Diplomarbeit

Constrained Random Sampling and Gap Filling Technique for Near-regular Texture Synthesis

Diego López Recas

July 28, 2011

Technische Universität Berlin
Fakultät IV: Fakultät Elektrotechnik und Informatik
Institut für Technische Informatik und Mikroelektronik
Computer Vision and Remote Sensing

Betreuender Hochschullehrer:	Prof. Dr.-Ing. Olaf Hellwich
Betreuender Mitarbeiter:	Dipl.-Ing. Anna Hilsmann (Fraunhofer HHI)

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig erstellt und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Berlin, den July 28, 2011

Diego López Recas

Acknowledgements

I am grateful to all authors that have worked on texture synthesis before us and whose publications have inspired our work, especially Efros and Freeman's and Liu et al.'s.

I would also like to thank Technische Universität Berlin, Fraunhofer HHI and Peter Eisert (head of Computer Vision & Graphics Group of Fraunhofer HHI) for giving me the opportunity to work with them. Thanks as well to Olaf Hellwich for his help and effort made to finally evaluate my work.

A warm thanks goes to my family, closest friends and especially my girlfriend for their unconditional support, love and a great amount of patience. I could not have done it without them.

And finally, I cannot forget that I would probably have failed to write this thesis without Anna Hilsmann. Her constant support, encouragement, motivation and technical advice has mean everything to me. Thank you very much.

Contents

List of Figures

1 Introduction

Textures have traditionally been used in computer graphics to enhance the appearance of scenes, e.g., texture mapping provides an inexpensive way of representing surface detail for special effects, image editing, content creation, rendering, and animation. In most cases, textures are taken from pictures of the real world or hand-drawn textures and are wanted to cover surfaces for wich the input texture does not fit well or is simply too small and an "enlargement" is needed.

The objective of *texture synthesis* is to generate an arbitrarily sized image that reproduces the texture of a relatively small sample image. That is, the output should be a "new" image that is different from the input image but a human eye perceives them containing the same texture. This thesis presents a novel texture synthesis approach for *near-regular textures*.

During the last years, many researchers in computer vision and computer graphics have proposed methods for texture synthesis and achieved impressive results for many kinds of textures. Especially successful and relatively simple approaches are those commonly classified as *non-parametric example-based* texture synthesis techniques [WLKT09]. These non-parametric methods are based on the idea of applying the Markov Random Field (MRF) model to textures, thus supposing that they have stationary and local statistics and that any pixel of the texture is fully characterized by its neighbourhood around. Then, the output is composed by sequencially selecting pixels whose neighbourhood agrees with the already synthesized part of the output. In this way, the output is generated by sampling the input texture example.

Non-parametric example-based methods have very good results for many different types of texture. However, the so-called *near-regular textures* [LTL05] have been specially difficult to reproduce faithfully. There are many examples of this type of near-regular textures, such as brick walls, tiled floors, carpets, woven sheets, where the texture patterns (each brick, tile, straw or bamboo strip) vary only locally. Our work is focused on this specific kind of textures where a global regular structure coexists with subtle yet characterisitic stochastic deviations from regularity.

Figure 1.1 [LLH04] shows examples of textures with different levels of regularity. Mathematically speaking, *regular texture* refers to periodic patterns that present non-trivial translation symmetry. *Near-regular texture* is referring to textures that are not strictly symmetrical. The irregularity can be caused by various statistical departures from regular textures. Liu et al. [LLH04] proposed a categorization of near-regular textures depending on the nature of the deformations, whether the deviations are in geometry or in color (see Figure 1.2). The focus of this thesis is on faithful texture synthesis of near-regular textures where departure from regularity is primarily caused by statistical color and intensity variations, while the underlying structural regularity

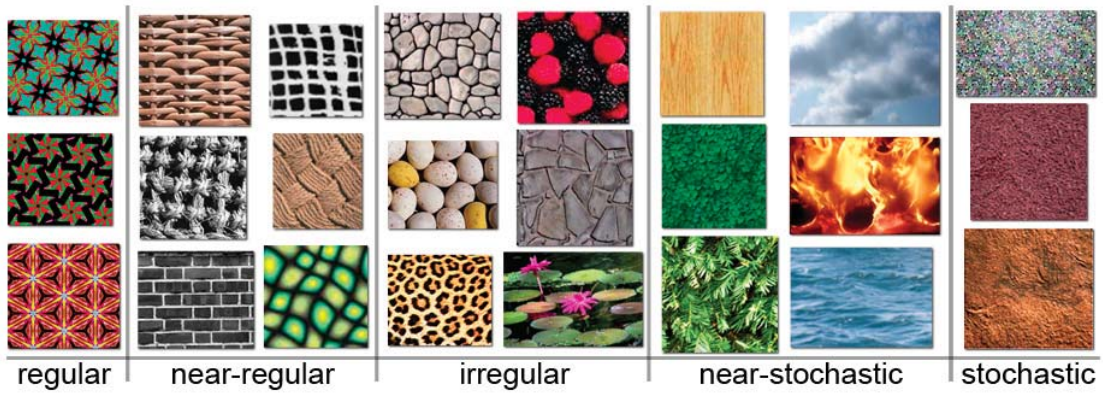


Figure 1.1: Texture spectrum.

Type	Geometry	Color	Symbols
0	**R**egular	**R**egular	GRCR
I	**R**egular	**I**rregular	GRCI
II	**I**rregular	**R**egular	GICR
III	**I**rregular	**I**rregular	GICI

Figure 1.2: Categorization of Near-regular Textures from [LLH04].

remains (Type I in Figure 1.2).

The synthesis of near-regular textures is especially troublesome because two very different properties coexist. The regular part is periodic, deterministic and *global*, whereas the irregular part is stochastic with *local* statistics. This makes non-parametric neighbourhood-based methods often fail to reproduce the large-scale global structure of the input texture, whereas a simple tiling approach is unable to introduce the characteristic randomness of the irregularities and the output would look rather unnatural.

In order to overcome these difficulties, we propose to synthesize near-regular textures in a *constrained random sampling* approach. In a first analysis step, we treat the texture as regular and analyze the global regular structure of the input texture sample to estimate two translation vectors defining the translation symmetry of the texture under analysis. In a subsequent synthesis step, this structure is exploited to *guide* or *constrain* a random sampling process so that random samples of the input are introduced into the output preserving the regular structure previously detected. This ensures the stochastic nature of the irregularities in the output yet preserving the regular pattern of the input texture.

Although our method was developed for near-regular textures we observed that it produces also very good results for irregular and stochastic textures if the analysis step is skipped.

The remainder of this work is structured as follows:

- Chapter 2 gives an overview of related work and our contribution. An overview of several existing non-parametric example-based synthesis techniques is presented

along with some reported or potential weaknesses. The chapter is concluded with an introduction to the kind of improvements that our technique is intended to introduce.

- Chapter 3 describes the analysis step that we propose to estimate the regular structure (or lattice) of the texture sample. By an observation of the local maxima distribution of a normalized autocorrelation of the input texture image, two independent translation vectors defining the translational symmetry of the input texture are estimated.

- Chapter 4 focuses on the synthesis stage. In Section 4.1, a method to find the best self-similar tile within the input is presented. A simple tiling of the best self-similar tile produces the best results for regular textures and helps to illustrate that the output looks unnatural if there are no random irregularities in the case of near-regular textures. On the other hand, Section 4.2 explains the alternative *constrained random sampling and gap filling* synthesis method that does introduce randomness in the output and enhances the natural appearance of the result.

- Chapter 5 discusses the performance, advantages and disadvantages of the proposed method and gives several examples and comparisons with other approaches.

- A conclussion and a discussion and thoughts about future work is presented in Chapter 6.

2 Related Work and Contribution

Texture analysis and synthesis has had a long history in psychology, statistics and computer vision. During the last years, it has been devoted significant work from researchers in the areas of computer graphics and computer vision.

The objective of texture synthesis is to generate images that reproduce a distribution of textural features which humans perceive as a specific type of texture. For a limited class of textures this distribution can be modeled using for example Perlin Noise [Per85] or reactiondiffusion systems [Tur91]. These types of procedural texture synthesis offer the advantage of user control and extremely compact representation. Statistical modeling is applicable to more general types of texture. Motivated by research on human texture perception, they mostly use statistics of filter response vectors. The actual synthesis is performed by iteratively matching statistics of a sample texture and the synthesized result. Heeger and Bergen [HB95], for example, matched marginal histograms of filter response vectors at different spatial scales. Follow-up publications [dB97, PS00, BJEYLW01] improved upon this scheme by enforcing more complex joint statistics of filter coefficients but still fail on highly structured textures. Few publications (e.g. [ZWM98]) propose parametric texture models based on the Markov Random Field model of the texture. Texture synthesis involves fitting the model to a sample texture and sampling from the resulting distribution which can be computationally very expensive and still reproduces mainly stochastic textures only. These elaborate models are outperformed in speed, quality and applicability by simple non-parametric sampling that was first proposed in the seminal paper by Efros and Leung [EL99] and many other publications improved on the idea (e.g. [WL00, TZL$^+$02, ZG02]).

Our texture synthesis technique is closely related to the last mentioned idea, for which there are numerous examples. In this chapter, we try to give an overview of the existing non-parametric sampling approaches. The inclined reader can follow references in the computer vision literature [LM99, LM01, ZWM97, ZWM98, ZGWW02] to get an overview of other existing work, which we do not further discuss. We list the work most relevant to ours in the following loose classification.

2.1 Pixel-Based Texture Synthesis

Pixel-based texture synthesis algorithms are generally based on the theory of Markov Random Fields (MRF's), a two-dimensional extension to Markov Chains, inspired by Shannon's work on modelling the English language using *n-grams* [SW63]. Using MRF's, a texture is modelled as a *local* and *stationary* random process: each pixel is classified by a small set of neighboring pixels (local causality) and this classification is the same for all pixels (stationary). In this context, Wei and Levoy [WL00] give a very nice description on the difference between images and textures as depicted in Figure 2.1.

