optimise quality per-pixel, and consequently produce smearing artefacts, deforming the elements of the input texture, such as the red smartie in the centre of figure 1.2(d). Because all published results rely on self-assessed quality measures on small samples of textures, it is crucial that a structured way of evaluating texture synthesis algorithm quality across textures is created to aid further development, and to enable the quality of existing methods to be objectively assessed.

1.2 Research Question

In order to advance the research in example-based texture synthesis, a key question has to be answered:

How can the quality in offline and online parallel texture synthesis algorithms be rigorously improved?

(a) exemplar

(b) Resynthesizer [Cornet and Rouquier, 2011]

(c) Image Quilting [Efros and Freeman, 2001]

(d) Self-Tuning [Kaspar et al., 2015]

(e) LazyFluids [Jamriška et al., 2015]

Figure 1.2: The smarties texture synthesized with state-of-the-art methods shows visible flaws

In the context of this work, *quality* refers to the perceived similarity to the original texture exemplar by users. *Users* are chosen to be application-specific, as those who will judge the texture in a given application.

1.3 Method Overview

After a detailed review of the current methods and a thorough analysis of their applicability, a new algorithm was devised. This algorithm uses a preprocessed set of patches and a version of categorical Perlin noise [Perlin, 1985]. A key property of the newly presented algorithm is that the quality of textures synthesised in the parallel online stage depends only on the quality of the offline preprocessing. Therefore, the texture created online can be of the same quality as that of any offline algorithm. The evaluation of quality for offline texture synthesis algorithms is then considered.

While automated "perceived satisfaction" metrics have been devised, none of them capture the broad properties of natural textures sufficiently well for practical ap-

plications. In previous work, which created synthesis algorithms to find global optima for Texture Energy [Kwatra et al., 2005] or Texture Energy with a Markov Random Field [Jamriska et al., 2012], the quality of outputs is insufficient for certain textures. This demonstrates that these metrics do not capture texture properties sufficiently well for automatic quality evaluation.

As no accepted metrics for automated evaluation of texture quality exist, texture synthesis algorithms need to be evaluated manually by users. For this purpose, dozens of texture datasets have been published and made available, each with a specific focus, for example Brodatz Textures [Brodatz, 1966], VisTex Textures [Graczyk, 1995], Colored Brodatz [Abdelmounaime and Dong-Chen, 2013], or the PSU Near-Regular Texture Database [Lee and Liu, 2005]. By synthesising textures with various algorithms on these datasets, one may subjectively gain insight into how well a given method handles each of the texture properties. However, these datasets contain thousands of textures in total, and do not necessarily contain textures which aggregate all known texture properties. In this thesis, these properties are agglomerated, and a minimal set of 12 textures is selected which covers all of these. These properties are either ordinal or binary, and the selected textures cover each ordinal property in the low-mid-high range, and the true and false cases of each binary property.

By synthesising these textures, a given algorithm can be evaluated visually. This can give meaningful insight into the quality performance of the algorithm across all texture properties. Furthermore, the textures are selected in such a way that a reference output is available. To make the manual comparison to other methods possible, outputs for six existing algorithms are also made available. These six algorithms and this reference output are compared in an anonymous user study, to create statistically significant orderings of perceived quality. By comparing outputs with a reference image, the results of this study objectively answer the question of absolute quality of the selected algorithms. This finding allows the use the best performing algorithms in conjunction with the parallel online sampling method from this work, making it possible to synthesise textures of high quality in a fast, online fashion.

Furthermore, the study reveals textures and texture properties for which current

algorithms do not perform sufficiently well, thus ushering further advancement in the field of example-based texture synthesis.

1.4 Thesis Overview

This thesis is composed of nine chapters, which logically follow the topic of texture synthesis from problem analysis to a structured evaluation of the solution. The contents of chapters 5, 7, and 8 have been published in peer-reviewed journals [Kolar et al., 2015, Kolar et al., 2017].

Chapter 2 provides a context for the goals and justifications of this work, explaining the applications of texture synthesis and outlining practical problems. Chapter 3 provides a detailed literature review of texture synthesis methods, datasets, quality perception, and related topics. This is followed by Chapter 4, which discusses the Research Methodology underpinning this work. This chapter explains the ultimate aim, and introduces a methodical storyline which the rest of the work then follows.

Chapter 5 presents the algorithm developed for fast online high quality texture synthesis. Here, state-of-the-art methods to achieve high quality, speed, and repeatability are analysed and compared, and a new method is developed so that it combines their strengths and benefits. The work of this chapter has been published in The Visual Computer [Kolar et al., 2015]. The following chapter, Chapter 6, presents a user study performed to justify the quality preserving aspect of the new algorithm.

After presenting the new algorithm, the thesis turns to the topic of subjective quality assessment. An extensive list of texture properties is compiled from existing work, to create a classification system across all textures. Binary and ordinal texture properties are concretised, and used to create a minimal set of textures which are representative of all potential textures. Chapter 7 explains the properties of textures used in texture synthesis, and shows which textures were selected to evaluate texture synthesis algorithms.

Then, in Chapter 8, a user study is prepared to analyse desirable and undesirable properties of texture synthesis algorithms, and the results are analysed to link the research back to the goals set out in the Research Methodology. Preferences across tex-

tures and algorithms are analysed for participants of the user study, to facilitate further development of texture synthesis algorithms. Chapters 7 and 8 have been published jointly at Eurographics [Kolar et al., 2017], the most prestigious conference on computer graphics in Europe.

Finally, the research in the thesis is concluded in Chapter 9, which includes a discussion listing emerging challenges. Various applications improving the ceramic tile manufacturing process are discussed throughout the text.

Chapter 2

Context

This work was funded by a CASE studentship with Johnson Tiles. This chapter describes the context within which texture synthesis is used as part of the Tile Manufacturing process.

2.1 Texture Synthesis for Tile Manufacturing

In the ceramic industry, texture synthesis is used to create tiles that are visually similar to an example of a natural material, but crucially minimise repetitions in the texture that may be immediately obvious on a wall of tiles. The current industrial practice is labour intensive. A texture "canvas" (typically $2m \times 2m$) is produced from an exemplar, or a series of exemplars, often by hand. A section of this canvas is then selected on the fly, and the chosen texture reproduced on a ceramic tile through an inkjet production process. The limited size of the canvas means that in runs of many thousands of tiles, some undesirable repetition will inevitably occur. The entire process is illustrated in Figure 2.1 using the example of a marble texture.

2.2 Profiling in Tile Manufacturing

It is crucial that the colour palette of the finished tile closely matches the desired design, but this is a challenging problem. The tile printing process requires an RGB[1] palette to

[1] Red, Green, and Blue

(a) scanned marble slabs (b) combined “canvas”

(c) generated tiles

Figure 2.1: Marble texture detail during various stages of production

be converted to CMYK[2], the design to be printed using industrial ink-jet printers of varying intensity for each channel, the tiles are glazed, and are then fired in kilns of varying temperature. As well as the size and shape of the tile itself, each of these factors affects colours in complex ways. The process of predicting how colours will be affected, and applying the inverse transformation to the desired template is called Profiling, and is a labour intensive manual process. As this printing set-up is a very labour intensive process, it could greatly benefit from automation.

2.3 Perceived Satisfaction

Rather than simply focusing on repeatability and accuracy measures, which are well defined [Wyszecki and Stiles, 1982], it is important that the produced tiles are visually pleasing. Repeatability quantifies similarity between two different outputs, and accuracy quantifies the similarity between the actual output and the desired output. Various metrics have been suggested to best account for human perception, to best model differences in colours [McLaren, 1976, Larson, 1998, Joblove and Greenberg, 1978], to understand which features are visually more important [Law and Siu, 1999, Banerjee et al., 2009], or a combination of both [DiCarlo and Wandell, 2000].

Pixels in a digital image can be defined in a panoply of ways, typically a combination of colour and brightness. Measuring differences in colours in a methodical way requires an understanding of how colours are perceived. In order to account for human perception, several metrics have been developed to be "perceptually uniform" (CIELUV [Robertson, 1977]), CIELAB [McLaren, 1976], logLUV [Larson, 1998]). In a perceptually uniform colour space, an equal distance in the colour space corresponds to equal differences in perceived colour (Figure 2.2). However, extending pixel metrics to texture comparison is not trivial. No consensus has yet been reached as to the quality of a synthetic texture, and for a given natural example, this is a key aspect of what needs to be done.

[2] Cyan, Magenta, Yellow, and Black

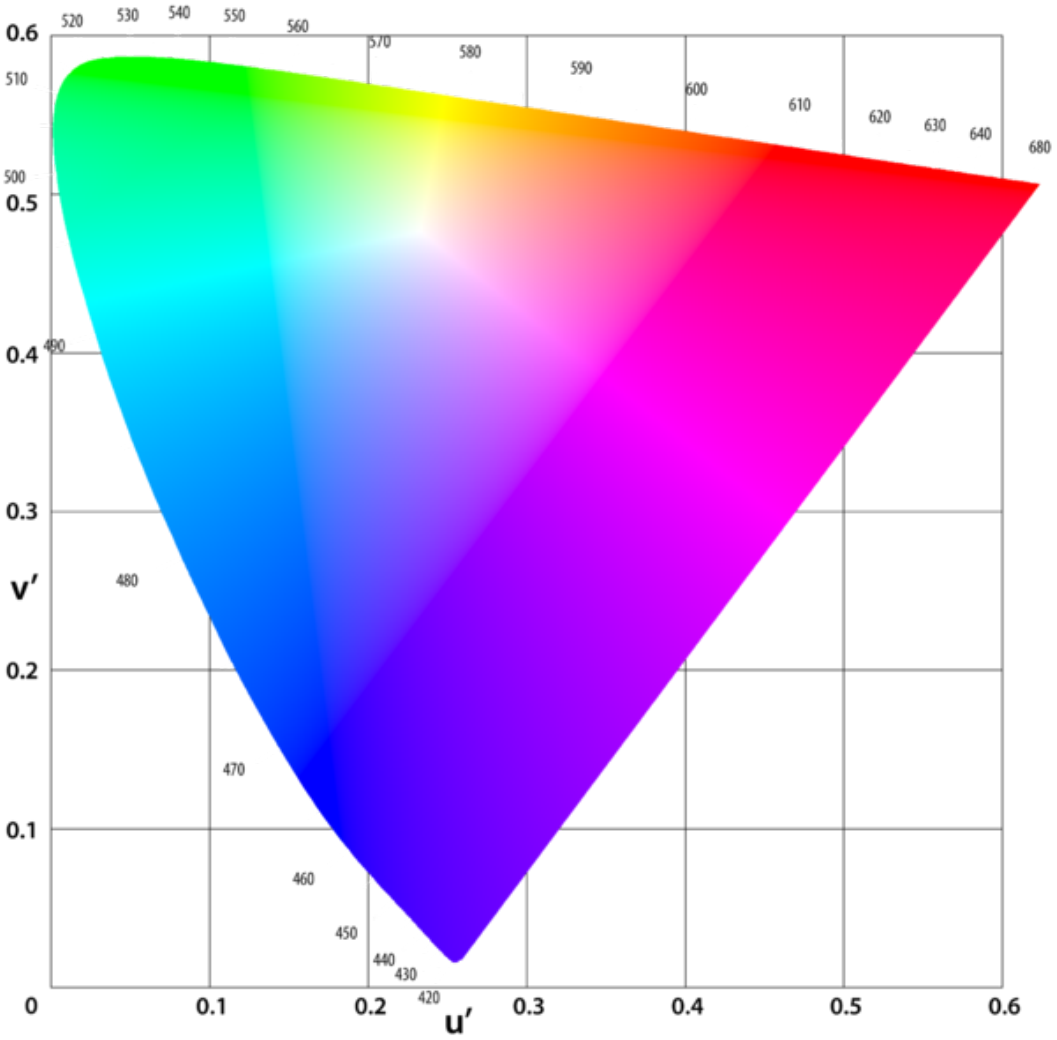


Figure 2.2: The CIE 1976 Uniform Chromaticity Scale (u', v') chromaticity diagram.

2.4 Objectives

The desired qualities of Texture Synthesis algorithms are speed, quality, repeatability, high resolution, and it is important to find an appropriate tradeoff of these to suit the application at hand. Live embedded applications, such as inpainting on a mobile platform [Criminisi et al., 2004], must focus on reduced computational requirements, and may do so at the expense of resolution, given the small display size. Texture synthesis for tile manufacturing requires high quality, a fixed resolution given by the ceramic tile inkjet printer, and are allowed a computationally intensive pre-processing step. Such

tradeoffs drive decisions in the algorithm selection process for a given application, and every application has a need for quality, albeit to different degrees.

Since quality is a subjective measure, evaluation needs to be done with a carefully selected dataset. Current datasets will be compared, and one will be created which consolidates their shortcomings. In order to bridge the gap between subjective user satisfaction and repeatable objective metrics, a user study will be conducted to compare correlations between computed and perceived quality.

The goal of this work is to create an objectively pleasing Texture Synthesis algorithm suitable for the tile manufacturing process.

Chapter 3

Related Work

This chapter provides an overview of the field of Texture Synthesis, with a focus on topics most relevant for this work. Because this thesis covers a wide range of subjects, from Parallel Texture Synthesis to Perceived Quality, so are the topics within this chapter very broad.

This chapter is organised as follows: Section 3.1 provides an overview of Texture Synthesis methods and applications. Section 3.2 discusses colours and colour management, called Profiling. Section 3.3 discusses the specifics relating to Texture Synthesis in the Ceramic Tile Manufacturing industry. The following section 3.4 is Perceived Satisfaction. Here, Colour Spaces, Visual Metrics, and Texture Energy Integration are discussed in the context of Texture Synthesis for Tile Manufacturing, and beyond. The following three sections offer overviews of the topics of the three following chapters: Parallel Texture Synthesis in section 3.5, Texture Datasets in section 3.6, and Texture Synthesis Quality in section 3.7.

Finally, the Related Work chapter ends in a Summary in section 3.8.

3.1 Texture Synthesis

In numerous computer graphics and computer design applications, texturing is a non-trivial key process. The process can be divided into three parts:

1. texture acquisition,

2. texture mapping, and,

3. texture rendering, which includes a variety of issues such as access, sampling, and filtering.

Textures can be created manually, using a variety of methods, such as manual drawing or photography, but example-based texture synthesis remains one of the most powerful methods as it works on a large variety of textures, is easy to use - the user only needs to supply an exemplar - and can provide high output quality. Manual texture acquisition becomes a problematic alternative for textures which cover a concrete three dimensional model, or when non-repeating textures are required on a large scale. The result of synthesis algorithms is an arbitrarily large output texture that is visually similar to the exemplar and does not contain any unnatural artefacts, repetition, or verbatim copying. In addition to making the process of texture creation easier, in many settings the example-based texture synthesis algorithms also provide benefits to other parts of the rendering pipeline. Distortion free textures are automatically generated over complex geometries, and on-the-fly generation of texture content strongly reduces storage requirements. [Wei et al., 2009] gives a thorough enumeration of historic methods, discussed briefly below.

Offline Exemplar-based texture synthesis falls into one of three classes: pixel-based methods, patch-based methods, and texture-optimisation methods.

Sequential pixel-based methods [Efros and Leung, 1999] are the oldest in the field of exemplar based texture synthesis. Given a random initialization, a pixel is copied from the example texture into the output such that the input-space neighbourhood matches the output-space neighbourhood as best as possible.

Sequential patch-based methods [Efros and Freeman, 2001] work in the same way, but copy an entire patch from the input to the output, while optimising the shape and location such that their neighbourhoods are similar. A popular method to find the optimal cut is Graph Cut [Kwatra et al., 2003].

Iterative optimisation methods do not pass through the output space sequentially, but instead attempt to find the least successfully synthesised region, and improve it by filling it with pixels or a patch from the input texture.

In order to approach the issue of speed and inherent sequentiality in creating

the output texture, several efforts have been made [Wei and Levoy, 2002, Lefebvre and Hoppe, 2005]. While allowing the synthesis of a subset of the output space, these algorithms are not suitable for simultaneous synthesis of disjoint regions, because the space between them needs to be synthesised as well. This issue is dealt with by [Wei and Levoy, 2002] by creating a multi-scale map, and relying only on the smaller level for synthesis. This information is kept in cache, because pixels synthesised later are dependent on the ones that have been synthesised already.

An approach to fast texture synthesis given a complex preprocessing step is **tile-based runtime synthesis**, which is done in a number of different ways: Ammann tiles [Grunbaum and Shephard, 1986], Wang tiles [Cohen et al., 2003], stochastic tiles [Wei, 2004], and s-tiles [Xue et al., 2007]. However, these all rely on a regular (in practice rectangular) grid. These approaches make the output pixel retrieval a constant-time operation for output textures of arbitrary size. However, verbatim copying is likely to produce undesirable visible artefacts, especially in an application such as ceramic tile-printing.

3.2 Profiling

In colour management, a profile describes the colour information regarding the input and output devices. This is done using a colour space, in which a mapping describes the transformations to colours that each device creates. To varying degrees, the changes can be described by linear mappings in the colour space. Certain printing processes may require more complex mappings, but any repeatable change in colour can be quantified by using appropriate colour spaces.

An industry standard is the ICC Profile [Specification, 2004], which produces a "profile connection space" (in either CIELAB or CIEXYZ spaces), by creating a table of transformations for many colours, and interpolating for the rest. Colour spaces are explored in more depth in section 3.4.1. Given a standard input, any device can be profiled using another device to exactly measure its output. These profiles are often supplied with the device, or can be created using an instrument such as a spectrometer.

(a) scanned marble (b) printed tile (c) fired tile

Figure 3.1: Marble texture detail during various stages of production

3.3 Ceramic Tile Industry

Although tile production has increasingly been subject to the introduction of automated technology, with high speed lines now approaching production rates in the order of 200 tiles/min, the texture synthesis and profiling, each a repeatable task which could be automated for consistent quality, which is considered fundamental to the maintenance of market share, has often remained essentially a manual operation. Boukouvalas et al.[Boukouvalas et al., 1995] reported a customer complaint rate as high as 70-80% in relation to certain aspects of product quality, and hence there is a clear window allowing for the increased application of automated methods. The streamlining of repetitive adjusting and labour-intensive synthesis from an exemplar offers a number of significant commercial and social advantages and reduced labour costs. Among these advantages are the elimination of human error and/or subjective judgement, improved operational efficiency, and the creation of timely statistical product data, improved safety, and better working conditions via a reduced workload. These important considerations are regarded as fundamental in order to secure a mechanism for competitive improvements within the world tile manufacturing market.

In practice, methods for decorated ceramic tile manufacture require the underlying unfired tile, an ink-jet printer, and a kiln. First, a design is printed onto an unfired tile with a specialised inkjet printer [Baldi et al., 2007]. Tiles then pass through a kiln with a set colour gradient, slowly being heated as they proceed on the conveyor, up to a maximum temperature, and then slowly cooling as they exit. Figure 3.1 shows how these steps affect the printed image on an example marble texture printed on a tile. The delicate heating process ensures that tiles are solid, and the smooth cooling ensures they do not crack. It is important to note that a larger tile will require a different firing con-

figuration from a smaller one, requiring adjustment of kiln settings and conveyor speed [Nicoletti et al., 2002, Sánchez et al., 2010, Mckee, 1996]. Finally, tiles are inspected and compared to the desired pattern, to detect variations and potential defects [Poirier et al., 2013]. This information then serves to adjust variables in the complex manufacturing process, a manual task requiring much expertise.

In order to optimise printing costs, image colours in printed ceramic tile designs are represented using the subtractive CMYK model. The advantage is lower cost over red, green, and blue colours, although manual tuning of colours is typically performed to adjust the variability and reduce cost for certain desired printed colours. Note the difference in colours between the desired scan and the printed ceramic tile in figure 3.1, caused by such a mapping.

3.4 Perceived Satisfaction

This section describes colour spaces, human-inspired visual metrics, and how Texture Energy can be integrated with Markov Chain Monte Carlo methods.

3.4.1 Colour spaces

A colour space is a multi-dimensional representation of colour, dividing colour into continuous components which are treated as axes. The simplest colour spaces are one-dimensional grayscale, or three-dimensional RGB (Red, Green, and Blue).

The range of luminance human vision can handle is quite large [Fairchild, 2013]. While the luminance of starlight is around 0.001 cd/m^2, that of a sunlit scene is around 100,000 cd/m^2, which is hundred millions times higher. The luminance of the sun itself is approximately 1,000,000,000 cd/m^2. The human eye can accommodate a dynamic range of approximately 10,000:1 in a single view and is capable of distinguishing about 10,000 colours at a given brightness. By comparison, typical computer monitors have a luminance range less than 100:1 and cover about half of the visible colour gamut. Despite this difference, most digital image formats are geared to the capabilities of conventional displays, rather than the characteristics of human vision. LogLUV [Larson, 1998]) is a compact encoding suitable for the transfer, manipulation, and storage of high

dynamic range colour images. This format is a replacement for conventional RGB images, and encodes colour pixels as log luminance values and CIE [McLaren, 1976, Pointer, 1981] (u',v') chromaticity coordinates.

LogLUV TIFF's design solves two specific problems: storing high dynamic image data and doing so within a reasonable amount of space [Larson, 1998]. Traditional image formats generally store pixel data in RGB-space occupying 24 bits, with 8 bits for each colour component. This limits the representable colours to a subset of all visible and distinguishable colours, introducing quantization and clamping artefacts clearly visible to human observers. Using a triplet of floats to represent RGB would be a viable solution, but it would quadruple the size of the file (occupying 32 bits for each colour-component, as opposed to 8 bits).

Instead of using RGB, logLUV uses the logarithm of the luminance and the CIELUV (u', v') chromaticity coordinates in order to provide a perceptually uniform colour space. LogLUV allocates 8 bits for each of the u' and v' coordinates, which allows for the encoding of the full visible gamut with imperceptible step sizes. In order to provide the required high dynamic range with imperceptible luminance steps, logLUV uses 16 bits to encode a fixed-point base2-logarithm of the luminance, which allows an EV range of nearly 128 stops. The space occupied by one pixel is thus 32 bits (L16 + U8 + V8), marginally bigger than a standard 8 bit RGB-image [Larson, 1998].

There are actually three variants of this logarithmic encoding [Thompson et al., 2011]. The first pairs a 10-bit log luminance value together with a 14-bit CIE (u', v') lookup to squeeze everything into a standard-length 24-bit pixel. This demonstrates that following a perceptual model allows making much better use of the same number of bits. In this case, the full visible gamut was extended and 4.8 orders of magnitude of luminance is achieved in just imperceptible steps. The second variation uses 16 bits for a pure luminance encoding, allowing negative values and covering a dynamic range of 38 orders of magnitude in 0.3% steps, which are comfortably below the perceptible level. The third variation uses the same 16 bits for signed luminance, then adds 8 bits each for CIE u' and v' coordinates to encompass all the visible colours in 32 bits/pixel.

Typically computer graphics effects, including alpha masking, operate in an opti-

cal intensity environment as opposed to a visually uniform one. Uniform colour spaces are designed with the aim that equal distances in the space correspond to colour differences of equal perceptual magnitude. From this perspective, uniform colour spaces are of potential interest because they should reflect visual representations of colour that are matched to properties of the colour environment. For example, the relative scaling along different axes should reflect the gamut of colour signals along these axes. The spaces might, therefore, provide clues about the structure of the environment that the visual system is calibrated for [McDermott and Webster, 2012].

Another commonly used colour representation is CMYK (Cyan, Magenta, Yellow, Black). It is a subtractive colour model typically used for specialised printing equipment, such as industrial ceramic tile printers.

3.4.2 Visual metrics

The purpose of visually uniform colour spaces is that the similarity between two colours becomes trivial to compute, and is thus important in many applications of computer graphics, especially Textures and Compression [Sheikh and Bovik, 2006, Sheikh et al., 2006]. The difference at the pixel level is simply the Euclidean distance in the chosen visually uniform space. This defines the distance between two pixels, and there are various ways of extending it to images. For instance, in Root-Mean-Squared-Error (RMSE) the difference between two realisations of an image is simply the sum of differences over pixels. Peak-Signal-to-Noise-Ratio (PSNR) calculates the ratio between the maximum possible true value and the amount of noise over the image, and is typically defined over the logarithmic decibel scale. By calculating the mean squared error (MSE) over two representations of the same scene, PSNR is defined by Equation 3.1.

$$PSNR = 20 * log_{10}(\max(pixel\ value)) - 10 * log_{10}(MSE) \tag{3.1}$$

Structural Similarity (SSIM) [Wang et al., 2004] is an extension of PSNR designed to be consistent with human eye perception, which has shown to be excellent for assessing the quality of a rendered or compressed image. However, these visual metrics were first developed to quantify the difference between two realisations of the same image.

They can therefore only be applied to exactly overlapping images, where identical pixels are expected to match, and are not appropriate for textures.

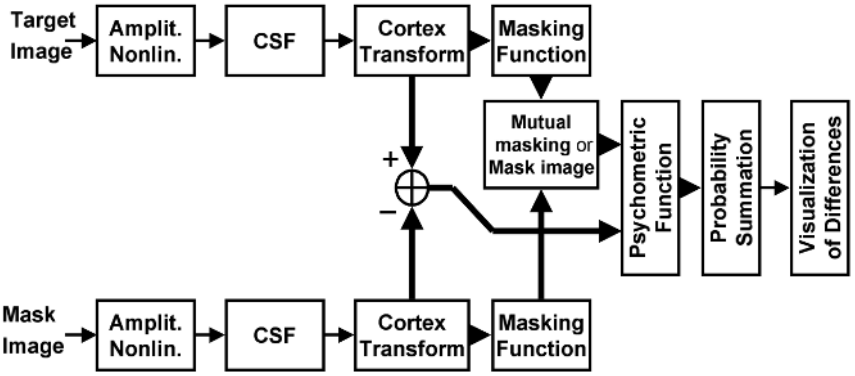


Figure 3.2: Block diagram of The Visible Difference Predictor [Myszkowski, 1998]

Visible Difference Predictor [Myszkowski, 1998] is an example with wider applications, and here, the assumption for the necessity of an overlapping section is made to a smaller degree. This is done by allowing wider variability in the overlapping section. The implementation and reasoning behind the selection of functions are not trivial. For an example, see Figure 3.2 for the block diagram for VDP[Myszkowski, 1998]. However, it provides a method which can be used for images with differences beyond the pixel level. For images with slight differences, such as those caused by printing, video frame synchronisations, or lens distortions, the overlap issue is typically solved by first undoing the distortions, and then applying a predictor which expects overlap, such as the Visible Difference Predictor. This creates additional challenges for distorted images, but in the context of quality metrics, different images cannot be applied altogether.

In the literature, texture synthesis results are typically compared visually, by synthesising textures from the same input using different methods and presenting the different outputs to an expert. Although this allows the assessment of a wide variety of potential errors or artefacts, it requires human input, and thus cannot be run as part of an automated, or objective, algorithm.

Some objective metrics for measuring the quality of synthesised texture with respect to a given input sample are presented in [Ismert et al., 2003] and [Kwatra et al., 2005]. In [Ismert et al., 2003], the values of the Jacobian matrix of the imaging trans-

form are interpreted as sample distances within the environment, practically serving as a metric of difference between input and output textures.

In the paper by Ismert et al. [Ismert et al., 2003], a solution to the problem of poor texture quality due to uneven sampling is presented by using a detail synthesis approach. A physical, sampling-based measure of the quality of a texture is introduced. This process injects statistically correct high frequency information into areas with poor detail, preserving any detail present. It introduces two important contributions to image-based texturing:

- A physical, sampling-based metric using the Jacobian matrix of the imaging transform for evaluating texture quality
- A method that relates the multi-resolution pyramids used by many current texture synthesis techniques to the Jacobian-based metric, allowing synthesis driven by a physically-based, quantitative measure of texture quality

This method extends existing texture synthesis algorithms to generate only missing detail using the metric, which is relevant for Detail Synthesis (but not Texture Synthesis). Texture Energy, a Markov Random Field-based metric, is introduced by Kwatra et al. [Kwatra et al., 2005] in the context of texture synthesis, and will be further explained here because it can be extended for our use.

3.4.3 Texture Energy Integration

Texture Energy (figure 3.3, equation 3.2), is a powerful method of quantifying the quality of a synthesised texture with respect to the original, but is very computationally demanding to calculate. The energy of neighbourhood x_p centred around pixel p is given by its distance to the closest input neighbourhood z_p. When two neighbourhoods x_p and x_q overlap, then any mismatch between z_p and z_q will lead to accumulation of error in the overlapping region. In essence, for every square patch of the output texture, find the most similar patch in the input texture, and calculate their difference. Then, sum this difference over all patches in the output, to create a single number. The lower the number, the more similar the output texture is to the input texture. By first defining the

operation of patch similarity as the sum of similarities of corresponding pixels, the entire Texture Energy function is defined as follows:

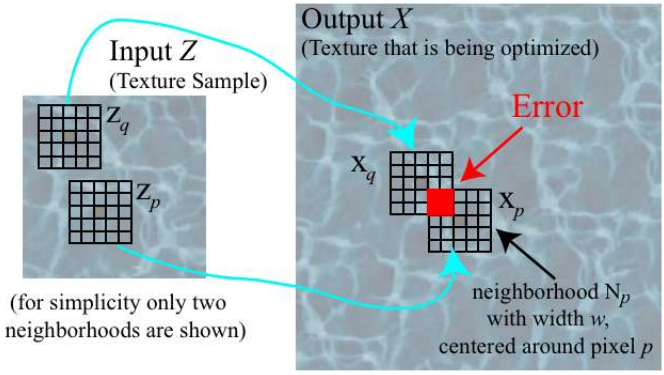


Figure 3.3: Schematic demonstrating the Texture Energy metric

In the equation below, Texture Energy (ϵ_1) is computed on input image Z and output image X, where $|X|_x$ and $|X|_y$ denote the dimensions of image X along x and y dimensions, $X_{x+l,y+m}$ denotes the pixel of image X at coordinates $(x+l, y+m)$, δ is the pixel difference in a perceptually uniform colour space, and s is the scale parameter which defines the texel size.

$$\epsilon_1(X_{output}, Z_{input}) = \sum_{l=1,m=1}^{l=|X|_x-s,m=|X|_y-s} \min_{a=1..|Z|_x-s,b=1..|Z|_y-s} \sum_{x=1,y=1}^{x=s,y=s} \delta(X_{x+l,y+m}, Z_{a+l,b+m}) \tag{3.2}$$

Hence, assuming δ can be computed in constant time, the computational complexity of exactly calculating Texture Energy ϵ_1 for a given input-output image pair is $O((|X|_x - s) * (|X|_y - s) * (|Z|_x - s) * (|Z|_y - s) * s^2)$, or simply $O(n^4 s^2)$.

3.5 Parallel Texture Synthesis

Sequential exemplar-based texture synthesis falls into one of three classes [Wei et al., 2009]: pixel-based methods, patch-based methods, and texture-optimisation methods. Parallel texture synthesis can be divided into an additional three: dependency-tree methods, constant-time tiling methods, and non-constant-time tiling methods.

Sequential pixel-based methods [Efros and Leung, 1999] consider each output

pixel in sequence, while sequential patch-based methods [Efros and Freeman, 2001] replicate entire patches, optimising seams using an algorithm such as Graph Cut [Kwatra et al., 2003]. Instead of performing the process once, patches can be placed iteratively over the output until desired quality is achieved [Wei and Levoy, 2002, Kwatra et al., 2005]. However, sequential algorithms are not suitable for simultaneous synthesis of disjoint regions, because the space between them needs to be synthesised as well.

Where the entire texture cannot be held in memory but needs to be generated on-the-fly, parallel texture synthesis methods can be used. The naïve approach to reduce rendering time is to create a repeating tile from an input exemplar, such that the edges fit [Wei and Levoy, 2000]. This causes visibly noticeable "tiling" effects. Tile-based runtime synthesis relies on offline-preparation contents of a texture map, which are then placed on a rendered surface. Such placement schemes can be done in a number of different ways using a rectangular grid: Ammann tiles [Grunbaum and Shephard, 1986], Wang tiles [Cohen et al., 2003], stochastic tiles [Wei, 2004], s-tiles [Xue et al., 2007], and coloured corners [Lagae and Dutré, 2006]. Triangular [Neyret and Cani, 1999] grids have also been used. These approaches make the output pixel retrieval a constant-time operation for output textures of arbitrary size. However, they create visible repeated edges and grid patterns when zoomed out, as shown in Figure 3.4.

Figure 3.4: Comparison with Wang tiles using 2 different borders (16 tiles) [Cohen et al., 2003] (left), [Vanhoey et al., 2013] (middle) and our results (right). Sampled linearly. Top exemplar is 268×230, bottom 512 × 512

In turn, this visible aberration is addressed by non-rectangular region copying, such as megatexture [Obert et al., 2012], virtual texturing [Ephanov and Coleman, 2006, Taibo et al., 2009], and patch-based methods [Praun et al., 2000]. However, these meth-

ods rely on cached information, which can cause temporal artefacts when a scene is re-rendered in a different order. To allow a different part of the scene to be rendered elsewhere, in the next video frame, or to be able to revisit a texture in a virtual environment, it is desirable to guarantee that a texture rendered again from the same compact seed will be identical to one rendered previously. This quality is referred to as "repeatability". (Not to be confused with "repetition", which is generally undesirable in synthesised textures)

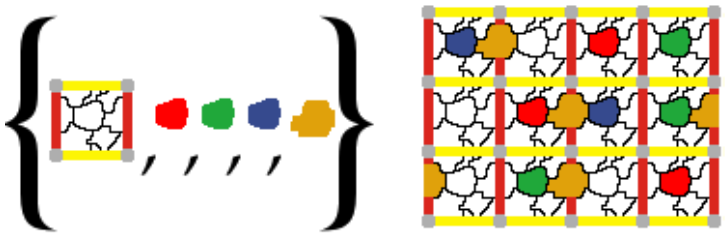


Figure 3.5: Patch map of [Vanhoey et al., 2013]

Parallel non-constant-time patch replacement methods [Vanhoey et al., 2013, Gilet et al., 2012, Praun et al., 2000] perform a non-constant overhead operation while rendering, to address grid artefacts. These methods place patches of precomputed shape on the texture at run-time, according to a run-time computation, but require a fixed patch map whose boundaries cannot be overlaid with a patch. For example, in the offline step of [Vanhoey et al., 2013], a fixed repeating patch map is created, along with various interchangeable patches which fit the patch map boundaries (Figure 3.5). The texture can then be sampled independently online with a pseudo-random number generator at each repeating patch map. However, none of these resolves the local adjacency constraint for patches overlapping repeatable tile boundaries.

Methods which use a statistical shape model for the texture [Gilet et al., 2012] are able to outperform rendering speeds and quality of exemplar-based parallel synthesis, at the expense of relying on additional user input to model the texture. As this is no longer automated example-based texture synthesis, such methods are not included in Figure 3.6, and are not considered here.