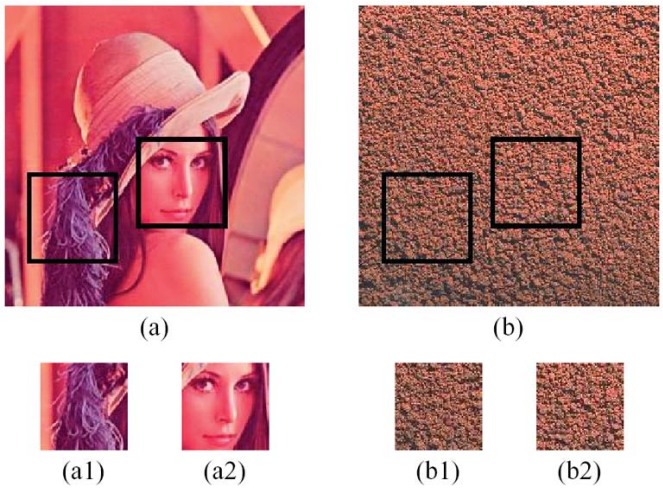


Figure 2.1: How textures differ from images. (a) is a general image while (b) is a texture. A movable window with two different positions are drawn as black squares in (a) and (b), with the corresponding contents shown below. Different regions of a texture are always perceived to be similar (b1,b2), which is not the case for a general image (a1,a2). In addition, each pixel in (b) is only related to a small set of neighboring pixels. These two characteristics are called stationarity and locality, respectively. Image and caption taken from [WL00].

For a full MRF realization of texture analysis and synthesis, an explicit probability distribution must be constructed from the input texture, and then sampled by the synthesizer. This process is computationally expensive, both in size and speed, which is why state-of-the-art pixel-based synthesis algorithms prefer a non-parametric approach. Therein, new pixels are synthesized based solely on already synthesized regions by maintaining local similarity, and no explicit probability distribution is needed.

Efros and Leung [EL99] pioneered this approach with their non-parametric sampling. They synthesize a texture I_{out} by repeatedly matching the neighborhood around the target pixel in the synthesis result with the neighborhood around all pixels in the input texture I_{in}, starting from a seed pixel and growing outwards. For each to-be-synthesized pixel p_{out} and its neighborhood of already synthesized pixels $\omega(p_{out})$, an approximation to the conditional probability $P(p_{in}|\omega(p_{out}))$ is constructed for each $p_{in} \in I_{in}$. This is achieved by computing a gaussian weighted, normalized sum of square differences (SSD) between $\omega(p_{out})$ and the pixel neighborhoods of each candidate in the input texture, $\omega(p_{in})$. A target pixel is then selected from a set of pixels p_{in} with high conditional probability. The algorithm performs an exhaustive search in I_{in} for each synthesized pixel and is therefore quite slow. Also, the algorithm has a tendency to *slip* into the wrong part of the search space and start *growing garbage* [EL99] or to perform verbatim copying of the input.

Wei and Levoy [WL00] introduced some significant changes to enhance both quality and speed of Efros and Leung's [EL99] work. In Efros and Leung's approach, the pixel

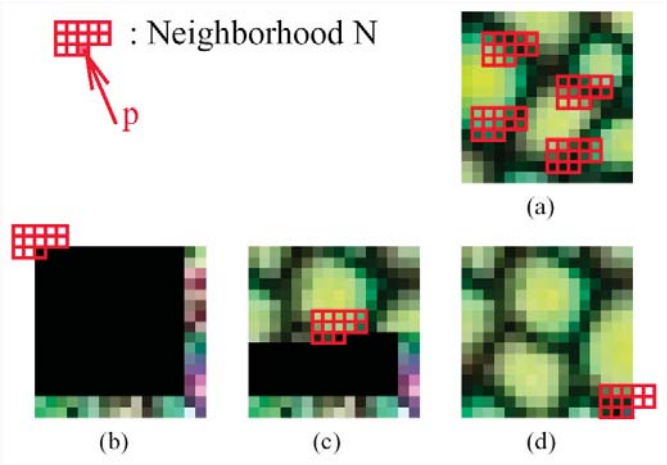


Figure 2.2: Wei/Levoy single resolution texture synthesis. (a) is the input texture and (b)-(d) show different synthesis stages of the output image. Pixels in the output image are assigned in a raster scan ordering. The value of each output pixel p is determined by comparing its spatial neighborhood N(p) with all neighborhoods in the input texture. The input pixel with the most similar neighborhood will be assigned to the corresponding output pixel. Neighborhoods crossing the output image boundaries (shown in (b) and (d)) are handled toroidally. Although the output image starts as a random noise, only the last few rows and columns of the noise are actually used. For clarity, we present the unused noise pixels as black. (b) synthesizing the first pixel, (c) synthesizing the middle pixel, (d) synthesizing the last pixel. Image and caption taken from [WL00].

neighborhood of already synthesized pixels in the output image is not known a priori, since they grow texture from a single seed pixel outwards. Wei and Levoy start off with a white random noise output image and then iterate through it in scanline order. At each iteration, the pixel in the input texture is picked, which best matches the L-shaped, fixed neighborhood around the current to be synthesized pixel (Figure 2.2). This results in their *seed* being the last few rows and columns of white random noise. The algorithm makes excellent use of the fixed neighborhood size by interpreting all possible neighborhoods in the input texture as a set of 1D vectors (each vector is an ordered concatenation of RGB triples) and preprocessing these high dimensional neighborhood vectors using tree structured vector quantization (TSVQ). This preprocess results in logarithmic complexity for each best-pixel-search and an overall speedup by two orders of magnitude compared to Efros and Leung's algorithm, at the price of some artifacts. The algorithm is furthermore extended to a multiresolution synthesis pyramid, progressing from coarse to fine. This results in smaller search neighborhoods, with synthesis quality comparable to using larger neighborhoods with single resolution synthesis. The reason for this is that larger, low frequency features are captured in low pyramid resolutions and that these lower resolution pixels constrain the added high frequency features to be consistent with the already synthesized low frequency structure.

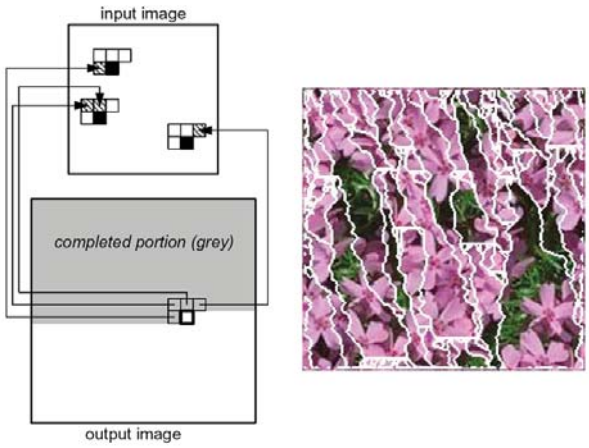


Figure 2.3: Ashikhmin's algorithm. Left: candidate pixels for the algorithm (the Ashikhmin Set 2.3). Each pixel in the current L-shaped neighborhood generates a *shifted* candidate pixel (black) according to its original position (hatched) in the input texture. The best pixel is chosen among these candidates only. Several different pixels in the current neighborhood can generate the same candidate. Right: region-growing nature of the algorithm. Boundaries of texture pieces are marked white on the right. Images and captions taken from [Ash01].

Ashikhmin [Ash01] introduced and intelligent modification to significantly reduce search space and achieve interactive framerates. He realized that, at a given step in the Wei/Levoy synthesis process, pixels in the input sample with neighborhoods similar to the shifted current neighborhood in the output image have already been found. Exploiting this observation leads to an algorithm, which encourages verbatim copying (or *region growing* as stated in [Ash01]) to a certain degree (Figure 2.3), as the candidate set (we call this set the Ashikhmin Set or A_s) for each pixel is very small compared to Wei/Levoy synthesis (at most $size(A_s) = (n2 - 1)/2$ with n being the corner length of the L-shaped neighborhood in pixels). For each pixel, the set A_s is constructed as outlined in Figure 2.3, which requires the storage of an additional source map with source locations of already synthesized pixels. Synthesis runs at interactive rates, allows user control (texture transfer with a user-provided target image) and works well for a class of textures titled natural textures [Ash01], where pure Wei/Levoy synthesis shows a tendency to blur out small objects [Ash01]. Note that this blurring is not problematic when using input textures with low color variance (in the extreme case, a binary image), since Wei/Levoy synthesis only chooses pixels which exist in the input texture [Ash01].

Merging both Ashikhmin and Wei/Levoy synthesis into a framework, **Hertzmann et al.** [HJO+01] introduce the problem statement titled Image Analogies: given a pair of images A and A' (the unfiltered and filtered source images, respectively) along with an unfiltered target image B, synthesize a new filtered target image B' such that B' relates to B in the same way A' relates to A [HJO+01]. Texture synthesis then reduces

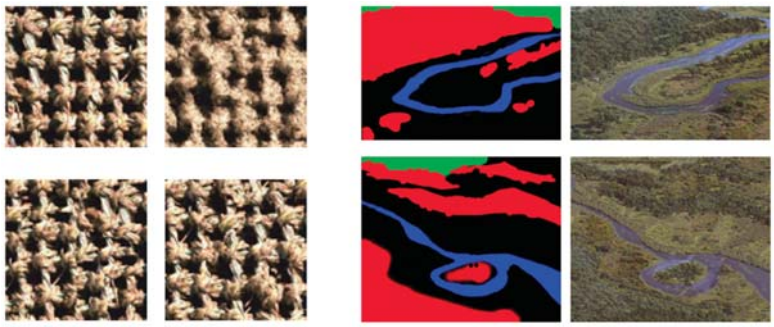


Figure 2.4: Some Image Analogies results. Left (weave): applying the framework to simple texture synthesis. The input weave texture (upper left) and results of Wei/Levoy (upper right), Ashikhmin (lower left), and Image Analogies (lower right). Right: texture-by-numbers. The pseudocolored texture area-map A (upper left), the original scene A' (upper right), the new pseudo-colored texture area-map B (bottom left, created interactively by the user) and the resulting new scene B' (bottom right). Images taken from [HJO+01].

to a trivial case of image analogies, where the images A and B are zero-dimensional or constant, and B' is synthesized from the input texture A'. By using both approximate nearest neighbors (ANN [AMN+95]) and coherence (Ashikhmin [Ash01]) search, their framework manages to combine the best of both worlds, as seen in the results of the weave texture in Figure 2.4, left. One of the most interesting applications attributed to Image Analogies is *texture-by-numbers*, where an unfiltered, pseudo-colored texture area-map A is created from the original image A', and thereafter used to allow the user to interactively paint a new, pseudo-colored scenario B, from which the algorithm generates the new scene B' (Figure 2.4, right).

Zelinka and Garland [ZG02] create a datastructure from the input texture in a preprocess, similar to *Video Textures* [SSSE00]. They term this datastructure *jump map* and use it to synthesize texture per-pixel in real-time. The jump map stores a set of k-nearest (pixel) neighbors (in feature space) for each pixel in the input texture. Each pixel in this set is a minimum distance away (in image space) from all other pixels in the set, analogous to poisson disk sampling, thus ensuring diversity in the synthesis result. During per-pixel synthesis (using various pixel orderings), the algorithm simply performs a random walk through the jump map, resulting in algorithmic complexity linear in the number of output pixels, and therefore realtime performance. Judging from the results in their paper, the synthesis quality is generally inferior to Ashikhmin's algorithm [Ash01].

2.2 Patch-Based Texture Synthesis

Patch-based texture synthesis methods preserve global structure by generating the texture on a per-patch basis, and then (in most cases) attempt to repair the patch

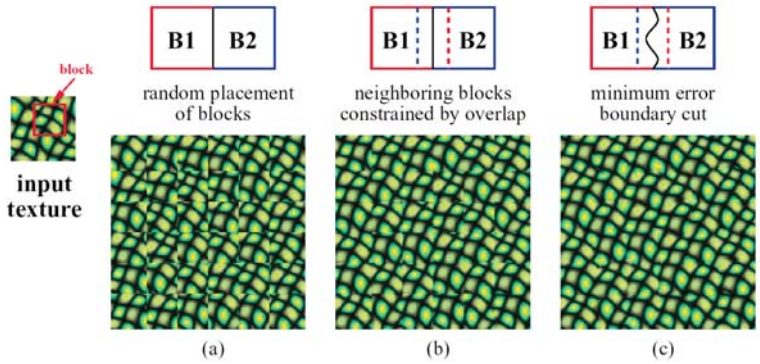


Figure 2.5: Image Quilting. Square blocks from the input texture are patched together to synthesize a new texture sample: (a) blocks are chosen randomly, (b) the blocks overlap and each new block is chosen so as to agree with its neighbors in the region of overlap, (c) to reduce blockiness, the boundary between blocks is computed as a minimum cost path through the error surface at the overlap. Image and caption taken from [EF01].

overlap regions using different strategies.

Efros and Freeman [EF01] proposed a patch-based technique called Image Quilting (IQ). IQ iterates through a uniform quadrilateral grid of patches, which, combined, resemble a tiling of the output texture. In scanline order, the algorithm selects, for each output patch, a congruent patch of pixels from the input texture, constrained by overlap with the already synthesized result. It then performs a minimum-error-boundary-cut (MEBC) within the overlap region of adjacent texture patches to reduce artifacts (Figure 2.5).

The MEBC is implemented by employing dynamic programming (the authors mention that Dijkstra's algorithm would also do the job). The algorithm is also well suited for *texture transfer*, which is demonstrated in the paper. Synthesis results presented in [EF01] are equal to or better than Efros/Leung-like, pixel-based algorithms. Still, as also pointed out by Liang et al. [LLX+01], hard color changes along patch boundaries, termed *boundary mismatch*, tend to occur.

Liang et al.'s Patch-Based Sampling [LLX+01] (PBS) uses the same technique as IQ for patch placement, but simply alpha-blends the overlap regions (feathering), as their primary concern is speeding up the algorithm to real-time performance. They also prefer *blurring* artifacts to IQ's *boundary mismatch* artifacts. The uniform patch sampling size, and therefore small set of overlap cases, gives way to an input texture preprocess, using an optimized kd-tree, a quadtree pyramid and principal component analysis (PCA) for feature vector dimension reduction. This accelerates the entire algorithm to real-time performance at negligible visual drawback. Blurring artifacts along patch boundaries do remain a problem though, especially in the presence of high frequency features.

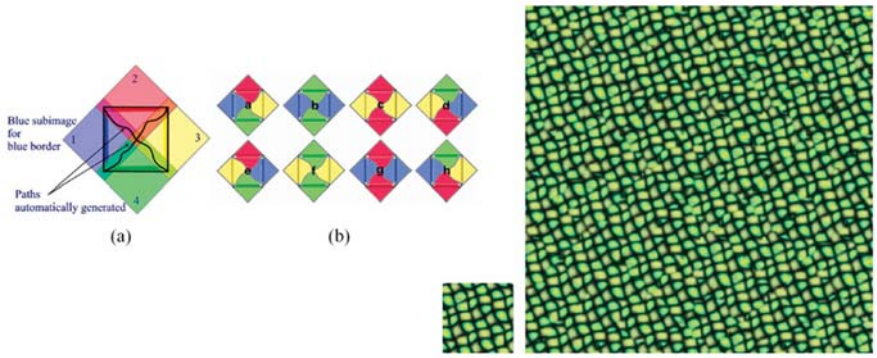


Figure 2.6: Wang Tile Textures. Left: (a) four subimages are combined to form each Wang Tile; (b) construction of an eight tile set. Right: a texture using 18 Wang Tiles. Images and captions taken from [CSHD03].

Kwatra et al. [KSE+03] developed a generalization of IQ called Graphcut Textures (GCT). The uniform patch sizes used in IQ and PBS is generalized to arbitrarily shaped patches. Their shapes are determined entirely by performing a minimum cost graph-cut with the underlying (partially synthesized, or perhaps already fully synthesized) result. The process is iterative and can therefore correct badly matched seams (with high cost) by pasting a new patch from the input texture (constrained by overlap) over the visually displeasing seam and repeating the graph-cutting process.

Cohen et al.'s Wang Tile Textures [CSHD03] use principles of provably non-periodic tilings of the plane to generate arbitrary amounts of non-repetitive texture (see the seminal work on aperiodic tilings by Grünbaum and Shepard [GS86]).

Their tile generation procedure randomly selects a set of base tiles from the input texture, one tile for each Wang Tile edge color, and constructs Wang Tiles by assembling the necessary 4-permutations of the base tileset (Figure 2.6, left). For each 4-permutation, the overlap region is repaired by performing a minimum-error-boundary-cut (MEBC [EF01]). If the resulting Wang Tileset is below a visual quality threshold (i.e. artifacts along the MEBC), a new base tileset is selected and the assembly procedure is repeated (optimization). Still, we assume as a result of the random selection process, some diamond-shaped artifacts might appear (Figure 2.6, right).

2.3 Texture Synthesis over Surfaces

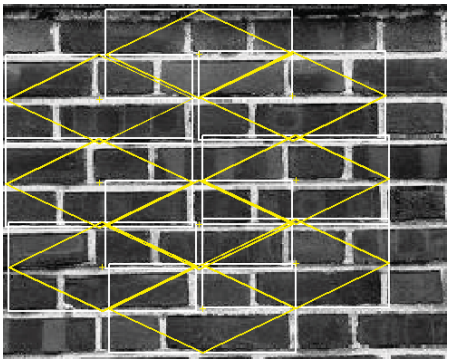
The synthesis methods previously introduced work in 2D, i.e. the output is a plain image that will tipically be deformed later to cover a 3D surface. An alternative is to directly synthesize over the 3D surface itself. We show here relevant examples of this methodology as most surface texture synthesis methods are direct extensions of pixel-based [Tur01, WL01, YHBZ01] or patch-based [PFH00, SCA02] algorithms.

Figure 2.7: Surface synthesis results. Wei/Levoy [WL01] (left), Ying et al. [YHBZ01] (center, multiscale synthesis) and Soler et al. [SCA02] (right). Each image taken from the respective publication.

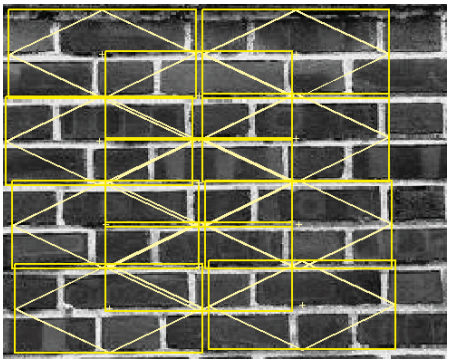
Turk's [Tur01] (TS) and **Wei and Levoy**'s [WL01] (WLS) surface synthesis methods both densely tessellate the 3-dimensional input mesh of an object using Turk's re-tiler [Tur92] and then perform a per-vertex color synthesis. These two approaches are very similar (both use a multiresolution mesh hierarchy), yet have three quite significant differences. (1) WLS uses both random and symmetric vector fields, whereas TS always uses a user defined, smooth vector field. (2) TS uses a sweeping order derived from the smooth vector field for vertex traversal, WLS visits the mesh vertices in random order. (3) TS uses surface marching to construct the mesh neighborhood, while WLS performs flattening and resampling of the mesh. Results of the two methods are comparable in quality. A texture synthesized on the Stanford bunny using WLS can be viewed in Figure 2.7, left.

Ying et al. [YHBZ01] worked on overcoming the drawbacks of surface marching methods (such as [Tur01]): (1) the sampling pattern is not guaranteed to be even in the presence of irregular geometry, (2) the sampling is numerically unstable, as small surface variations can cause large variations in the pattern and (3) the method is slow due to many geometric intersection and projection operations. In their approach, texture is synthesized on surfaces per-texel using a texture atlas of the polygonal mesh (a collection of rectangular domains U_i on which the surface is smoothly parameterized) and a common planar domain, the *chart* V, from which neighborhood sample positions in the domains U_i are gathered. These positions in the U_i then correspond to the neighborhood on the original surface along a previously defined, orthogonal unit tangent vector field. They apply both Wei/Levoy and Ashikhmin per-pixel synthesis strategies with convincing results (Figure 2.7, middle).