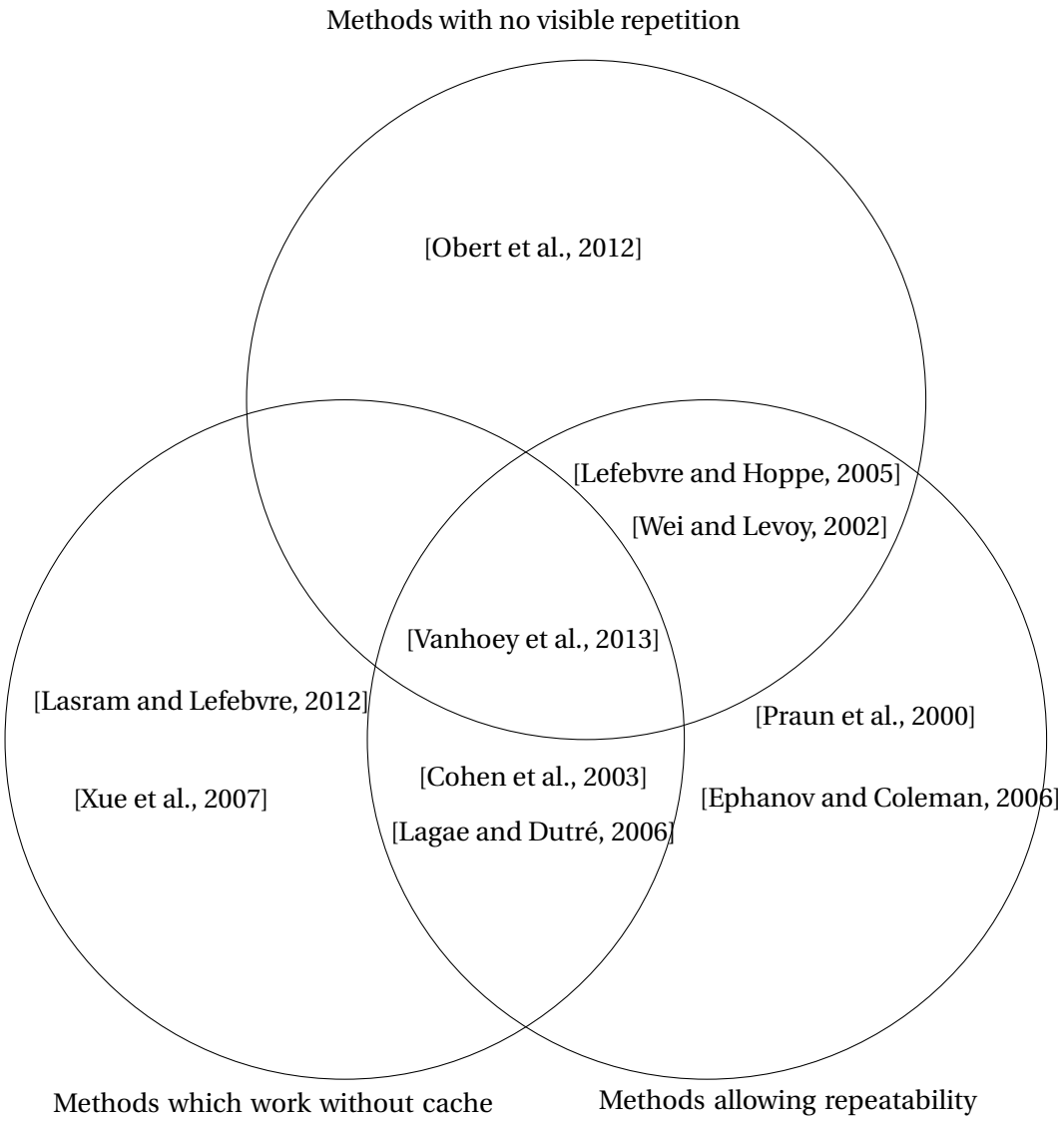


Figure 3.6: Venn diagram of the trade-offs between current example-based parallel texture synthesis algorithms

3.6 Texture Datasets

Several texture datasets have been previously created to demonstrate and evaluate texture synthesis with: Brodatz Textures [Brodatz, 1966], VisTex Textures [Graczyk, 1995], DeBonet Textures [De Bonet, 1997], Colored Brodatz [Abdelmounaime and Dong-Chen, 2013], the PSU Near-Regular Texture Database [Lee and Liu, 2005], Simoncelli Textures

[Portilla and Simoncelli, 2000], and many others. See [Hossain and Serikawa, 2013] for a complete survey of Texture Databases. None of these covers the full range of published texture properties [Lous, 1990, Fellner and Helmberg, 1993, Kobbelt et al., 1997, Lafortune et al., 1997, Buhmann et al., 1998, Foley et al., 1993] which various methods claim to handle, and none of them attempts to represent all texture properties.

Texture synthesis is an inherently subjective problem. In order to clarify the meaning of texture synthesis, various schemes have been devised to classify textures by properties, so that appropriate synthesis algorithms may be created. Such schemes are the continuous texture spectrum from regular to stochastic [Lin et al., 2004b], and the a-score and g-score of [Liu et al., 2004].

3.7 Texture Synthesis Quality

Texture Synthesis algorithms are typically compared on a small sample chosen by the authors, possibly resulting in inconsistent evaluations of new methods and misjudging of method properties.

Unlike general images, textures must satisfy stationarity and locality. The first criterion requires any two patches to be visually similar, and the second criterion requires that pixels are only related to a small set of neighbouring pixels. As these criteria are satisfied to different degrees (a random noise image satisfies them perfectly), so must texture synthesis algorithms be able to handle textures with limited locality, stationarity, and various other properties.

Many texture synthesis algorithms have been devised over the past 30 years [Wei et al., 2009], with a focus on various aspects: quality, speed, parallelism, use for animation, manufacturing quality, and others. In this work, we only focus on online rendering speed, and the quality of the synthesised texture. Despite attempts to quantify synthesised texture quality (texture energy metric [Kwatra et al., 2003], image statistics [Balas, 2006]), texture quality remains a subjective notion. However, no previous user study has been performed to assess texture synthesis algorithms.

There have also been comparative studies, but only through individual subjective evaluation with example textures on which the algorithms don't fail [Lin et al., 2004a].

3.7.1 Selected Texture Synthesis Algorithms

In order to provide a comparison across texture synthesis methods, six methods were chosen. The selected algorithms belong to one of three categories: state-of-the-art methods with a focus on quality [Kaspar et al., 2015, Jamriška et al., 2015, Li and Wand, 2016], well-known historic methods [Efros and Freeman, 2001, Ashikhmin, 2001], and methods whose code is public, and known to perform well beyond the research community [Cornet and Rouquier, 2011, Ashikhmin, 2001].

Texture Synthesis by image quilting [Efros and Freeman, 2001] places square subsamples of the exemplar onto the output texture, optimally choosing transitions by finding the nearest matches according to the overlap. Each overlapping region is then optimally cut with a minimum cost path. See figure 3.7 for a visual explanation of this process.

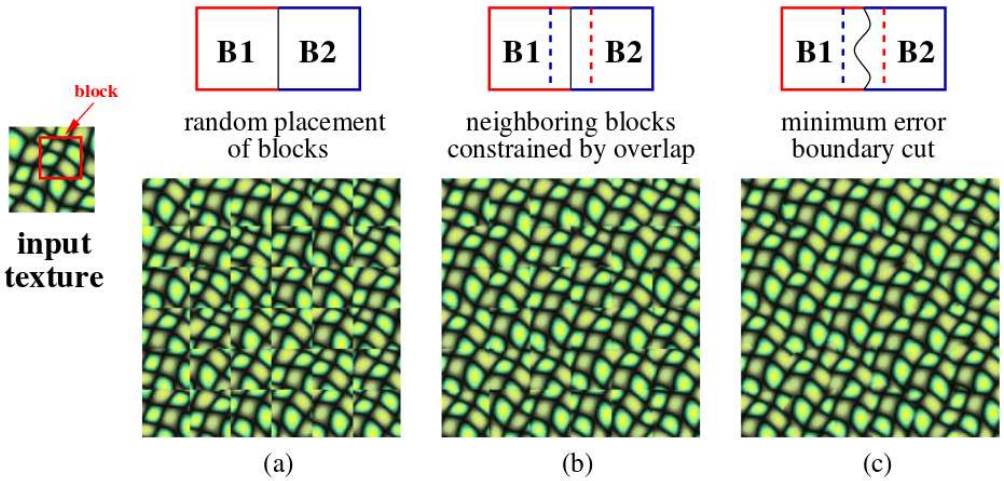


Figure 3.7: Image Quilting: Square blocks from the input texture are patched together to synthesise a new texture sample. The figure is taken from [Efros and Freeman, 2001]

Ashikhmin Natural Texture Synthesis [Ashikhmin, 2001] is based on Wei and Levoy Texture Synthesis [Wei and Levoy, 2000], where pixels are added individually row-by-row by finding the best matching candidate according to surrounding pixel similarity, as seen in figure 3.8. Each pixel in the current L-shaped neighbourhood generates a “shifted” candidate pixel according to its original position in the input texture. The best pixel is chosen among these candidates. Ashikhmin improves this search by focusing on several

candidates, thus encouraging verbatim copying instead of blurring.

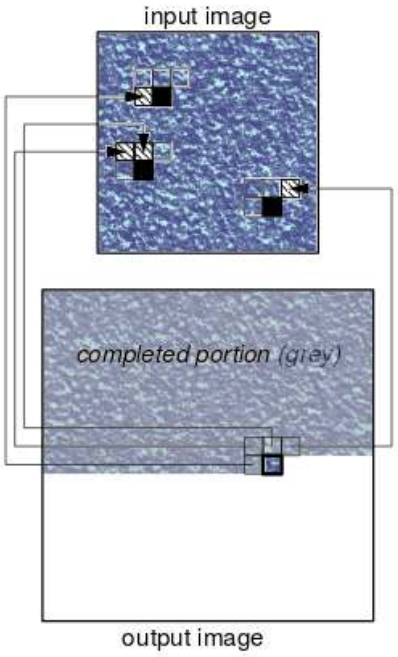
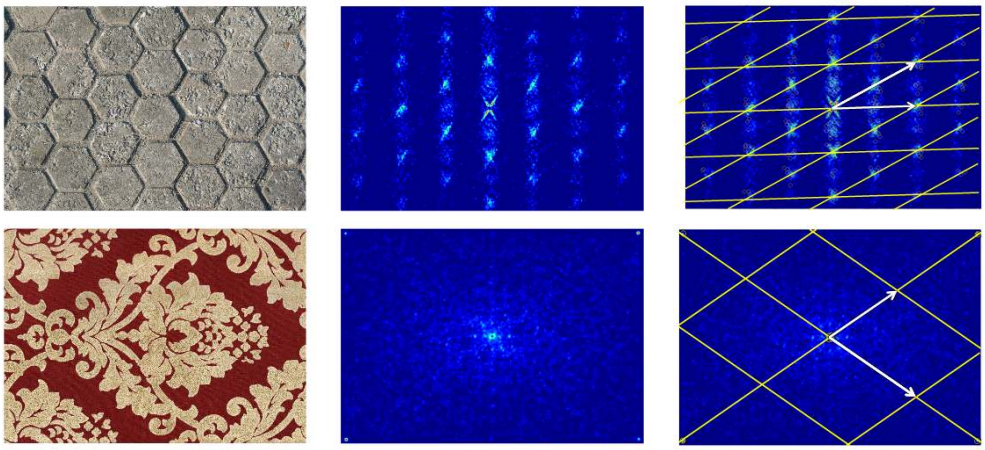


Figure 3.8: Candidate pixels for the Ashikhmin algorithm. Figure taken from [Ashikhmin, 2001]

Resynthesizer [Cornet and Rouquier, 2011, Harrison, 2001] is an open-source texture synthesis plugin. In this algorithm, pixel values are chosen one at a time in random order. When choosing the value of a pixel, the n nearest pixels that already have values are located, and the input image is searched for a good match to the pattern these pixels form. Once a good match is found, the appropriate pixel value is copied from the input texture to the output. To increase quality, some earlier chosen pixel values are updated after later pixel values have been chosen.

Self Tuning Texture Optimization [Kaspar et al., 2015] is a general-purpose and fully automatic self-tuning non-parametric texture synthesis method. Various parameters and weights are tuned by focusing on three aspects of texture synthesis: irregular large scale structures are faithfully reproduced through the use of automatically generated and weighted guidance channels, repetition and smoothing of texture patches is avoided by new spatial uniformity constraints, and a smart initialization strategy is used in order to improve the synthesis of regular and near-regular textures [Liu et al., 2004],



(a) Exemplars (b) Offset Kernel Map (c) Extracted Lattice

Figure 3.9: Self Tuning Texture Optimization Lattice Extraction Step demonstration on a repeating texture (top row), and a tileable image (bottom row). Figure taken from [Kaspar et al., 2015]

without affecting textures that do not exhibit regularities. Figure 3.9 shows the automatic lattice extraction step used in this method for modelling of regular and near-regular textures.

LazyFluids Appearance Transfer [Jamriška et al., 2015] extends Graphcut Textures [Kwatra et al., 2003] which minimizes Texture Energy

$$E(Z,X) = \sum_{p \in X} \min_{q \in Z} ||x_p - z_q||^2 \tag{3.3}$$

where Z is the source texture, and X is the output. However, this is known to create an output image which matches only a portion of the input, for example blurred parts, and quality is highly dependent on selected parameters. LazyFluids resolves these issues by using a nearest-neighbour field to assure uniform source patch usage.

CNNMRF [Li and Wand, 2016] is based on Neural Style [Gatys et al., 2015], which uses statistics of higher levels of a pre-trained Convolutional Neural Network, and iteratively adjusts a random noise image to match the statistics of a texture exemplar. However, CNNMRF also fits a Markov Random Field over neuron activations, in order to better match the texture properties.

Note that Self Tuning [Kaspar et al., 2015] and CNNMRF [Li and Wand, 2016]

require hours to compute results for large textures, while the other algorithms run in seconds or minutes. The source code for all selected algorithms is available online, with the exception of LazyFluids.

3.8 Summary

In order to research the topic of texture synthesis, a wide array of topics need to be covered. These include the physical manufacturing process, to colour space analysis and image visual metrics, up to texture synthesis. In this chapter, topics such as Texture Energy and Exemplar-based Texture Synthesis were treated in extensive detail, as they will be crucial building blocks for the research explained in further in this thesis.

This chapter has covered the basic topics of general Example-based Texture Synthesis, and questions of Colour and Quality Analysis. Building on these, this chapter discussed the relevance of these topics for Ceramic Tile Manufacturing, and provided an overview of existing practices in industry.

Finally, a literature review of each of the topics of the three following chapters has been presented: Parallel Texture Synthesis, Texture Datasets, and Texture Synthesis Quality.

The clearest challenge emerging from the literature review is a lack of quantified claims regarding quality. Furthermore, there is a growing divide between the methods of high quality and the real-time rendering methods. High quality methods take hours per texture, and real-time rendering methods produce results of visibly lower quality. The need for a quality metric has been tackled by the introduction of Texture Energy and Texture Energy over a nearest-neighbour field, which quantify quality of a given generated output for an input. However, as outputs which optimise these metrics contain visual flaws, these objective evaluation methods fail to capture texture quality. This emphasises the need for a subjective user-centred quality metric. The evaluation of quality must be a process with the user in the loop.

These challenges underpin the work done in the remainder of this thesis.

Chapter 4

Research Methodology

4.1 Introduction

In order to systematically approach the complex research question of this work, it is crucial to consider appropriate research methodology. This is to be considered in view of the research question posed earlier:

How can the quality in offline and online parallel texture synthesis algorithms be rigorously improved?

The ultimate aim posed for the research to be completed here is the creation of a comprehensive approach for the analysis of quality of texture synthesis algorithms, and the development of a method which improves the state-of-the-art with respect to this approach.

4.2 Justification

As the previous chapter has shown, current published work lacks the rigour of a well-defined quality analysis test over presented results, which is necessary to objectively support the selection of any given algorithm in practice. Existing publications lack a holistic view, comparing published methods according to particular properties being tested, and typically with a proposed algorithm that performs well over these. In order to approach the problem outlined in the chapter 2, handle the emerging challenges of the previous

chapter, and answer the research question, the work preformed throughout this research is planned in the following steps.

4.3 Objectives

1. Review existing approaches and identify their benefits and weaknesses, so that these may be improved upon. Study the problem of colour analysis for various applications related to tile manufacture, with a special focus on textures, perception, and reproducibility. This will ensure that the work performed here is novel, repeatable, and beneficial in a wide array of use cases.

2. Create an algorithm capable of reproducing high quality textures in real time, irrespectively of the complexity of the underlying preprocessing step. Such an algorithm will enable the use of varying high-quality textures in industrial applications of the tile manufacturing design process.

3. Prepare a dataset of textures which are representative of real use cases across known fields where texture synthesis is relevant; this should also be small enough to make experiments with participants practical in varied settings relevant for each application. Justify the selection of textures in the representative subset with thorough analysis of known texture properties. This will allow the results to be practically used in future work, and in the tile manufacturing industry.

4. Perform a comprehensive perceptive experiment comparing the highest quality texture synthesis algorithms available. This requires the acquisition of state-of-the-art algorithms through contact with the original authors, or through re-implementation and optimisation of the involved parameters. Finally, the qualitative user study is to be rigorously prepared, tested, executed, and its results analysed with appropriate statistical tests. This step will enable the selection of the highest quality synthesis method for use in conjunction with the algorithm of step 2, so that potential applications benefit from high speed in conjunction with the highest currently achievable quality.

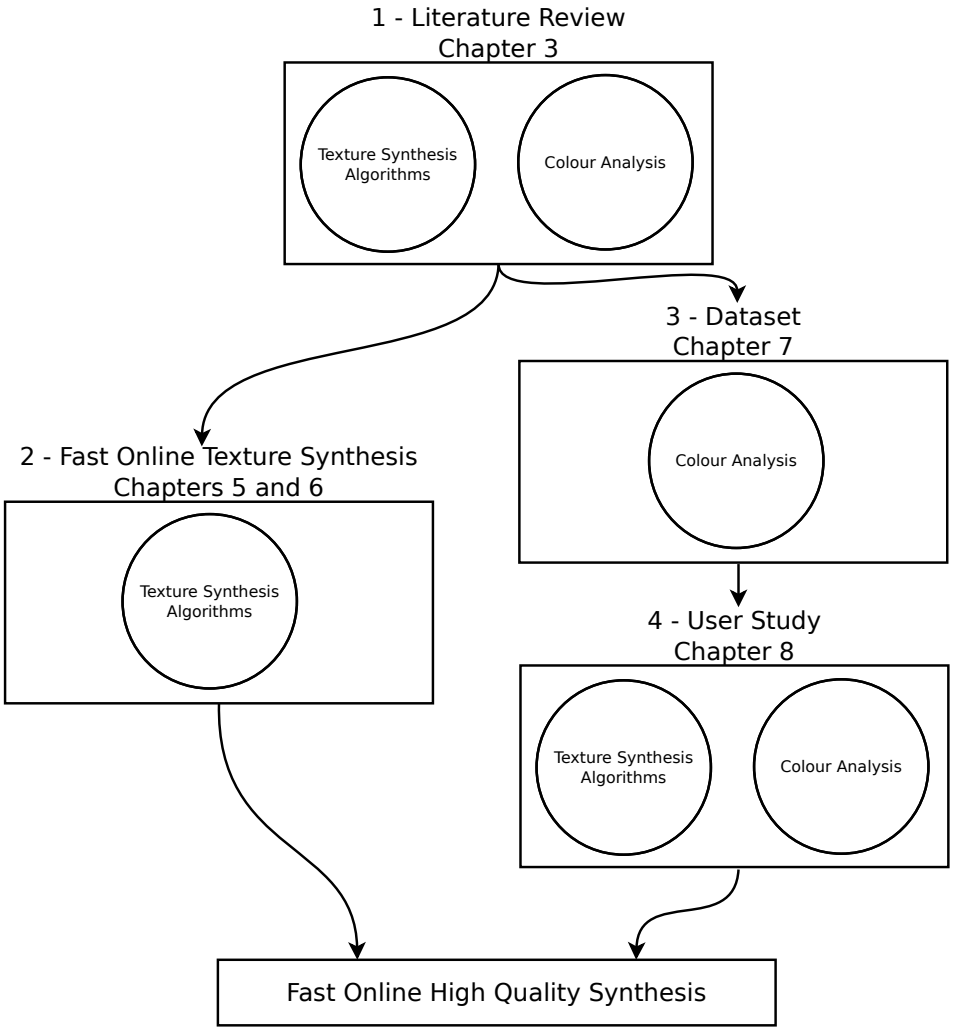


Figure 4.1: Research Plan

4.4 Methods Used

See Figure 4.1 for an illustration of the research process outline as undertaken in this work. The steps outlined above are directed toward achieving the aim of creating a novel fast high quality texture synthesis algorithm in a rigorously verified manner. These steps correspond to the chapters dividing the work of the doctorate.

Step 1, where existing approaches a reviewed, is performed in chapter 3 - Related Work. Step 2, where a texture synthesis algorithm is prepared, corresponds to chapter 5 - Repeatable Texture Sampling with Interchangeable Patches, and the claims regarding quality preservation are verified in chapter 6 - Repeatable Texture Sampling Evaluation

Study. Step 3 requires the preparation of a representative dataset, which is performed in chapter 7 - Synthesis Evaluation Dataset. Finally, step 4, which contains a comprehensive experiment to judge the relative and absolute quality of existing approaches, is in chapter 8 - A Subjective Evaluation of Texture Synthesis Methods.

As the results of both threads are brought together, a process is proposed to use a computationally intensive high-quality pre-processing step together with the fast online texture synthesis method. This process allows a general approach for high-speed synthesis, where the globally optimal pre-processing algorithm is selected. Furthermore, it enables application-specific solutions to be devised by taking into account additional knowledge regarding quality and memory requirements.

A crucial part of the validation of this work is the user study, which will be undertaken according to formal methods of experimental design. Informal experimental designs may include questionnaires, semi-structured interviews, and subjective observations. Significantly informative formal experimental designs are available, such as ranking, which compares all instances against all other instances.

In fact, the experimental process can be formulated as a Completely Randomised Design (C.R. Design) [Salkind, 2010], where each participant is asked to compare every method to every method across every free variable. Similar studies have been performed to compare High Dynamic Range compression methods [Mukherjee et al., 2016], perceived video quality [Seshadrinathan et al., 2010], or the emotional experience in games [Mandryk et al., 2006]. Such a process will enable a more statistically meaningful analysis of the findings. The specifics of this process will be handled in thorough detail in chapter 7, where the experimental setup will be clearly defined.

Furthermore, a crucial question regarding the experiment is the selection of participants. These should come from broad backgrounds, and perform the experiment across a wide range of lighting conditions, on varied display devices, all while maintaining consistence among their ratings of the textures. If statistical significance and agreement can be shown to hold even in such broadly varying settings, the results of the qualitative user study will be widely applicable to many users, customers, and applications.

4.5 Conclusion

The research methodology planned for this project, as explained in this chapter, assures a standard of quality and the successful completion of the final goal set by this research. In order to fully justify the relevance of all work performed as part of this doctorate, a clear plan has been drawn, and it will be logically performed in the following chapters. Therefore, the work will not only be applicable for the specific needs of the industrial partner Johnson Tiles, but its novelty and benefit will usher progress for the wider scientific community.

Chapter 5

Repeatable Texture Sampling with Interchangeable Patches

5.1 Introduction

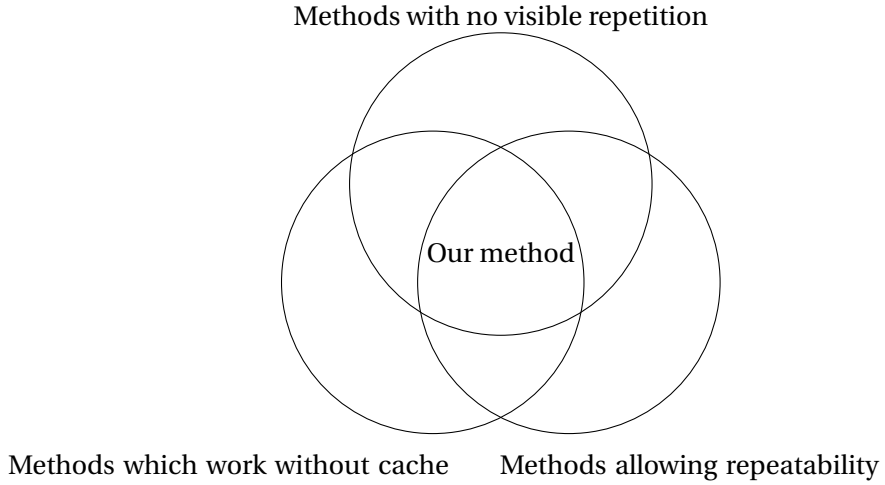


Figure 5.1: Venn diagram describing properties of the published method

Crucial points on which current Texture Synthesis methods compete are perceived texture quality, rendering speed, scale considerations, and memory requirements.

In order to allow rendering with ray-tracing, and to render textures in real time, current methods need to run in parallel and on a GPU, without hindering the perceived

quality.

As shown in Figure 5.1, our method addresses the main requirements for example-based parallel texture synthesis algorithms. This chapter introduces a method which allows parallel texture synthesis with patches of arbitrary shape, without the necessity of a fixed repeating patch boundary. In previous work [Vanhoey et al., 2013], selection of interchangeable patches at runtime was also possible, but required a repeating fixed patch boundary. As discussed later, the method presented here has similar preprocessing complexity, memory consumption, and rendering speed, but allows a wider class of interchange types with higher variability, resulting in a higher quality texture.

Our method allows the sampling of any pixel in the output texture with a deterministic algorithm, without requiring any information from the pixels that have already been synthesised. Therefore, adjacent pixels can be synthesised in parallel in separate threads, which do not communicate.

Large textures are rendered by randomly selecting subsets of prepared patches in a parallel and repeatable manner. These precomputed patches are stored efficiently, allowing seamless integration in GPU, and the texture is rendered independently for each pixel on-the-fly, allowing repeatable parallel access to an infinite, non-periodic texture, appropriate for ray tracing applications.

The chapter is structured as follows: Section 5.2 defines the objective that this work set out to accomplish, and section 5.3 offers a justification of this objective. Section 5.4 outlines the high-level concept behind the method, and section 5.5 goes through how these points are implemented. Method outputs, their comparison to other work, and other results are presented in section 5.6, and future work and the conclusion are in section 5.8.

A visual comparison of methods which precompute tiles and select placement during rendering is shown for: Wang tiles (Figure 5.2), fixed map patches of [Vanhoey et al., 2013] (Figure 5.3), and patches without map boundaries presented here (Figure 5.4). In each figure, the precomputed set of patches or shapes is on the left, with colours corresponding to places of interchangeability.

This chapter describes how interchangeable patches can be applied online to

repeating tiles without a fixed patch map, and without posing constraints on boundaries and adjacencies. A discussion of the benefits of this approach is in the results section.

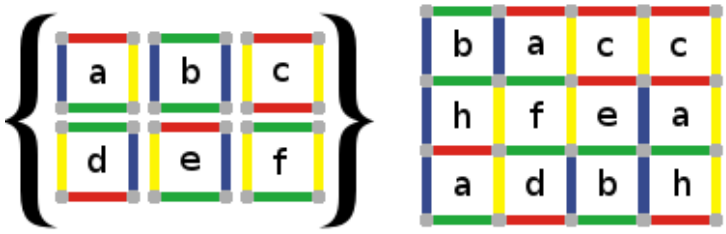


Figure 5.2: Precomputed Wang tiles a to f are represented on the left, and a synthesised image is on the right

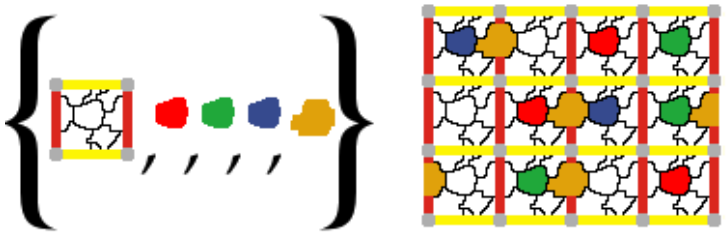


Figure 5.3: Method of [Vanhoey et al., 2013]. Patch map on the left, synthesised image on the right

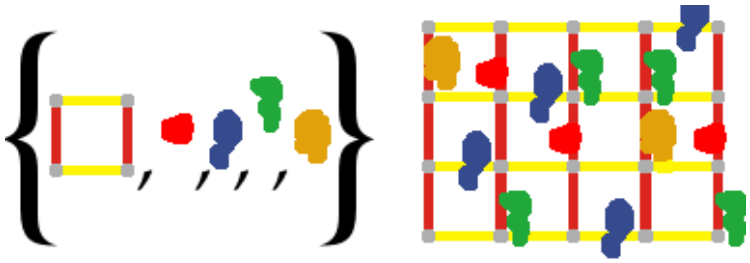
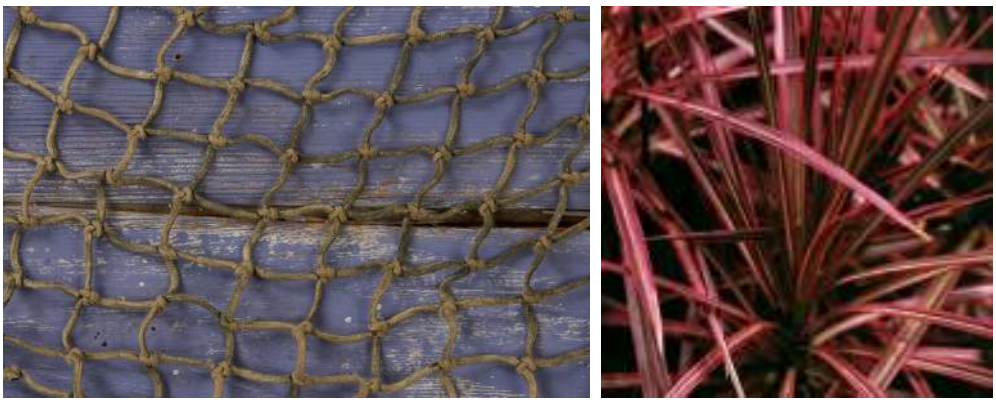


Figure 5.4: Our method. The precomputed tile and interchangeable patches are on the left, and a synthesised image is on the right

5.2 Objective

The presented algorithm was developed with the objectives of improved speed and high synthesis quality. While offering this enhancement over the state-of-art, our method



(a) Texture with a non-repeating patch map (1024x682). See section 5.6 for a discussion of the properties of such textures.

(b) Texture where patch boundaries are crossed by other patches. (512x512)

Figure 5.5: Irregular textures

maintains the benefits of parallel non-constant-time tiling: texture quality depends only on the quality of a pre-processing step and available GPU space. Memory and load on a GPU are addressed to show that even complex textures can fit into limited GPU memory, and the non-constant runtime overhead is only a light logic operation. Quantitative results, such as speed and memory requirements, are to be assessed through measurements. Furthermore, in order to justify claims regarding the preservation of quality, a user study regarding the perceived quality is additionaly performed in Chapter 6.

5.3 Justification

The benefit of a high speed and high quality texture synthesis algorithm is wider accessibility of quality textures in real-time applications. By creating such an algorithm, high quality online texture synthesis will become available to applications in rendering, video games, and high-resolution ceramic tile design.

Previous available algorithms do not satisfy the wide array of use cases where preprocessing time complexity is not a crucial factor. For such applications, texture synthesis must be performed in high quality, quickly with low computational overhead, and online in a repeatable fashion to minimise disk usage. These factors are crucial for the rendering of ceramic tile designs during production, which justifies the development of

such a method.

Furthermore, texture synthesis on the fly has been a complex, important problem in the wider texture synthesis community. By bridging the gap between high quality and high speed methods, this work enables the development of further new techniques and applications.

5.4 Patch-based Texture Synthesis without Spatial Dependency

The algorithm described in this work is divided into two steps: preprocessing and rendering, see Figure 5.6.

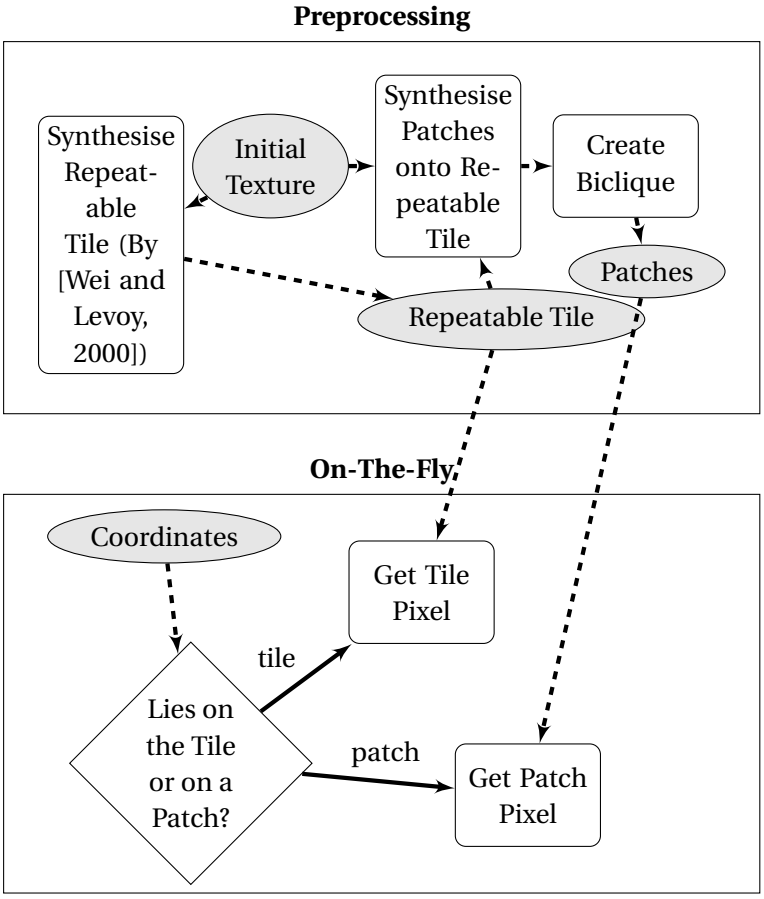


Figure 5.6: Flowchart overview of our method

The method starts with a simple repeatable tile created from the exemplar texture. Next, interchangeable patches of varying size are precomputed on the repeatable

tile. These can reach over the repeatable tile boundary, so they are created such that they form two sets, which mutually do not overlap. During rendering, this non-overlapping criterion permits parallel rendering, while guaranteeing local adjacency because “active" patches cannot overlap.

The tile is made larger than the largest visual repeating element of the texture in the exemplar. For example, in the left image of figure 5.13, this corresponds to the number of pixels spanned by one apple. By using interchangeable patches of various sizes (from a few pixels to a large portion of the tile), the synthesised texture will contain elements on multiple scales.

The patches are saved, and at runtime are chosen in each tile in a random, repeatable process, without any cached information. A binary map of the patch allows constant-time retrieval of pixel values.

Preprocessing guarantees that patches of adjacent tiles do not overlap, and the online selection guarantees that patches chosen within each specific tile are selected so that they do not overlap. Due to these constraints, every sampled pixel is copied from one of two regions: the repeatable tile, or a selected patch of this tile or the neighbouring tile. At runtime, pixel lookup is performed based on a simple logical operation which makes this decision, and the selected pixel is retrieved from the input texture.

5.5 Algorithm Implementation

The algorithm is composed of an offline preprocessing step which creates the texture map representation, and an online on-the-fly algorithm called for each requested pixel coordinate (see Figure 5.6). The texture map consists of a repeatable tile, and patches divided into two sets with offset vectors on their specific tile (but not locations in the final output texture). (See Figure 5.4)

5.5.1 Preprocessing

From an input texture, we create

- a repeatable tile

- difference vectors for each patch, denoting its location in the base texture, and in the repeatable tile
- a 2D binary array containing the shape of each interchangeable patch
- a binary matrix for each of two sets of patches with the information whether any given pair overlaps

(a) repeatable tile (300 × 300px) (b) a large patch (91 × 112px) (c) a small patch (15 × 19px)

Figure 5.7: Sample repeatable tile and patches for a given texture

First, using Image Quilting [Efros and Freeman, 2001], we synthesise the “repeatable tile" from the exemplar (Figure 5.7).

The input texture is used as the patch source. If this input texture does not fit into GPU memory, it may be desirable to render a smaller base texture from which patches will be copied, by [Wei and Levoy, 2000]. Patches are not stored explicitly, but are indexed from this base texture using the difference vector. Therefore, each pixel of a patch takes 1 bit of memory, instead of a minimum 3 bytes for a naïve RGB pixel representation.

Next, we generate candidate patches by associating random pixel locations between the repeatable tile and the base texture, and by executing GridCut [Jamriska et al., 2012] to find the optimal cut. Patches of various sizes are generated by weighting the cuts by a Gaussian bell curve of varying width. The cut is allowed to overflow over borders of the repeatable tile, but not the source tile.

As in previous work, the cut cost is Euclidean distance in CIELab colour space [Zhang and Wandell, 1996] of pixels in the original texture (“base"), and the repeatable tile (“tile"). For each potential patch, we find the maximum pixel cut cost along the boundary, and choose a predefined number of patches (P = 100 to 1000) with smallest maximum cut cost. This removes poorly matched patches.

The following step, involving a rhombus and pseudo-biclique, ensures that when patches are selected in adjacent tiles, they will not interact by potentially overlapping.

The output space is divided into tiles A and B, and there are two sets of precomputed interchangeable patches, one for each. The same repeatable tile is used for A and B, to make the texture map compact, but the patch sets differ, to allow greater variability. These patches can lie on the boundary between A and B, but must be entirely within the rhombus around the region they lie in (Figure 5.8). It is important that patches do not overlap in the output texture, because the region simultaneously covered by multiple patches is not guaranteed to fit. The patches which lie over the tile boundaries ensure that there is no straight repeating boundary in the output texture, and the bounding rhombus assures independence between patches “active" in adjacent tiles.

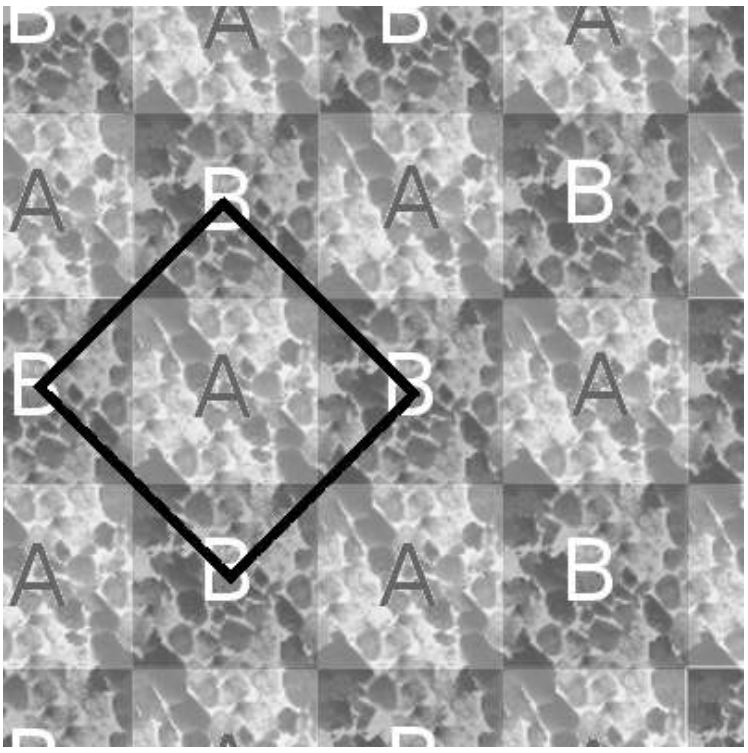


Figure 5.8: The output space alignment. The black rhombus represents the boundary that patches in A cannot overlap

The patches are divided into a pseudo-biclique such that all patches in set A never overlap with any patch in set B (Figure 5.9). Using the P patches of varying size (Figure 5.7), a graph is constructed where each patch is a node and each edge is a “does-

not-overlap" relationship (Figure 5.9). This division is done heuristically, using algorithm 1. Note: if edges are made to represent an “overlaps" relationship instead, this pseudo-biclique becomes the union of two disjoint graphs.

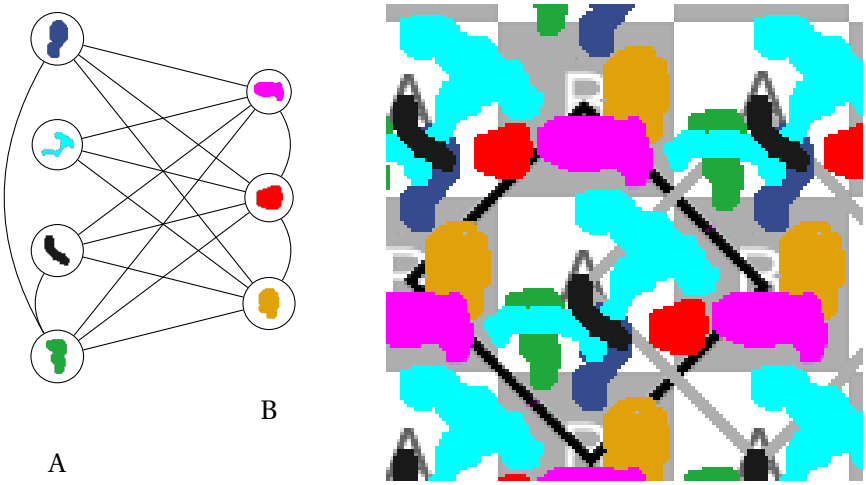


Figure 5.9: A pseudo-biclique. Every edge between patches represents a “does not overlap" relationship, and edges between patches in either group are allowed. On the right, corresponding patch positions are shown.

The selected patches are then saved in a binary array, along with the following variables: width, height, top left corner location in the base texture, and top left corner location in the repeatable tile. Square subsets of the matrix of overlaps are saved as well, one for the overlaps among patches in A and a second for B.

5.5.2 On-the-fly Sampling

Given a single (x, y) coordinate, determine which tile it lies in (t_x, t_y) by rounding to the nearest tile, and its location in the tile (p_x, p_y) by modulo. It is then determined whether the desired pixel is on an A tile or a B tile, by whether $t_x + t_y$ is odd or even.

For each tile type, there is a set of precomputed patches P_A and P_B. For each patch ρ in each set, there is a pseudo-random function $r(a,b)$ which lies in the binary domain. For example, our implementation uses the following function:

$$r_\rho(a,b) = mod((\alpha_\rho + a + cos(b))^2 + (\beta_\rho + b + sin(a))^2, 1) < \eta \tag{5.1}$$

input : square binary matrix M, where true at (a, b) represents that patches a and b do not overlap
output: subset of patches, divided into a pseudo-biclique
lists A and B are initialized with the row and column indexes of a random true point in M
while *sum of lengths of A and B* $< P$ **do**
    a := index of randomly selected row of M, such that all intersections with elements in B are true
    if *a is not empty* **then**
        add a to A
    end
    b := index of randomly selected column of M, such that all intersections with elements in A are true
    if *b is not empty* **then**
        add b to B
    end
end

Algorithm 1: pseudo-biclique graph division algorithm

where α_ρ and β_ρ are initialization parameters of the function, specific for each patch ρ. This binary Perlin noise function was chosen because it can be executed efficiently on a CPU and GPU, and it passes the Diehard battery of randomness tests[1]. The parameter $\eta \in [0,1]$ varies incidence (in our implementation, we set $\eta = 10/|\text{patches}|$. Figure 5.10 shows values of this function near the origin with $\eta = 0.5$.

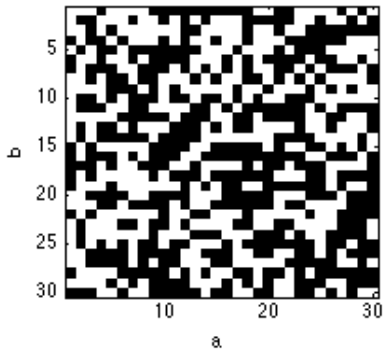


Figure 5.10: Values of the pseudorandom binary function

$r_\rho(t_x, t_y)$ is evaluated for each patch in the appropriate set. For “active" patches, those where $r_\rho(t_x, t_y)$ is true, the precomputed binary overlap matrix is used to find over-

[1] `http://stat.fsu.edu/~geo/diehard.html`

input : square binary matrix, where true at (a, b) represents that patches a and b overlap
output: subset of patches, such that there is no overlap
while *the matrix contains at least one true* **do**
    find first row with most true values;
    remove this row, and the same column;
end

Algorithm 2: Deterministic creation of non-overlapping patch subset

laps between them. Algorithm 2, which is deterministic, is then used to eliminate overlaps. This creates a small non-constant overhead, which is at most linear, but always terminates in very few iterations. As patches have been divided during offline preprocessing, this operation does not need to consider more than a few potential overlaps, each requiring one clock cycle.

This creates a subset of patches on this tile, called the “active subset". For each, the 2D precomputed binary map of the patch is used to find whether (t_x, t_y) is inside. If the point (t_x, t_y) is inside the patch, the associated pixel is retrieved from the synthesised base texture. This operation is a trivial array lookup in the appropriate saved patch, since patches are not saved as polygons.

If the point is not inside the patch, the nearest edge is found, and the procedure repeated for the adjacent tile. Note that, thanks to the rhombus-shaped boundary for patches overlaying the boundary between A and B, a pixel can only be affected by interchangeable patches from the adjacent tile which is nearest (Figure 5.8). If the point is found to be in one of the non-overlapping patches in the adjacent tile, the associated pixel is retrieved from the synthesised base texture.

If the point does not lie in a patch chosen in this tile, and does not lie in a patch chosen in the adjacent tile, we retrieve the pixel from the tile itself.

5.5.3 Complexity

The computational complexity of the live sampling is near-constant, thanks to the structure in which precomputed information is stored. The process for each pixel is to find which patches are active, then to find the active subset, and retrieve the pixel. Deciding which patches are active is a constant time operation, choosing the active patch subset is

at most quadratic in the number of patches, and retrieving the relevant pixel is also constant. As with other tiling methods, memory consumption is completely independent on the number of sampled pixels and the size of the output texture.

The computational complexity of the preprocessing step is comparable to sequential patch-based texture synthesis methods, but the contribution of this work is the texture map compression and on-the-fly synthesis. In practice, the preprocessing can even be done semi-automatically, allowing the user to manually choose patches which are visually satisfactory. The quality of the synthesised texture can be made arbitrarily tuned at the scale of patches, and patches can be chosen to have any size. Therefore, by definition, the runtime algorithm of our method can theoretically synthesise a texture of the same quality as any offline patch-based algorithm, repeatably, in parallel, and as fast as other tiling methods. This only depends on the quality of the preprocessing selection. Note that our implementation and results are of a fully automated algorithm, to allow a fair comparison to results published elsewhere.

On the GPU, pixel values are stored in DRAM and cached in texture memory, and all other variables can be optimized to fit into limited L1 processor shared memory. The SIMD model of the GPU allows each multiprocessor to evaluate equation 5.1 for multiple pixels, and the quadratic selection operation can be performed to the earliest stopping among pixels sharing a multiprocessor. The pixel coordinates in the repeating tile and base texture are returned. Since both images can fit into the texture cache, non-local pixel value retrieval will happen quickly, without reading DRAM memory.

5.6 Results

Our method produces textures whose quality is not dependent on the runtime computational complexity, but on the quality of the preprocessing step. Therefore, at equal memory footprint, our runtime performance is comparable to simple tiling methods (repeating precomputed tiles, as in [Grunbaum and Shephard, 1986], [Cohen et al., 2003], [Wei, 2004], and [Xue et al., 2007]), but the texture quality is comparable with patch-based iterative approaches.

In our experiments, precomputing was set to chose the 1000 best patch inter-

changes found over a 4 hour period, comparing tens of millions patch interchanges at different scales. For the textures used here, these settings proved satisfactory. Out of these, 300 patches were used in each tile type, and 400 patches were discarded, as discussed in Section 5.5.1. These amounts have been selected because of memory constraints, because a long search improves the patch quality, and because this many patches provide ample variation in the rendered texture. The upper bound on possible distinct rendered tiles is $300!^2$, but because there are up to 75% overlaps within each group, this is reduced by a few orders of magnitude.

The speed of the sampling process itself compares favourably with current approaches, despite the overhead to be calculated at every pixel. Speeds are reported on a single core of a 3GHz Xeon with 667 MHz DDR2 RAM. This overhead is tuned by changing η, which was set to $\eta = 0.1$. Since the time required to calculate a single pixel is constant, the algorithm scales linearly in the number of sampled pixels, and is parallelisable in a straightforward manner with speedup proportional to the number of cores.

Our method has multiple applications: ray tracing, rendering from bundled texture maps, or creating non-repeating patches, such as for ceramic tile-printing.

For irregular textures, synthesis requires handling complex properties, such as layering and overlapping, which are not handled automatically by optimal seam selection algorithms. For these, our method allows manual selection of an appropriate repeating tile and patches. Multiple repeating tiles can be synthesised, and the best one is chosen by an expert. Patches are prepared offline, so they can be shown to an expert user, who determines if they make a believable substitution, and selects the best. Synthesis results in figure 5.11 show how human intervention can improve synthesis quality, while maintaining the storage and run-time speed advantages of our method. Interestingly, by allowing patches to assume the locally optimal shape at numerous scales, interchanged patches often contain visual or semantic features of the example texture.

While [Vanhoey et al., 2013] works well for regular and stochastic textures, repeating a fixed patch map across an image cannot capture certain irregular textures. Certain irregular textures cannot be faithfully replicated by simply repeating patches of a given shape, no matter what the shape is (Figure 5.5). Our method does not restrict patches

to replace contents only within precomputed boundaries, instead allowing the boundaries themselves to be replaced by other patches, thanks to a patch biclique division. Section 5.6 contains a deeper discussion of the limitations for certain irregular textures.

The fishing net in figure 5.11a changes orientation, so patches replacing strings won't align with the wood texture. However, If patch boundaries lie on strings, the underlying wood texture will not align. In figure 5.11b, if patches are chosen to contain parts of leaves rather than leaves, faithful reproduction won't be guaranteed.

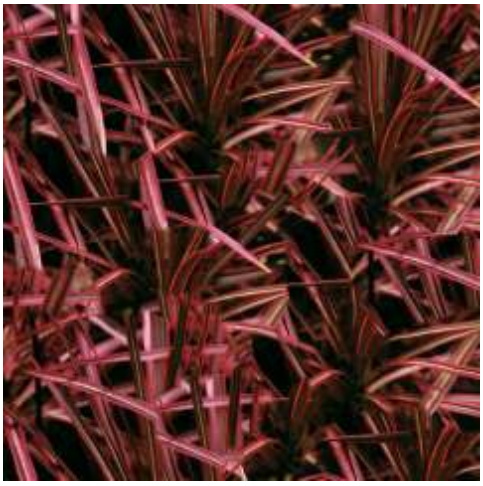
Note that all other textures in this chapter have been created without manual intervention.

5.6.1 Independent Pixel sampling for Ray Tracing

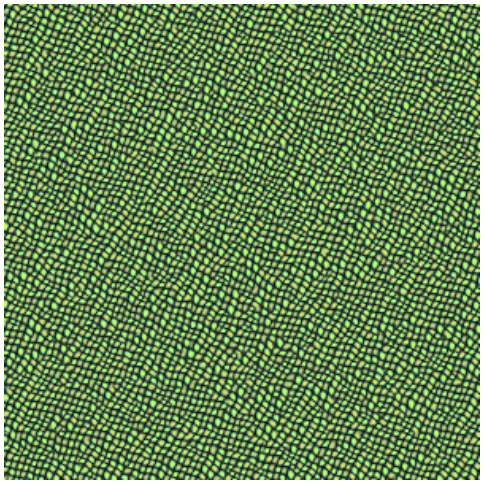
In various applications, a crucial property of texture synthesis algorithms is that the error produced when sampling distant pixels will not be constructive, but will have the same properties as randomly selecting pixels. This is referred to as the white noise property, and is particularly desirable with ray tracing. See results in figure 3.4. We benchmarked the algorithm's speed by randomly sampling pixels. Across the different textures we have tested, the time required to sample one pixel did not significantly vary, because the key parameters (number of patches, tile size, patch size) were set similarly for all textures. On the tested platform, the time required to sample one pixel averages 60 microseconds, with little variation. Out of this, 45 microseconds are required for pseudo-random binary sampling. Testing various random access scenarios for different textures has no effect on the retrieval time.

Unlike [Wei and Levoy, 2002], the algorithm does not place a spatial dependency on sampled pixels, so their performance cannot be compared in practice. Any algorithm which places dependency on sampled pixels would rely on cache, making it slower for larger output textures, so that sampling the texture millions of pixels apart would give our method an advantage with predictable results. Therefore, such a test was not performed in practice.

(a) High-level properties of textures can be handled by selecting a repeating tile and patches which satisfy the properties.



(b) Overlapping can be interactively modelled with our technique by manually appropriately selecting patches during preprocessing.



(c) exemplar size 64 × 64

(d) exemplar size 512 × 512

Figure 5.11: Synthesis results for irregular textures

(a) 192x192px

(b) 185x124px

Figure 5.12: Input textures

5.6.2 Patch sampling

If pixels forming a rectangle are not calculated independently, the pseudo-random binary sampling and patch selection is performed unnecessarily. As the largest part of time is spend precomputing which patches are used within a specific tile, performing sampling for adjacent pixels is significantly sped up when the tile properties are already known. Here, simply retrieving the information which patch the pixel is in and returning the appropriate pixel is even faster.

Such an implementation can be beneficial when entire neighbourhoods are required for further processing, such as in filtering.

Figure 3.4 compares the quality of a texture synthesised with Wang tiles [Cohen et al., 2003], with our method. Note the diamond shaped artefacts, which our method inherently avoids.

Figures 5.13, 5.14, and 5.15 show textures rendered from the inputs in Figures 5.7a and 5.12. Output textures from our algorithm display white-noise properties, without using cache (images are linearly transformed without interpolation to vanish at the horizon, results without offline manual tuning). Figure 5.13 demonstrates that our method doesn't create constructive repetitions at scales far larger than the input texture, but instead displays properties of white noise necessary for certain applications. Flaws in images generated by our method are seams, visible when the texel is large, and discontinuous objects (Figure 5.14). This problem is inherent to patch-based methods, as can be

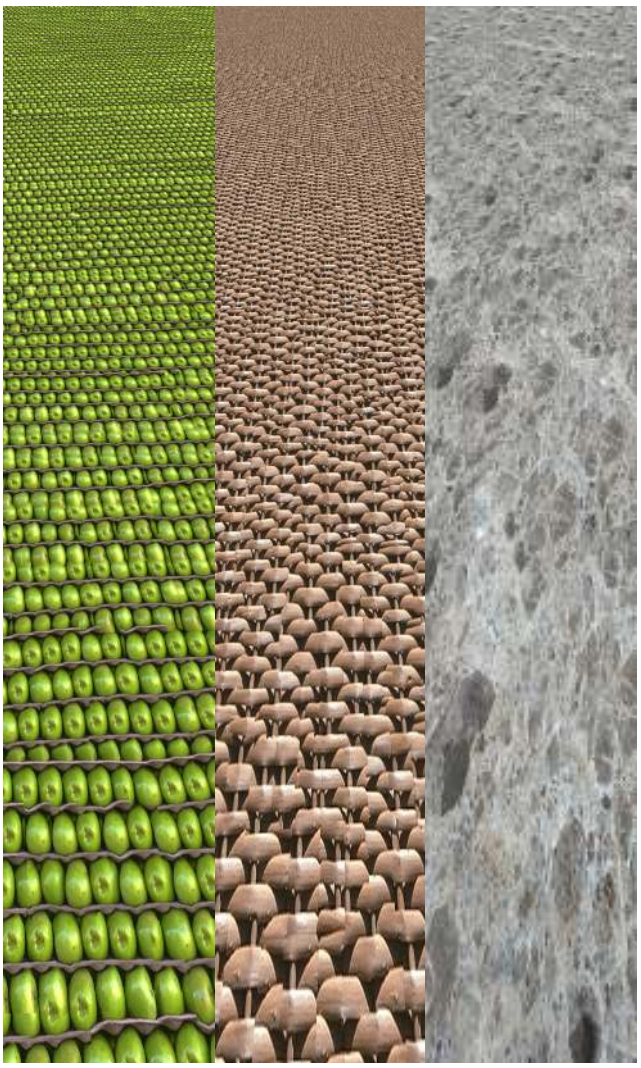


Figure 5.13: Output textures from our algorithm without offline manual tuning

seen in the results of the baseline method (Figure 5.15). Both issues can be mitigated by manual selection of the tile and patches when precomputing.

If adjacent pixels are rendered naïvely, the speedup is be linear. However, if congruous sections are retrieved simultaneously in each thread, the algorithm can be sped up further, because the decision process selecting patches need only be executed once.

Figure 5.14: Output textures from our algorithm (450 × 800px)

Figure 5.15: Output textures using Image Quilting [Efros and Freeman, 2001] (450x800px)

5.7 Evaluation

The proposed method is quantitatively evaluated in two scenarios: repeated pixel sampling, and repeated patch samling. The quantitative measurements presented here demonstrate high speed and low memory footprint. Along with perceived quality, which is evaluated in the next chapter, these constitute the goals that this algorithm was set to meet.

Memory consumption is determined by parameters of the preprocessing step, allowing fine tuning to best balance the tradeoff between quality and use of available memory. This was determined to be in the range of tens of kilobytes (for simple textures such as Figure 5.12a), to 1MB (for complex textures such as Figure 5.12b). If the first texture uses 600 patches of 5x5 to 86x86 (2400 bytes for difference vectors and 47kB of binary maps) pixels from a texture map of 96x96 (27kB), and a repeatable tile of 96x96 (27kB), the texture map totals 103kB. Note that memory consumption is a factor of four of the texture map, which has been approximately true for all included textures. For larger textures, the total memory footprint will always be determined by these four factors, and each of them can be tuned for the specific application.

A continuous texture of 512×512 pixels is synthesised in thirty to eighty of milliseconds, while the unoptimised pixel-based approach takes over a second on the evaluating PC.

5.8 Conclusion and Future work

We have presented a method with the benefits of current parallel texture synthesis algorithms, allowing texture synthesis in real-time environments from a minimal texture map. By sampling every pixel independently, parallel processing can be exploited for a synthesised texture of arbitrary size, while avoiding repetition along lines or rectangles to avoid visible seams. Our method makes it possible to perform exemplar-based texture synthesis of arbitrary size in times comparable with much simpler image retrieval operations. Our results are of the same quality as sequential patch-based synthesis, while significantly reducing retrieval time.

The benefits of this method over previous parallel patch-replacement [Vanhoey

et al., 2013] is two-fold: precomputed patches are not limited to lie on a precomputed fixed patch map, and rendered patches can lie over the boundaries of other precomputed patches. This makes the method suitable even to complex irregular textures.

A major drawback of the method is the repetition of corners of the repeatable tile. Careful selection of patches would allow these corners to be overlapped by patches as well, and future work could investigate this. Thanks to the read-only access of precomputed texture information, our method will make it possible to efficiently utilise GPU hardware to improve rendering times, which is also future work.

In future work, high-level ideas from this work will make it possible to create replaceable patches which do not follow any repeating grid. Patch sets are not inherently limited to two groups. The non-overlapping biclique introduced here is not limited to patches which follow the checkerboard pattern of precomputed interchange locations 5.8, but can follow a random Wang tile pattern. This will further combine benefits of the two techniques.

It is clear that with a single underlying tile, repetitions in areas not covered by patches will be visible. Future work could combine our technique with Wang tiles to deal with both this issue, and the issue of repeating edges in Wang tiles.

Thanks to the inherent parallelism, many further applications can be explored. This method offers novel benefits to texture compression, bundling of preprocessed textures with graphical design packages, virtual environment platforms, video games, rendering engines, and mobile applications.

The work presented in this chapter has been published in The Visual Computer [Kolar et al., 2015], a widely acclaimed journal of computer graphics with Impact Factor 1.62.

Chapter 6

Repeatable Texture Sampling Evaluation Study

The fast random patch placement method developed here creates textures of high quality. In order to quantify the quality similarity between the core synthesis algorithm and online synthesis, a subjective evaluation has been run and will be described in this section.

6.1 Objective

Quantitative measurements regarding computational complexity, speed and memory footprint are possible through quantitative analysis. However, a user study is required to justify claims regarding quality. Therefore, 15 participants were asked to complete an experiment comparing the online synthesised textures with Image Quilting textures. Since Image Quilting is the method used in the offline preprocessing step, its quality is an upper bound for the expected quality of textures synthesised online with the method of this chapter. Furthermore, it performed well in the comparative quality study of chapter 8, tying for 3rd.

6.2 Justification

By performing a user study on the perceived quality of textures produced by the algorithm of chapter 5, the algorithm can be verified to produce outputs of equal quality to the underlying offline algorithm. In conjunction with the a speed increment of several orders of magnitude, this proves the algorithm to be high speed and high quality, as originaly aimed for.

6.3 Participants

The 15 participants were computer graphics researchers. There were 12 males and 3 females, between the ages of 19 and 53, with a mean of 32 years. All confirmed to have good vision, with or without glasses (7 and 8 participants, respectively), and none of the participants were colour blind.

6.4 Materials

The study was performed with the 12 textures of the following chapter. For each exemplar, two methods of synthesis were used: the online method (chapter 5), and the underlying offline preprocessing method [Efros and Freeman, 2001]. A new texture was rendered for each participant with a random seed. Each printed page contains three images of the same texture. On the left, the input example, and on the right one generated by Image Quilting [Efros and Freeman, 2001], and the other by Repeatable Texture Sampling with Interchangeable Patches (chapter 5). The generated outputs are three times higher and three times wider than the input. Figure 6.1 shows an example page from the user study, as presented to participants.

6.5 Design

The experimental methodology chosen for this evaluation must compare two possibilities across randomly selected participants and randomly selected textures. Therefore, a no-intervention observational binary study is performed.

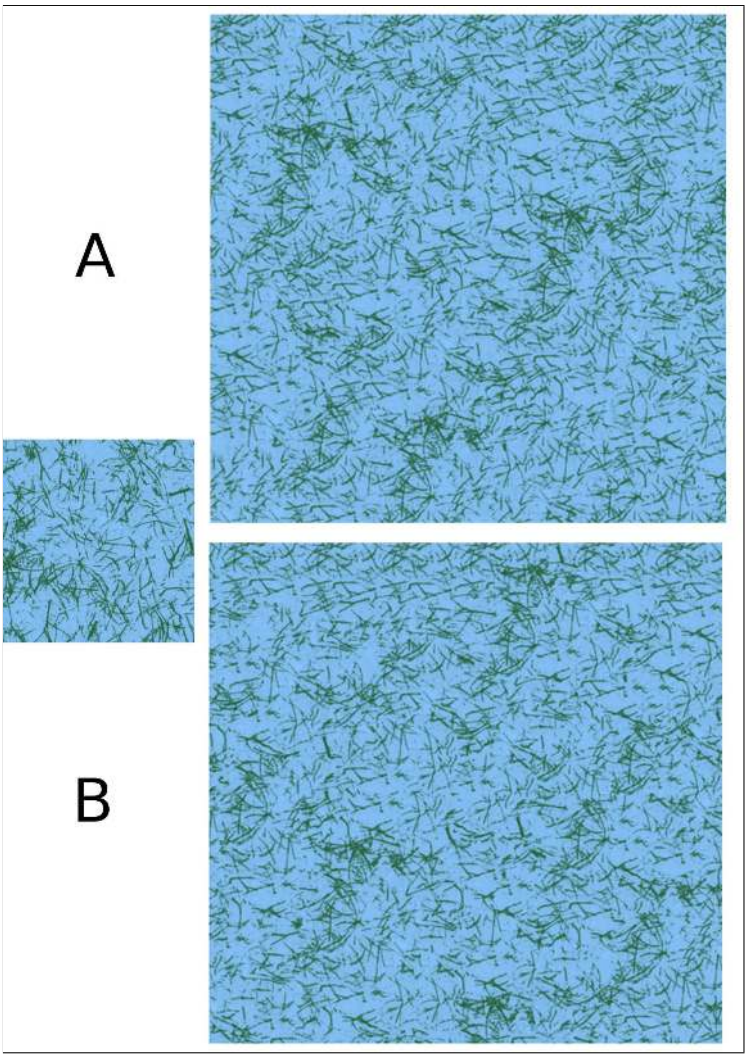


Figure 6.1: Sample page from the user experiment. Here, A is the Image Quilting result, and B is the Repeatable Sampling result.

The experiment is prepared as follows: each participant is given instructions, and twelve printed pages.

Participants were then asked to identify the output texture of higher quality. Every participant evaluates all 12 textures. Each pair is ordered randomly, and each participant sees a different render of the same algorithms.

The selected textures are from the texture dataset of chapter 7, so that they form a representative subset of textures across varying applications.

6.6 Analysis

This randomised test without intervention was completed by 15 participants. The results are ranked, where the selected original receives rank 1, and the other rank 0.

The collected measurements are paired two-sample variables, therefore the Wilcoxon signed rank test is appropriate for analysis. The test is two-tailed, because there exists a possibility of improvement over the vanilla version. While unlikely, the texture generated by the underlying algorithm may be of lower quality due to lower variance, or other factors.

The Wilcoxon two-tailed signed-rank test is performed on the data, which is aggregated rather than averaged. The resulting Z-value is -0.4708, and a p-value of 0.55675. The null-hypothesis, that there is no significant difference between the samples, is therefore not rejected.

The analysis is also performed on each texture separately, with resulting p-values between p=0.6752 and p=0.4383, therefore not rejecting the null-hypothesis for every texture independently.

6.7 Results

The user study did not demonstrate any difference between the quality of the methods. Therefore, the textures generated by the fast online rendering algorithm were not distinguishable from the textures generated by Image Quilting. In fact, there is no evidence that either method is of higher quality, demonstrating that speeding up rendering by several orders of magnitude with the method developed in chapter 5 does not involve statistically significant loss of quality.

As shown in the analysis, this result is true for the algorithm across all twelve textures, and for each texture independently as well.

6.8 Conclusion

The user study performed here compared the quality of textures synthesised online with the algorithm of chapter 5 with the underlying offline method. Fifteen participants demon-

strated with statistical significance that the perceived quality is not significantly different with either method, thus proving the hypothesis of high quality, and justifying the use of this method for high quality, high speed texture synthesis.

These results support the aims set out in the research methodology of chapter, where high quality is achieved in real-time texture synthesis.

Chapter 7

Synthesis Evaluation Dataset

7.1 Introduction

This chapter describes the process taken to establish a small representative set of textures. First, a clear objective, justification, and methodology is outlined regarding the preparation of the texture dataset in section 7.2, section 7.3, and 7.4. Previous popular datasets have shortcomings which are listed and addressed here. Then, in section 7.5, known properties are individually analysed, each of them being applied as a filtering method to select appropriate texture. The resulting textures are shown in section 7.6, followed by a listing of each property for each texture in section 7.7. Finally, 7.8 concludes this chapter.

7.2 Objective

The objective of this chapter is the creation of a texture dataset which is representative of real-world applications. The set of textures devised here must satisfy a range of perceptual and quantitative criteria, so that findings made with these images have clearly interpretable implications.

Previously published datasets suffer from a number of shortcomings. Most contain too many textures to enable all participants to judge all textures, few contain textures from varying domains, and none of the existing datasets contain any justification regard-

ing texture selection. The objective of this chapter is the creation of a dataset which will satisfy these criteria, as well as others that may be discovered to bear relevance.

7.3 Justification

A rigorously selected dataset of textures will allow meaningful analysis of texture synthesis methods. As this analysis will result in the clear knowledge of which texture synthesis algorithms create results of the highest quality, it will ultimately enable the accomplishment of the aim for this work set out in the Methodology chapter 4. Along with results directly applicable to this work, such a dataset will be of benefit to future researchers in the field of texture synthesis, by allowing them to develop and evaluate future methods.

The justification of work done in this chapter is therefore two-fold: enabling quality analysis for the purposes of the aim set forth in this thesis, and quantifying the texture properties on which future research must focus.

7.4 Methodology

In order to devise the appropriate dataset, the desired texture properties must be identified. Then, giving consideration to the desired measurements of the experiment in the next chapter, specific needs must be identified. This section identifies the concrete shortcomings of existing texture image datasets with regard to the objective, so that these specific needs are clearly identified.

Previous texture datasets have been devised with various goals, from the creation of comprehensive tileable textures for computer games to demonstrations of artistic renderings. However, despite the number and size of available datasets, none of them fulfill requirements for a minimal set of textures which are representative of known texture properties. None of these datasets attempt to cover texture properties as listed below in a structured way.

No existing texture dataset contains exemplars as well as reference textures. By making this available, new methods of evaluation are possible, namely comparison to a ground truth.

Certain datasets contain non-textures as well as textures, making them an interesting tool for understanding the inner workings and limitations of texture synthesis algorithms. However, non-texture images are not informative when assessing quality.

Numerous datasets lack the variety of properties which is necessary for a robust evaluation: all textures are at the same scale, or at the same resolution, or produced with the same camera. This creates a bias toward certain properties, making the results of algorithm evaluation challenging to extrapolate.

Few datasets contain less than 50 images, making it impractical to evaluate algorithms in publications or experiments. Conversely, this leads to cherry-picking among textures to demonstrate where a given algorithm performs well, rather than simplifying the evaluation process. In order to be able to create experiments where each participant judges all textures, the number of textures should be within the range of 10 to 15, while covering all known texture properties.

Texture properties are mentionned throughout previous work, but a complete list has not been compiled to date. The properties compiled here have been gathered from seminal work which considers texture properties [Lous, 1990, Fellner and Helmberg, 1993, Kobbelt et al., 1997, Lafortune et al., 1997, Buhmann et al., 1998, Foley et al., 1993], and named and unified here.

7.5 Review of Texture Properties

A texture is, by definition, an image which exhibits stationarity and locality [Wei et al., 2009], but only a random noise image can satisfy these perfectly. Texture synthesis algorithms must be able to handle textures with varying locality, stationarity, and various other properties.

The properties fall into two classes: ordinal and binary. Texture properties are measurements of global and local distributions of colour and shape in images, as well as their technical specifications. Some refer to aggregations of pixels, such as texel size, and others to individual pixels, such as resolution. Just as texture synthesis algorithms are expected to synthesise arbitrary textures, the texture properties presented here are expected to be covered in their full range by textures selected for the dataset.

Ordinal properties may cover a continuous or discrete range. However, defining a precise definition of these ranges is ill-defined and beyond the scope of this work. For example, consider the case of resolution, which has no upper bound, or the case of how natural a texture is, which is subjective. Therefore, since multiple range definitions can be created for these texture properties, the following convention is adopted: each ordinal property is defined by practical extremes, and each texture is labelled in the low-mid-high range. While these labels are subjective, they provide a practical indication to be used for texture selection to cover the full range of variability for each property. Through application of this heuristic, representative textures may be selected.

Similarly, binary properties are defined as true or false for any given texture. The full list of properties compiled from the literature is described below, and in tables 7.1 and 7.2. Ordinal properties are listed individually in sections 7.5.1 to 7.5.14, followed by binary properties in section 7.5.15.

7.5.1 Scale

The size of texture elements in relation to the size of the entire example. Textures with a small repeating element are considered fine-grained (low scale), such as the rough texture. Conversely, textures where the repeating element is large are considered blown-up (high scale), such as straw or green marble, where certain texture elements continue across the entire texture. Figure 7.1 shows examples of Scale across the selected dataset.

(a) low (b) mid (c) high

Figure 7.1: Texture Scale

Furthermore, textures may be Multi-Scale, exhibiting texture elements at various

scales simultaneously. Note that this property is independent of resolution.

7.5.2 Stochasticity

Stochasticity, or randomness, refers to the random variability within a texture. This can be formulated as follows:

Given information of other pixels in the texture, how predictable is another pixel?

Therefore, even textures which are entirely random globally (such as the smarties texture, where each element is placed randomly) do not exhibit perfect stochasticity, because nearby pixels are likely to belong to the same texture element. Similarly, textures where pixels are likely to change locally are not entirely stochastic if the structure is periodic (regular), such as the grid texture. Figure 7.2 shows examples of Stochasticity across the selected dataset.

(a) low (b) mid (c) high

Figure 7.2: Texture Stochasticity

7.5.3 Stationarity

Stationarity is the property which defines to what degree variance is linked with neighbourhood. A stationary pattern is similar to a local pattern, in that texture elements are dependent on local pixel neighbourhoods. However, a non-stationary pattern may be local or non-local, because both local and global underlying patterns can affect pixels to varying degrees. Figure 7.3 shows examples of Stationarity across the selected dataset.

Note that texture stationarity is independent of texture locality, because a non-stationary pattern may simultaneously be highly local, such as the flow-guided texture

(a) low (b) mid (c) high

Figure 7.3: Texture Stationarity

ink of figure 7.3(a).

7.5.4 Locality

Locality refers to the property of textures to depend on a local neighbourhood. Regular textures, those which fit onto a repeating lattice, exhibit low locality, because pixel values depend across the entire texture, such as the straw or grid textures. Textures exhibiting locality are for example the blades and smarties textures, because small neighbourhoods are locally independent from the rest of the texture. Figure 7.4 shows examples of Locality across the selected dataset.

(a) low (b) mid (c) high

Figure 7.4: Texture Locality

7.5.5 Flow-guided

This property chiefly refers to the underlying creation process of a texture. If a texture was generated in a process involving flow, it is said to be flow-guided. For example, the green marble and orange marble textures both exhibit flow, although the green marble shows less local phenomena. As seen in this example, this property can be seen to varying degrees, so the Flow-guided property is ordinal, rather than binary. Figure 7.5 shows examples of Flow across the selected dataset.