Praun et al.'s Lapped Textures [PFH00] extend the chaos mosaic [XGS00] to surfaces with a pre-computed vector field to direct anisotropy. In their system, the user specifies a tangential vector field over the surface, controlling texture scale and orientation. A (possibly irregular) input texture sample is then repeatedly pasted onto the surface by growing a surface patch and parameterizing it in texture space. The parametrization is optimized (by solving a sparse linear system) such that the vector field aligns with the frame of the texture patch. They render the resulting model both with a generated texture atlas and by runtime-pasting, the latter significantly reducing



(a) Sample set of selected user-aligned tiles (T).



(b) Sample set of selected half-way shifted tiles (T_h).

Figure 2.8: Sample tiles of The Promise and Perils of Near-regular Texture. The sample tiles are shown (rhombic shaped tiles are *minimum tiles* and rectangle shaped tiles are *maximum tiles*), they are carved from the input brick texture. (a) and (b) show the two different lattice positions. Images taken from [LTL05].

texture memory at the cost of rendering some faces multiple times.

Soler et al. demonstrated in Hierarchical Pattern Mapping (HPM) [SCA02] how a mesh can be seamlessly textured with only the input texture and a set of texture coordinates for each vertex. They set up a hierarchy of face clusters for the input mesh and then, for each cluster, (1) flatten it (if distortion is too high, the cluster is subdivided for later processing), (2) find a texture patch in the example texture for the flattened cluster which best matches already textured neighbors (if the error due to texture discontinuities with the existing neighbors is too high, the cluster is subdivided for later processing), (3) if (1+2) pass, map texture coordinates onto all polygons in the cluster. Unlike previous methods, HPM does not use a vector field, instead letting possible texture anisotropy propagate itself (Figure 2.7, right). Similar to Lapped Textures runtime-pasting [PFH00], the texture memory overhead for rendering is minimal, and additionally, all faces are rendered only once.

2.4 Near-Regular Texture Synthesis

A few authors have proposed specialized methods for synthesizing a specific type of textures known as Near-Regular Textures (see techreport [LHW+04] to have an overview, for example). The work of this thesis is as well focused on the synthesis of this particular kind of textures. These textures are ubiquitous in the real world, e.g. brick walls, tiled floors, carpets and woven sheets fall in this category where a dominant global structure or pattern (each brick, tile, straw or bamboo strip) varies only locally.

Liu et al.'s The Promise and Perils of Near-regular Texture [LTL05] focuses on the synthesis of near-regular textures whose local irregularities are mainly color deviations

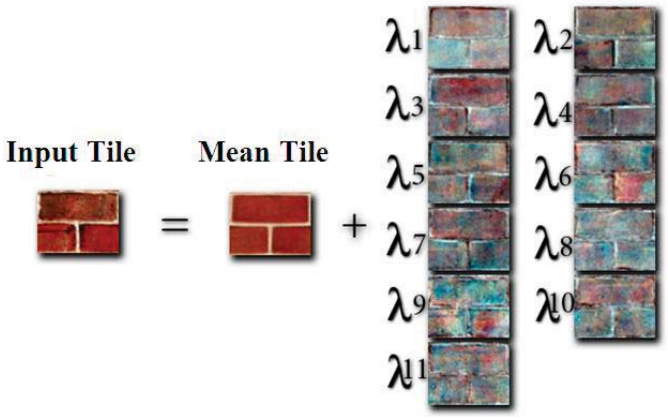


Figure 2.9: Color deformation field as modeled in Near-Regular Texture Analysis and Manipulation [LLH04]. In this example, each input tile can be represented as a linear combination of its mean tile with the top 11 PCA bases. The blue colors on PCA bases reflect negative values. Images taken from [LLH04].

and no important geometric deformation of the global pattern exists. Similar to ours, they analyze the input texture image to estimate the translational symmetry of the input texture by using a correlation-based method proposed in [LCT04]. After a user assisted alignment of the detected lattice, two sets of *maximum tiles* are identified within the input: aligned maximum tiles (T) and half-way shifted maximum tiles (T_h) (Figure 2.8). Starting with one randomly selected maximum tile, the output is composed then by sequentially stitching maximum tiles in the directions of the detected lattice. Each new maximum tile is alternatively selected from T or T_h and pasted at lattice points or half-way shifted lattice points respectively. The selection is constrained by overlapping and the selected candidate is registered with a correlation-based method such that small movements around the current lattice point are possible. At each step, the current tile is stitch to what has already been synthesized in a similar manner to [EF01].

Liu et al. further worked on near-regular textures and developed a multimodal framework to treat a wider range of near-regular textures in Near-Regular Texture Analysis and Manipulation [LLH04]. They extend the idea developed in [LL03] to address geometry, lighting and color deviations from regularity.

User assistance helps detect a coarse irregular lattice from which a geometric deformation field d_{geo} is inferred and extracted to obtain a regularized version of the input texture (flattened texture). Tsin et al.'s algorithm [TLR01] is then applied to this flattened version to estimate a lighting deformation field d_{light} that is afterwards mapped back to the original input texture applying the inverse deformation field. The previous deformation and lighting models are extracted from the input and individual tiles (pattern units) are identified to estimate a color deformation field from a PCA analysis. The color deformation field is modeled with the mean tile and a set of PCA basis that are representative tile color deviations (Figure 2.9). In the synthesis stage, the geometry and lighting deformation fields are related. A geometric deformation field

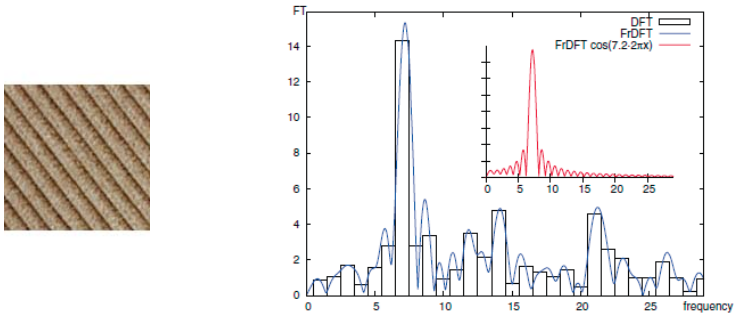


Figure 2.10: Comparison of absolute DFT and FrDFT coefficients (right) for the first scanline of a Corduroy texture sample (left) after subtraction of mean value. The dominant regular structure is caused by frequency 7.2. The small plot shows the FrDFT of the function $\cos(7.2 \cdot 2\pi x)$. Images taken from [NMMK05].

synthesis algorithm [LL03] is applied to synthesize D_{geo} from d_{geo} first, then image analogies [HJO+01] is used to synthesize the lighting deformation field D_{light} with $A = d_{geo}$, $A' = d_{light}$, $B = D_{geo}$ and $B' = D_{light}$. Finally, the color deformation field is synthesized by sampling the multidimensional space defined by the PCA basis as axis.

Nicoll et al. [NMMK05] used the concept of fractional Fourier analysis to perform an automatic separation of the global regular structure from the irregular structure. The actual synthesis is performed by generating a fractional Fourier texture mask from the extracted global regular structure which is used to guide the synthesis of irregular texture details.

First, they separate the dominant regular structure from irregular texture detail using the fractional Fourier transform (FrDFT, see Figure 2.10) and an intensity filter. This allows them to generate a *fractional Fourier texture mask* (FFTM) (procedural texture for the regular part), which is derived by "enlarging" the regular structure obtained from the fractional Fourier analysis to a desired size. That is, the FrDFT analysis identifies a set of dominant fractional frequency pairs $b_1, \ldots, b_n$ and their corresponding coefficients $F_1, \ldots, F_n$ that synthesize the FFTM by using the inverse DFT formula in a process known as Fourier synthesis [WW91]. The addition of irregular texture detail is finally done by an extended version of either pixel-based or patch-based texture synthesis algorithms.

2.5 Contribution

Our work is focused on the improvement of the synthesis of near-regular textures (NRT Type I, see Figure 1.2), as we see that this kind of textures especially pose problems to exisiting synthesis methods.

Pixel-based texture synthesis approaches are generally unable to capture global largescale structures and fail to reproduce the regularity of NRTs. In addition, the synthesis is sequential, hence newly synthesized pixels are dependent of what has been synthesized before. This may sometimes lead to garbage accumulation problems if the

process "slips" into a wrong part of the search space [EL99]. Furthermore, synthesizing one pixel at a time may cause blurriness in the output.

Patch-based texture synthesis techniques base largescale pattern reproduction on taking groups of contigous pixels as the sampling unit instead of individual pixels and constraining the selection of each new group (or patch) by overlapping with the already synthesized part of the output. However, this does not generally ensures faithful gobal regular structure reproduction [LTL05]. Moreover, although garbage accumulation problems of pixel-based methods are reduced, they report the appearance of boundary missmatch artifacts that may be propagated to new patches in our experience. Furthermore, new patches are still dependent on the previously synthesized output and this may also cause visible repetitions (poor randomness of the irregularities).

Texture synthesis approaches specialized in near-regular textures generally require an important amount of user intervention and specific tuning [LLH04, LTL05, NMMK05]. In addition, they are still sequential, what we see as an incovenience because errors may propagate across the output and visible repetitions may appear if no especial care is taken. Nicoll et al. [NMMK05] also report that fractional Fourier texture masks suffer a degeneration if the extracted frequencies are not completely accurate, what in our experience is always if the output is big enough.

In this thesis, we focus on the improvement of two of the main drawbacks of existing synthesis approaches for near-regular textures. On the one hand, we propose a lattice estimation method that works successfully in most cases with no user intervention at all. On the other hand, we exploit the estimated lattice to break the tradicional sequencial approach of the synthesis process: independent input samples are sparsely introduced into the output to ensure that local errors are not futher propagated and that the irregularities of the texture are stochastically rich. Again, special attention is given to avoid user intervention in the synthesis process as its configuration is generally automatizable. Following sections 3 and 4 explain our methodology in detail.

3 Regular Structure Detection (Analysis)

This thesis describes a example-based synthesis approach for a specific kind of textures referred to as *near-regular textures* [LTL05]. According to Liu et al. [LLH04], a near-regular texture (NRT) can be categorized according to Figure 1.2. The type of NRT we are interested in is Type 1, where a regular structure (i.e. an identifiable repeated pattern) is combined with stocastic deviations from regularity that are mainly photometric rather than geometric. Examples of this type of texture are frequently found in the real world: most textiles (e.g. used for clothing, furniture, or car interiors) and construction elements (walls, floors, grid structures, corrugated sheet roofs) fall into this category, and yet they are are still very difficult to synthesize faithfully. Many existing example-based methods typically focus on preserving local properties and fail to reproduce the large-scale global structure of the texture as reported in [LTL05].

In order to avoid this, we perform an analysis step prior to the synthesis to derive the strictly regular structure from the input and use the result to guide (constrain) the subsequent synthesis step so that the stocastic deviations respect the inferred periodicity. In practice, the regular structure is defined by two independent vectors describing the displacements that cause the texture to repeat itself [GS86]. These vectors are called *translation vectors* in this thesis, and define what we call the *tile* of the regular pattern.

To estimate the translation vectors, we make use of a normalized cross-correlation (NCC) of the input image with itself. Normalized cross-correlation is known to be a good tool for template matching [Lew95]. Its invariance with respect to image brightness and contrast makes it especially appropriate for template matching in non-ideal lighting conditions. We use it to compute the autocorrelation of the texture sample, and derive the two translation vectors from its local maxima. The common formula of the normalized cross-correlation has some characteristics that do not completely suit our problem domain.

As we do not want to make any assumption in the size of the underlying pattern of the input texture, we correlate the whole sample image with itself (autocorrelation). Section 3.1 describes why a generalized reformulation of the common NCC is needed in order to compute the normalized autocorrelation correctly and shows how this generalized calculation can be kept acceptably efficient by a computation in the frequency domain.

Moreover, we perform the autocorrelation of a 3-channel image (RGB), so we need a way to combine the autocorrelation of each separate channel. Section 3.2 describes how we combine the cross-correlation of various channels into a single measure.

From this measure we estimate two translation vectors $\mathbf{v_1}, \mathbf{v_2}$ that characterize the regular structure of the input texture from an observation of the local maxima distri-

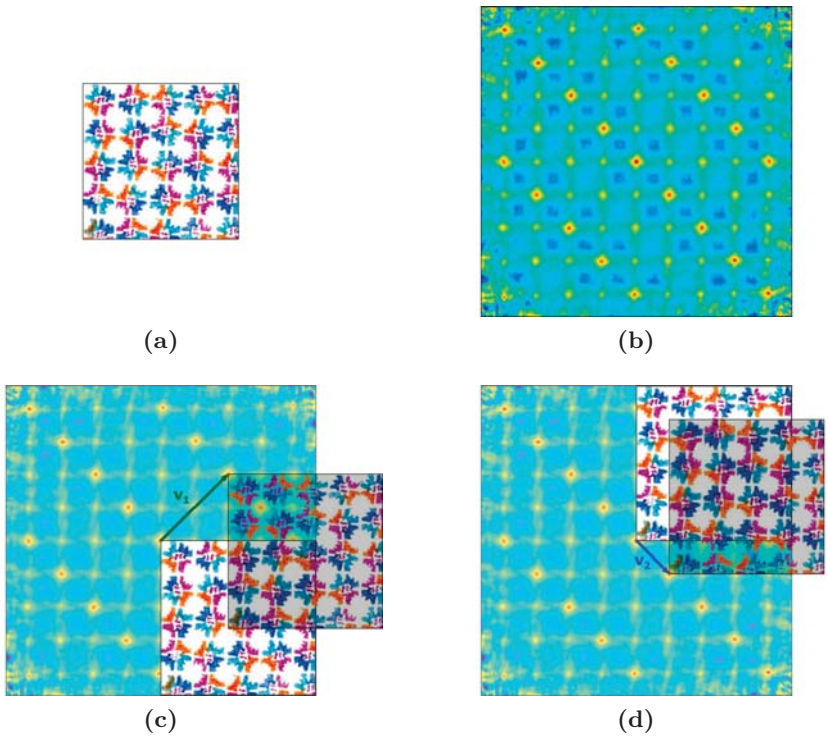


Figure 3.1: An example describing the analysis procedure. The normalized autocorrelation (b) of the input near-regular texture sample (a) has high local maxima where the underlying regular structure repeats itself. Two independent *translation vectors* $\mathbf{v_1}, \mathbf{v_2}$ (c) (d) are found to describe the underlying repetition pattern.

bution. Section 3.3 explains the translation vectors estimation process.

Figure 3.1 illustrates the goal of the analysis process. The high local maxima of the normalized autocorrelation of the input texture sample are related to the underlying repetition pattern. We analyze the local maxima distribution to get two independent translation vectors describing this regularity.

3.1 Generalized Normalized Cross-Correlation (GNCC)

The common formulation of the normalized cross-correlation (NCC) supposes that a full *template* (i.e. a relatively small image) is to be found within a bigger *image* and thus does not allow partial matches (or gives them invalid values). This is specially inconvenient if we want to perform an autocorrelation or a cross-correlation of two images of the same size, as every point but the origin is a partial match. We show here

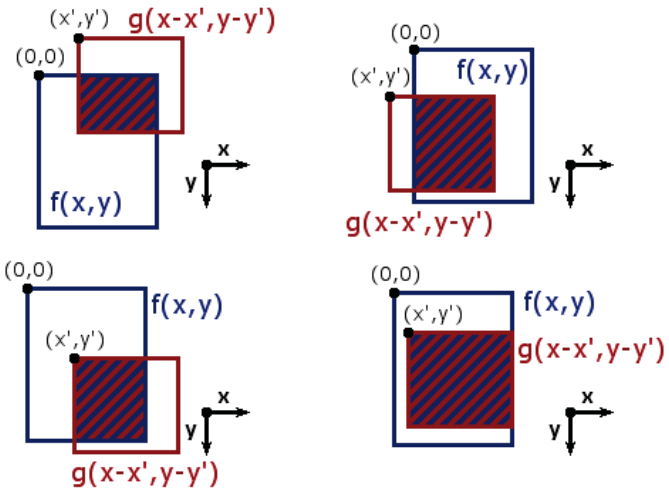


Figure 3.2: Overlapping region (in stripes) of the generalized normalized cross-correlation at different positions. Only the example at the bottom-right corner has an overlapping region that covers $g(\mathbf{x} - \mathbf{x}')$ entirely.

how we can reformulate the NCC formula to get a generalized computation that allows partial matches. We call it generalized normalized cross-correlation GNCC.[1]

We will apply this generalized formula to perform the autocorrelation of the input image, which will allow us to detect the translation vectors related to underlying regular structure in section 3.3.

The common formula of the normalized cross-correlation NCC is [Lew95]:

$$\gamma(\mathbf{x}') = \frac{\sum_{\mathbf{x}}(f(\mathbf{x}) - \bar{f}_{\mathbf{x}'})(t(\mathbf{x} - \mathbf{x}') - \bar{t})}{\sqrt{\sum_{\mathbf{x}}(f(\mathbf{x}) - \bar{f}_{\mathbf{x}'})^2}\sqrt{\sum_{\mathbf{x}}(t(\mathbf{x} - \mathbf{x}') - \bar{t})^2}} \tag{3.1}$$

where $\bar{t}$ and $\bar{f}$ denote the mean of the template and the covered image region. Both the mean of the image $\bar{f}_{\mathbf{x}'}$ and the sums are over all pixels $\mathbf{x} = [x, y]^T$ under the window containing the template positioned at a pixel position $\mathbf{x}' = [x', y']^T$.

This categorization in *image* and *template* implies that it is assumed that the template is small compared to the image and thus the correlation is usually invalid or undefined where the template is not entirely contained in the size of the image.

The generalization overcomes this categorization in *image* and *template* and considers the cross-correlation of two arbitrarily sized images with no distinction between the two inputs. It gives a normalized scalar product of the overlapping between the two images for every relative displacement with non-void intersection (or overlapping). Figure 3.2 depicts some examples of the overlapping region between two images f, g for different

[1] We sometimes overuse the acronym GNCC to refer to a normalized autocorrelation as its computation is our main application of the Generalized Normalized Cross Correlation.

relative displacements $\mathbf{x}' = [x', y']^T$. The formula of the generalized normalized cross-correlation GNCC between an $N_f \times M_f$ image f and an $N_g \times M_g$ image g now yields:

$$\gamma(\mathbf{x}') = \frac{\sum_{\mathbf{x}}(f(\mathbf{x}) - \bar{f}_{\mathbf{x}'})(g(\mathbf{x}-\mathbf{x}') - \bar{g}_{\mathbf{x}'})}{\sqrt{\sum_{\mathbf{x}}(f(\mathbf{x}) - \bar{f}_{\mathbf{x}'})^2}\sqrt{\sum_{\mathbf{x}}(g(\mathbf{x}-\mathbf{x}') - \bar{g}_{\mathbf{x}'})^2}} \tag{3.2}$$

where the sums and the mean of both images $\bar{f}_{\mathbf{x}'}$, $\bar{g}_{\mathbf{x}'}$ are over all pixels $\mathbf{x}$ in the overlapping region between the images with g positioned at $\mathbf{x}'$. Note that the new formulation is equal to the original NCC for $g = t$, $N_g < N_f, M_g < M_f$ and $0 \leq x' \leq N_f - N_g$, $0 \leq y' \leq M_f - M_g$.