(a) low (b) mid (c) high

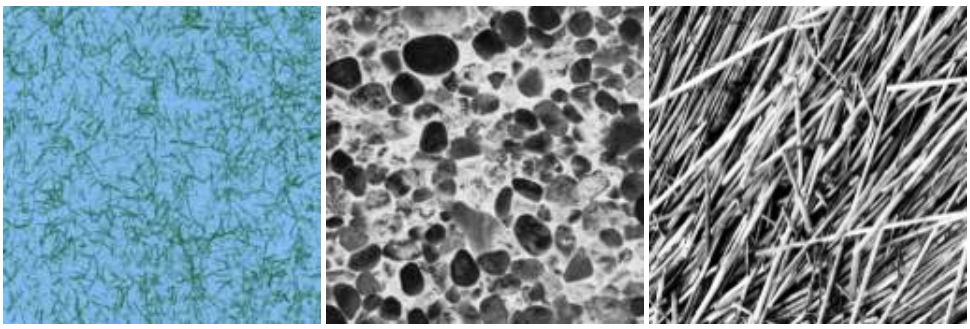
Figure 7.5: Textures which are Flow-guided to varying degrees

7.5.6 Shape Regularity

This texture property refers to variation of the shape of texture elements. For example, the smarties texture has low shape variation of its elements, therefore exhibiting high regularity. Conversely, high shape variance or lack of shape altogether mean low shape regularity, such as textures ink or blades. Since this property is not related to the concept of a loose lattice, it is different from the G-score of section 7.5.11. However, whenever a loose lattice can be fitted onto a texture, shape regularity corresponds with G-regularity. Figure 7.6 shows examples of Shape Regularity across the selected dataset.

7.5.7 Colour Regularity

Similarly to shape regularity, colour regularity refers to the variation of the colour across elements of a texture. This can be attributed to variability between elements, and variability within a given element. For example, the blades texture of figure 7.6(c) displays



(a) low (b) mid (c) high

Figure 7.6: Texture Shape Regularity

only two major colours, with little variation (high regularity). On the other hand, the smarties texture of figure 7.7(a) shows both types of colour variation (low regularity). Figure 7.7 shows examples of Colour Regularity across the selected dataset.

(a) low (b) mid (c) high

Figure 7.7: Texture Colour Regularity

7.5.8 Natural

This property refers to the degree to which a texture has been produced in a natural process. The marble textures show unedited images of stone, thus being entirely natural. The ink texture has been generated in an entirely artificial way. Others, such as chicago and smarties are produced in a partly natural process, consisting of both a repeatable generative process as well as an element of natural randomness. Figure 7.8 shows textures which are Natural to varying degrees across the selected dataset.

This property, together with the Flow-guided property 7.5.5, can serve as a help-

(a) low (b) mid (c) high

Figure 7.8: Textures Natural to varying degrees

ful indicator of repeatability of texture synthesis results across natural and synthetic textures. Because arbitrarily large samples from synthetic textures can be drawn, it is a potential benefit for texture synthesis algorithm development.

7.5.9 Absolute Texel Size

This property refers to the size in pixels of a repeating texture element. This is included to shift focus from algorithms which work at a given scale, to evaluate texture synthesis methods which perform well on textures with repeating elements at any scale. Here, "absolute" refers to the fact that the total number of pixels is measured, rather than a relative measure of variation in texel size within the texture.

7.5.10 Texels per Sample

The number of texels per sample refers to the repeating texture elements of sections 7.5.9 and 7.5.1. These three properties are inevitably interlinked, and are listed here for completeness.

7.5.11 A-score and G-score

As introduced in [Liu et al., 2004], Geometric (G) Regularity and Appearance (A) Regularity of near-regular textures provide a quantitative measure of deviation from a perfectly regular texture in terms of shape (G-regularity), and colour (A-regularity). See figure 7.9 for examples.

These differ from Shape Regularity and Colour Regularity defined earlier by requiring a texture to be near-regular. Textures which cannot fit into a loose lattice structure, such as the smarties texture of figure 7.7(a) are not near-regular, and therefore cannot have an A-score and G-score.

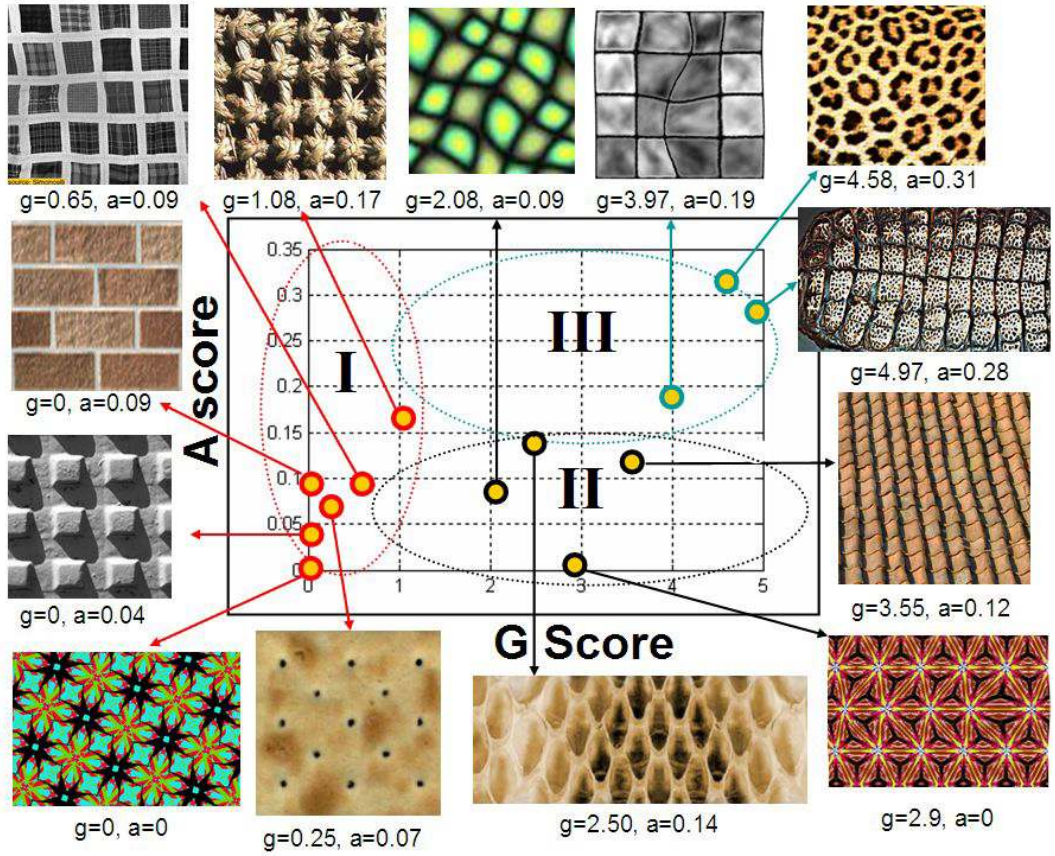
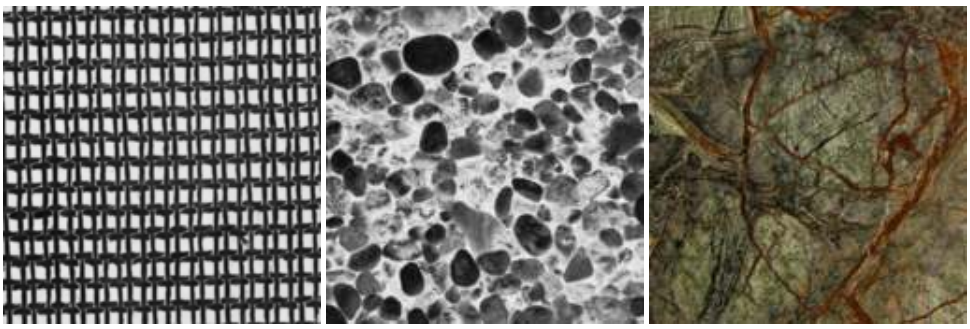


Figure 7.9: Near-Regular Textures plotted by Geometric (G) and Appearance (A) regularity [Liu et al., 2004]

7.5.12 Scale Variance

Scale Variance refers to the difference in scale between repeating elements in a texture. For instance, the pebbles texture of figure 7.10(b) contains pebbles of varying size, creating scale variance. On the other hand, the grid texture is scale invariant, because texture elements are of constant size. Figure 7.10 shows texture Scale Variance across the selected dataset.



(a) low (b) mid (c) high

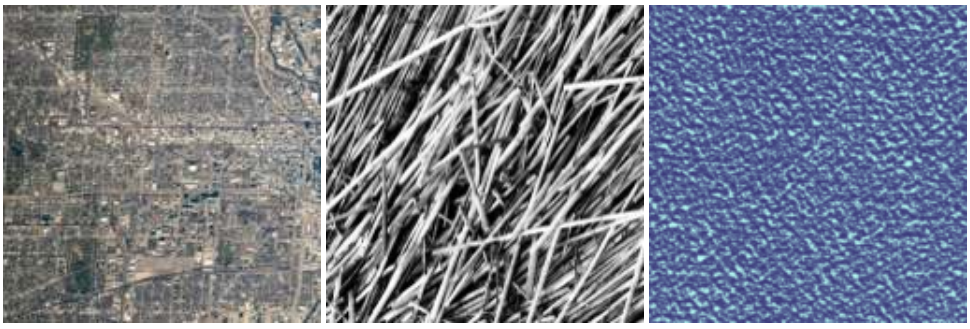
Figure 7.10: Texture Scale Variance

7.5.13 Example Resolution

The pixel size of the example texture. Similarly to Absolute Texel Size 7.5.9, this property serves to bring focus to textures of various resolutions, and to allow the evaluation of algorithms which can handle small as well as large example images. Table 7.3 shows selected texture resolutions.

7.5.14 Rotation

Texture Rotation refers to rotation invariance of repeatable texture elements. If elements of a texture can be rotated, such as the ink texture, it exhibits high Rotation. Conversely, if elements of the texture cannot be rotated, such as the grid texture, Rotation is low. Note that this property does not refer to rotating the entire texture, because this is always assumed to be possible. Figure 7.11 shows texture Rotation across the selected dataset.



(a) low (b) mid (c) high

Figure 7.11: Texture Rotation

Property	**Definition**	**Low**	**Mid**	**High**
Scale	fine-grained(fine) to blown-up(coarse)	rough	ink	pebbles
Stochasticity	entirely deterministic to entirely random	grid	straw	ink
Stationarity	different regions are perceived to be similar	smarties	orange marble	chicago
Locality	pixel values depend on small neighbourhood	ink	pebbles	rough
Flow-guided	generative process includes flow	pebbles	straw	green marble
Shape regularity	shapes don't vary to wide shape variance	ink	smarties	grid
Colour regularity	colours don't vary to high colour variance	smarties	green marble	blades
Natural	natural versus artificial	ink	chicago	orange marble
Absolute Texel size (resolution)	number of pixels in a texel	rough	blades	smarties
Texels per sample	number of texels in example	pebbles	chicago	blades
A-score	appearance regularity	chicago	straw	rough
G-score	geometric regularity	blades	chicago	grid
Scale variance	variation in scale of elements within texture	smarties	ink	green marble
Rotation	texture rotation invariance	grid	straw	blades
Example resolution	tiny to huge	pebbles	ink	smarties

Table 7.1: Ordinal properties of selected textures

7.5.15 Binary Properties

In addition to the ordinal properties discussed here, it is important to consider certain categorical binary properties to quantify textures. The properties listed in table 7.2 are discussed in depth here, to clarify and justify their definition and selection.

Historic textures were selected because various past publications have already demonstrated their performance on them in sets of synthesised images available from the authors. Rather than requiring some textures to be old, the purpose of this property is to allow a clear link to past work.

The two combinations of colour and shape regularity and irregularity have been included to match all four categories of near-regular textures according to their A-score and G-score. This produces the classification of [Liu et al., 2004], where near-regular textures are divided into four types, all of which are represented here:

0 Regular Geometry, Regular Colour

I Regular Geometry, Irregular Colour

II Irregular Geometry, Regular Colour

III Irregular Geometry, Irregular Colour

As mentioned in section 7.5.1, Multi-resolution textures contain texture elements at various scales, such that they interact with each other. For example, the chicago texture demonstrates this property: the road grid is a lower scale texture than the houses, but both of these properties satisfy the properties required to be a texture.

Global variance is the property that texture elements differ either by colour or geometry across the texture in a global, predictable manner. While this is at odds with the basic requirement for locality, a texture demonstrating this property is included in the dataset to judge how this affects texture synthesis algorithms.

An additional property of textures is that there may be a combination of overlapping textures. Unlike Multi-resolution, this property requires independence of the combined textures, instead of interaction between a higher-scale repeating texture element with a lower-scale one. The texture element may even be on the same scale. The

Property	**true**	**false**
Historic	straw	ink
Regular colour, irregular shape	pebbles	chicago
Irregular colour, regular shape	smarties	straw
Multi-resolution	chicago	blades
Global variance	green marble	rough
Overlapping multiple textures	ink	grid
Colour	blades	straw

Table 7.2: Binary properties of selected textures

ink texture demonstrates this property, because there are different independent overlapping generative processes involved.

Finally, textures in both colour and greyscale are included, to facilitate testing of single-channel texture synthesis algorithms. Single-channel texture synthesis algorithms can be applied on textures with three or more colour channels as well, by simply converting colours to a single measure, or for certain methods by providing a similarity metric. However, these results would be skewed by the chosen colour mapping, hence the benefit of providing two greyscale textures (straw and pebbles).

7.6 Results

Textures were selected such that at least one covers each extreme of any ordinal properties listed in Table 7.1, and each case of the binary properties 7.2.

Some popular textures which have been widely applied by the community had to be discarded because the texture was not represented in a 1/3 width and 1/3 height subimage.

Ten textures were selected, such that all properties of interest are covered, as per table 7.1 and table 7.2, and two additional textures were included to serve as a sanity check. These are the checkerboard and noise textures.

As with other datasets, another benefit of these textures are that they are freely available. However, by selecting only textures for which an input-output reference pair is available, new types of analysis are possible, such as automated quality analysis in comparison to the reference, and the possibility to manually compare synthesised outputs

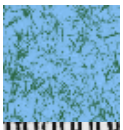
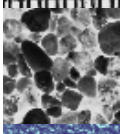
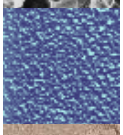
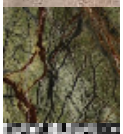
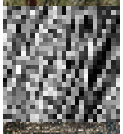
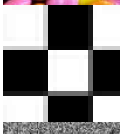
Texture		Source	Size
	blades	[Abdelmounaime and Dong-Chen, 2013]	640 × 640
	grid	[Brodatz, 1966]	640 × 640
	pebbles	[Brodatz, 1966]	640 × 640
	rough	[Abdelmounaime and Dong-Chen, 2013]	640 × 640
	orange marble	own work	1024 × 1340
	green marble	own work	657 × 876
	straw	[Brodatz, 1966]	256 × 256
	chicago	[crew ISS NASA, 2014]	2266 × 2267
	ink	own work	1281 × 653
	smarties	[Schwarzkopf, 2012]	3543 × 2362
	checkerboard	own work	256 × 256
	noise	own work	1000 × 1000

Table 7.3: Selected textures

with a ground truth texture. The latter is the subject of the following chapter.

7.7 Properties of Selected Textures

The selected textures were chosen in such a way that they cover the space of properties in a uniform way, handling as many varied combinations of properties as possible. For each of the selected textures, the scalar properties are listed in table 7.4, and the binary properties in table 7.5. An X signifies that the property is undefined for the texture, either because texel size is considered for a multi-scale texture or because a regular grid cannot be fitted onto the texture.

As can be seen in these tables, every property contains a balanced uniform categorical distribution of values. Furthermore, no property value correlations emerge from this data, demonstrating that the textures are selected in a uniform, unbiased fashion.

Texture	Scale	Stochasticity	Stationarity	Locality	Flow-guided	Shape variance	Colour variance	Natural
blades	low	high	low	high	mid	high	low	high
grid	mid	low	high	low	low	low	low	low
pebbles	high	mid	mid	mid	low	mid	low	high
rough	low	low	mid	high	low	low	low	mid
orange marble	high	high	mid	mid	mid	high	mid	high
green marble	high	high	mid	high	high	high	mid	high
straw	high	mid	high	low	mid	high	mid	mid
chicago	X	mid	high	high	mid	mid	high	mid
ink	mid	high	low	low	high	high	mid	low
smarties	mid	mid	low	high	low	mid	high	mid
checkerboard	high	low	high	mid	low	low	low	low
noise	low	high	low	low	low	high	low	low

Texture	Absolute texel size	Texels per sample	A-score	G-score	Scale variance	Rotation	Example resolution
blades	mid	high	X	X	low	high	mid
grid	low	mid	low	high	low	low	mid
pebbles	low	low	X	X	high	high	mid
rough	low	high	X	X	low	mid	mid
orange marble	high	low	X	X	mid	high	high
green marble	mid	low	X	X	high	mid	mid
straw	low	low	mid	mid	low	mid	low
chicago	X	mid	low	mid	low	low	high
ink	mid	mid	X	X	mid	high	mid
smarties	high	low	X	X	low	low	high
checkerboard	high	low	high	high	low	low	low
noise	low	high	X	X	low	low	mid

Table 7.4: Ordinal properties in the low, middle, high range for selected textures. An X signifies that the property is undefined.

Texture	Historic	Regular colour, irregular shape	Irregular colour, regular shape	Multi-scale	Global variance	Overlapping multiple textures	Colour
blades	true	false	false	false	false	false	true
grid	true	false	false	false	false	false	false
pebbles	true	true	false	false	false	false	false
rough	true	false	false	false	false	false	true
orange marble	false	false	false	false	false	false	true
green marble	false	false	false	false	true	false	true
straw	true	false	false	false	false	false	false
chicago	false	false	true	true	true	true	true
ink	false	true	false	false	false	false	true
smarties	false	false	true	false	false	true	true
checkerboard	true	false	false	false	false	false	false
noise	true	true	false	false	false	false	false

Table 7.5: Binary Properties for selected textures

7.8 Conclusion

This chapter has discussed the topics of texture properties and texture datasets. We have created a minimal set of textures which are representative of desired properties, with the hope that the performance of any method may be judged over all textures simply by observing results on these.

A new minimal set of 12 textures has been proposed, such that a wide number of known ordinal and binary properties are covered with few textures. This contribution has several benefits: standardising texture synthesis algorithm evaluation, discouraging selective publication of positive results only, making evaluation for as many properties as possible simpler than before, and making an interpretation of results across various properties possible.

Finally, the proposed dataset contains natural textures sufficient for synthesis, as well as natural reference samples. This will allow absolute evaluation of quality and insightful analysis into user preference across textures.

The work in this chapter has been published at Eurographics [Kolar et al., 2017], Europe's most prestigious conference in computer graphics. All textures are freely available under various licences at full resolution in the online Additional Material at `https://github.com/mrmartin/ordering_study`.

Chapter 8

A Subjective Evaluation of Texture Synthesis Methods

8.1 Introduction

Exemplar-based texture synthesis has numerous applications across computer graphics, such as inpainting, rendering, and manufacturing. Thirty years of research has resulted in a wide array of approaches, many of which claim wide applicability across all textures.

However, texture quality is subjective, and independent evaluation of known methods is lacking. There is no structured way of evaluating the quality of texture synthesis algorithms, allowing anyone to claim superiority, making it possible to evaluate on heavily-tuned results, and stagnating progress. In this chapter, a study is performed using the texture dataset of the preceding chapter, by rendering each texture with selected algorithms, and by performing a ranking study. The chapter is structured as follows: section 8.2 describes the objective of the study, section 8.3 explains the justification and benefits of this work, and section 8.4 contains the methods used to prepare, perform, and analyse the user experiment. Finally, sections 8.5 and 8.7 contain the results and conclusion, respectively.

8.2 Objective

The objective of the user study documented in this chapter is a structured qualitative evaluation of texture synthesis methods. By quantifying the performance of state-of-the-art algorithms by various participants, across varied textures, and in varied lighting conditions, the method of the highest quality will be identified.

8.3 Justification

Once the method of the highest quality is identified, it may be incorporated into the fast online rendering algorithm of chapter 5. As outlined in the overall plan for the thesis, this will enable the combination of speed and quality necessary for real-time texture rendering applications, such as ceramic tile design rendering.

The user study is rigorously prepared, performed, and analysed with statistical tests in order to produce results useful in the selection of the highest quality algorithm, as outlined in the Research Methodology chapter 4.

The user study materials have been prepared by focusing on specific types of textures, which covers known properties, as described in the previous chapter. To this end, several texture classification schemes have been proposed, dividing textures by regularity of shape and colour, and other properties. This has served to narrow down the problem on textures with specific properties.

The benefit of this work is twofold: First, this work is the first to compare texture synthesis algorithms in a user study on textures selected to represent the wide variability of all textures. Statistically significant preferences are identified for algorithms, as well as over textures. Second, we offer this minimal set of textures with which future work may be subjectively evaluated, to facilitate the development of future texture synthesis methods.

8.4 Experiment and Methods Used

This section describes the experimental method, including design, choice of algorithms, materials and procedure.

8.4.1 Design

The experiment compares six algorithmic methods across twelve textures. The experiment is based around a ranking design based on similar experiments for HDR texture comparison [Mukherjee et al., 2016]. The ranking procedure can be seen in Figure 8.1 were a reference exemplar, acting as ground truth is presented and asked to be compared with the presented texture stimuli. The participant ranks the stimuli interactively according to how close to the exemplar they believe the stimuli to be. Ranking gives the opportunity to participants to be able to view all stimuli an equal number of times, and be able to view them concurrently within a reasonable amount of time reducing the fatigue associated to other design choices, such as pairwise comparisons. The *texture* variable corresponds to a within-participants independent variable encompassing the different texture stimuli. The *method* variable is also a within-participants independent variable referring to the distinct texture synthesis algorithms. The *method* variable also includes a hidden reference besides the texture synthesis generated textures. The hidden reference is added to help identify differences between the reference and texture synthesis methods and to see how close these are to the ground truth. The experimental question was explicitly formulated to allow multiple interpretations, asking participants to "Sort [...] by how much they look like the original.", "Sort [...] by order of realism.", and "Order [...] according to how similar to the reference you think they are."

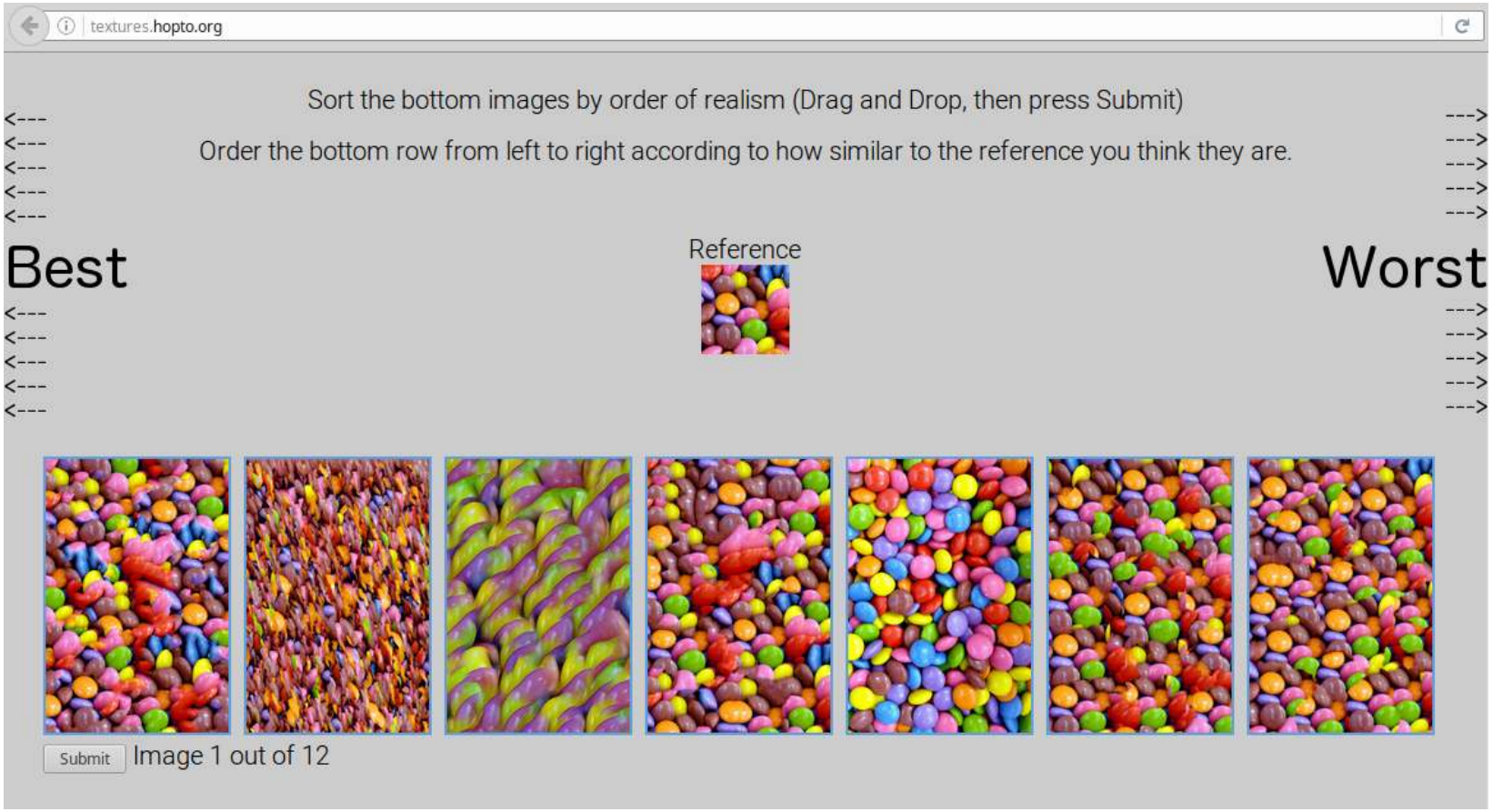


Figure 8.1: Experiment User Interface

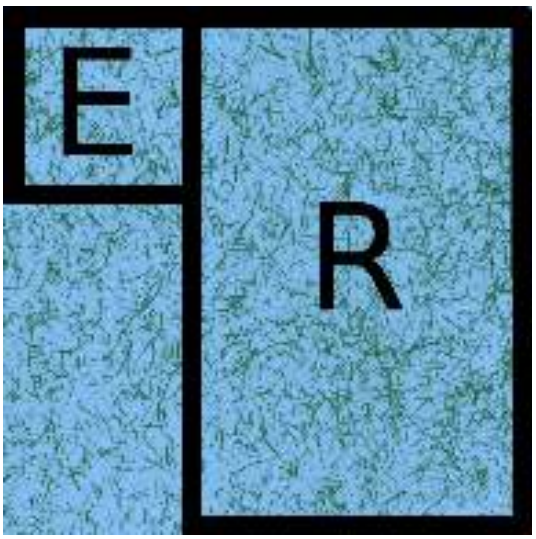


Figure 8.2: Process of selection of the input exemplar E and the reference output R from a square sample texture.

8.4.2 Materials

The selection of textures and texture synthesis algorithms for the study has been guided by various constraints. First, in order to compare synthesis outputs to a ground truth texture, it was necessary to chose textures large enough to be divided into an input and output sample. Furthermore, this output sample needed to be larger than the input, in order to clearly demonstrate algorithmic properties. An output aspect ratio of $2:3$ was chosen, and because it was desirable to present all textures equally on one row, the number of outputs was limited to 7 (see user interface in figure 8.1). The textures can be reordered interactively, and the entire page scales up to maintain correct aspect ratios between the reference and the synthesized textures.

The input and output images are generated by first taking a texture, and cropping it to a square. Then, the top-left corner of $1/3$ width and $1/3$ height is extracted to give the exemplar. The right band of $2/3$ width and full height is used as the ground truth. See figure 8.2.

The six selected algorithms are:

Image Quilting [Efros and Freeman, 2001]

Ashikhmin Natural Texture Synthesis [Ashikhmin, 2001]

Resynthesizer	[Harrison, 2005]
Self Tuning Texture Optimization	[Kaspar et al., 2015]
LazyFluids Appearance Transfer	[Jamriška et al., 2015]
CNNMRF	[Li and Wand, 2016]

These algorithms were selected for various reasons, but always focusing on finding representative algorithms of highest quality. Image Quilting and Ashikhmin are baselines, cited as performing best and often compared against in previous work. Resynthesizer and CNNMRF are widely available and used outside the research community, and results of impressive quality have been accomplished with them. Self Tuning and LazyFluids were selected because they are widely acclaimed state-of-the-art methods with regard to quality. See section 3.7.1 for a discussion of how these algorithms perform synthesis. All selected algorithms were run with the same default parameters across all textures, to simulate a non-expert user environment. The work here does not focus on the expert user, who may adapt method parameters.

User Interface

Figure 8.1 shows the GUI used for selection. The experiment code runs entirely within a browser, so that it can be performed online; this enables reaching a wide variety of participants with various monitors, resolutions, preferences, and lighting conditions. The GUI presents the exemplar as a ground truth reference in the top centre and the six methods plus hidden reference right under. The GUI allows the selection and movement of any of the stimuli corresponding to the methods along the x-axis to be ordered according to participant preferences.

The background is a neutral grey, and the participant is requested to maximize the window. All images scale to fill as much of the screen as possible, and the aspect ratio between the input and output are maintained.

8.4.3 Participants and Procedure

Participation was anonymous. In order to guarantee serious participants in the evaluation of the textures, jurors were selected from Masters students at the Warwick Manufacturing Group of the University of Warwick, graduate students from the Computer Graphics and Multimedia group of the Brno University of Technology, and interested domain specialists. Therefore, their performance can be interpreted as indicative, and the collected data are not noisy.

Screen resolution was recorded, and participants with less than HD 720p were discarded. Participants who did not change the order of any textures, and those who did not reach the end of the experiment were discarded. In total, the user study was performed by 67 participants.

A mass e-mail was sent out to students and staff of two universities to recruit participants on a voluntary basis. Those who responded were sent a URL corresponding to the experiment. No personal data was collected. The experiment was run entirely in browser and results stored on a server.

Furthermore, the experimental design focuses on various use-cases, by not restricting participants in the use of their own monitors in various lighting conditions. Statistical analysis allows us to quantitatively answer questions regarding juror performance, sensitivity, and experiment repeatability. These are discussed individually in the next sections.

8.5 Results and Analysis

Analysis was conducted using the non-parametric Kendall test for Concordance. Kendall's test for Concordance (W) provides a statistic between 0 and 1 conforming to the agreement across participants. A W of 1 indicates perfect agreement among participants and 0 means perfect disagreement. W can also be tested for significance and this is also reported in the results. Pairwise comparisons among all the methods for each of the textures were also conducted, these give an indication of if there are any significant differences across methods for a given texture, or in the overall.

Results for each texture, as well as for across all textures using collapsed scores, are in table 8.1. All orderings are significant to $p < 0.01$, except orderings within each group. Kendall's W coefficient of concordance ranges from 0 (no agreement) to 1 (complete agreement). The groupings in each row demonstrate significant differences. Methods not grouped together indicate significant differences across those methods for that particular texture.

8.5.1 Juror Performance

Statistical analysis throught the calculation Kendall's W shows consistency amongs participants. Juror performance is highly consistent across the entire experiment, and across individual textures as well.

Disagreements can be interpreted in two ways. Either as similar quality among the textures, or as disagreement among participants. Over 85% of all rankings are significant. For the smarties texture, for example, there are 19 significant ranking agreements and 2 non-significant agreements. The fact that most rankings are statistically significant shows that the disagreements do not stem from the jurors, who are not given information about the textures, but in the textures themselves.

8.5.2 Experimental Sensitivity

While the results of the ranking study are statistically significant, it is important to consider the uncertainty of the measurements with regard to unknowns and other sources of uncertainty in the experiment. Much of this is answered by two factors: the variability among participants, devices, and lighting conditions, and the significance and agreement amont rankings.

However, it remains possible that the results of the study performed here could be different if the experiment were performed differently. For example, by selecting older participants, other rankings may be significant. The results presented here are limited by the experimental setup chosen here, and while results may be extrapolated to printed materials or specialised textures, it is important to remember the limitations of the experimental setup.

8.5.3 Experiment Repeatability

Along with the W-agreement among jurors, the significance level p of rankings is calculated in the analysis. A significance level $p < 0.01$ demonstrates that a portion of participants may be discarted, and the results would remain the same. Therefore, repeating the experiment with different participants may produce different rankings, but not significantly so.

The experimental setup is made available online, and thanks to the simplicity of the experiment, it may be repeated to verify our results. The experimental setup may also be adapted to new data or testing platforms, so this work facilitates the quantification of similar qualitative questions in different fields.

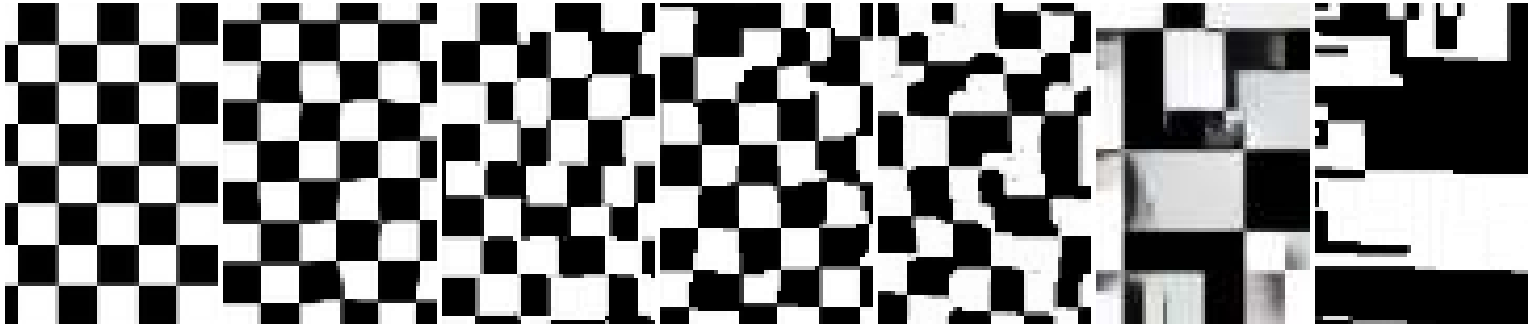
The results with the checkerboard texture are visually clearest, and also show the higher Kendall's coefficient of concordance. However, other textures cause wide disagreement. This is caused by differing views on what constitutes an ideal output.