This generalization makes the computation potentially much costlier, but its computation in the frequency domain allows us to keep it acceptably efficient. This computation in the frequency domain needs some considerations to be made. A reformulation of equation (3.2) makes clear what operations need to be done:

$$\gamma(\mathbf{x}') = \frac{\mathcal{N}\mathcal{S}_{fg} - \mathcal{S}_f\mathcal{S}_g}{\sqrt{\mathcal{N}\mathcal{S}_{f^2} - \mathcal{S}_f^2}\sqrt{\mathcal{N}\mathcal{S}_{g^2} - \mathcal{S}_g^2}} \tag{3.3}$$

with

$$\begin{aligned}
\mathcal{S}_{fg}(\mathbf{x}') &= \sum_{\mathbf{x}} f(\mathbf{x})g(\mathbf{x}-\mathbf{x}') \\
\mathcal{S}_f(\mathbf{x}') &= \sum_{\mathbf{x}} f(\mathbf{x}) \\
\mathcal{S}_g(\mathbf{x}') &= \sum_{\mathbf{x}} g(\mathbf{x}-\mathbf{x}') \\
\mathcal{S}_{f^2}(\mathbf{x}') &= \sum_{\mathbf{x}} f(\mathbf{x})^2 \\
\mathcal{S}_{g^2}(\mathbf{x}') &= \sum_{\mathbf{x}} g(\mathbf{x}-\mathbf{x}')^2 \\
\bar{f}_{\mathbf{x}'} &= \frac{\mathcal{S}_f(\mathbf{x}')}{\mathcal{N}(\mathbf{x}')} \\
\bar{g}_{\mathbf{x}'} &= \frac{\mathcal{S}_g(\mathbf{x}')}{\mathcal{N}(\mathbf{x}')}
\end{aligned} \tag{3.4}$$

and $\mathcal{N}(\mathbf{x}')$ is the number of pixels in the overlapping region between the images when g is at position $\mathbf{x}'$. Equation (3.3) is equivalent to equation (3.2) and shows that the computation can be obtained from a few integrations and the (unnormalized) cross-correlation between the two images.

We need to compute $\mathcal{S}_{fg}$, $\mathcal{S}_f$, $\mathcal{S}_g$, $\mathcal{S}_{f^2}$, $\mathcal{S}_{g^2}$ and $\mathcal{N}$, where the unnormalized cross-correlation $\mathcal{S}_{fg}$ is the costliest operation, as the others can be computed in linear time. Fortunately, it is equivalent to the convolution $f(\mathbf{x}) * g(-\mathbf{x})$ and so it can be computed as $\mathcal{F}^{-1}\{\mathcal{F}(f)\mathcal{F}^*(g)\}$, where $\mathcal{F}, \mathcal{F}^{-1}$ stand for the Direct and Inverse Fourier Transforms

respectively[2]. Regarding the other computations, [Lew95] showed that they can be done in linear time from tables with the precomputed integration (running sum) of the respective images.

The recursive definition of the precomputed integration s_f of image f is:

$$s_f(\mathbf{x}') = f(\mathbf{x}') + s_f(\mathbf{x}' - [1,0]^T) + s_f(\mathbf{x}' - [0,1]^T) - s_f(\mathbf{x}' - [1,1]^T) \tag{3.5}$$

where $s_f(\mathbf{x}') = 0$ when either $x', y' < 0$. This integration allows the reduction of the computation of $\mathcal{S}_f$ to:

$$\begin{aligned} \mathcal{S}_f(\mathbf{x}') &= s_f(\mathbf{x}' + [N_g - 1, M_g - 1]^T) \\ &\quad -s_f(\mathbf{x}' + [N_g - 1, -1]^T) \\ &\quad -s_f(\mathbf{x}' + [-1, M_g - 1]^T) \\ &\quad +s_f(\mathbf{x}' + [-1, -1]^T) \end{aligned} \tag{3.6}$$

where N_g and M_g are the horizontal and vertical dimensions of image g respectively. A similar procedure can be applied for $\mathcal{S}_{f^2}$, by integrating $f(\mathbf{x})^2$ instead of $f(\mathbf{x})$.

On the other hand, $\mathcal{S}_g$ needs the integration of g to be done backwards, what is equivalent to a forward integration of a 180°-rotated version $\tilde{g}$ of g , i.e.:

$$\begin{aligned} \tilde{g}(\mathbf{x}) &= g([N_g - 1, M_g - 1]^T - \mathbf{x}) \\ s_{\tilde{g}}(\mathbf{x}') &= \tilde{g}(\mathbf{x}') + s_{\tilde{g}}(\mathbf{x}' - [1,0]^T) + s_{\tilde{g}}(\mathbf{x}' - [0,1]^T) - s_{\tilde{g}}(\mathbf{x}' - [1,1]^T) \end{aligned} \tag{3.7}$$

where $s_{\tilde{g}}(\mathbf{x}') = 0$ when either $x', y' < 0$. And analogously, this can be used to easily compute $\mathcal{S}_g$:

$$\begin{aligned} \mathcal{S}_g(\mathbf{x}') &= s_{\tilde{g}}(\mathbf{x}' + [N_f - 1, M_f - 1]^T) \\ &\quad -s_{\tilde{g}}(\mathbf{x}' + [N_f - 1, -1]^T) \\ &\quad -s_{\tilde{g}}(\mathbf{x}' + [-1, M_f - 1]^T) \\ &\quad +s_{\tilde{g}}(\mathbf{x}' + [-1, -1]^T) \end{aligned} \tag{3.8}$$

where N_f and M_f are the horizontal and vertical dimensions of f respectively. Again, a similar procedure gives $\mathcal{S}_{g^2}$ by integrating $\tilde{g}(\mathbf{x})^2$ instead of $\tilde{g}(\mathbf{x})$.

The computation of $\mathcal{N}$ has also linear cost:

$$\mathcal{N}(\mathbf{x}') = \min(N_g, x' + N_g, N_f - x') \cdot \min(M_g, y' + M_g, M_f - y') \tag{3.9}$$

where min stands for the minimum and $-N_g < x' < N_f$, $-M_g < y' < M_f$.

[2] As our domain is discrete, we have to zero-pad the images to be size $(N_f + N_g - 1) \times (M_f + M_g - 1)$ before computing their DFTs in oder to do this correctly.

3.2 Multi-channel GNCC

The definition of the generalized normalized cross-correlation in section 3.1 is only applicable to a pair of single channel images (e.g. grayscale images). As we usually deal with multi-channel images (usually RGB images), we need to find a way to deal with more than one channel at the same time. We do this by evaluating the cross-correlation of each channel separately and then combining them in a single measure as a weighted sum of the independent correlations.

A normalized measure is always confined to the interval $[-1, 1]$ regardless of the actual magnitude of the inputs. This makes it more general and comparable for a wide range of different inputs. On a first thought, we could try to take advantage of this and simply multiply the correlation coefficients of each channel to get a combined result. However, that way, a poor match in one of the channels would make the combined measure as poor or even poorer. This is specially undesirable considering that normalization implies that the magnitude of information that the channel actually carries (or its relevance in the combined measure) is no longer taken into account. So, for instance, the correlation coefficient of a channel where one or both of the images are constant (and so with no structure information and no relevance at all) would be undetermined (see equation (3.2)) and usually taken as zero, causing the combined measure be zero as well. Similarly, a channel with little changes, maybe even imperceptible to human eyes, could have a poor match and mislead the overall measure.

An alternative is the mean coefficient, but still a poor match in a channel with little or no relevance would affect the overall measure considerably. For instance, with 3-channel RGB images, if one channel with no structure information has a correlation coefficient of zero, one third of the combined result is lost at once, and it would be even worse if it had a negative value. This undesired effect can be effectively reduced performing a weighted sum of the correlation coefficients of each channel instead of a blind mean.

Having a look at equation (3.2), we can see that its denominator is the product of the square root of the zero-mean energy of each of the two images in the current overlapping. That is the product of their standard deviations and can be thought of as a measure of the magnitude of structure information they carry (a constant value or small changes mean no structure information). This measure specially emphasizes sharp edge-like changes, as they are more energetic than smooth transitions. This makes it appropriate to weighting in our combined measure, considering that human perception is specially sensitive to edges when recognizing objects and patterns.

Let γ^i be the generalized normalized cross-correlation of the f^i, g^i channels of two multi-channel f, g images (with $i = R, G, B$ typically). Then, the combined correlation coefficient γ is:

$$\gamma(\mathbf{x}') = \frac{\sum_i \gamma^i(\mathbf{x}')\sigma_{f^i}\sigma_{g^i}}{\sum_i \sigma_{f^i}\sigma_{g^i}} \tag{3.10}$$

where

$$\sigma_{f^i} = \sqrt{\sum_{\mathbf{x}} [f^i(\mathbf{x}) - \overline{f^i}_{\mathbf{x}'}]^2} \tag{3.11}$$

$$\sigma_{g^i} = \sqrt{\sum_{\mathbf{x}} [g^i(\mathbf{x} - \mathbf{x}') - \overline{g^i}_{\mathbf{x}'}]^2} \tag{3.12}$$

Reformulated as in equation (3.3) and expanded yields:

$$\gamma(\mathbf{x}') = \frac{\sum\limits_i [\mathcal{NS}_{f^i g^i} - \mathcal{S}_{f^i} \mathcal{S}_{g^i}]}{\sum\limits_i \sqrt{\mathcal{NS}_{f^{i2}} - \mathcal{S}^2_{f^i}} \sqrt{\mathcal{NS}_{g^{i2}} - \mathcal{S}^2_{g^i}}} \tag{3.13}$$

The denominator normalizes the applied weighting to sum one so that the final result falls into $[-1, 1]$ again. The applied weights in the two alternative formulations are:

$$w^i = \frac{\sigma_{f^i} \sigma_{g^i}}{\sum\limits_k \sigma_{f^k} \sigma_{g^k}} \tag{3.14}$$

$$w^i = \frac{\sqrt{\mathcal{NS}_{f^{i2}} - \mathcal{S}^2_{f^i}} \sqrt{\mathcal{NS}_{g^{i2}} - \mathcal{S}^2_{g^i}}}{\sum\limits_k \sqrt{\mathcal{NS}_{f^{k2}} - \mathcal{S}^2_{f^k}} \sqrt{\mathcal{NS}_{g^{k2}} - \mathcal{S}^2_{g^k}}} \tag{3.15}$$

Note that the denominator of each γ^i (see Equation (3.3))is simplified by the numerator of the applied weighting in Equation (3.13) and the formula reduces to the sum of numerators divided by the sum of denominators.

3.3 Translation Vectors Estimation

The previous Sections 3.1 and 3.2 provide a tool for obtaining the normalized cross-correlation of two arbitrarily sized RGB images. Now, we use this technique to perform a cross-correlation of our texture sample with itself, i.e. the normalized autocorrelation of the input texture sample image. This allows us to analyse the repetition pattern of the input near-regular texture.

In practice, we are looking for two independent displacements (in the form of 2-dimensional vectors) defining the *tile* of the underlying regular pattern. What we call the *tile* is the 2-dimensional counterpart of the 1-dimensional period, as the regular structure we want to characterize is a periodic signal in two dimensions. For simplicity, we will first reduce the problem to the 1-dimensional case.

Consider the 1-dimensional ideally periodic discrete signal $f(x) = f(x + kT)$ with $x, k, T \in \mathbb{Z}$ and T constant and equal to its minimum period. Now, imagine we only have a sample f_{sam} of this periodic signal confined to the interval $[0, N-1]$ with $N > T$, thus

$$f_{sam}(x) = \begin{cases} f(x) & \text{if } 0 \leq x \leq N-1, \\ 0 & \text{otherwise.} \end{cases} \tag{3.16}$$

Then, for each $-(N-1) \leq x' \leq N-1$, the normalized autocorrelation (as analogously defined in equation (3.2) for the 2-dimensional case) of the sample signal f_{sam} is the scalar dot product of two multidimensional unit vectors (cosine similarity):

$$\gamma(x') = \begin{cases} \frac{\mathbf{u_{x',N-1}} \cdot \mathbf{u_{0,N-x'-1}}}{\|\mathbf{u_{x',N-1}}\| \|\mathbf{u_{0,N-x'-1}}\|} & \text{if } 0 \leq x' \leq N-1, \\ \gamma(-x') & \text{if } -(N-1) \leq x' < 0, \\ 0 & \text{otherwise} \end{cases} \tag{3.17}$$

where

$$\mathbf{u_{a,b}} = \begin{bmatrix} f(a) \\ f(a+1) \\ \vdots \\ f(b-1) \\ f(b) \end{bmatrix} - \frac{1}{b-a+1} \sum_{x=a}^{b} f(x) \tag{3.18}$$

This means that our measure γ is confined to the interval $[-1, 1]$ and has the maximum value ($\gamma = 1$) whenever $\mathbf{u_{x',N-1}}, \mathbf{u_{0,N-x'-1}}$ are linearly dependent $\mathbf{u_{x',N-1}} = \alpha \mathbf{u_{0,N-x'-1}}$ with $\alpha > 0 \in \mathbb{R}$. In the case of a periodic signal, we have $f(x) = f(x + kT)$, $k \in \mathbb{Z}$, and then $\mathbf{u_{x',N-1}} = \mathbf{u_{0,N-x'-1}}$, $\forall x' = kT,\ k \in \mathbb{Z}$:

$$\mathbf{u_{kT,N-1}} = \begin{bmatrix} f(kT) \\ f(kT+1) \\ \vdots \\ f(N-2) \\ f(N-1) \end{bmatrix} - \frac{1}{N-kT} \sum_{x=kT}^{N-1} f(x) \tag{3.19}$$

$$\mathbf{u_{kT,N-1}} = \begin{bmatrix} f(0) \\ f(1) \\ \vdots \\ f(N-kT-2) \\ f(N-kT-1) \end{bmatrix} - \frac{1}{N-kT} \sum_{x=0}^{N-kT-1} f(x+kT) = \mathbf{u_{0,N-kT-1}} \tag{3.20}$$

So, γ gets its maximum value every multiple of the period T.

A regular texture can be described as a 2-dimensional periodic signal, and the sample image as a rectangular piece of texture. A 2D periodic signal $f(\mathbf{x})$, $\mathbf{x} = [x, y]^T$ obeys the rule

$$f(\mathbf{x}) = f(\mathbf{x} + a\mathbf{v_1} + b\mathbf{v_2}) \tag{3.21}$$

(a) (b)

Figure 3.3: (a) The translation vectors $\mathbf{v_1}, \mathbf{v_2}$ define a parallelogram that is the *tile* of the 2D periodic signal. (b) The *tile* is the 2D counterpart of the 1D period.

where $a, b \in \mathbb{Z}$ and $\mathbf{v_1}, \mathbf{v_2}$ are two independent vectors called *translation vectors* that define the *tile* of the signal. The *tile* is the parallelogram that has the translation vectors as its non-parallel sides and is the 2D counterpart of the 1D period (see Figure 3.3). Anagously to the 1-dimesional space, the GNCC of a 2D regular texture sample image with itself has absolute maxima at linear combinations of the translation vectors $\mathbf{x} = a\mathbf{v_1} + b\mathbf{v_2}$ with a, b integers.

Although we want to analyze non-strictly regular but near-regular texture samples, we expect their normalized autocorrelation to still have high local maxima at multiples of the translation vectors related to their regular structure. Therefore, it is reasonable to believe that we can infer the shortest (defining the smallest *tile*), independent translation vectors from an observation of the local maxima of the normalized autocorrelation of the input texture sample. However, detecting which peaks are related to the regular structure of the input texture is not a trivial task. Deviations from regularity present at the texture sample can cause the ideally absolute maxima to decrease and become only local maxima that may be confused with other spurious local maxima.

Figure 3.4 shows two examples of near-regular texture samples and their normalized autocorrelation. The horizontal profile passing through the origin (i.e. the center row of the correlation matrix) of both correlations is also represented to better illustrate the magnitude of the peaks.

In the first example (upper row in Figure 3.4), the normalized autocorrelation presents local maxima wherever the shape of the little paintings roughly coincide. However, as well as the shape, the colors of the little paintings also follow a regular distribution and only those displacements where the colors are also matched in the overlapping truly define the translation symmetry of the texture (see Figure 3.1). Fortunately, the normalized autocorrelation has higher local maxima where both the shape and colors are matched (peak D) than where only the shape is matched (peaks A, B and C), so on first thought we could think of thresholding to detect the correct repetition pattern.

On the other hand, the second example (lower row in Figure 3.4) shows that simple thresholding may be inconvenient in some cases. The input texture is made of squared gray tiles of varying brightness. We would like to detect one single tile as the smallest unit of repetition of the regular structure, so the relatively low peaks A and B are not

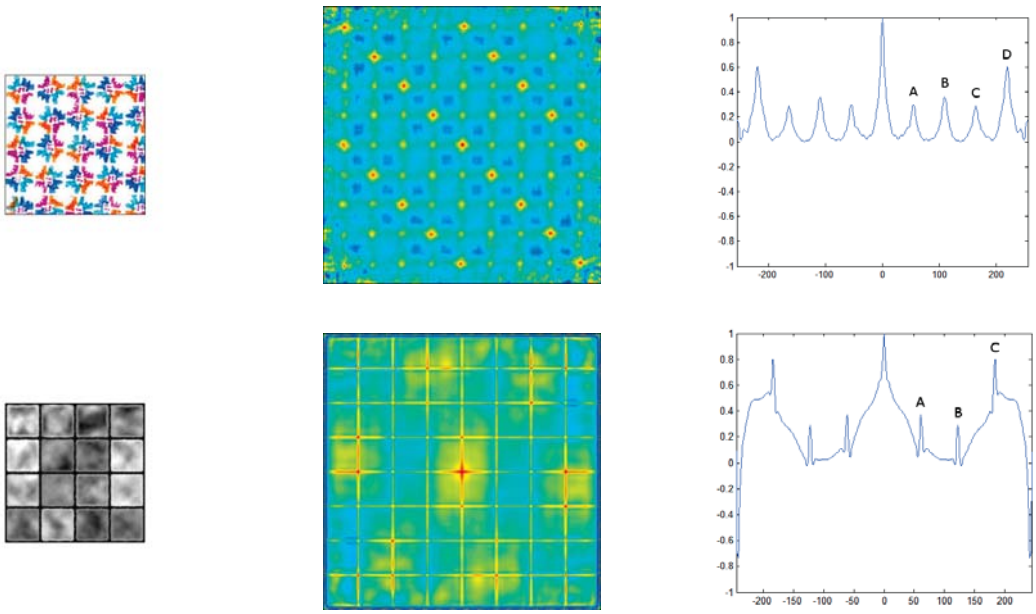


Figure 3.4: Some spurious relatively high local maxima may appear in the normalized autocorrelation. Left: input texture sample. Center: RGB-combined normalized autocorrelation. Right: horizontal profile passing through the origin of the combined GNCC (center row).

spurious but are caused by the repetition pattern that we want to estimate. So, if we applied the hypothetical thresholding proposed in the previous paragraph, we would fail to detect the smallest translation vectors in this case.

The method to estimate the two smallest translation vectors related to the regularity of input texture from the analysis of the normalized autocorrelation of the texture sample that we propose here tries to overcome these difficulties. On the one hand, it intends to find the smallest possible translation vectors, and on the other hand, avoid misleadings that spurious maxima may cause (and do it without user intervention).