Texture	ranks							Kendall's W
smarties	reference	resynthesizer	quilting	self tuning	LazyFluids	Ashikhmin	CNNMRF	0.737
checkerboard	reference	self tuning	quilting	resynthesizer	LazyFluids	CNNMRF	Ashikhmin	0.819
blades	reference	resynthesizer	LazyFluids	self tuning	quilting	Ashikhmin	CNNMRF	0.44
grid	reference	quilting	self tuning	LazyFluids	resynthesizer	Ashikhmin	CNNMRF	0.811
pebbles	LazyFluids	reference	self tuning	resynthesizer	quilting	Ashikhmin	CNNMRF	0.461
ink	self tuning	LazyFluids	reference	resynthesizer	Ashikhmin	quilting	CNNMRF	0.558
rough	quilting	resynthesizer	self tuning	LazyFluids	reference	Ashikhmin	CNNMRF	0.447
chicago	resynthesizer	self tuning	quilting	LazyFluids	reference	Ashikhmin	CNNMRF	0.619
noise	reference	LazyFluids	quilting	resynthesizer	self tuning	Ashikhmin	CNNMRF	0.361
green marble	self tuning	LazyFluids	reference	resynthesizer	quilting	Ashikhmin	CNNMRF	0.517
orange marble	LazyFluids	self tuning	reference	resynthesizer	CNNMRF	Ashikhmin	quilting	0.266
straw	self tuning	quilting	resynthesizer	LazyFluids	reference	Ashikhmin	CNNMRF	0.363
ALL	reference	self tuning	resynthesizer	LazyFluids	quilting	Ashikhmin	CNNMRF	0.402

Table 8.1: Subjective ranks with Kendall W, for each texture separately, and over all textures. Ranks are from left to right

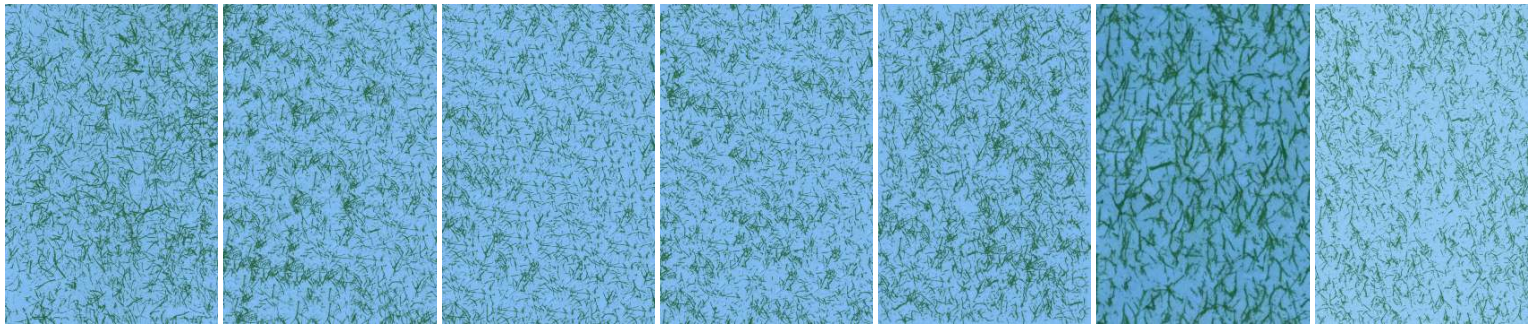
(a) reference (b) resynthesizer (c) quilting (d) Self-Tuning (e) LazyFluids (f) Ashikhmin (g) CNNMRF

Figure 8.3: The smarties texture, ordered by user preference



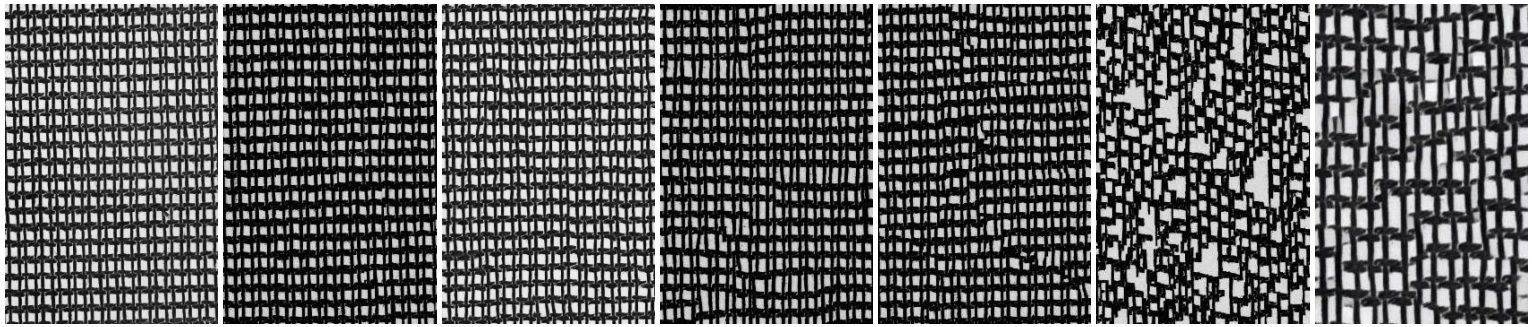
(a) reference (b) self-tuning (c) quilting (d) resynthesizer (e) LazyFluids (f) CNNMRF (g) Ashikhmin

Figure 8.4: The checkerboard texture, ordered by user preference



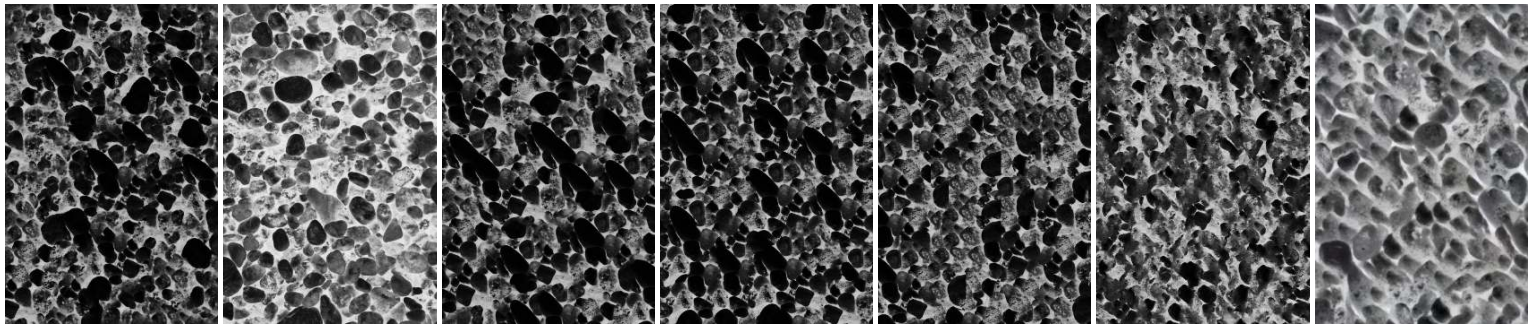
(a) reference (b) Self-Tuning (c) quilting (d) resynthesizer (e) LazyFluids (f) CNNMRF (g) Ashikhmin

Figure 8.5: The blades texture, ordered by user preference



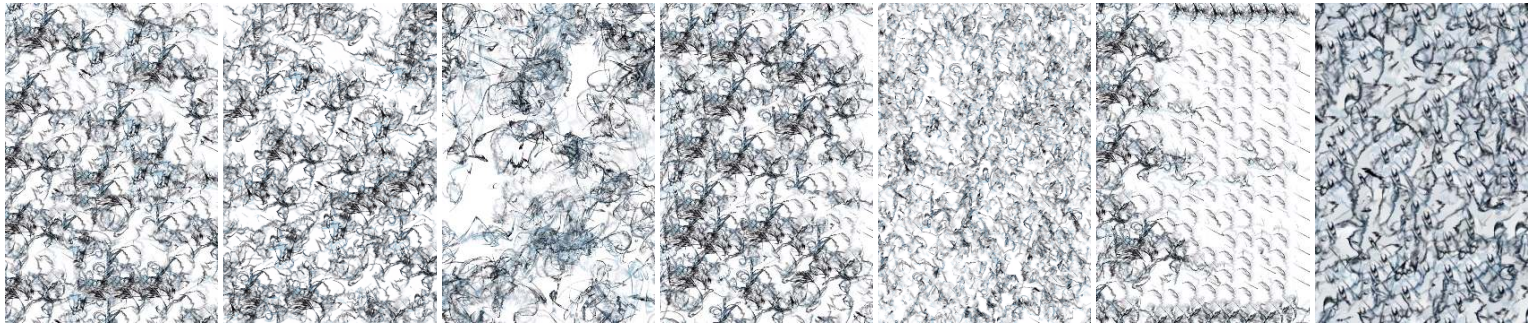
(a) reference (b) quilting (c) Self-Tuning (d) LazyFluids (e) resynthesizer (f) Ashikhmin (g) CNNMRF

Figure 8.6: The grid texture, ordered by user preference



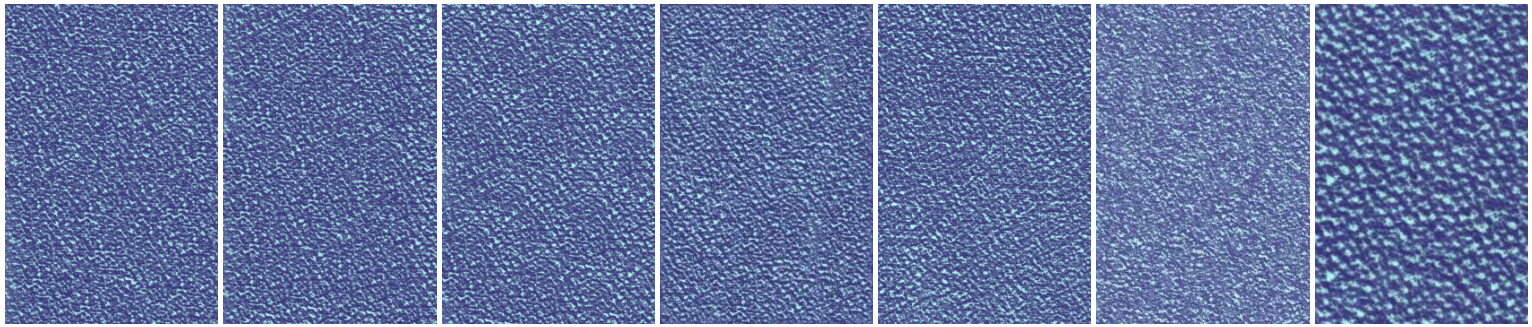
(a) LazyFluids (b) reference (c) Self-Tuning (d) resynthesizer (e) quilting (f) Ashikhmin (g) CNNMRF

Figure 8.7: The pebbles texture, ordered by user preference



(a) Self-Tuning (b) LazyFluids (c) reference (d) resynthesizer (e) Ashikhmin (f) quilting (g) CNNMRF

Figure 8.8: The ink texture, ordered by user preference

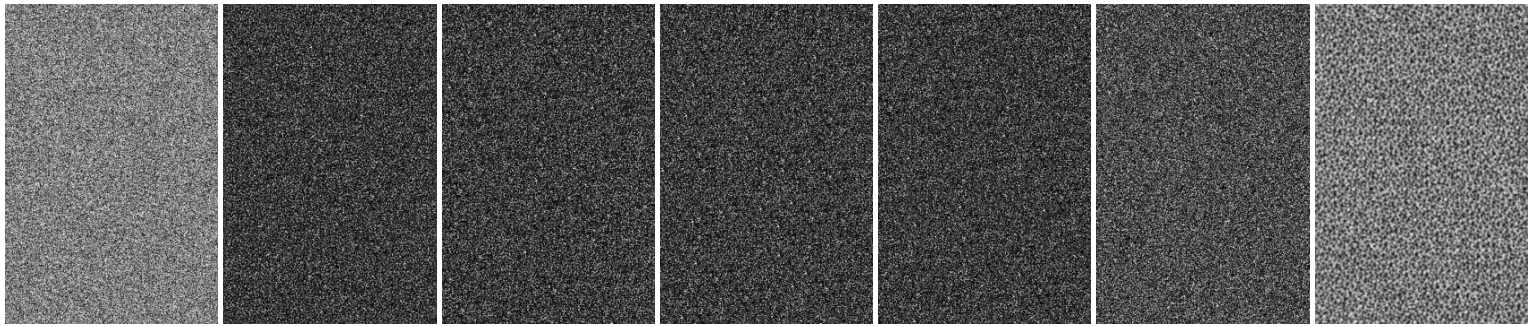


(a) quilting (b) resynthesizer (c) Self-Tuning (d) LazyFluids (e) reference (f) Ashikhmin (g) CNNMRF

Figure 8.9: The rough texture, ordered by user preference

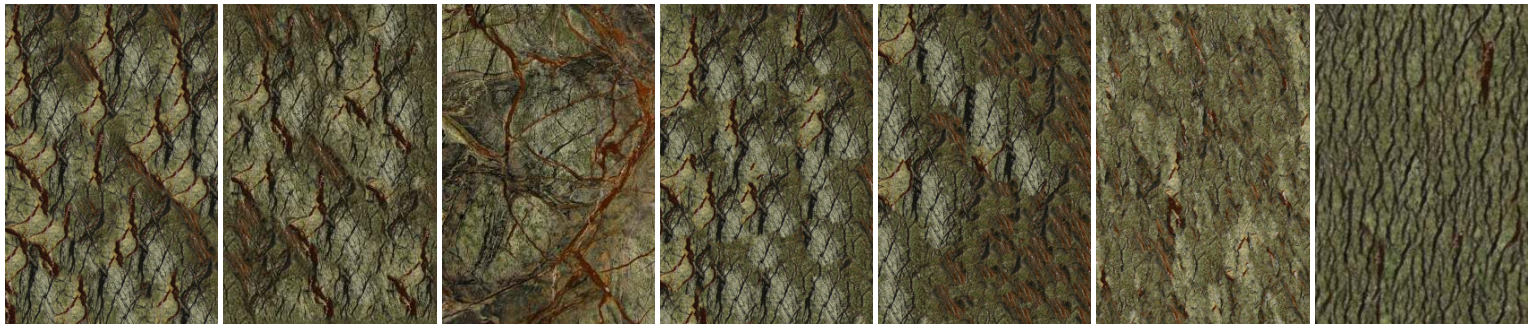
(a) resynthesizer (b) Self-Tuning (c) quilting (d) LazyFluids (e) reference (f) Ashikhmin (g) CNNMRF

Figure 8.10: The chicago texture, ordered by user preference



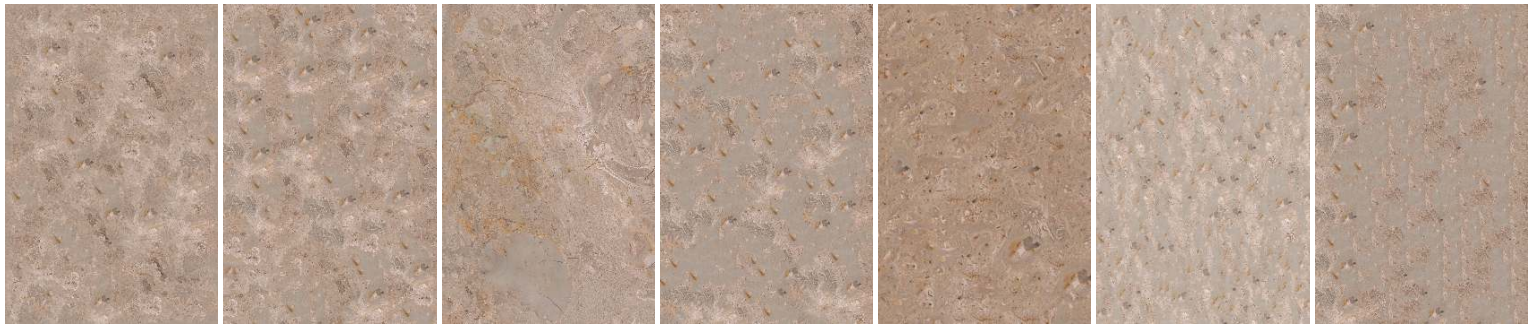
(a) reference (b) LazyFluids (c) quilting (d) resynthesizer (e) Self-Tuning (f) Ashikhmin (g) CNNMRF

Figure 8.11: The noise texture, ordered by user preference



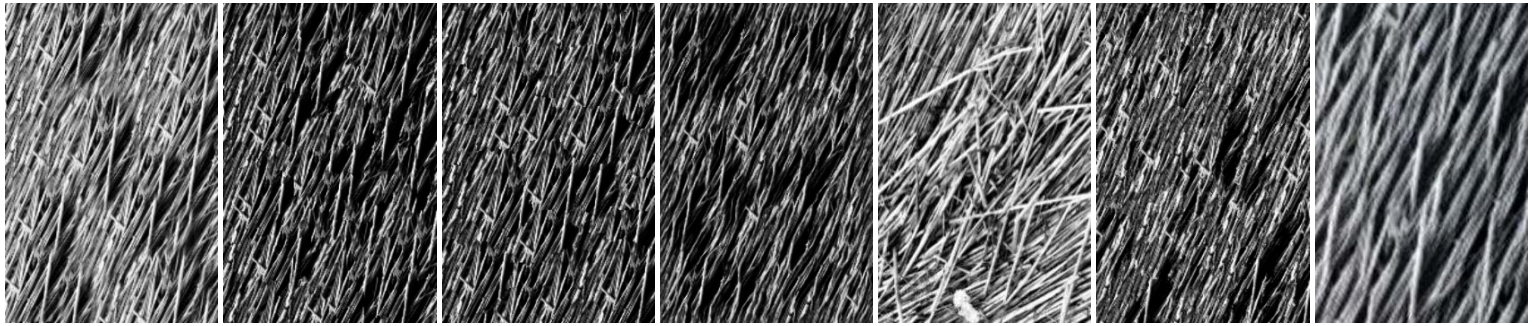
(a) Self-Tuning (b) LazyFluids (c) reference (d) resynthesizer (e) quilting (f) Ashikhmin (g) CNNMRF

Figure 8.12: The green marble texture, ordered by user preference



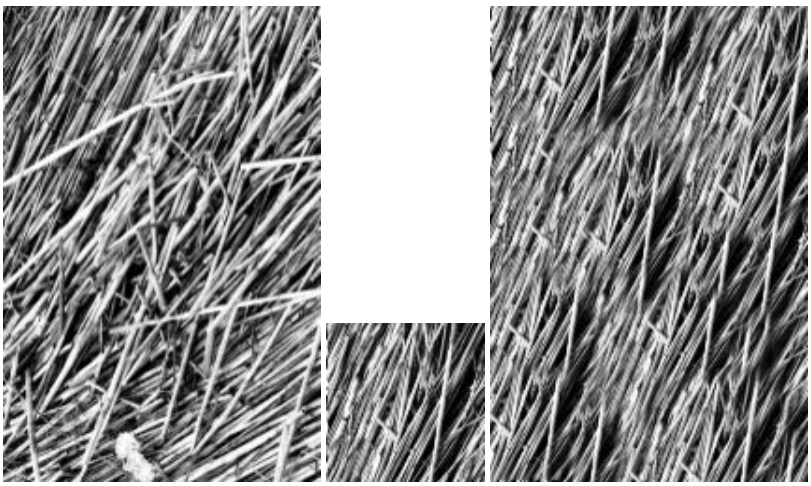
(a) LazyFluids (b) Self-Tuning (c) reference (d) resynthesizer (e) CNNMRF (f) Ashikhmin (g) quilting

Figure 8.13: The orange marble texture, ordered by user preference



(a) Self-Tuning (b) quilting (c) resynthesizer (d) LazyFluids (e) reference (f) Ashikhmin (g) CNNMRF

Figure 8.14: The straw texture, ordered by user preference



(a) reference (b) exemplar (c) self-tuning output

Figure 8.15: A texture for which the reference shows greater variance than the exemplar, which is not replicated in synthesis

In fact, for several textures participants consistently agree that the reference is not the best (pebbles, rough, chicago, green marble). This is because the reference image contains greater variance than the example, while synthesized textures closely match the inputs' visual properties (figure 8.15).

Consistently with assumptions, the noise texture is easy to synthesize correctly, because four out of six methods create results indistinguishable from the reference. There are seven textures for which some methods produce results not significantly worse than the reference (blades, pebbles, ink, noise, green marble, orange marble, straw). The texture synthesis methods for these textures may be broadly considered successful, because a single algorithm (Self Tuning) is indistinguishable from the reference in these textures.

Out of the remaining five textures (smarties, checkerboard, grid, rough, chicago), two reference textures are evaluated as significantly worse than synthesized outputs (rough, chicago), and in the other three, the reference is significantly better than synthesized textures.

The chicago texture does not strictly fit the definition of a texture, because patches are potentially identifiable. Therefore, the selected example is different from the selected reference, unlike outputs of the four best algorithms.

The synthesis of three textures is not solved by any evaluated method. These are

Figure 8.16: Blended images of Texture Synthesis methods ordered by user preference

the smarties texture (figure 8.16), checkerboard (figure 8.4), and grid. These share some common properties: mid-to-high shape regularity, mid-to-low stochasticity, and low locality. These properties, together with complex shape patterns, form patterns which are hard to recreate without considering the underlying generative process.

High shape regularity poses challenges which none of the algorithms handle well. Regular texture elements are not replicated perfectly (figure 8.4) in the output, and a human observer readily identifies even minute flaws, so the relative perceived quality is significantly worse than the reference.

The results demonstrate that numerous complex properties are well handled by the top four methods: texel size, example resolution, texels per sample, scale, scale variance, stochasticity, and shape.

Considering the shared properties of textures which are not well synthesized, and the fact that reference images contain greater variance than the exemplar, the major focus of texture synthesis research should be regularity mapping, learning to replicate the texture's generative process, and a structured approach to estimate variability to increase it in the output.

Figure 8.10 shows the full-scale vs the details of the chicago texture, revealing additional clues regarding method quality. For example, LazyFluids seems best at generating novel textures of the scale of the exemplar, but does not perform well when generating a large sample.

8.6 Discussion

The findings of this research enable us to provide a novel process that a user can follow

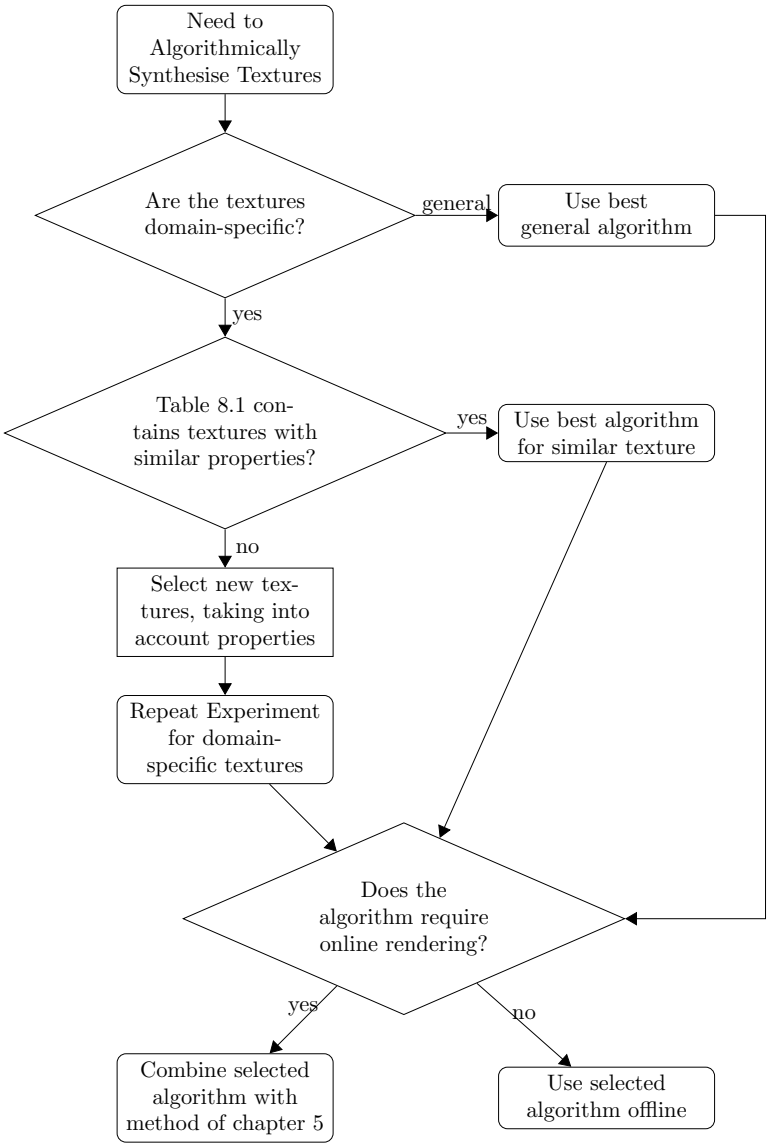


Figure 8.17: Flowchart for application-specific texture synthesis algorithm selection

for improving their required texture quality. As shown in the flowchart in figure 8.17, this process can assist the user to select the texture synthesis algorithms best suited for their specific application. The process is as follows.

For any application requiring Example-based Texture Synthesis, consider first whether the textures it will be applied to are domain specific, or general. This may require the consideration of texture properties, and assessment of whether the full range is covered in the application. For example, an in-game terrain synthesis application may require only natural and stochastic textures, while an inpainting application may require processing of general textures. In the general case, the optimal algorithm to use is self-tuning [Kaspar et al., 2015], as per bottom row of table 8.1.

If the application is deemed to require specific types of textures, their properties are considered. If one or more textures with the same properties are identified among the 12 with which the user study was performed, their resulting rankings are to be selected. For instance, if an algorithm to generate wood textures is required, it will correspond to natural textures of colour, with high scale, high stochasticity, and high variance. As per tables 7.4 and 7.5, the green marble and orange marble textures satisfy the same properties. As per table 8.1, the best algorithms for these may be selected for this application: self-tuning [Jamriška et al., 2015], LazyFluids [Jamriška et al., 2015], or resynthesizer [Harrison, 2005].

If the user study contains no textures with the same properties as the desired application, an informed selection of the appropriate algorithm will require execution of a new subjective study. Thanks to the availability of the code for all methods, as well as the experimental setup for the study, this does not require any re-implementation.

Once the offline texture synthesis algorithm has been selected, the application may or may not require fast online rendering. If it does, combine the selected algorithm with Repeatable Texture Sampling with Interchangeable Patches from chapter 5. If not, process with selected algorithm offline.

This process enables the integration of results published in this work with practical industrial applications of texture synthesis algorithms.

8.7 Conclusion

We have qualified existing texture synthesis algorithms by means of a subjective experiment, demonstrating that the problem of example-based texture synthesis is not yet fully solved. It appears there are types of textures and texture properties where existing algorithms fail; although certain textures are found to be very well synthesized. Previously published findings have been thoroughly validated, and new findings regarding properties of textures which are not well synthesized have been made.

The goal of the study, the identification of the highest quality texture synthesis method, has been achieved. Therefore, the method can now be incorporated into the fast online texture sampling algorithm of chapter 5. By combining these findings, a fast and high quality method can now be produced.

Only three of these textures show significant flaws with state-of-the-art methods, allowing for compact and straightforward analysis. While it is our hope that by performing well on these an algorithm should perform well on any new textures, it is likely that this set will need to be extended once the texture synthesis algorithms perform well on all of them. The code of our experimental set-up is made available with this publication, to encourage such future extensions.

A shortcoming of our final analysis is that it assumes equal weight across the selected textures, but in reality this weighting is application-specific. We have attempted to allow application-specific analysis by showing rankings for each texture, but the overall ranking will not be representative of all applications.

In future work, it would be beneficial to define appropriate metrics for texture quality assessment. Furthermore, as the raw experiment data is being made available with this publication, future analysis can show more complex relationship across participants. It would be interesting to quantify properties for clusters of participants, and infer the specific criteria that affect their preferences. While algorithms exist which will work well on a single texture of this set, this is not a general solution to the texture synthesis problem.

By identifying properties of textures which have consistently not been synthesized as well as the reference, we hope to help advance research in the field of example-

based texture synthesis, so that algorithm may be devised which handles any texture.

The work in this chapter has been published at Eurographics [Kolar et al., 2017], Europe's most prestigious conference in computer graphics.

Chapter 9

Conclusion and Future Work

The goal of work performed in the course of this doctorate is the creation of a comprehensive approach for the analysis of quality of texture synthesis algorithms, and the development of a method which improves the state-of-the-art with respect to this approach. This goal has been accomplished, along with additional findings along the way.

The research presented in this thesis has investigated in detail the quality of offline texture synthesis, and introduced a new method for texture synthesis and made advances in quality assessment. The following contributions have been made: The parallel algorithm of chapter 5, the texture dataset of chapter 7, and the user study of chapter 8. This leads to several conclusions regarding the research question posed in the the introduction:

First, a new texture synthesis method has been presented with improved quality for fast, parallel online synthesis. The quality of this algorithm depends only on a preprocessing step, which can be performed with any method. The new method, presented in chapter 5 (figure 9.1) achieves the benefits of other parallel texture synthesis algorithms, namely speed in a real-time environment, and a small memory footprint. Because sampling is independent for each pixel, the method runs in parallel, works for arbitrarily large textures, and the deterministic nature of the algorithm allows repeatability, which is necessary for certain applications, such as tile manufacture.

The resulting quality of the new method depends only on the quality of patches produced in an offline preprocessing step, thus making it possible for the other advances

in texture quality in offline algorithms to be utilised on-the-fly as well. The benefits of this parallel patch-replacement method over previous parallel patch-replacement [Vanhoey et al., 2013] are two-fold: precomputed patches are not limited to lie on a precomputed fixed patch map, and rendered patches can lie over the boundaries of other precomputed patches. This makes the method presented in this thesis suitable even for complex irregular textures, and enhances the flexibility of fitting pre-processed textures. In conclusion, this demonstrates that online texture synthesis can achieve the quality of any offline texture synthesis algorithm.

Furthermore, the preservation of quality has been verified through a randomized user study, where statistically significant results demonstrate that no significant loss of quality occurs when a high-quality offline algorithm is converted into a fast online rendering algorithm with the use of this method. This new method allows the integration of a non-real-time high quality texture synthesis method, thus allowing real-time speed and high quality.

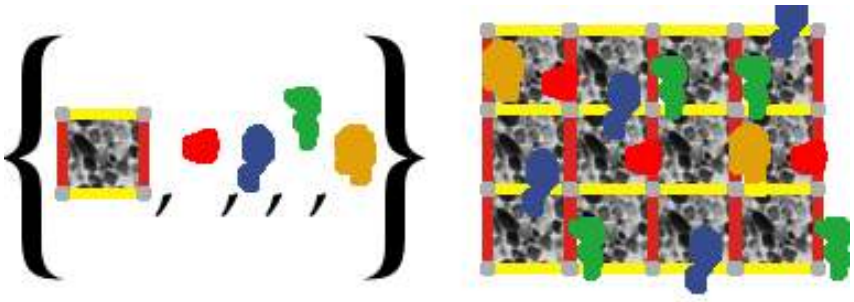


Figure 9.1: The pebbles texture rendered using the algorithm of chapter 5

Secondly, known properties of textures are compiled into a logical classification in chapter 7. The texture properties assessed here are a structured reorganisation of previously published lists of properties. Whereas evaluation previously only relied on a set of textures satisfying the stationarity and locality constraints, no fixed set was used to compare results with each other. In order to offer guarantees regarding quality over new textures, this wide array of ordinal and binary texture properties has been compiled. These properties are used to create a small set of 12 textures which exhibit these properties. Such an approach allows texture synthesis algorithms to be visually compared with each other in a standardised way.

Figure 9.2: Synthesis Results of Self-Tuning Texture Optimization [Kaspar et al., 2015] over the dataset. Top row contains the examples, and the bottom row the synthesis results.

Thirdly, in chapter 8 these textures are compared in a user study over six texture synthesis algorithms, to produce relative rankings and absolute comparisons with a reference texture. This experiment has controlled over personal preferences, over screen configurations, and over the individual definition of synthesised texture quality, in order to produce significant results across multiple applications. Overall, the study showed that people prefer the quality of the reference over any algorithm, conclusively demonstrating that none of the available algorithms perform perfectly. Nonetheless, when ordering algorithms in relation to each other, Self-Tuning Texture Optimization [Kaspar et al., 2015] is objectively the best available method (figure 9.2). Such a method is an appropriate choice for use in future work and integration with parallel online synthesis.

Finally, as set out in the research methodology, these findings are to be joined. By developing a quality-preserving real-time texture synthesis method, and identifying the highest-quality texture synthesis algorithm, an offline and online high quality texture synthesis algorithm has been created. This joint method satisfies the original requirements set at the beginning.

9.1 Limitations

The parallel texture synthesis algorithm of chapter 5 has limitations regarding the number of repeated pixels in each patch. It also requires replication of corner pixels, which

cannot be overlapped.

Further research with the chosen textures of chapter 7 will inevitably lead to algorithms which perform increasingly better on textures with the listed properties. As the quality over all selected textures becomes indistinguishable from the reference, it will be necessary to extend the selection of textures, because new properties will be found for which improved texture synthesis algorithms are yet to produce results of sufficient quality. For example, irregular textures with non-repeating patch maps and textures where patch boundaries are crossed by other patches of figure 5.5 may prove challenging candidates.

Because all measurements were taken on computer screens, the user study of chapter 8 does not take into account user's preferences regarding various environments. It may be beneficial to perform a similar study with printed textures, with ceramic tiles, or inside virtual environments. Similarly, certain applications may require specific types of textures, and additional properties not discussed here, such as embossing. Because the focus is different, these must be evaluated separately.

The key contributions of the thesis are thus: enhancing the quality of parallel texture synthesis, creating a dataset for evaluation, performing an objective perceived quality study, and interlinking the findings.

9.2 Applications and Impact

This work was funded by a CASE studentship with Johnson Tiles. This section describes the importance of the work undertaken in this thesis for the tile industry. Emerging challenges are underpinned below, four focusing on further applications in Tile Manufacturing, and the others on future work in texture synthesis research.

9.2.1 Future Research Questions

The results presented in this thesis clearly show which texture properties fail to be well synthesised in existing texture synthesis algorithms. In particular, textures which exhibit mid-to-high shape regularity without a loose lattice, those with mid-to-low stochasticity, and those with low locality are not synthesised well by any available algorithm. Further-

more, complex shape patterns and regular texture elements which are not replicated perfectly create clearly visible flaws. These results contribute to the field of texture synthesis by reducing the need for subjective evaluation in the development of new algorithms.

The work performed in this thesis has opened new directions for research:

1. Making use of the fact that the quality of online texture synthesis can be as good as offline texture synthesis.

2. Find new textures and texture properties on which relative and absolute algorithm quality can be evaluated meaningfully.

3. Although one algorithm performs very well, texture synthesis remains an open problem, because not all results are indistinguishable from the reference.

4. Future work in texture synthesis should focus on shape regularity without a loose lattice, stochasticity, locality, and regular textures.

9.2.2 Texture Synthesis for Tile Manufacturing

In the ceramic industry, texture synthesis is used to create tiles that are visually similar to an example of a natural material, but crucially minimise repetitions in the texture that may be immediately obvious on a wall of tiles. The current industrial practice is labour intensive.

The results of this work are directly applicable in the Texture Synthesis processes used in Tile Manufacturing, especially in design and printing.

9.2.3 Perceived Satisfaction on Tiles

The quality of synthesised textures in the user study was assessed in a general sense, and on computer screens. Although the results are applicable in Tile Manufacturing, there may be specific factors affecting quality preference for the types of textures used and the ceramic tiles that they are printed on.

Results of the user study have also shown that multiple methods are indistinguishable from the reference on the two marble textures in the dataset, suggesting that

for this type of texture, the problem of texture synthesis is solved. These results are directly applicable, given assumptions that quality is perceived similarly on screens and printed tiles, and that various marble textures behave similarly when synthesised. In order to verify these findings and apply them in tile manufacturing, a similar study should be performed to assess quality on printed tiles with the types of textures which are typically printed on ceramic tiles.

9.2.4 Integration of Tile Manufacturing Processes

The desired qualities of Texture Synthesis algorithms are speed, quality, repeatability, resolution, and it is important to find an appropriate tradeoff of these to suit the application at hand. Since for the ceramic tile industry quality is key for a subjective measure, any evaluation needs to be done with a carefully selected dataset. In order to bridge the gap between subjective user satisfaction and repeatable objective metrics, analysis of computed results and answers in the user study will be conducted to compare correlations between computed and perceived quality.

In order to integrate the results of an objectively pleasing Texture Synthesis algorithm into the tile manufacturing process, several steps need to be taken.

- The profiling will need to be automated by taking into account a number of factors
- The RGB texture will need to be represented optimally using CMYK colours, to minimise printing costs
- The numerous variables involved in the printing process will need to be measured on-the-fly and accounted for during the manufacturing process, in order to produce an accurate mapping from the CMYK representation to finished tile
- This changing mapping will then be applied to tiles being printed, in order to reduce variation between the desired and the finished tile

The texture synthesis method and user study results have already made significant contributions to the creation of texture canvasses by Johnson Tiles. Future work will integrate the profiling step, colour representation, the printing process, and the quality assurance measurements.

9.2.5 Android App

In order to make the texture synthesis algorithm easily usable to Johnson Tiles, a smartphone app utilising these findings has been delivered. It allows designers and engineers working with textures for ceramic tiles to test synthesis with novel textures, wherever they will see them. By allowing a demonstration of the final synthesised texture at tradeshows and in the wild, new texture patterns can be found and created.

As well as allowing the capture and use of new textures, a mobile application demonstrating texture synthesis allows the quick evaluation of the usability of a given texture. Certain textures cannot be replicated well, and others display necessary properties to be rendered well, but may prove inappropriate for manufacture because of the large-scale appearance of the finished product, which is difficult to assess in advance. The application will allow an expert texture designer to assess this quickly for any new texture.

Finally, while making work related to texture synthesis easier and more flexible, the application should be economically useful for Johnson Tiles by allowing them to capture and assess textures without acquiring physical slabs of marble, and dealing with additional expenses of international transport and flatbed scanning. Objective metrics of quality can be extrapolated if any of the evaluated methods are used to create the textures in the mobile application, therefore assuring user satisfaction and quality in the synthesised texture.

9.2.6 Parallel Synthesis Preprocessing Quality

The presented parallel texture synthesis method allows textures of arbitrary quality in theory, but uses image quilting for preprocessing in practice. By using the best available algorithm for patch and tile preprocessing, the quality of the online rendering will be vastly improved.

Furthermore, a known limitation of the method is that corner pixels cannot be overlaid with patches, because of the non-deterministic overlap which would arise. Furthermore, only 50% of each repeating tile may be replaced with patches. By combining this method with underlying Wang tiles, rather than a single repeating S-tile, these issues

would be efficiently resolved.

9.2.7 Extending the Texture Dataset

This work covers many texture properties, and meaningful combinations of these properties. However, new properties may be explored, and new types of textures may be discovered. Properties may include roughness, embossing, and textures whose texture elements do not lie on a loose lattice, and are therefore non-regular. Furthermore, textures where patch boundaries are crossed require manual intervention to synthesise well, and this will be interesting to explore in future work.

Furthermore, only three of the selected textures are conclusively synthesised with poor quality. By extending this with an additional set of textures for which most algorithms do not perform well, a more precise understanding of new avenues for improvement may be defined.

9.2.8 Generative Quality Metric

Because results from the user study have demonstrated interportability of quality perception from synthetic to natural textures, it may be possible to develop a quality metric using a generatively rendered texture. A texture may be algorithmically created, and the properties of any synthesised texture can be tracked, measured, and compared the its properties. Given such a process, it may be used to evaluate texture synthesis algorithm quality.

Future work may use measurable properties of textures, and by creating appropriate generative processes, input textures may be automatically generated and synthesised textures may be automatically evaluated. Findings regarding perceived quality and usability of specific algorithms across various texture properties are applicable here. Given a new texture which matches existing texture properties for which the objective user study has been performed, the same results in user satisfaction can be accomplished.

9.2.9 Summary of Emerging Challenges

Future work related to delivering additional facilities to Johnson Tiles, the CASE studentship sponsor of this PhD, has been discussed. A wide array of engineering and scientific problems has been opened with this research. Future work includes Texture Synthesis algorithms in Ceramic Tile Manufacturing, Perceived Satisfaction on Tiles, the optimisation of the entire manufacturing process, and the development of an Android App. Additional future work in texture synthesis research includes the improvement of the preprocessing step of the proposed algorithm, and the design of a generative quality metric.

9.3 Final Remarks

Example-based Texture Synthesis is a complex problem with numerous applications. The perceived quality of known methods, which was hypothesised to be insufficient, was proven to be significantly and repeatably inferior to a ground truth reference, as shown in the bottom row of table 8.1. An objective method optimisation scheme was assessed for quality as well, and a need for a user-in-the-loop subjective evaluation method was shown to be necessary.

This work has improved the quality of offline and online texture synthesis algorithms in three steps: The creation of a parallel algorithm whose quality is the same as the quality of an algorithm used for pre-processing, the creation of a dataset of textures to make quality evaluations possible, and the execution of an anonymous user-study to draw conclusions on quality. By combining these findings and assessing requirement trade-offs for various applications, a workflow was proposed to deliver a general texture synthesis method, as well as application-specific solutions with respect to texture-specific quality requirements.

These three contributions have fulfilled the objectives set out in the research methodology well as a multitude of possible Future Work in research and in industry. In this work, High Quality Texture Synthesis has been improved upon in a rigorous, structured way. This thesis has contributed to research in the field of texture synthesis, pro-

viding a firm foundation for future work.

Appendix A

Ethical Proposal

A Ranking Comparison Based Subjective Evaluation of Example-based Texture Synthesis Methods[1]

Martin Kolář, Kurt Debattista and Alan Chalmers

WMG, University of Warwick

A.1 Summary

This document describes a proposal for subjective evaluation of leading Texture Synthesis methods via two evaluation techniques viz. ranking and pairwise comparison. The evaluation involves human participants who are asked to compare and rank on computer displays, the quality of a synthesised canvas, random tiles, and a tiled wall using one of 7 published methods against a displayed input sample. In the following sections, Example-based Texture Synthesis methods will be referred to as SMs (Synthesis Methods). The experiment will involve different natural textures (inputs), and 7 different synthesised textures for each (outputs). The benefit of this experiment will be twofold: to determine observed quality of Texture Synthesis methods, and to validate a computed quality metric which seeks to mimic human texture perception.

[1] edited for formatting

A.1.1 Related Work - Evaluation

Related psychophysical evaluations involving ranking and pairwise comparison have been conducted to compare several Tone-Mapping operators (TMO) with each other and/or with reference HDR frames using a combination of LDR and HDR displays. Moreover, such comprehensive subjective evaluation of SMs, to the best of our knowledge, has never been conducted to date. This proposal will undertake a comprehensive psychophysical evaluation involving human participants who are tasked to determine the position on a scale of each SM from best to worst by means of ranking method according to their preference. Doing so, will help identify the best SM according to the ranking criteria, the ranked order of preference for each SM, and finally provide a better understanding of the evaluation of SMs; more so if the data between the objective and subjective evaluation correlate to a large degree. Data will also be used to validate if the results of our quality metric correspond to human preferences.

A.2 Aims/Objectives

The objectives of these experiments are: 1. Identify the best SM out of the ones proposed to date. 2. Quantify perceived expectations of texture synthesis by comparing how favourably a second natural texture compares to various algorithms. 3. Find and quantify the correlation between perceived and computer quality, in order to allow fully automated tests in the future, and the incorporation into our own method.

A.3 Design/Methodology

We propose to conduct a psychophysical evaluation by ranking, in three different experiments. The experiments are performed identically, but against three different sets of images: A large canvas, A single tile, and a tiled wall. The second and third experiments will only be performed with textures desirable for tile manufacture, and will be analysed separately. The first experiment, however, will be of crucial importance to the broader Texture Synthesis community, and will be executed with multiple textures. The ranking test itself will be the same for all three experiments, so they are now described together

in detail.

A.3.1 Ranking-based methodology

Each experiment will be performed for textures in Figure 6 independently. Participants will be shown the source texture, and all the other textures simultaneously. They will also have the possibility of clicking on any texture to enlarge it over a larger portion of the screen, and clicking a shortcut will activate the magnifying glass tool to examine details. They will then be asked to rearrange the synthesised textures on their screen, with the mouse, in their perceived order of quality. The preferred order can be analysed to identify significant statistical differences amongst the SMs. Synthesised textures will be resized to 1/8th of the monitor, so that all can fit on the screen simultaneously, and will be displayed in high resolution, to allow the participants to focus on fine details in the textures.

A.3.2 General Texture Synthesis experiment

In order to assess the method over the wide variety of synthesis methods, it is important to use a wide variety of textures, including textures on which current algorithms do not perform well. It is of most benefit to quantify what drawbacks each method has, and this will be done by assessing the ranking performance of each method over the various textures. These textures have been selected to account for numerous types of variation: input size, texel size, orientability, patch 4interchangeability, colour, large-scale variance, and size of repeatable elements. Therefore, textures in Figure 6 have been chosen. Some are well known in the literature, while others have been specifically chosen to address potential shortcomings of today's methods.

This part of the experiment will only have the participants evaluating the large canvas, in contrast with the experiment in the following subsection.

The experiments will be done only using a single calibrated screen (Samsung U28D590D 28 inch Ultra HDMI 4K), a computer, and a USB stick to store all data. The use of a specific screen is important, and to eliminate external stimuli, the experiments will be performed in a closed dark room. Certain textures will be provided by our partner

Johnson Tiles. Chosen textures will be shown for as long as each participant wishes, with a recommended length of 30 to 90 seconds. It is to be noted that care has been taken to not include any material which might be objectionable to the participants. A visual description of some of the chosen textures used for the experiments is given below in Figure 6. Other textures, chosen from the near-regular, irregular, and near-stochastic categories of Figure 1 will be selected, because these reflect best where texture synthesis is important, and may underperform.

A.3.3 Participants

The number of participants is defined by sample size 's'. Typical sample sizes for comparison of images studies in the past have been in the region of 30 to 100 participants, and we will strive to gather 5 staff, 10 graduate students, and 85 undergraduate students to perform the study. The ideal sample size should be large enough such that the ranking distribution is big. The advantage of a large sample size is that while analysing the results a high degree of agreement can be measured through Kendall's coefficient of consistency and other non-parametric tests such as Friedman's test or analysis of variance (ANOVA). The age criteria for participant eligibility is 18 years and above and all participants should have normal or corrected to normal vision. In an ideal scenario, the distribution between males and females should be equal and from a wide ranging demographic distribution.

A.3.4 Recruitment strategy

The participants will be selected from either University of Warwick or our industrial partners Johnson Tiles. In case of University of Warwick, the primary recruitment strategy will be through formal email sent to the research director which is subsequently forwarded to the rest of the dept. In the case of Johnson Tiles, a formal email will be sent to the senior project management to be subsequently forwarded to other employees. If a participant is interested in taking part, he/she will be provided the participant information leaflet and the basics of the experiment will be discussed as described. Upon his/her approval, the participant will be requested to sign the consent form following which the experiment will start. In no situation will any personal data of the partici-

pant viz. name, phone number and email address be collected as the data will remain completely anonymous.

A.3.5 Analysis

Results can be analysed in a number of ways. Ranking data will be analysed directly over all textures and SMs using ANOVA (or a non-parametric equivalent such as Friedman's test; ANOVA is preferred if it is possible as it has more power). Pairwise comparisons will also be used to identify the differences between each of the SMs; this will be done per SCN and over all HDRVs. Kendall's co-efficient of consistency will be used to provide a value of agreement amongst the participants.

A.4 Ethical Considerations

The experiment will be conducted ensuring all ethical considerations are adhered to including the right to withdraw at any moment during the experiment. The results of the participant who withdraws from the test will not used in any way.

A.4.1 Informed consent

Informed consent will be acquired from each participant before the start of the experiment by means of a thorough briefing including a short training session and a consent letter in addition to participant information leaflet to each participant in which a description of the experiment will be provided. The participant will be debriefed after the completion of the experiment and have the opportunity to ask any further questions.

A.4.2 Participant confidentiality

The experiment has been designed in such a way that the data collection will be completely anonymous and all forms of bias will be eliminated. The evaluation does not require any case studies or personal interviews. Therefore, the data collected will consist of only the ranks of different SMs assigned to the synthesised textures, and the times that the participants took to make their final decision.

A.4.3 Data security

Since the experiment does not include any case studies or personal interviews, the data collected will not be sensitive. Additionally, care will be taken such that the collected data does not serve any other interests except for further research in this field. The collected data will be retained till January 2020 in an encrypted optical disc in a locked filing cabinet, following which it will be destroyed. The data will only be accessible to authorised personnel i.e. the student conducting the experiment and the two research supervisors.

A.5 Other considerations

A.5.1 Right of withdrawal

The right to withdrawal has been previously discussed in section A.4. No data is stored if the participant decides to withdraw from the experiment midway. In case the participant decides to opt out of the experiment after the completion of the test, the stored data will be deleted instantly without further analysis.

A.5.2 Process of sensitive disclosures

Not applicable

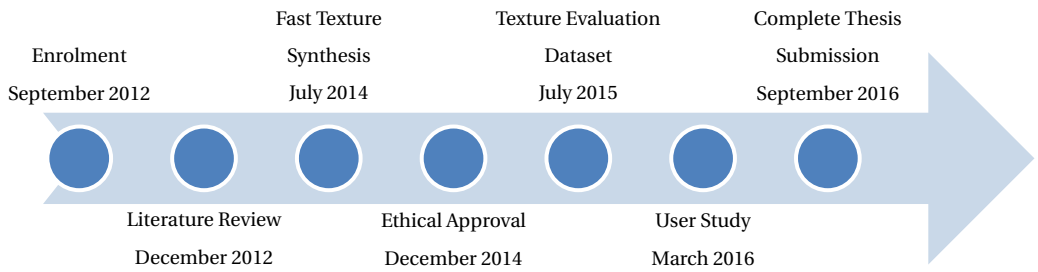
A.5.3 Benefits and risks

The psychophysical experiment will be of great benefit to the research field of example-based Texture Synthesis. As various algorithms perform well on different textures, it is crucial to identify which ones produce the best results, and validate an automatic metric for future use. Therefore, they need to be evaluated on a wide spectrum of potentially desired textures, as well as a wide range of synthesis algorithms.

Appendix B

Timeline

This work has been created over the course of 4 years, as follows:



Appendix C

Publications

Vis Comput (2016) 32:1263–1272
DOI 10.1007/s00371-015-1161-4

ORIGINAL ARTICLE

Repeatable texture sampling with interchangeable patches

Martin Kolář[1] · **Alan Chalmers**[1] · **Kurt Debattista**[1]

Published online: 24 October 2015

Abstract Rendering textures in real-time environments is a key task in computer graphics. This paper presents a new parallel patch-based method which allows repeatable sampling without cache, and does not create visual repetitions. Interchangeable patches of arbitrary shape are prepared in a preprocessing step, such that patches may lie over the boundary of other patches in a repeating tile. This compresses the example texture into an infinite texture map with small memory requirements, suitable for GPU and ray-tracing applications. The quality of textures rendered with this method can be tuned in the offline preprocessing step, and they can then be rendered in times comparable to Wang tiles. Experimental results demonstrate combined benefits in speed, memory requirements, and quality of randomisation when compared to previous methods.

Keywords Texture synthesis · Texture mapping · Parallel rendering · Ray tracing

1 Introduction

Texture synthesis is a core process in computer graphics and design. It is used extensively in a wide range of applications, including computer games, virtual environments, manufacturing, and rendering. Crucial points on which current methods compete are perceived texture quality, rendering speed, scale considerations, and memory requirements.

In order to allow rendering with ray tracing, and to render textures in real time, current methods need to run in parallel and on a GPU, without hindering the perceived quality (Fig. 1).

As shown in Fig. 2, our method addresses the main requirements for example-based parallel texture synthesis algorithms. This paper introduces a method which allows parallel texture synthesis with patches of arbitrary shape, without the necessity of a fixed repeating patch boundary. In previous work [17], selection of interchangeable patches at runtime was also possible, but required a repeating fixed patch boundary. As discussed later, the method presented here has similar preprocessing complexity, memory consumption, and rendering speed, but allows a wider class of interchange types with higher variability, resulting in a higher quality texture.

Our method allows the sampling of any pixel in the output texture with a deterministic algorithm, without requiring any information from the pixels that have already been synthesised. Therefore, adjacent pixels can be synthesised in parallel in separate threads, which do not communicate.

Large textures are rendered by randomly selecting subsets of prepared patches in a parallel and repeatable manner. These precomputed patches are stored efficiently, allowing seamless integration in GPU, and the texture is rendered independently for each pixel on-the-fly, allowing repeatable parallel access to an infinite, non-periodic texture, appropriate for ray-tracing applications.

The paper is structured as follows: Sect. 2 discusses previous work, and how our method improves on the state of the art in parallel texture synthesis. Section 3 outlines the high-level concept behind the method, and Sect. 4 goes through how these points are implemented. Method outputs, their

✉ Martin Kolář
m.kolar@warwick.ac.uk

Alan Chalmers
alan.chalmers@warwick.ac.uk

Kurt Debattista
k.debattista@warwick.ac.uk

[1] WMG, University of Warwick, Coventry, UK

Fig. 1 Output textures from our algorithm display white noise properties, without using cache (*images* are linearly transformed without interpolation to vanish at the *horizon*, results without offline manual tuning)

comparison to other work, and other results are presented in Sect. 5, and future work and the conclusion are in Sect. 6.

2 Previous work

Sequential exemplar-based texture synthesis falls into one of three classes [19]: pixel-based methods, patch-based methods, and texture-optimisation methods. Parallel texture synthesis can be divided into an additional three: dependency-tree methods, constant-time tiling methods, and non-constant-time tiling methods.

Sequential pixel-based methods [3] consider each output pixel in sequence, while sequential patch-based methods [2] replicate entire patches, optimising seams using an algorithm such as Graph Cut [9]. Instead of performing the process

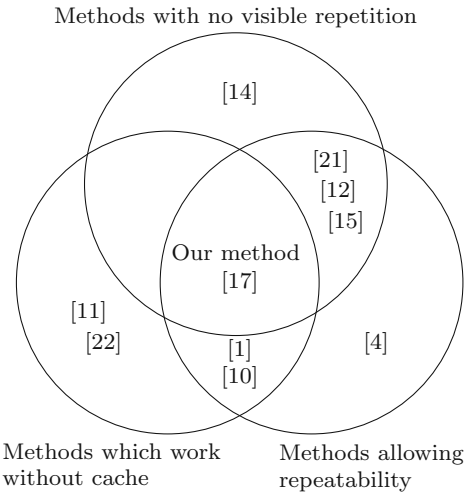


Fig. 2 Venn diagram of the trade-offs between current example-based parallel texture synthesis algorithms

once, patches can be placed iteratively over the output until the desired quality is achieved [8,21]. However, sequential algorithms are not suitable for simultaneous synthesis of disjoint regions, because the space between them needs to be synthesised as well.

Where the entire texture cannot be held in memory, but needs to be generated on-the-fly, parallel texture synthesis methods can be used. The naïve approach to reduce rendering time is to create a repeating tile from an input exemplar, such that the edges fit [20]. This causes visibly noticeable "tiling" effects. Tile-based runtime synthesis relies on offline-preparation contents of a texture map, which are then placed on a rendered surface. Such placement schemes can be done in a number of different ways using a rectangular grid: Ammann tiles [6], Wang tiles [1], stochastic tiles [18], s-tiles [22], and coloured corners [10]. Triangular [13] grids have also been used. These approaches make the output pixel retrieval a constant-time operation for output textures of arbitrary size. However, they create visible repeated edges and grid patterns when zoomed out, as shown in Fig. 3.

In turn, this visible aberration is addressed by non-rectangular region copying, such as megatexture [14], virtual texturing [4,16], and patch-based methods [15]. However, these methods rely on cached information, which can cause temporal artefacts when a scene is re-rendered in a different order. To allow a different part of the scene to be rendered elsewhere, in the next video frame, or to be able to revisit a texture in a virtual environment, it is desirable to guarantee that a texture rendered again from the same compact seed will be identical to one rendered previously. This quality is referred to as "repeatability". (Not to be confused with "repetition", which is generally undesirable in synthesised textures.)

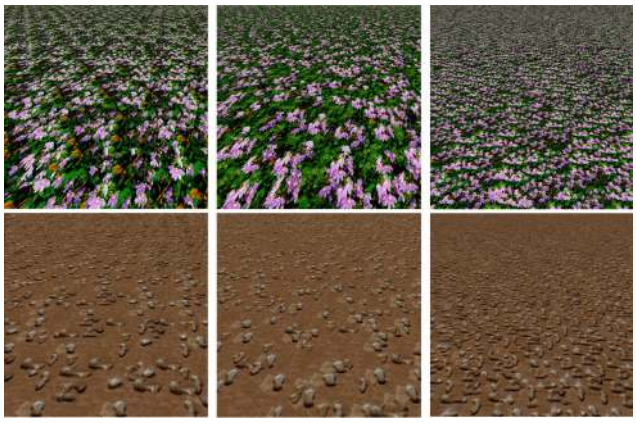


Fig. 3 Comparison with Wang tiles using two different borders (16 tiles) [1] (*left*), [17] (*middle*) and our results (*right*). Sampled linearly. *Top* exemplar is 268 × 230, *bottom* 512 × 512

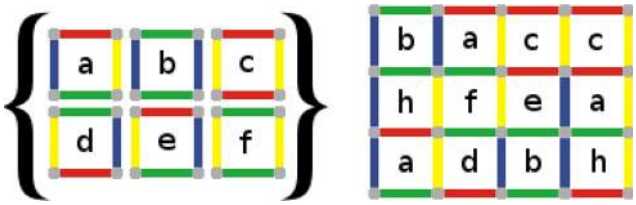


Fig. 4 Precomputed Wang tiles *a* to *f* are represented on the *left*, and a synthesised image is on the *right*

Parallel non-constant-time patch replacement methods [5,15,17] perform a non-constant overhead operation while rendering, to address grid artefacts. These methods place patches of precomputed shape on the texture at runtime, according to a runtime computation, but require a fixed patch map whose boundaries cannot be overlaid with a patch. For example, in the offline step of [17], a fixed repeating patch map is created, along with various interchangeable patches which fit the patch map boundaries (Fig. 5). The texture can then be sampled independently online with a pseudo-random number generator at each repeating patch map. However, none of these resolve the local adjacency constraint for patches overlapping repeatable tile boundaries.

Methods which use a statistical shape model for the texture [5] are able to outperform rendering speeds and quality of exemplar-based parallel synthesis, at the expense of relying on additional user input to model the texture. As this is no longer automated example-based texture synthesis, such methods are not included in Fig. 2.

A visual comparison of methods which precompute tiles and select placement during rendering is shown for: Wang tiles (Fig. 4), fixed map patches of [17] (Fig. 5), and patches without map boundaries presented here (Fig. 6). In each figure, the precomputed set of patches or shapes is on the left, with colours corresponding to places of interchangeability.

This paper describes how interchangeable patches can be applied online to repeating tiles without a fixed patch map,

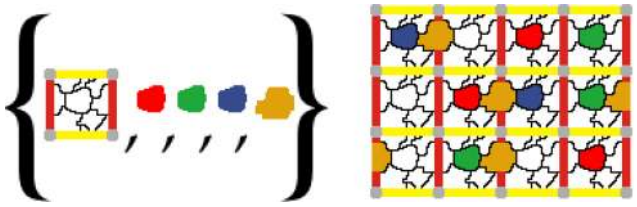


Fig. 5 Method of Vanhoey et al. [17]. Precomputed fixed patch map and patches for content exchange on the *left*, and a synthesised image on the *right*

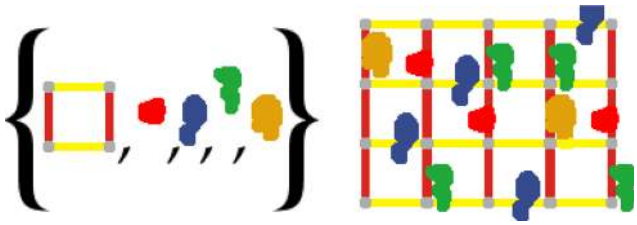


Fig. 6 Our method. The precomputed tile and interchangeable patches are on the *left*, and a synthesised image is on the *right*

and without posing constraints on boundaries and adjacencies. A discussion of the benefits of this approach is in the results section.

While offering this enhancement over the state of the art, our method maintains the benefits of parallel non-constant-time tiling: texture quality depends only on the quality of a preprocessing step and available GPU space. Memory and load on a GPU are addressed to show that even complex textures can fit into limited GPU memory, and the non-constant runtime overhead is only a light logic operation.

3 Patch-based texture synthesis without spatial dependency

The algorithm described in this paper is divided into two steps: preprocessing and rendering, see Fig. 7.

The method starts with a simple repeatable tile created from the exemplar texture. Next, interchangeable patches of varying size are precomputed on the repeatable tile. These can reach over the repeatable tile boundary, so they are created such that they form two sets, which mutually do not overlap. During rendering, this non-overlapping criterion permits parallel rendering, while guaranteeing local adjacency because "active" patches cannot overlap.

The tile is made larger than the largest visual repeating element of the texture in the exemplar. For example, in the left image of Fig. 1, this corresponds to the number of pixels spanned by one apple. By using interchangeable patches of various sizes (from a few pixels to a large portion of the tile), the synthesised texture will contain elements on multiple scales.

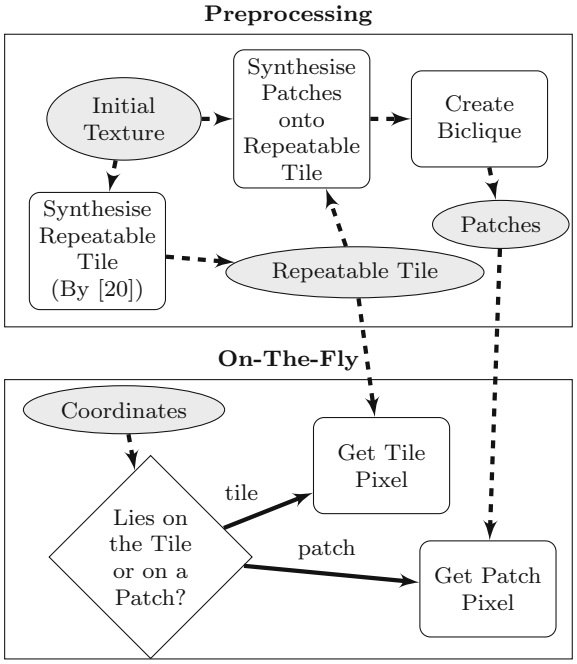


Fig. 7 Flowchart overview of our method

The patches are saved, and at runtime are chosen in each tile in a random, repeatable process, without any cached information. A binary map of the patch allows constant-time retrieval of pixel values.

Preprocessing guarantees that patches of adjacent tiles do not overlap, and the online selection guarantees that patches chosen within each specific tile are selected so that they do not overlap. Because of these constraints, every sampled pixel is copied from one of two regions: the repeatable tile, or a selected patch of this tile or the neighbouring tile. At runtime, pixel lookup is performed based on a simple logical operation which makes this decision, and the selected pixel is retrieved from the input texture.

4 Algorithm implementation

The algorithm is composed of an offline preprocessing step which creates the texture map representation, and an online on-the-fly algorithm called for each requested pixel coordinate (see Fig. 7). The texture map consists of a repeatable tile, and patches divided into two sets with offset vectors on their specific tile (but not locations in the final output texture) (see Fig. 6).

4.1 Preprocessing

From an input texture, we create

– a repeatable tile
– difference vectors for each patch, denoting its location in the base texture, and in the repeatable tile
– a 2D binary array containing the shape of each interchangeable patch
– a binary matrix for each of two sets of patches with the information whether any given pair overlaps.

First, using Image Quilting [2], we synthesise the "repeatable tile" from the exemplar (Fig. 8).

The input texture is used as the patch source. If this input texture does not fit into GPU memory, it may be desirable to render a smaller base texture from which patches will be copied, by [20]. Patches are not stored explicitly, but are indexed from this base texture using the difference vector. Therefore, each pixel of a patch takes 1 bit of memory, instead of a minimum 3 bytes for a naïve RGB pixel representation.

Next, we generate candidate patches by associating random pixel locations between the repeatable tile and the base texture, and by executing GridCut [7] to find the optimal cut. Patches of various sizes are generated by weighting the cuts by a Gaussian bell curve of varying width. The cut is allowed to overflow over borders of the repeatable tile, but not the source tile.

As in previous work, the cut cost is Euclidian distance in CIELab colour space [23] of pixels in the original texture ("base"), and the repeatable tile ("tile"). For each potential patch, we find the maximum pixel cut cost along the boundary, and choose a predefined number of patches ($P = 100$–1000) with smallest maximum cut cost. This removes poorly matched patches.

The following step, involving a rhombus and pseudo-biclique, ensures that when patches are selected in adjacent tiles, they will not interact by potentially overlapping.

The output space is divided into tiles A and B, and there are two sets of precomputed interchangeable patches, one for each. The same repeatable tile is used for A and B, to make the texture map compact, but the patch sets differ, to allow greater variability. These patches can lie on the boundary between A and B, but must be entirely within the rhombus around the region they lie in (Fig. 9). It is important that patches do not overlap in the output texture, because the region simultane-

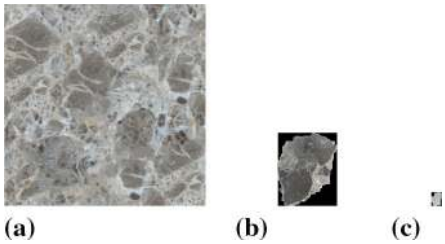


Fig. 8 Sample repeatable tile and patches for a given texture. **a** Repeatable tile (300 × 300 px), **b** a large patch (91 × 112 px), **c** a small patch (15 × 19 px)

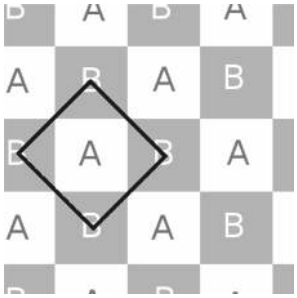


Fig. 9 The output space alignment. *The black rhombus* represents the boundary that patches in A cannot overlap

ously covered by multiple patches is not guaranteed to fit. The patches which lie over the tile boundaries ensure that there is no straight repeating boundary in the output texture, and the bounding rhombus assures independence between patches "active" in adjacent tiles.

The patches are divided into a pseudo-biclique such that all patches in set A never overlap with any patch in set B (Fig. 10). Using the P patches of varying size (Fig. 8), a graph is constructed where each patch is a node and each edge is a "does-not-overlap" relationship (Fig. 10). This division is done heuristically, using Algorithm 1. Note: if edges are made to represent an "overlaps" relationship instead, this pseudo-biclique becomes the union of two disjoint graphs.

input : square binary matrix M, where true at (a, b) represents that patches a and b do not overlap
output: subset of patches, divided into a pseudo-biclique
lists A and B are initialized with the row and column indexes of a random true point in M
while *sum of lengths of A and B* $< P$ **do**
  a := index of randomly selected row of M, such that all intersections with elements in B are true
  if *a is not empty* **then**
    add a to A
  end
  b := index of randomly selected column of M, such that all intersections with elements in A are true
  if *b is not empty* **then**
    add b to B
  end
end

Algorithm 1: pseudo-biclique graph division algorithm

The selected patches are then saved in a binary array, along with the following variables: width, height, top left corner location in the base texture, and top left corner location in the repeatable tile. Square subsets of the matrix of overlaps are saved as well, one for the overlaps among patches in A and a second for B.

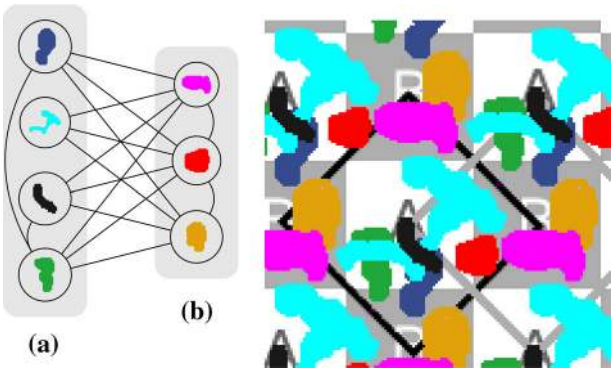


Fig. 10 A pseudo-biclique. Every edge between patches represents a "does not overlap" relationship, and edges between patches in either group are allowed. On the *right* corresponding patch positions are shown

4.2 On-the-fly sampling

Given a single (x, y) coordinate, determine which tile it lies in (t_x, t_y) by rounding to the nearest tile, and its location in the tile (p_x, p_y) by modulo. It is then determined whether the desired pixel is on an A tile or a B tile, by whether $t_x + t_y$ is odd or even.

For each tile type, there is a set of precomputed patches P_A and P_B. For each patch ρ in each set, there is a pseudo-random function $r(a, b)$ which lies in the binary domain. For example, our implementation uses the following function:

$$r_\rho(a, b) = \mathrm{mod}((\alpha_\rho + a + \cos(b))^2 + (\beta_\rho + b + \sin(a))^2, 1) < \eta, \quad (1)$$

where α_ρ and β_ρ are initialisation parameters of the function, specific for each patch ρ. This binary Perlin noise function was chosen because it can be executed efficiently on a CPU and GPU, and it passes the Diehard battery of randomness tests.[1] The parameter $\eta \in [0, 1]$ varies incidence (in our implementation, we set $\eta = 10/|\text{patches}|$.

input : square binary matrix, where true at (a, b) represents that patches a and b overlap
output: subset of patches, such that there is no overlap
while *the matrix contains at least one true* **do**
  find first row with most true values;
  remove this row, and the same column;
end

Algorithm 2: Deterministic creation of non-overlapping patch subset

$r_\rho(t_x, t_y)$ is evaluated for each patch in the appropriate set. For "active" patches, those where $r_\rho(t_x, t_y)$ is true, the pre-

[1] http://stat.fsu.edu/~geo/diehard.html.

computed binary overlap matrix is used to find overlaps between them. Algorithm 2, which is deterministic, is then used to eliminate overlaps. This creates a small non-constant overhead, which is at most linear, but always terminates in very few iterations. Because patches have been divided during offline preprocessing, this operation does not need to consider more than a few potential overlaps, each requiring one clock cycle.

This creates a subset of patches on this tile, called the "active subset". For each, the 2D precomputed binary map of the patch is used to find whether (t_x, t_y) is inside. If the point (t_x, t_y) is inside the patch, the associated pixel is retrieved from the synthesised base texture. This operation is a trivial array lookup in the appropriate saved patch, since patches are not saved as polygons.

If the point is not inside the patch, the nearest edge is found, and the procedure repeated for the adjacent tile. Note that, thanks to the rhombus-shaped boundary for patches overlaying the boundary between A and B, a pixel can only be affected by interchangeable patches from the adjacent tile which is nearest (Fig. 9). If the point is found to be in one of the non-overlapping patches in the adjacent tile, the associated pixel is retrieved from the synthesised base texture.

If the point does not lie in a patch chosen in this tile, and does not lie in a patch chosen in the adjacent tile, we retrieve the pixel from the tile itself.

4.3 Complexity, memory, and quality

The computational complexity of the live sampling is near-constant, thanks to the structure in which precomputed information is stored. The process for each pixel is to find which patches are active, then to find the active subset, and retrieve the pixel. Deciding which patches are active is a constant-time operation, choosing the active patch subset is at most quadratic in the number of patches, and retrieving the relevant pixel is also constant. As with other tiling methods, memory consumption is completely independent of the number of sampled pixels and the size of the output texture.

Memory consumption is determined by parameters of the preprocessing step, allowing fine tuning to best balance the tradeoff between quality and use of available memory. This was determined to be in the range of tens of kilobytes (for simple textures such as Fig. 11a), to 1 MB (for complex textures such as Fig. 11b). If the first texture uses 600 patches of 5×5 to 86×86 (2400 bytes for difference vectors and 47 kB of binary maps) pixels from a texture map of 96×96 (27 kB), and a repeatable tile of 96×96 (27 kB), the texture map totals 103 kB. Note that memory consumption is a factor of four of the texture map, which has been approximately true for all included textures. For larger textures, the total memory footprint will always be determined by these four factors, and each of them can be tuned for the specific application.

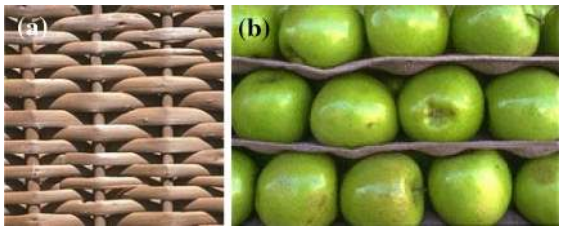


Fig. 11 Input textures. **a** 192×192 px, **b** 185×124 px

The computational complexity of the preprocessing step is comparable to sequential patch-based texture synthesis methods, but the contribution of this work is the texture map compression and on-the-fly synthesis. In practice, the preprocessing can even be done semi-automatically, allowing the user to manually choose patches which are visually satisfactory. The quality of the synthesised texture can be made arbitrarily tuned at the scale of patches, and patches can be chosen to have any size. Therefore, by definition, the runtime algorithm of our method can theoretically synthesise a texture of the same quality as any offline patch-based algorithm, repeatably, in parallel, and as fast as other tiling methods. This only depends on the quality of the preprocessing selection. Note that our implementation and results are of a fully automated algorithm, to allow a fair comparison to results published elsewhere.

On the GPU, pixel values are stored in DRAM and cached in texture memory, and all other variables can be optimised to fit into limited L1 processor shared memory. The SIMD model of the GPU allows each multiprocessor to evaluate Eq. 1 for multiple pixels, and the quadratic selection operation can be performed to the earliest stopping among pixels sharing a multiprocessor. The pixel coordinates in the repeating tile and base texture are returned. Since both images can fit into the texture cache, non-local pixel value retrieval will happen quickly, without reading DRAM memory.

5 Results

Our method produces textures whose quality is not dependent on the runtime computational complexity, but on the quality of the preprocessing step. Therefore, at equal memory footprint, our runtime performance is comparable to simple tiling methods (repeating precomputed tiles, as in [1,6,18,22]), but the texture quality is comparable with patch-based iterative approaches.

In our experiments, precomputing was set to chose the 1000 best patch interchanges found over a 4-h period, comparing tens of millions patch interchanges at different scales. For the textures used here, these settings proved satisfactory. Out of these, 300 patches were used in each tile type, and 400 patches were discarded, as discussed in Sect. 4.1. These

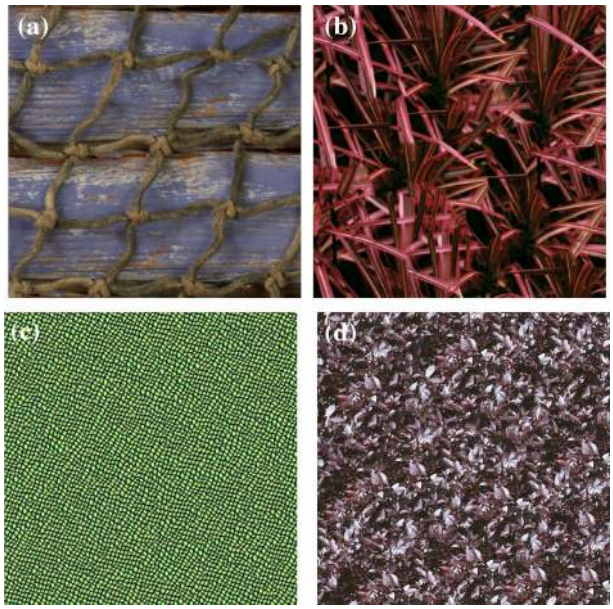


Fig. 12 Synthesis results for irregular textures. **a** High-level properties of textures can be handled by selecting a repeating tile and patches which satisfy the properties. **b** Overlapping can be interactively modelled with our technique by manually appropriately selecting patches during preprocessing. **c** Exemplar size 64 × 64. **d** Exemplar size 512 × 512

amounts have been selected because of memory constraints, because a long search improves the patch quality, and because this many patches provide ample variation in the rendered texture. The upper bound on possible distinct rendered tiles is $300!^2$, but because there are up to 75 % overlaps within each group, this is reduced by a few orders of magnitude.

The speed of the sampling process itself compares favourably with current approaches, despite the overhead to be calculated at every pixel. Speeds are reported on a single core of a 3 GHz Xeon with 667 MHz DDR2 RAM. This overhead is tuned by changing η, which was set to $\eta = 0.1$. Because the time required to calculate a single pixel is constant, the algorithm scales linearly in the number of sampled pixels, and is parallelisable in a straightforward manner with speedup proportional to the number of cores.

Our method has multiple applications: ray tracing, rendering from bundled texture maps, or creating non-repeating patches, such as for ceramic tile-printing.

For irregular textures, synthesis requires handling complex properties, such as layering and overlapping, which are not handled automatically by optimal seam selection algorithms. For these, our method allows manual selection of an appropriate repeating tile and patches. Multiple repeating tiles can be synthesised, and the best one is chosen by an expert. Patches are prepared offline, so they can be shown to an expert user, who determines if they make a believable substitution, and selects the best. Synthesis results in Fig. 12 show how human intervention can improve synthesis quality, while maintaining the storage and runtime speed

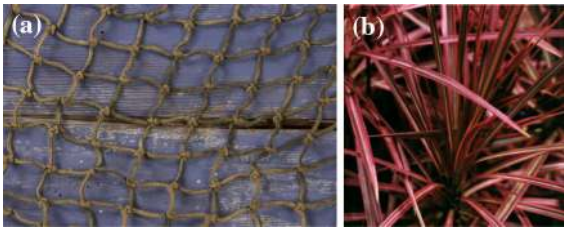


Fig. 13 Irregular textures. **a** Texture with a non-repeating patch map (1024 × 682). See Sect. 5 for a discussion of the properties of such textures. **b** Texture where patch boundaries are crossed by other patches (512 × 512)

advantages of our method. Interestingly, by allowing patches to assume locally optimal shape at numerous scales, interchanged patches often contain visual or semantic features of the example texture.

While [17] works well for regular and stochastic textures, repeating a fixed patch map across an image cannot capture certain irregular textures. Certain irregular textures cannot be faithfully replicated by simply repeating patches of a given shape, no matter what the shape is (Fig. 13). Our method does not restrict patches to replace contents only within precomputed boundaries, instead allowing the boundaries themselves to be replaced by other patches, thanks to a patch biclique division. Section 5 contains a deeper discussion of the limitations for certain irregular textures.

The fishing net in Fig. 12a changes orientation, so patches replacing strings will not align with the wood texture. However, If patch boundaries lie on strings, the underlying wood texture will not align. In Fig. 12b, if patches are chosen to contain parts of leaves rather than leaves, faithful reproduction will not be guaranteed.

Note that all other textures in this paper have been created without manual intervention.

5.1 Independent pixel sampling for ray tracing

In various applications, a crucial property of texture synthesis algorithms is that the error produced when sampling distant pixels will not be constructive, but will have the same properties as randomly selecting pixels. This is referred to as the white noise property, and is particularly desirable with ray tracing; see results in Fig. 3. We benchmarked the algorithm's speed by randomly sampling pixels. Across the different textures we have tested, the time required to sample one pixel did not significantly vary, because the key parameters (number of patches, tile size, patch size) were set similarly for all textures. On the tested platform, the time required to sample 1 pixel averages 60 μs, with little variation. Out of this, 45 μs are required for pseudo-random binary sampling. Testing various random access scenarios for different textures has no effect on the retrieval time.

Fig. 14 Output textures from our algorithm (450 × 800 px)

Fig. 15 Output textures using Image Quilting [2] (450 × 800 px)

Unlike [21], the algorithm does not place a spatial dependency on sampled pixels, so their performance cannot be compared in practice. Any algorithm which places dependency on sampled pixels would rely on cache, making it slower for larger output textures, so that sampling the texture millions of pixels apart would give our method an advantage with predictable results. Therefore, such a test was not performed in practice.

5.2 Patch sampling

If pixels forming a rectangle are calculated independently, the pseudo-random binary sampling and patch selection is performed unnecessarily. As the largest part of time is spend precomputing which patches are used within a specific tile, performing sampling for adjacent pixels is significantly sped up when the tile properties are already known. Here, simply retrieving the information which patch the pixel is in and returning the appropriate pixel is even faster. This allows us to synthesise a continuous texture of 512×512 pixels in the order of tens of milliseconds, while our naïve pixel-based approach takes over a second.

Such an implementation can be beneficial when entire neighbourhoods are required for further processing, such as in filtering.

Figure 3 compares the quality of a texture synthesised with Wang tiles [1], with our method. Note the diamond shaped artefacts, which our method inherently avoids.

Figures 1, 14, and 15 show textures rendered from the inputs in Figs. 8a and 12. Figure 11 demonstrates that our method does not create constructive repetitions at scales far larger than the input texture, but instead displays properties of white noise necessary for certain applications. Flaws in images generated by our method are seams, visible when the texel is large, and discontinuous objects (Fig. 14). This problem is inherent to patch-based methods, as can be seen in the results of the baseline method (Fig. 15). Both issues can be mitigated by manual selection of the tile and patches when precomputing.

If adjacent pixels are rendered naïvely, the speedup is be linear. However, if congruous sections are retrieved simultaneously in each thread, the algorithm can be sped up further, because the decision process selecting patches need only be executed once.

6 Conclusion and future work

We have presented a method with the benefits of current parallel texture synthesis algorithms, allowing texture synthesis in real-time environments from a minimal texture map. By sampling every pixel independently, parallel processing can be exploited for a synthesised texture of arbitrary size, while avoiding repetition along lines or rectangles to avoid visible seams. Our method makes it possible to perform exemplar-based texture synthesis of arbitrary size in times comparable with much simpler image retrieval operations. Our results are of the same quality as sequential patch-based synthesis, while significantly reducing retrieval time.

The benefits of this method over previous parallel patch-replacement [17] is twofold: precomputed patches are not limited to lie on a precomputed fixed patch map, and rendered patches can lie over the boundaries of other precomputed patches. This makes the method suitable even to complex irregular textures.

A major drawback of the method is the unavoided repetition of corners of the repeatable tile. Careful selection of patches would allow these corners to be overlapped by patches as well, and future work could investigate this. Thanks to the read-only access of precomputed texture information, our method will make it possible to efficiently utilise GPU hardware to improve rendering times, which is also future work.

In future work, high-level ideas from this work will make it possible to create replaceable patches which do not follow any repeating grid. Patch sets are not inherently limited to two groups. The non-overlapping biclique introduced here is not limited to patches which follow the checkerboard pattern of precomputed interchange locations (Fig. 9), but can follow a random Wang tile pattern. This will further combine benefits of the two techniques.

It is clear that with a single underlying tile, repetitions in areas not covered by patches will be visible. Future work could combine our technique with Wang tiles to deal with both these issues, and the issue of repeating edges in Wang tiles.

Thanks to the inherent parallelism, many further applications can be explored. This method will greatly benefit texture compression, bundling of preprocessed textures with graphical design packages, virtual environment platforms, video games, rendering engines, and mobile applications.

References

1. Cohen, M.F., Shade, J., Hiller, S., Deussen, O.: Wang tiles for image and texture generation. ACM Trans. Graph. **22**(3), 287 (2003)
2. Efros, A.A., Freeman, W.T.: Image quilting for texture synthesis and transfer. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, pp. 341–346. ACM, New York (2001)
3. Efros, A.A., Leung, T.K.: Texture synthesis by non-parametric sampling. In: The Proceedings of the Seventh IEEE International Conference on Computer Vision, 1999, vol. 2, pp. 1033–1038. IEEE (1999)
4. Ephanov, A., Coleman, C.: Virtual texture: a large area raster resource for the GPU. In: The Interservice/Industry Training, Sim-

ulation and Education Conference (I/ITSEC), vol. 2006. NTSA (2006)
5. Gilet, G., Dischler, J.M., Ghazanfarpour, D.: Multi-scale assemblage for procedural texturing. Comput. Graph. Forum **31**(7 PART1), 2117–2126 (2012). doi:10.1111/j.1467-8659.2012.03204.x
6. Grunbaum, B., Shephard, G.C.: Tilings and Patterns. W.H. Freeman & Company, New York (1986)
7. Jamriska, O., Sykora, D., Hornung, A.: Cache-efficient graph cuts on structured grids. In: Computer Vision and Pattern Recognition (CVPR), 2012, pp. 3673–3680. IEEE (2012)
8. Kwatra, V., Essa, I., Bobick, A., Kwatra, N.: Texture optimization for example-based synthesis. In: ACM Transactions on Graphics (TOG), vol. 24, pp. 795–802. ACM, New York (2005)
9. Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A.: Graphcut textures: image and video synthesis using graph cuts. In: ACM Transactions on Graphics (TOG), vol. 22, pp. 277–286. ACM, New York (2003)
10. Lagae, A., Dutré, P.: An alternative for wang tiles: colored edges versus colored corners. ACM Trans. Graph. **25**(4), 1442–1459 (2006)
11. Lasram, A., Lefebvre, S.: Parallel patchbased texture synthesis. In: Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics (2012)
12. Lefebvre, S., Hoppe, H.: Parallel controllable texture synthesis. In: ACM Transactions on Graphics (TOG), vol. 24, pp. 777–786. ACM, New York (2005)
13. Neyret, F., Cani, M.P.: Pattern-based texturing revisited. In: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, pp. 235–242. ACM Press/Addison-Wesley Publishing Co., New York (1999)
14. Obert, J., van Waveren, J., Sellers, G.: Virtual texturing in software and hardware. In: ACM SIGGRAPH 2012 Posters, p. 5. ACM, New York (2012)
15. Praun, E., Finkelstein, A., Hoppe, H.: Lapped textures. In: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 00 (1), pp. 465–470 (2000)
16. Taibo, J., Seoane, A., Hernández, L.: Dynamic virtual textures. J. WSCG **17**(1–3), 25–32 (2009)
17. Vanhoey, K., Sauvage, B., Larue, F., Dischler, J.M.: On-the-fly multi-scale infinite texturing from example. ACM Trans. Graph. **32**(6), 208 (2013)
18. Wei, L.Y.: Tile-based texture mapping on graphics hardware. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, pp. 55–63. ACM, New York (2004)
19. Wei, L.Y., Lefebvre, S., Kwatra, V., Turk, G., et al.: State of the art in example-based texture synthesis. In: Eurographics 2009, State of the Art Report, EG-STAR, pp. 93–117 (2009)
20. Wei, L.Y., Levoy, M.: Fast texture synthesis using tree-structured vector quantization. In: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, pp. 479–488 (2000)
21. Wei, L.Y., Levoy, M.: Order-independent texture synthesis. Technical report, TR 2002 (2002)
22. Xue, F., Zhang, Y.S., Jiang, J.L., Hu, M., Wu, X.D., Wang, R.G.: Real-time texture synthesis using s-tile set. J. Comput. Sci. Technol. **22**(4), 590–596 (2007)
23. Zhang, X., Wandell, B.A., et al.: A spatial extension of CIELAB for digital color image reproduction. SID Int. Symp. Dig. Tech. Pap. **27**, 731–734 (1996)

Martin Kolář M.Sc. is a PhD student working in Texture Synthesis for Tile Manufacturing at Warwick Manufacturing Group, University of Warwick. He is working with industrial partner Johnson Tiles to research new methods for automating and enhancing example-based texture synthesis design, specializing in HDR, color profiling, and real-time texture synthesis.

Alan Chalmers is a Professor of Visualisation at the International Digital Laboratory, WMG, University of Warwick, UK. He has published over 200 papers in journals and international conferences on high-fidelity graphics, multi-sensory perception, high dynamic range (HDR) imaging, virtual archaeology and parallel rendering. He is Honorary President of Afrigraph and a former Vice President of ACM SIGGRAPH. Chalmers' research goal is "Real Virtuality", obtaining physically-based, multi-sensory, high-fidelity virtual environments at interactive rates through a combination of parallel processing and human perception techniques.

Kurt Debattista holds a PhD from the University of Bristol, an MSc in Computer Science, an MSc in Psychology and a BSc in Mathematics. Dr Debattista has published over 70 papers in peer-reviewed international conferences and journals, four guest editorials in leading journals, co-edited six books and co-authored one book on advanced high dynamic range imagery. Dr Debattista is a member of a number of programme committees for international conferences. In 2013, he was also granted a Royal Society Industrial Fellowship in co-operation with Jaguar Land Rover.

EUROGRAPHICS 2017 / L. Barthe and B. Benes
(Guest Editors)

Volume 36 (2017), Number 2

A Subjective Evaluation of Texture Synthesis Methods

M.Kolář[1,2], K. Debattista[1] and A. Chalmers[1]

[1]University of Warwick
[2]Brno University of Technology

Abstract

This paper presents the results of a user study which quantifies the relative and absolute quality of example-based texture synthesis algorithms. In order to allow such evaluation, a list of texture properties is compiled, and a minimal representative set of textures is selected to cover these. Six texture synthesis methods are compared against each other and a reference on a selection of twelve textures by non-expert participants ($N = 67$). Results demonstrate certain algorithms successfully solve the problem of texture synthesis for certain textures, but there are no satisfactory results for other types of texture properties. The presented textures and results make it possible for future work to be subjectively compared, thus facilitating the development of future texture synthesis methods.

1. Introduction

Exemplar-based texture synthesis has numerous applications across computer graphics, such as inpainting, rendering, and manufacturing. Thirty years of research has resulted in a wide array of approaches. However, texture quality can be subjective, and independent evaluation of known methods is lacking. There is currently no structured way of evaluating the quality of texture synthesis algorithms, making it difficult to gain an understanding of which methods perform better, and under which circumstances.

In this work, a method for analyzing textures is proposed and used in a subjective experiment involving six representative texture synthesis methods and a reference (figure 1). A minimal set of twelve textures is created for this evaluation. These textures have been selected by compiling a list of 21 properties, and choosing textures such that the full range of each property is covered. This allows the analysis of texture synthesis quality for specific texture properties.

The benefit of this work is twofold: First, this work is the first to compare texture synthesis algorithms in a subjective study on textures selected to represent the wide variability of all textures. Statistically significant preferences are identified for algorithms, as well as over individual textures. Second, we offer a minimal set of textures with which future work may be subjectively compared, to promote the quality enhancement of future texture synthesis algorithms.

2. Background and Motivation

Texture Synthesis algorithms are typically compared on a small sample, possibly resulting in inconsistent evaluations of new methods and the misunderstanding of method properties. This work identifies textures which cover a wide range of properties, and uses them to synthesize outputs with six representative algorithms. A study using these outputs finds statistically significant user preferences between methods.

Unlike general images, textures must satisfy stationarity and locality. The first criterion requires any two patches to be visually similar, and the second criterion requires that any pixel be only related to a small set of neighboring pixels. As these criteria are satisfied to different degrees (for example a uniform noise image satisfies them perfectly), so must texture synthesis algorithms be able to handle textures with limited locality, stationarity, and other properties.

Many texture synthesis algorithms have been devised over the past 30 years [WLKT09], with a focus on various aspects: quality, speed, parallelism, use for animation, manufacturing quality, and others. In this paper, we focus on the quality of the synthesized texture. Despite attempts to quantify synthesized texture quality via texture energy metrics [KSE*03] or image statistics [Bal06], texture quality remains a subjective notion because optimizing these metrics does not guarantee textures of high visual quality [JFA*15]. However, no independent comparative user study has been performed to assess texture synthesis algorithms.

Several texture datasets have been previously created to demonstrate and evaluate texture synthesis: Brodatz Textures [Bro66], VisTex Textures [Gra95], DeBonet Textures [DB97], Colored Brodatz [ADC13], the PSU Near-Regular Texture Database [LL05], Simoncelli Textures [PS00], and many others. See Hossain and Serikawa [HS13], and Bianconi and Fernández [BF14] for a complete survey of Texture Databases. None of these cover the

(a) reference (b) resynthesizer (c) quilting (d) self-tuning (e) LazyFluids (f) Ashikhmin (g) CNNMRF

Figure 1: The *smarties* texture synthesized with selected methods, ordered by user preference

full range of published texture properties [Lou90, FH93, KSS97, LFTG97, BFH*98, FvDF*93] which the various methods claim to handle.

Texture synthesis quality evaluation is an inherently subjective problem. In order to clarify it, various schemes have focused on a subset of textures, by classifying textures by properties, so that appropriate synthesis algorithms may be created for specific applications. Some of the proposed classification schemes are the continuous texture spectrum from regular to stochastic [LHW*04a], and the A-score and G-score of Liu, Lin, and Hays [LLH04], promoting the creation of algorithms specifically for regular and near-regular textures.

Comparative studies have been performed for tone-mapping [LCTS05], inverse tone-mapping [BLD*09], image retargetting [RGSS10], and single image blind deblurring [LHH*]. There have also been comparative studies for texture synthesis, but only through individual subjective evaluation with example textures on which the algorithms do not fail [LHW*04b].

2.1. Selected Texture Synthesis Algorithms

In order to provide a comparison across texture synthesis methods, six methods were chosen. The selected algorithms belong to one of three categories: state-of-the-art methods with a focus on quality [KNL*15, JFA*15, LW16], well-known historic methods [EF01, Ash01], and methods whose code is public and known to be regarded, outside the research community, as useful and well-performing [CR11, Ash01]. The method will be referred to by the name described in brackets after its introduction in the following sections.

Texture Synthesis by image quilting [EF01] (**quilting**) places square subsamples of the exemplar onto the output texture, optimally choosing transitions by finding nearest matches according to the overlap. Each overlapping region is then optimally cut with a minimum cost path. Various implementation based on the original paper are available at `http://people.csail.mit.edu/thouis/efros-freeman/`.

Ashikhmin's Natural Texture Synthesis [Ash01] (**Ashikhmin**) is based on Wei and Levoy's Texture Synthesis [WL00], where pixels are individually added row-by-row by finding the best matching candidate according to the surrounding pixel similarity. Ashikhmin improves this search by focusing on several candidates, thus encouraging verbatim copying instead of blurring. An implementation based on the original paper are available at `http://www.cs.utah.edu/~michael/tscode/`.

Resynthesizer [CR11, Har05] is an open-source texture synthesis plugin. In this algorithm, pixel values are chosen one at a time in a random order. When choosing the value of a pixel, the n nearest pixels that already have values are located, and the input image is searched for a good match to the pattern these pixels form. Once a good match is found, the appropriate pixel value is copied from the input texture to the output. To increase quality, some earlier chosen pixel values are re-chosen after later pixel values have been chosen. The source code and precompiled binary are available at `http://www.logarithmic.net/pfh/resynthesizer`.

Self Tuning Texture Optimization [KNL*15] (**self tuning**) is a general-purpose and fully automatic self-tuning non-parametric texture synthesis method. Various parameters and weights are tuned by focusing on three aspects of texture synthesis: irregular large scale structures are faithfully reproduced through the use of automatically generated and weighted guidance channels, repetition and smoothing of texture patches is avoided by new spatial uniformity constraints, and a smart initialization strategy is used in order to improve the synthesis of regular and near-regular textures [LLH04] without affecting textures that do not exhibit regularities. The Matlab code is available from the authors.

LazyFluids Appearance Transfer [JFA*15] (**LazyFluids**) extends Graphcut Textures [KSE*03] which minimizes Texture Energy

$$E(Z,X) = \sum_{p \in X} \min_{q \in Z} ||x_p - z_q||^2 \quad (1)$$

where Z is the source texture, and X is the output. However, this is known to create an output image which matches only a portion of the input, for example, blurred parts, and quality is highly dependent on selected parameters. LazyFluids resolves these issues by using a nearest-neighbor field to assure uniform source patch usage.

CNNMRF [LW16] is based on Neural Style [GEB15], which uses statistics of higher levels of a pre-trained Convolutional Neural

Network, and iteratively adjusts a random noise image to match the statistics of a texture exemplar. However, CNNMRF also fits a Markov Random Field over neuron activations, in order to better match the texture properties. The implementation code is available at https://github.com/chuanli11/CNNMRF.

Note that Self Tuning [KNL*15] and CNNMRF [LW16] require hours for large textures, while the other algorithms run in seconds or minutes. Exact times are not reported here because they vary with implementations, and this work focuses on quality. For the data used in the experiment, we performed synthesis for all algorithms except LazyFluids. The code for this method is not public, and synthesis was performed on our request by the authors of the method.

3. Texture Dataset

This section describes the process taken to establish a small representative set of textures. First, the shortcomings of the most popular datasets and the goals of this dataset are discussed in section 3.1. Then, texture properties are individually listed and explained in section 3.2, and section 3.3 lists and shows chosen textures.

3.1. Dataset Goals

Previous texture datasets have been devised with various goals, from the creation of comprehensive tileable textures for computer games to demonstrations of artistic renderings. However, despite the number and size of available datasets, none of them fulfill requirements for a minimal set of textures which are representative of known texture properties. None of these datasets attempt to cover texture properties as listed below in a structured way.

Certain datasets contain non-textures as well as textures, making them an interesting tool for understanding the inner workings and limitations of texture synthesis algorithms. However, non-texture images are not informative when assessing quality.

Most datasets contain more than 50 textures. This makes it challenging to perform a detailed evaluation, as evaluation requires human observation of the results. It also leads necessarily to manual selection of representative textures when publishing, thereby making it challenging to compare algorithms where authors have chosen different exemplars.

Numerous datasets also lack the variety of properties which is necessary for a robust evaluation: all textures are at the same scale, at the same resolution, or produced with the same camera. This creates a bias toward certain properties, making the results of algorithm evaluation challenging to extrapolate.

Lastly, no existing application-independent texture dataset contains exemplars as well as reference textures. By making this available, new methods of evaluation are possible, namely comparison to a ground truth.

3.2. Texture Properties

Textures are expected to exhibit stationarity and locality [WLKT09], but only a random noise image can satisfy these perfectly. Texture synthesis algorithms must be able to handle textures with varying locality, stationarity, and various other properties described here, and in tables 1 and 2. Ordinal properties are listed individually in sections 3.2.1 to 3.2.14, followed by binary properties in section 3.2.15. The selection combines properties discussed in other texture synthesis publications. Whenever possible, a clear definition of how each property is measured is included, but in some cases it is subjective and relative.

3.2.1. Scale

Scale refers to the size of texture elements in relation to the size of the entire example. Textures with a small repeating element are considered fine-grained (low scale), such as the *rough* texture. Conversely, textures where the repeating element is large are considered blown-up (high scale), such as *straw* or *green marble*, where certain texture elements continue across the entire texture. Furthermore, textures may be Multi-Scale, exhibiting texture elements at various scales simultaneously. Note that this property is independent of resolution.

3.2.2. Stochasticity

Stochasticity, or randomness, refers to the random variability within a texture. This can be formulated as follows *Given information of other pixels in the texture, how predictable is another pixel?* Therefore, even textures which are entirely random globally (such as the *smarties* texture, where each element is placed randomly) do not exhibit perfect stochasticity, because nearby pixels are likely to belong to the same texture element. Similarly, textures where pixels are likely to change locally are not entirely stochastic if the structure is periodic (regular), such as the *grid* texture.

3.2.3. Stationarity

Stationarity is the property which defines to what degree variance is linked with neighborhood. A stationary pattern is similar to a local pattern, in that texture elements are dependent on local pixel neighborhoods. However, a non-stationary pattern may be local or non-local, because both local and global underlying patterns can affect pixels to varying degrees. Note that texture stationarity is independent of texture locality, because a non-stationary pattern may simultaneously be highly local, such as the flow-guided texture *ink*.

3.2.4. Locality

Locality refers to the property of textures to depend on a local neighborhood. Regular textures, those which fit onto a repeating lattice, exhibit low locality, because pixel values depend on the entire texture, such as the *straw* or *grid* textures. Textures exhibiting locality are for example the *blades* and *smarties* textures, because small neighborhoods are locally independent from the rest of the texture.

3.2.5. Flow-guided

If a texture was generated in a process involving motion with some continuity, it is said to be flow-guided. This property can be seen to varying degrees, so the flow-guided property is ordinal, rather than binary.

Property	Definition	Low	Mid	High
Scale	fine-grained(fine) to blown-up(coarse)	rough	ink	pebbles
Stochasticity	entirely deterministic to entirely random	grid	straw	ink
Stationarity	different regions are perceived to be similar	smarties	orange marble	chicago
Locality	pixel values depend on small neighborhood	ink	pebbles	rough
Flow-guided	generative process includes flow	pebbles	straw	green marble
Shape variance	shapes do not vary to wide shape variance	grid	smarties	ink
Color variance	color does not vary to high color variance	blades	green marble	smarties
Natural	natural versus simulated	ink	chicago	orange marble
Absolute Texel size (resolution)	number of pixels in a texel	rough	blades	smarties
Texels per sample	number of texels in example	pebbles	chicago	blades
A-score	appearance regularity	chicago	straw	rough
G-score	geometric regularity	blades	chicago	grid
Scale variance	variation in scale of elements within texture	smarties	ink	green marble
Rotation	texture rotation invariance	grid	straw	blades
Example resolution	tiny to huge	straw	ink	smarties

Table 1: Ordinal properties of selected textures. Note that the noise and checkerboard textures are not required for completeness.

Property	true	false
Historic	straw	ink
Regular color, irregular shape	pebbles	chicago
Irregular color, regular shape	smarties	straw
Multi-scale	chicago	blades
Global variance	green marble	rough
Overlapping multiple textures	chicago	grid
Color	blades	straw

Table 2: Binary properties of selected textures

Texture	Source	Size
 blades	[ADC13]	640 × 640
 grid	[Bro66]	640 × 640
 pebbles	[Bro66]	640 × 640
 rough	[ADC13]	640 × 640
 orange marble	own work	1024 × 1340
 green marble	own work	657 × 876
 straw	[Bro66]	256 × 256
 chicago	[cre14]	2266 × 2267
 ink	own work	1281 × 653
 smarties	[Sch12]	3543 × 2362
 checkerboard	own work	256 × 256
 noise	own work	1000 × 1000

Table 3: Selected textures, with images of the exemplar. All textures are freely available under various licenses at full resolution.

3.2.6. Shape Variance

Variance refers to variation of the shape of texture elements. For example, the *smarties* texture has low shape variation of its elements, therefore exhibiting high regularity. Conversely, low shape regularity or lack of shape altogether mean high shape variance, such as textures *ink* or *blades*. Because this property is not related to the concept of a loose lattice, it is different from the G-score of section 3.2.11. However, whenever a loose lattice can be fitted onto a texture, shape variance inversely corresponds to G-regularity.

3.2.7. Color Variance

Similarly to shape variance, color variance refers to the variation of the color across elements of a texture. This can be attributed to variability between elements, and variability within a given element. For example, the *blades* texture displays only two major colors, with little variation while the *smarties* texture shows both types of color variation.

3.2.8. Natural

This property refers to the degree to which a texture has been produced in a natural process, rather than being the product of a computer simulation. The marble textures show unedited images of stone, thus being entirely natural. The ink texture has been generated in an entirely artificial way. Others, such as *chicago* and *smarties* are produced in a partly natural process, consisting of both a repeatable generative process as well as an element of natural randomness. This property, together with the Flow-guided property, can serve as a helpful indicator of repeatability of texture synthesis results across natural and synthetic textures. Because arbitrarily large samples from synthetic textures can be drawn, it is a potential benefit for texture synthesis algorithm development.

3.2.9. Absolute Texel Size

This property refers to the size in pixels of a repeating texture element. This is included to shift focus from algorithms which work at a given scale, and used to evaluate texture synthesis methods which perform well on textures with repeating elements at any scale.

3.2.10. Texels per Sample

The number of texels per sample refers to the repeating texture elements of sections 3.2.1 and 3.2.9. These three properties are inevitably interlinked, and are listed here for completeness.

3.2.11. A-score and G-score

As introduced by Liu, Lin, and Hays [LLH04], Geometric (G) Regularity and Appearance (A) Regularity of near-regular textures provide a quantitative measure of deviation from a perfectly regular texture in terms of shape, and color. These differ from shape variance and color variance defined earlier by requiring a texture to be near-regular. Textures which cannot fit into a loose lattice structure, such as the *smarties* texture of figure 1 are not near-regular, and therefore cannot have an A-score and G-score.

3.2.12. Scale Variance

The difference in scale between repeating elements in a texture. For instance, the *pebbles* texture contains pebbles of varying size, creating scale variance. On the other hand, the *grid* texture is scale invariant, because texture elements are of constant size.

3.2.13. Example Resolution

The absolute pixel size of the example texture. Similarly to Absolute Texel Size, this property enables the evaluation of algorithms which can handle small as well as large example images. Table 3 shows selected texture resolutions.

3.2.14. Rotation

Texture Rotation refers to rotation invariance of repeatable texture elements. If elements of a texture can be rotated, such as the *ink* texture, it exhibits high rotation. Conversely, if elements of the texture cannot be rotated, such as the *grid* texture, rotation is low. Note that this property does not refer to rotating the entire texture, because this is always assumed to be possible.

3.2.15. Binary Properties

In addition to the ordinal properties discussed here, it is important to consider certain categorical binary properties to quantify textures. The properties listed in table 2 are discussed in depth here, to clarify and justify their definition and selection.

Historic textures were selected because various past publications have already demonstrated their performance on them in sets of synthesized images available from the authors. Rather than requiring some textures to be old, the purpose of this property is to allow a clear link to past work.

The two **combinations of color and shape variance and regularity** have been included to match all four categories of near-regular textures according to their A-score and G-score. This produces the classification of Liu, Lin, and Hays [LLH04], where near-regular textures are divided into four types:

0 Regular Geometry, Regular Color
I Regular Geometry, Irregular Color
II Irregular Geometry, Regular Color
III Irregular Geometry, Irregular Color

As mentioned in section 3.2.1, **Multi-scale textures** contain texture elements at various scales, such that they interact with each other. For example, the *chicago* texture demonstrates this property: the road grid is a lower scale texture than the houses, but both of these properties satisfy the properties required to be a texture.

Global variance is the property that texture elements differ either by color or geometry across the texture in a global, predictable manner. While this is at odds with the basic requirement for locality, a texture demonstrating this property is included in the dataset to judge how this affects texture synthesis algorithms.

An additional property of textures is that there may be a **combination of overlapping textures**. Unlike Multi-resolution, this property requires independence of the combined textures, instead of interaction between a higher-scale repeating texture element with a lower-scale one. The texture element may even be on the same scale. The *ink* texture demonstrates this property, because there are different independent overlapping generative processes involved.

Finally, textures in both **color and grayscale** are included, to facilitate testing of single-channel texture synthesis algorithms. Single-channel texture synthesis algorithms can be applied on textures with three or more color channels as well, by simply converting colors to a single measure, or for certain methods by providing a similarity metric. However, these results would be skewed by the chosen color mapping, hence the benefit of providing two grayscale textures (*straw* and *pebbles*). For work with single-channel methods, we recommend the method of Smith et al [SLTM08] which produces accurate and perceptually preferred color to grayscale conversions according to a prior comparative study [Č08].

3.3. Texture Selection

Textures were selected such that at least one covers the low, mid, and high values of every ordinal property listed in table 1, and each case of the binary properties 2. See Table 3 for exemplar images and sources of these textures.

To allow an example and reference output to be produced from the selected textures, it is required that a subimage of $1/3$ width and $1/3$ height is a sufficient sample of the texture, as per figure 2. Therefore, some popular textures which have been widely applied by the community had to be discarded because such a small subimage did not contain a representative patch.

4. Experiment

This section describes the experimental method, including design, materials and algorithm parameter configuration, and the procedure. The interface source code and textures used in the experiment are available at `https://github.com/mrmartin/ordering_study`.

4.1. Design

The experiment compares six algorithmic methods across twelve textures. The experiment is based around a ranking design based

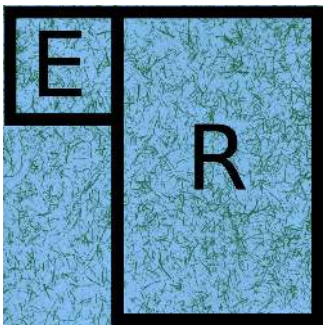


Figure 2: Process of selection of the input exemplar E and the reference output R from a square sample texture.

on similar experiments for HDR video compression comparisons [MDBR*16]. The ranking procedure can be seen in figure 3 where a reference exemplar, acting as ground truth is presented. The participant ranks the stimuli interactively according to how close to the exemplar they believe the stimuli to be. Ranking gives participants the opportunity to be able to view all stimuli an equal number of times, and be able to view them concurrently within a reasonable amount of time reducing the fatigue associated to other design choices such as pairwise comparisons. The *texture* variable corresponds to a within-participants independent variable encompassing the different texture stimuli. The *method* variable is also a within-participants independent variable referring to the distinct texture synthesis algorithms. The *method* variable also includes a hidden reference besides the texture synthesis generated textures. The hidden reference is added to help identify differences between the reference and texture synthesis methods and to see how close these are to the ground truth. The experimental question was explicitly formulated to allow multiple interpretations, asking users to "Sort [...] by how much they look like the original.", "Sort [...] by order of realism.", and "Order [...] according to how similar to the reference you think they are."

4.2. Materials

The selection of textures and texture synthesis algorithms for the study has been guided by various constraints. First, in order to compare synthesis outputs to a ground truth texture, it was necessary to choose textures large enough to be divided into an input and output sample. Furthermore, this output sample needed to be larger than the input, in order to clearly demonstrate algorithmic properties. An output aspect ratio of $2:3$ was chosen, and because it was desirable to present all textures equally on one row, the number of outputs was limited to seven, comprising the six methods and the hidden reference (see user interface in figure 3).

The input and output images were generated by first taking a texture, and cropping it to a square. Then, the top-left corner of $1/3$ width and $1/3$ height is extracted to give the exemplar. The right band of $2/3$ width and full height was used as the ground truth. See figure 2.

All selected algorithms were run with the same parameters across all textures, to simulate a non-expert user environment. Default parameters were set according to the cited publications. Note that certain algorithms required no tuning at all (self-tuning and resynthesizer), while for some finding default configurations was challenging and error-prone. Ashikhmin was executed in three passes over a 7×7 neighborhood, and Quilting was performed with a patch size half of the input, and left and top overlaps one third of the input.

4.2.1. User Interface

Figure 3 shows the GUI used for selection. The experiment was run entirely inside a browser to provide online availability; this permits easy of use and the ability to recruit a wide variety of users with various monitors, resolutions, preferences, and lighting conditions. The GUI presents the exemplar as a ground truth reference in the top center and the six methods plus hidden reference underneath. The GUI allows the selection and movement of any of the stimuli corresponding to the methods along the x-axis to be ordered according to participant preferences.

The background is a neutral gray, and the user is requested to maximize the window. All images scale to fill as much of the screen as possible, and the aspect ratio between the input and output are maintained.

4.3. Participants and Procedure

Participation was anonymous, and available to students at two international universities, and interested members of the public. Screen resolution was recorded, and participants with less than HD 720p were discarded. Participants who did not change the order of any textures, and those who did not reach the end of the experiment were discarded. In total, the subjective study was performed by 67 participants.

A mass e-mail was sent out to students and staff of two universities to recruit participants on a voluntary basis. Those who responded were sent a URL corresponding to the experiment. No personal data was collected. The experiment was run entirely in browser and results stored on a server.

5. Results and Analysis

Analysis was conducted using the non-parametric Kendall test for Concordance. Kendall's test for Concordance (W) provides a statistic between 0 and 1 conforming to the agreement across participants. A W of 1 indicates perfect agreement among participants and 0 means perfect disagreement. The overall W of this study is 0.402, which is compares well with other subjective ordering studies in Computer Graphics: 0.12 [LHH*], 0.095 [RGSS10], 0.282 [BLD*09]. W is also be tested for significance, and its p-value is below 10^{-10} for every texture, due to the high number of participants [Gwe14].

Pairwise comparisons among all the methods for each of the textures were also conducted, these give an indication of whether there are any significant differences across methods for a given texture or in the overall experiment.

Results for each texture, as well as across all textures using collapsed scores, are shown in table 4. The groupings in each row

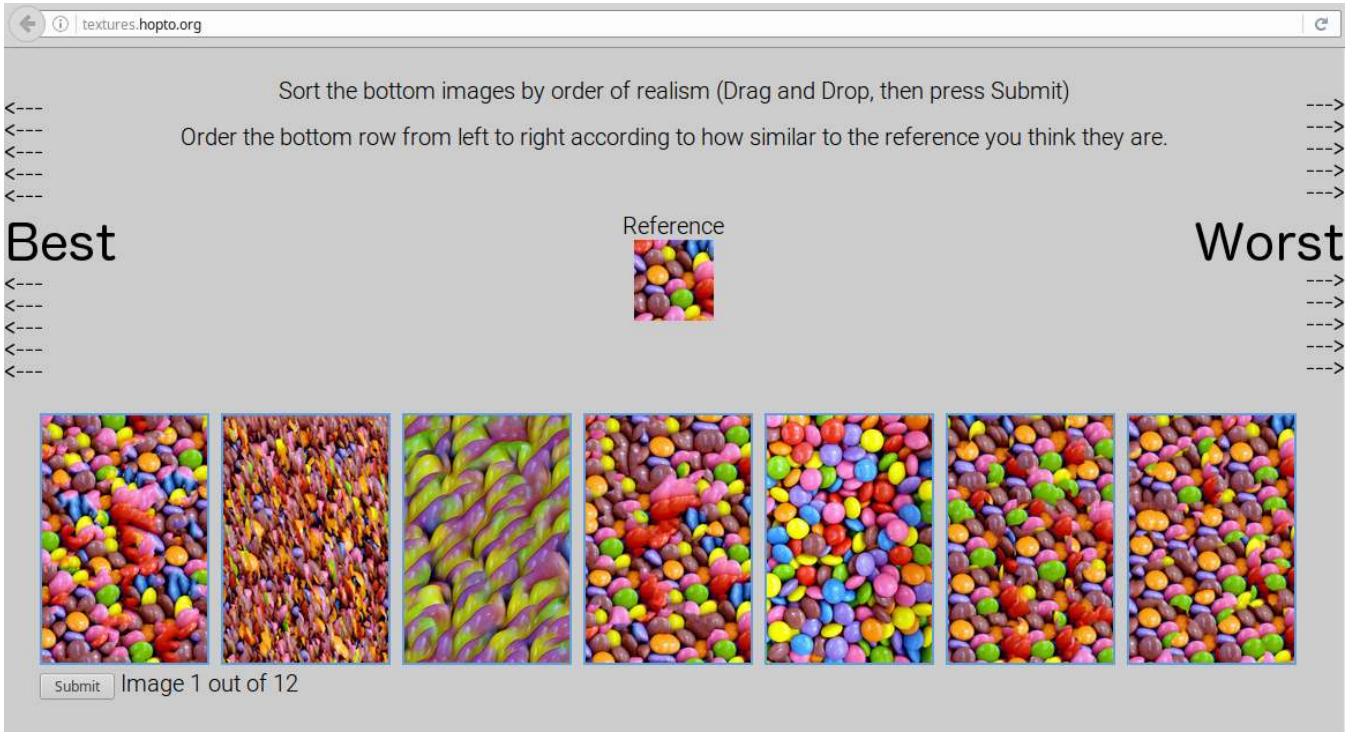


Figure 3: The experiment's user interface. The textures can be reordered interactively, and the entire page scales up to maintain correct aspect ratios between the reference and the synthesized textures.

demonstrate non-significant differences. Methods not grouped together indicate significant differences across those methods for that particular texture.

Results on the noise and checkerboard textures serve as a sanity check, verifying assumptions about the experiment. Consistently with assumptions, the noise texture is easy to synthesize correctly, because four out of six methods create results statistically indistinguishable from the reference. Results with the checkerboard texture are visually clearest (this can be seen in figure 5), they show a very high Kendall's coefficient of concordance.

Other textures cause wide disagreement. This is possibly caused by differing views on what constitutes an ideal output. For several textures, participants consistently agree that the reference is not the best (*pebbles*, *rough*, *chicago*, *green marble*). We hypothesize that this is because the reference image contains greater variance than the example, while synthesized textures closely match the inputs' visual properties (figure 6).

There are seven textures for which some methods produce results not significantly worse than the reference (*blades*, *pebbles*, *ink*, *noise*, *green marble*, *orange marble*, *straw*). The texture synthesis methods for these textures may be broadly considered successful.

Out of the remaining five textures (*smarties*, *checkerboard*, *grid*, *rough*, *chicago*), two reference textures are evaluated as significantly worse than synthesized outputs (*rough*, *chicago*), and in the other three, the reference is significantly better than synthesized textures.

5.1. Analysis of Texture Properties

There are three textures for which synthesis is not fully solved by any of the evaluated methods. These are the *smarties* texture (figure 1), *checkerboard* (figure 5), and *grid*. These share some common properties: low-to-mid shape variance, mid-to-low stochasticity, and low locality. These properties, together with complex shape patterns, form patterns which are hard to recreate without considering the underlying generative process.

Low shape variance poses challenges which none of the algorithms handle well. Regular texture elements are not replicated perfectly (figure 5) in the output, and a human observer readily identifies even minute flaws, so the relative perceived quality is significantly worse than the reference.

The results demonstrate that numerous complex properties are well handled by the top four methods: texel size, example resolution, texels per sample, scale, scale variance, stochasticity, and shape. Furthermore, the orderings of table 4 show constructive disagreement among textures, demonstrating each offers novel information. Therefore, they must exhibit uncorrelated properties, validating the selection of section 3.2.

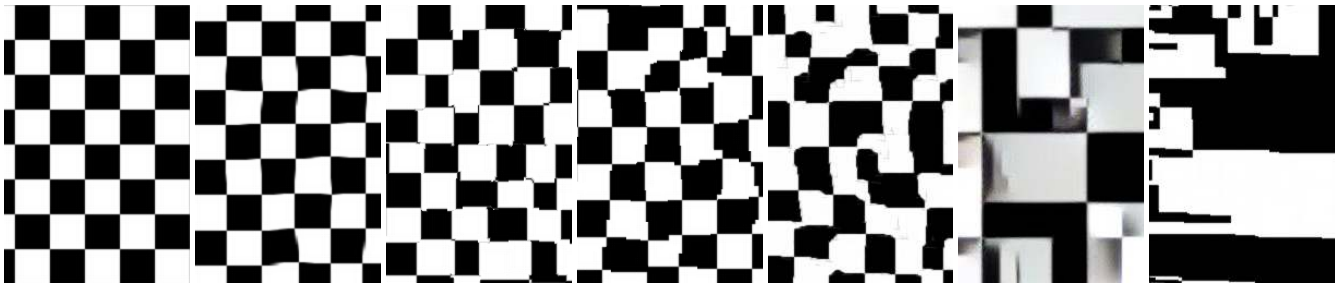
Considering the shared properties of textures which are not well synthesized, and the fact that reference images contain greater variance than the exemplar, the major focus of texture synthesis re-

Texture	ranks						Kendall's W
smarties	reference	resynthesizer	quilting	self tuning	LazyFluids	Ashikhmin	CNNMRF	0.737
checkerboard	reference	self tuning	quilting	resynthesizer	LazyFluids	CNNMRF	Ashikhmin	0.819
blades	reference	resynthesizer	LazyFluids	self tuning	quilting	Ashikhmin	CNNMRF	0.44
grid	reference	quilting	self tuning	LazyFluids	resynthesizer	Ashikhmin	CNNMRF	0.811
pebbles	LazyFluids	reference	self tuning	resynthesizer	quilting	Ashikhmin	CNNMRF	0.461
ink	self tuning	LazyFluids	reference	resynthesizer	Ashikhmin	quilting	CNNMRF	0.558
rough	quilting	resynthesizer	self tuning	LazyFluids	reference	Ashikhmin	CNNMRF	0.447
chicago	resynthesizer	self tuning	quilting	LazyFluids	reference	Ashikhmin	CNNMRF	0.619
noise	reference	LazyFluids	quilting	resynthesizer	self tuning	Ashikhmin	CNNMRF	0.361
green marble	self tuning	LazyFluids	reference	resynthesizer	quilting	Ashikhmin	CNNMRF	0.517
orange marble	LazyFluids	self tuning	reference	resynthesizer	CNNMRF	Ashikhmin	quilting	0.266
straw	self tuning	quilting	resynthesizer	LazyFluids	reference	Ashikhmin	CNNMRF	0.363
ALL	reference	self tuning	resynthesizer	LazyFluids	quilting	Ashikhmin	CNNMRF	0.402

Table 4: Subjective ranks with Kendall W, for each texture separately, and over all textures. Ranks are from left to right. All orderings are significant to $p < 0.01$, except orderings within each group. Kendall's W coefficient of concordance ranges from 0 (no agreement) to 1 (complete agreement).

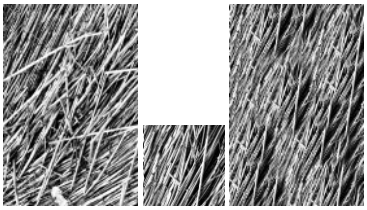
(a) resynthesizer (b) self-tuning (c) quilting (d) LazyFluids (e) reference (f) Ashikhmin (g) CNNMRF

Figure 4: The *chicago* texture synthesized with selected methods, ordered by user preference. Top row is the full synthesized texture as seen by participants, and the bottom row is a center crop of size $1/4 \times 1/4$ of the full output.



(a) reference (b) self-tuning (c) quilting (d) resynthesizer (e) LazyFluids (f) CNNMRF (g) Ashikhmin

Figure 5: The checkerboard texture, ordered by user preference



(a) reference (b) input (c) self-tuning

Figure 6: For this texture, the reference shows greater variance than the exemplar, which is not replicated in synthesis

search should be regularity mapping, learning to replicate the texture's generative process, and a structured approach to estimate variability to increase it in the output.

5.2. Analysis of Algorithm Performance

Overall, the study showed that people prefer the quality of the reference over any algorithm, conclusively demonstrating that none of the available algorithms in general produce convincing textures. Nonetheless, when ordering algorithms in relation to each other, **self tuning** is the best available method, and should be used in applications where quality is key.

Despite its simplicity, **resynthesizer** is indistinguishable from the far more computationally intensive **self tuning** on nine out of twelve textures, demonstrating that it is a powerful and flexible method, whose open-source implementation makes it ideal for portability.

LazyFluids appears best at generating novel textures of the scale of the exemplar, but does not perform well when generating a large sample. Figure 4 shows the full-scale vs the details of the *chicago* texture, revealing additional clues regarding method quality. Optimizing the quality metric of LazyFluids, which is an enhancement of Texture Energy [KSE*03], yield textures of comparatively low quality. Therefore, these quality metrics are not appropriate for quantitative quality evaluation.

Ashikhmin and **CNNMRF** have not performed as well as the other methods. Although these methods are known to produce impressive results, this demonstrates that their usefulness is limited to certain textures, and that they require careful parameter selection. Despite being the oldest tested method, **quilting** was consistently preferred over both of these, demonstrating the flexibility with which it achieves quality.

6. Conclusion

The quality of existing texture synthesis algorithms has been quantified by means of a subjective experiment, demonstrating that the problem of example-based texture synthesis is not yet fully solved. It appears there are types of textures and some texture properties where existing algorithms fail; although certain types of texture are found to be very well synthesized. The quality of previously published methods has been validated, and new findings regarding properties of textures which are not well synthesized have been made.

A set of textures which are representative of the desired properties have been proposed, with the hope that the performance of any method may be judged over all textures simply by observing results on these. Nine of these textures can be considered solved by the best algorithm, **self tuning**, and three show significant flaws with all state-of-the-art methods, allowing for compact and straightforward analysis of new methods.

A shortcoming of our final analysis is that the collapsed scores assume equal weight across the selected textures, but in reality this weighting is application-specific. We have attempted to allow application-specific analysis by showing rankings for each texture, but the overall ranking will not be representative of all applications. A limitation of the current work is that all participants were shown the same rendering, but independent renderings would have produced a more representative evaluation of methods.

By identifying properties of textures which have consistently not been synthesized as well as the reference, we hope to help advance research in the field of example-based texture synthesis, so that algorithms may be devised which handle any texture. Conclusions drawn regarding relationships between the texture properties and algorithm performance are informative, and hint to an area where additional studies may make statistically significant findings.

In future work, it will be beneficial to take into account user's preferences in application-specific environments, by performing the study with printed textures, ceramic tiles, or inside virtual envi-

ronments. The evaluation procedure with the proposed dataset requires human interaction, and it would be beneficial if quality assessment could be performed automatically. Finally, chosen properties and textures may need to be extended, either because future improvement is likely to solve the 12 proposed textures, or because certain applications may require texture properties not explored here, such as embossing or linear features. New textures may be found by first preparing a large dataset, then finding a representative subset.

Acknowledgment

The authors would like to thank the authors of compared methods for their code and cooperation. Debattista is partially supported by a Royal Society Industrial Fellowship (IF130053) and Kolář by the EPSRC (EP/I01585X/1).

References

[ADC13] ABDELMOUNAIME S., DONG-CHEN H.: New brodatz-based image databases for grayscale color and multiband texture analysis. *ISRN Machine Vision 2013* (2013). 1, 4

[Ash01] ASHIKHMIN M.: Synthesizing natural textures. In *Symposium on Interactive 3D graphics* (2001), ACM, pp. 217–226. 2

[Bal06] BALAS B. J.: Texture synthesis and perception: Using computational models to study texture representations in the human visual system. *Vision research 46*, 3 (2006), 299–309. 1

[BF14] BIANCONI F., FERNÁNDEZ A.: Appendix to texture databases-a comprehensive survey. *Pattern Recognition Letters 45* (2014). 1

[BFH*98] BUHMANN J. M., FELLNER D. W., HELD M., KETTERER J., PUZICHA J.: Dithered color quantization. *CGF 17*, 3 (1998). 2

[BLD*09] BANTERLE F., LEDDA P., DEBATTISTA K., BLOJ M., ARTUSI A., CHALMERS A.: A psychophysical evaluation of inverse tone mapping techniques. In *CGF* (2009), vol. 28. 2, 6

[Bro66] BRODATZ P.: *Textures: a photographic album for artists and designers*. Dover Pubns, 1966. 1, 4

[CR11] CORNET E., ROUQUIER J.-B.: Resynthesizer plugin for the gimp, 2011. 2

[cre14] CREW I. E. .: Image courtesy of the earth science and remote sensing unit, nasa johnson space center, 2014. 4

[DB97] DE BONET J. S.: Multiresolution sampling procedure for analysis and synthesis of texture images. In *Computer graphics and interactive techniques* (1997). 1

[EF01] EFROS A. A., FREEMAN W. T.: Image quilting for texture synthesis and transfer. In *Computer graphics and interactive techniques* (2001), ACM. 2

[FH93] FELLNER D. W., HELMBERG C.: Robust rendering of general ellipses and elliptical arcs. *ACM TOG 12*, 3 (July 1993), 251–276. 2

[FvDF*93] FOLEY J. D., VAN DAM A., FEINER S. K., HUGHES J. F., PHILLIPS R.: *Introduction to Computer Graphics*. Addison-Wesley, 1993. 2

[GEB15] GATYS L. A., ECKER A. S., BETHGE M.: A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576* (2015). 2

[Gra95] GRACZYK C.: Vistex vision texture database, 1995. 1

[Gwe14] GWET K. L.: *Handbook of inter-rater reliability: The definitive guide to measuring the extent of agreement among raters*. Advanced Analytics, LLC, 2014. 6

[Har05] HARRISON P.: *Image texture tools*. PhD thesis, PhD thesis, Monash University, 2005. 2

[HS13] HOSSAIN S., SERIKAWA S.: Texture databases–a comprehensive survey. *pattern recognition letters 34*, 15 (2013), 2007–2022. 1

[JFA*15] JAMRIŠKA O., FIŠER J., ASENTE P., LU J., SHECHTMAN E., SÝKORA D.: Lazyfluids: Appearance transfer for fluid animations. *ACM Transactions on Graphics 34*, 4 (2015). 1, 2

[KNL*15] KASPAR A., NEUBERT B., LISCHINSKI D., PAULY M., KOPF J.: Self tuning texture optimization. In *CGF* (2015), no. 2. 2, 3

[KSE*03] KWATRA V., SCHÖDL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: image and video synthesis using graph cuts. In *ACM Transactions on Graphics (ToG)* (2003), vol. 22, ACM, pp. 277–286. 1, 2, 9

[KSS97] KOBBELT L., STAMMINGER M., SEIDEL H.-P.: Using subdivision on hierarchical data to reconstruct radiosity distribution. *CGF 16*, 3 (1997), C347–C355. 2

[LCTS05] LEDDA P., CHALMERS A., TROSCIANKO T., SEETZEN H.: Evaluation of tone mapping operators using a high dynamic range display. In *ACM Transactions on Graphics (TOG)* (2005), vol. 24, ACM, pp. 640–648. 2

[LFTG97] LAFORTUNE E. P., FOO S.-C., TORRANCE K. E., GREENBERG D. P.: Non-linear approximation of reflectance functions. In *Proc. SIGGRAPH '97* (1997), vol. 31, pp. 117–126. 2

[LHH*] LAI W.-S., HUANG J.-B., HU Z., AHUJA N., YANG M.-H.: A comparative study for single image blind deblurring. 2, 6

[LHW*04a] LIN W.-C., HAYS J., WU C., KWATRA V., LIU Y.: *A comparison study of four texture synthesis algorithms on near-regular textures*. Tech. Rep. CMU-RI-TR-04-01, 2004. 2

[LHW*04b] LIN W.-C., HAYS J., WU C., KWATRA V., LIU Y.: A comparison study of four texture synthesis algorithms on near-regular textures. In *ACM SIGGRAPH 2004 Posters* (2004), ACM, p. 16. 2

[LL05] LEE S., LIU Y.: Psu near-regular texture database. 1

[LLH04] LIU Y., LIN W.-C., HAYS J.: Near-regular texture analysis and manipulation. In *ACM Transactions on Graphics (TOG)* (2004), vol. 23, ACM, pp. 368–376. 2, 5

[Lou90] LOUS Y. L.: Report on the First Eurographics Workshop on Visualization in Scientific Computing. *Computer Graphics Forum 9*, 4 (Dec. 1990), 371–372. 2

[LW16] LI C., WAND M.: Combining markov random fields and convolutional neural networks for image synthesis. *arXiv preprint arXiv:1601.04589* (2016). 2, 3

[MDBR*16] MUKHERJEE R., DEBATTISTA K., BASHFORD-ROGERS T., VANGORP P., MANTIUK R., BESSA M., WATERFIELD B., CHALMERS A.: Objective and subjective evaluation of high dynamic range video compression. *Signal Processing: Image Communication 47* (2016), 426–437. 6

[PS00] PORTILLA J., SIMONCELLI E. P.: A parametric texture model based on joint statistics of complex wavelet coefficients. *International journal of computer vision 40*, 1 (2000), 49–70. 1

[RGSS10] RUBINSTEIN M., GUTIERREZ D., SORKINE O., SHAMIR A.: A comparative study of image retargeting. In *ACM transactions on graphics (TOG)* (2010), vol. 29, ACM, p. 160. 2, 6

[Sch12] SCHWARZKOPF H.: Public domain photograph, 2012. 4

[SLTM08] SMITH K., LANDES P.-E., THOLLOT J., MYSZKOWSKI K.: Apparent greyscale: A simple and fast conversion to perceptually accurate images and video. In *CGF* (2008), vol. 27. 5

[Č08] ČADÍK M.: Perceptual evaluation of color-to-grayscale image conversions. In *Computer Graphics Forum* (2008), vol. 27. 5

[WL00] WEI L.-Y., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. In *Computer graphics and interactive techniques* (2000). 2

[WLKT09] WEI L.-Y., LEFEBVRE S., KWATRA V., TURK G.: State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report* (2009). 1, 3

Bibliography

[Abdelmounaime and Dong-Chen, 2013] Abdelmounaime, S. and Dong-Chen, H. (2013). New brodatz-based image databases for grayscale color and multiband texture analysis. *ISRN Machine Vision*, 2013.

[Ashikhmin, 2001] Ashikhmin, M. (2001). Synthesizing natural textures. In *Proceedings of the 2001 symposium on Interactive 3D graphics*. ACM.

[Balas, 2006] Balas, B. J. (2006). Texture synthesis and perception: Using computational models to study texture representations in the human visual system. *Vision research*, 46(3).

[Baldi et al., 2007] Baldi, G., Cinotti, E., and Zama, I. (2007). Industrial application of "bottom-up" nano ink, improvement of ink-jet printing to ceramic tile decoration. In *10th International Conference and Exhibition of the European Ceramic Society*. June.

[Banerjee et al., 2009] Banerjee, M., Kundu, M. K., and Maji, P. (2009). Content-based image retrieval using visually significant point features. *Fuzzy Sets and Systems*, 160(23).

[Boukouvalas et al., 1995] Boukouvalas, Kittler, Marik, Mirmehdi, and Petrou (1995). Ceramic tile inspection for colour and structural defects. *Proceedings of Advanced Materials and Processing Technologies*.

[Brodatz, 1966] Brodatz, P. (1966). *Textures: a photographic album for artists and designers*. Dover Pubns.

[Buhmann et al., 1998] Buhmann, J. M., Fellner, D. W., Held, M., Ketterer, J., and Puzicha, J. (1998). Dithered color quantization. 17(3). (Proc. Eurographics'98).

[Cohen et al., 2003] Cohen, M., Shade, J., Hiller, S., and Deussen, O. (2003). Wang Tiles for image and texture generation. *ACM Transactions on Graphics*, 22(3):287.

[Cornet and Rouquier, 2011] Cornet, E. and Rouquier, J.-B. (2011). Resynthesizer plugin for the gimp.

[crew ISS NASA, 2014] crew ISS NASA, E. . (2014). Image courtesy of the earth science and remote sensing unit, nasa johnson space center.

[Criminisi et al., 2004] Criminisi, A., Pérez, P., and Toyama, K. (2004). Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on image processing*, 13(9):1200–1212.

[De Bonet, 1997] De Bonet, J. S. (1997). Multiresolution sampling procedure for analysis and synthesis of texture images. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co.

[DiCarlo and Wandell, 2000] DiCarlo, J. M. and Wandell, B. A. (2000). Rendering high dynamic range images. In *Electronic Imaging*. International Society for Optics and Photonics.

[Efros and Freeman, 2001] Efros, A. and Freeman, W. (2001). Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. ACM.

[Efros and Leung, 1999] Efros, A. and Leung, T. (1999). Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1033–1038. IEEE.

[Ephanov and Coleman, 2006] Ephanov, A. and Coleman, C. (2006). Virtual texture: A large area raster resource for the gpu. In *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*, volume 2006. NTSA.

[Fairchild, 2013] Fairchild, M. D. (2013). *Color appearance models*. John Wiley & Sons.

[Fellner and Helmberg, 1993] Fellner, D. W. and Helmberg, C. (1993). Robust rendering of general ellipses and elliptical arcs. *ACM Transactions on Graphics*, 12(3).

[Foley et al., 1993] Foley, J. D., van Dam, A., Feiner, S. K., Hughes, J. F., and Phillips, R. (1993). *Introduction to Computer Graphics*. Addison-Wesley.

[Gatys et al., 2015] Gatys, L. A., Ecker, A. S., and Bethge, M. (2015). A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*.

[Gilet et al., 2012] Gilet, G., Dischler, J. M., and Ghazanfarpour, D. (2012). Multi-scale assemblage for procedural texturing. *Computer Graphics Forum*, 31(7 PART1).

[Graczyk, 1995] Graczyk, C. (1995). Vistex vision texture database.

[Grunbaum and Shephard, 1986] Grunbaum, B. and Shephard, G. (1986). *Tilings and patterns (book)*. W.H. Freeman & Company.

[Hafemann et al., 2014] Hafemann, L., Oliveira, L., and Cavalin, P. (2014). Forest species recognition using deep convolutional neural networks. In *ICPR*, pages 1103–1107.

[Harrison, 2001] Harrison, P. (2001). A non-hierarchical procedure for re-synthesis of complex textures.

[Harrison, 2005] Harrison, P. (2005). *Image texture tools*. PhD thesis, PhD thesis, Monash University.

[Hossain and Serikawa, 2013] Hossain, S. and Serikawa, S. (2013). Texture databases–a comprehensive survey. *pattern recognition letters*, 34(15).

[Ismert et al., 2003] Ismert, R. M., Bala, K., and Greenberg, D. P. (2003). Detail synthesis for image-based texturing. In *Proceedings of the 2003 symposium on Interactive 3D graphics*. ACM.

[Jamriska et al., 2012] Jamriska, O., Sykora, D., and Hornung, A. (2012). Cache-efficient graph cuts on structured grids. In *Computer Vision and Pattern Recognition (CVPR), 2012*, pages 3673–3680. IEEE.

[Jamriška et al., 2015] Jamriška, O., Fišer, J., Asente, P., Lu, J., Shechtman, E., and Sýkora, D. (2015). Lazyfluids: Appearance transfer for fluid animations. *ACM Transactions on Graphics*, 34(4).

[Joblove and Greenberg, 1978] Joblove, G. H. and Greenberg, D. (1978). Color spaces for computer graphics. In *ACM SIGGRAPH Computer Graphics*, volume 12. ACM.

[Kaspar et al., 2015] Kaspar, A., Neubert, B., Lischinski, D., Pauly, M., and Kopf, J. (2015). Self tuning texture optimization. In *Computer Graphics Forum*, volume 34. Wiley Online Library.

[Kobbelt et al., 1997] Kobbelt, L., Stamminger, M., and Seidel, H.-P. (1997). Using subdivision on hierarchical data to reconstruct radiosity distribution. 16(3). (Proc. Eurographics'97).

[Kolar et al., 2015] Kolar, M., Chalmers, A., and Debattista, K. (2015). Repeatable texture sampling with interchangeable patches. 32.

[Kolar et al., 2017] Kolar, M., Debattista, K., and Chalmers, A. (2017). A subjective evaluation of texture synthesis methods. 36.

[Kwatra et al., 2005] Kwatra, V., Essa, I., Bobick, A., and Kwatra, N. (2005). Texture optimization for example-based synthesis. In *ACM Transactions on Graphics (TOG)*, volume 24. ACM.

[Kwatra et al., 2003] Kwatra, V., Schödl, A., Essa, I., Turk, G., and Bobick, A. (2003). Graphcut textures: image and video synthesis using graph cuts. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 277–286. ACM.

[Lafortune et al., 1997] Lafortune, E. P., Foo, S.-C., Torrance, K. E., and Greenberg, D. P. (1997). Non-linear approximation of reflectance functions. In *Proc. SIGGRAPH '97*, volume 31.

[Lagae and Dutré, 2006] Lagae, A. and Dutré, P. (2006). An alternative for wang tiles: colored edges versus colored corners. *ACM Transactions on Graphics (TOG)*, 25(4):1442–1459.

[Larson, 1998] Larson, G. W. (1998). Logluv encoding for full-gamut, high-dynamic range images. *Journal of Graphics Tools*, 3(1).

[Lasram and Lefebvre, 2012] Lasram, A. and Lefebvre, S. (2012). Parallel patchâĂŞbased texture synthesis. *High Performance Graphics*.

[Law and Siu, 1999] Law, N.-F. and Siu, W.-C. (1999). Progressive image coding based on visually important features. In *Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on*, volume 2. IEEE.

[Lee and Liu, 2005] Lee, S. and Liu, Y. (2005). Psu near-regular texture database. *URL http://vivid. cse. psu. edu/texturedb/gallery*, 17.

[Lefebvre and Hoppe, 2005] Lefebvre, S. and Hoppe, H. (2005). Parallel controllable texture synthesis. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 777–786. ACM.

[Li and Wand, 2016] Li, C. and Wand, M. (2016). Combining markov random fields and convolutional neural networks for image synthesis. *arXiv preprint arXiv:1601.04589*.

[Lin et al., 2004a] Lin, W.-C., Hays, J., Wu, C., Kwatra, V., and Liu, Y. (2004a). A comparison study of four texture synthesis algorithms on near-regular textures. In *ACM SIGGRAPH 2004 Posters*. ACM.

[Lin et al., 2004b] Lin, W.-C., Hays, J. H., Wu, C., Kwatra, V., and Liu, Y. (2004b). A comparison study of four texture synthesis algorithms on regular and near-regular textures. Technical Report CMU-RI-TR-04-01, Robotics Institute, Pittsburgh, PA.

[Liu et al., 2004] Liu, Y., Lin, W.-C., and Hays, J. (2004). Near-regular texture analysis and manipulation. In *ACM Transactions on Graphics (TOG)*, volume 23. ACM.

[Lous, 1990] Lous, Y. L. (1990). Report on the First Eurographics Workshop on Visualization in Scientific Computing. 9(4).

[Mandryk et al., 2006] Mandryk, R., Atkins, S., and Inkpen, K. (2006). A continuous and objective evaluation of emotional experience with interactive play environments. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM.

[McDermott and Webster, 2012] McDermott, K. and Webster, M. (2012). Uniform color spaces and natural image statistics. *JOSA A*, 29(2).

[Mckee, 1996] Mckee, L. D. (1996). Automated tile mosaic creation system. US Patent 5,568,391.

[McLaren, 1976] McLaren, K. (1976). XiiiÑthe development of the cie 1976 (l* a* b*) uniform colour space and colour-difference formula. *Journal of the Society of Dyers and Colourists*, 92(9).

[Mukherjee et al., 2016] Mukherjee, R., Debattista, K., Bashford-Rogers, T., Vangorp, P., Mantiuk, R., Bessa, M., Waterfield, B., and Chalmers, A. (2016). Objective and subjective evaluation of high dynamic range video compression. *Signal Processing: Image Communication*, 47.

[Myszkowski, 1998] Myszkowski, K. (1998). The visible differences predictor: Applications to global illumination problems. In *Rendering Techniques 98*. Springer.

[Neyret and Cani, 1999] Neyret, F. and Cani, M.-P. (1999). Pattern-based texturing revisited. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 235–242. ACM Press/Addison-Wesley Publishing Co.

[Nicoletti et al., 2002] Nicoletti, G. M., Notarnicola, B., and Tassielli, G. (2002). Comparative life cycle assessment of flooring materials: ceramic versus marble tiles. *Journal of Cleaner Production*, 10(3).

[Obert et al., 2012] Obert, J., van Waveren, J., and Sellers, G. (2012). Virtual texturing in software and hardware. In *ACM SIGGRAPH 2012 Posters*. ACM.

[Perlin, 1985] Perlin, K. (1985). An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3).

[Pietikäinen, 2000] Pietikäinen, M. (2000). *Texture analysis in machine vision*, volume 40. World Scientific.

[Pointer, 1981] Pointer, M. (1981). A comparison of the cie 1976 colour spaces. *Color Research & Application*, 6(2).

[Poirier et al., 2013] Poirier, T., Bertrand, P., and Coddet, C. (2013). Colour matching in decorative thermally sprayed glass coatings. *Journal of thermal spray technology*, 22(1).

[Portilla and Simoncelli, 2000] Portilla, J. and Simoncelli, E. P. (2000). A parametric texture model based on joint statistics of complex wavelet coefficients. *International journal of computer vision*, 40(1).

[Praun et al., 2000] Praun, E., Finkelstein, A., and Hoppe, H. (2000). Lapped textures. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques SIGGRAPH 00*, (1).

[Robertson, 1977] Robertson, A. R. (1977). The cie 1976 color-difference formulae. *Color Res. Appl.*, 2.

[Salkind, 2010] Salkind, N. (2010). *Encyclopedia of research design*, volume 1. Sage.

[Sánchez et al., 2010] Sánchez, E., García-Ten, J., Sanz, V., and Moreno, A. (2010). Porcelain tile: almost 30 years of steady scientific-technological evolution. *Ceramics International*, 36(3).

[Schwarzkopf, 2012] Schwarzkopf, H. (2012). Public domain photograph.

[Seshadrinathan et al., 2010] Seshadrinathan, K., Soundararajan, R., Bovik, A. C., and Cormack, L. (2010). Study of subjective and objective quality assessment of video. *IEEE transactions on image processing*, 19(6).

[Sheikh and Bovik, 2006] Sheikh, H. R. and Bovik, A. C. (2006). Image information and visual quality. *Image Processing, IEEE Transactions on*, 15(2).

[Sheikh et al., 2006] Sheikh, H. R., Sabir, M. F., and Bovik, A. C. (2006). A statistical evaluation of recent full reference image quality assessment algorithms. *Image Processing, IEEE Transactions on*, 15(11).

[Sonka et al., 2014] Sonka, M., Hlavac, V., and Boyle, R. (2014). *Image processing, analysis, and machine vision*. Cengage Learning.

[Specification, 2004] Specification, I. (2004). 1: 2004-10 (profile version 4.2. 0.0). *International Color Consortium*.

[Taibo et al., 2009] Taibo, J., Seoane, A., and Hernández, L. (2009). Dynamic virtual textures. *Journal of WSCG*, 17(1-3):25–32.

[Thompson et al., 2011] Thompson, W., Fleming, R., Creem-Regehr, S., and Stefanucci, J. K. (2011). *Visual perception from a computer graphics perspective*. CRC Press.

[Vanhoey et al., 2013] Vanhoey, K., Sauvage, B., Larue, F., and Dischler, J.-M. (2013). On-the-fly multi-scale infinite texturing from example. *ACM Transactions on Graphics (TOG)*, 32(6).

[Wang et al., 2004] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4).

[Wei, 2004] Wei, L.-Y. (2004). Tile-based texture mapping on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. ACM.

[Wei et al., 2009] Wei, L.-Y., Lefebvre, S., Kwatra, V., Turk, G., et al. (2009). State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR*, pages 93–117.

[Wei and Levoy, 2000] Wei, L.-Y. and Levoy, M. (2000). Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*.

[Wei and Levoy, 2002] Wei, L.-Y. and Levoy, M. (2002). Order-independent texture synthesis. Technical report, TR 2002.

[Wyszecki and Stiles, 1982] Wyszecki, G. and Stiles, W. S. (1982). *Color science*. Wiley New York.

[Xue et al., 2007] Xue, F., Zhang, Y.-S., Jiang, J.-L., Hu, M., Wu, X.-D., and Wang, R.-G. (2007). Real-time texture synthesis using s-tile set. *Journal of Computer Science and Technology*, 22(4):590–596.

[Yang et al., 2015] Yang, J., Reed, S., Yang, M.-H., and Lee, H. (2015). Weakly-supervised disentangling with recurrent transformations for 3d view synthesis. In *Advances in Neural Information Processing Systems*, pages 1099–1107.

[Zhang and Wandell, 1996] Zhang, X. and Wandell, B. (1996). A spatial extension of cielab for digital color image reproduction. In *SID international symposium digest of technical papers*, volume 27, pages 731–734.