The procedure mainly derives from the following observations:

i. The autocorrelation has the symmetry $\gamma(\mathbf{x}) = \gamma(-\mathbf{x})$.

ii. If two vectors are translation vectors for the near-regular texture under analysis, the normalized autocorrelation of the input sample has high local maxima not only at those displacements but also at every of their multiples.

iii. A translation vector $\mathbf{v}$ is equivalent to its opposite $-\mathbf{v}$ as they have the same multiples.

iv. A pair of translation vectors $\{\mathbf{v_1}, \mathbf{v_2}\}$ is equivalent to $\{\mathbf{v_2}, \mathbf{v_1}\}$, $\{\mathbf{v_1}, \mathbf{v_2} + k_1\mathbf{v_1}\}$ and to $\{\mathbf{v_1} + k_2\mathbf{v_2}, \mathbf{v_2}\}$ for any $k_1, k_2 \in \mathbb{Z}$ as they all have the same set of multiples and so define the same periodicity.

v. Some spurious maxima are likely to appear at the borders of the autocorrelation matrix, where the values are obtained from the comparison of a smaller pixel region (see Figure 3.2).

vi. Other spurious maxima not at the borders of the autocorrelation are generally lower than the peaks caused by the periodicity of the texture.

Given these observations, we decide to proceed as follows:

1. Mark every local maximum of the right half of the autocorrelation matrix (the left half is redundant given the symmetry of γ).

2. Form a candidate list c_1 of vectors $\mathbf{v_{1(1)}}, \ldots, \mathbf{v_{1(n)}}$ with the displacements of each marked local maximum from the origin (i.e. the center of the matrix) sorted in ascending length $\|\mathbf{v_{1(1)}}\| \leq \ldots \leq \|\mathbf{v_{1(n)}}\|$.

3. Explore each vector $\mathbf{v_{1(i)}}$ in c_1 in order and form $c_2 = \{\mathbf{v_{1(i+1)}}, \ldots, \mathbf{v_{1(n)}}\}$ with the vectors that are after $\mathbf{v_{1(i)}}$ in the sorted list c_1.

4. Reduce every $\mathbf{v_{2(j)}}$ in c_2 to $\mathbf{v_{2(j)}} = \mathbf{v_{2(j)}} - \left[\frac{\mathbf{v_{2(j)}} \cdot \mathbf{v_{1(i)}}}{\|\mathbf{v_{1(i)}}\|^2}\right] \mathbf{v_{1(i)}}$ (and take $\mathbf{v_{2(j)}} = -\mathbf{v_{2(j)}}$ if it lay on the left semi-plane) and eliminate duplicates and vectors shorter than $\mathbf{v_{1(i)}}$, where [.] is the round-to-nearest-integer operator and the dot is the scalar dot product. Among all equivalent pairs (observation (iv)) we keep the one with the greatest angle between vectors.

5. Given the normalized autocorrelation, $\mathbf{v_{1(i)}}$ and for all $\mathbf{v_{2(j)}}$ in c_2, evaluate the *goodness*:
$$g(\{\mathbf{v_{1(i)}}, \mathbf{v_{2(j)}}\}) = \frac{\left(\sum\limits_{a,b \in \mathbb{Z}} \gamma(a\mathbf{v_{1(i)}} + b\mathbf{v_{2(j)}})\right)^{\alpha}}{n} \tag{3.22}$$
where the sum is over the *significant* multiples of $\{\mathbf{v_{1(i)}}, \mathbf{v_{2(j)}}\}$ that are on the right half of the autocorrelation excluding the origin and n is the final number of summands.

6. The pair of vectors with the higher *goodness* are the final estimation of the translation vectors.

Note that the formula defining the *goodness* of a candidate pair $\{\mathbf{v_1}, \mathbf{v_2}\}$ is equivalent to:

$$g(\{\mathbf{v_1}, \mathbf{v_2}\}) = (\bar{\gamma}_{\mathbf{v_1},\mathbf{v_2}})^{\alpha} \cdot n^{\alpha - 1} \tag{3.23}$$

with

$$\bar{\gamma}_{\mathbf{v_1},\mathbf{v_2}} = \frac{\sum\limits_{a,b \in \mathbb{Z}} \gamma(a\mathbf{v_1} + b\mathbf{v_2})}{n} \tag{3.24}$$

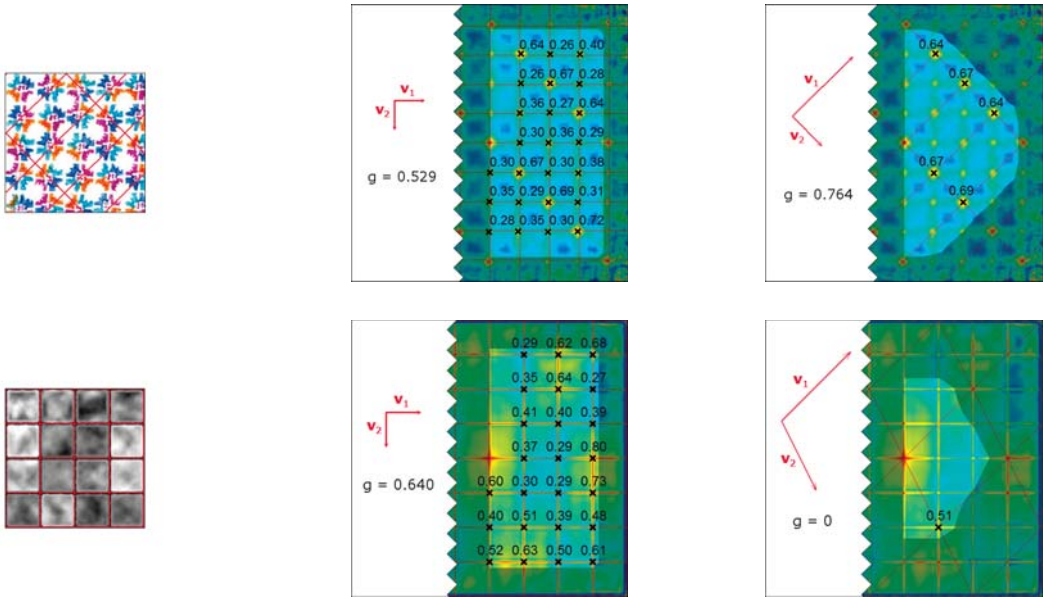


Figure 3.5: Texture samples with their detected lattice and examples of goodness evaluation. Black crosses mark the values of the autocorrelation that compose the sum for the goodness evaluation. The shading differenciates which values of the autocorrelation are considered *significant* with the current candidate translation vectors (*invalid* border).

the mean GNCC at the *significant* multiples of the translation vectors $\{\mathbf{v_1}, \mathbf{v_2}\}$. The idea is to mainly rate two candidate vectors $\{\mathbf{v_1}, \mathbf{v_2}\}$ with the mean GNCC at their *significant* multiples, but assign a slightly greater than 1 value to α to give priority to smaller tiles in case of similar mean GNCC. All our results were achived with $\alpha = 1.12$.

Note that only *significant* multiples of the candidate translation vectors are taken into account. Section 3.3.1 explains what *significant* means in this context.

Figure 3.5 is a summary of the estimation procedure. It shows that the tile shape and size of both example textures of Figure 3.4 were estimated correctly and without user intervention. In the first example (upper row), the bigger tile scores better because the spurious peaks not at the border of the autocorrelation make the mean γ of the smaller tile lower. In the second example however (lower row), the bigger tile has no valid multiple of $\mathbf{v_1}$, so its goodnes is automatically 0.

3.3.1 *Significant* Values of the Autocorrelation

Recall that the estimation is based on the normalized autocorrelation (see Sections 3.1 and 3.2) of the texture sample image. The normalized autocorrelation gives a measure of similarity between the image and a displaced version of itself in their overlapping region for every possible relative displacement. If this overlapping region is small compared to the tile of the texture (i.e. the periodicity unit), high spurious local maxima are likely to appear (borders of the autocorrelation matrix). Therefore, only those values of the

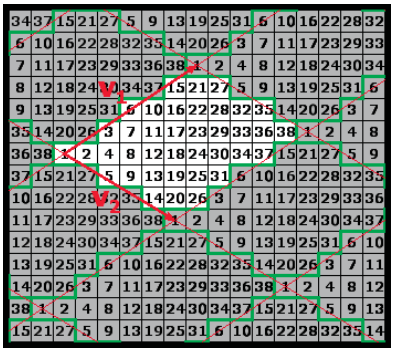


Figure 3.6: Input space split in tiles and each pixel numbered in function of its relative position within its tile.

autocorrelation that come from a *significant* set of pixels are taken into account in the goodness evaluation of two candidate translation vectors (see Equation (3.22)).

Note that whether an overlapping is significant or not depends on the specific texture, but it is sure enough if it contains a full tile of the repetition pattern and we cannot assure the validity of γ if only a little part of the tile is included. So when rating the goodness of a candidate pair of vectors $\{\mathbf{v_1}, \mathbf{v_2}\}$, we only consider values of the autocorrelation whose overlapping contains at least a certain percentage of the candidate tile (i.e. the parallelogram defined by the candidate pair of vectors). In all our results, we chose an 85% as the minimum percentage of the candidate tile that *significant* overlappings have to contain.

Seeing how much of the tile an overlapping contains is not a trivial task. The intersection of two rectangular images is always rectangular itself, but the tile is a parallelogram that can have any orientation. Depending on the orientation and angle between the two translation vectors, the border of the autocorrelation matrix that we consider *invalid* may have different shape and width (see Figure 3.5). Although other faster conservative approximations could be applied, we propose an exact method for determining the percentage of the tile that a given overlapping contains.

We split the input space in disjoint contiguous tiles (i.e. tile-shaped pieces) and assign an identifier to each pixel in function of its relative position within its tile (see Figure 3.6). Then, we find out how many unique identifiers a given overlapping contains.

Figure 3.7 shows two examples of overlapping regions. They suppose the input dimensions and the candidate vectors $\{\mathbf{v_1}, \mathbf{v_2}\}$ depicted in Figure 3.6. The first example (upper row) is the overlapping corresponding to the computation of $\gamma(14, 7)$ and shows that it only includes a 57.9% of the candidate tile, so it would not pass the threshold of 85% that we impose. However, the second example (lower row) corresponds to the computation of $\gamma(12, 8)$ and includes a 92.1% of the candidate tile, so it would pass the imposed threshold of 85%.

Note that the percentage of the tile included in an overlapping region only depends on the shape and size of both the tile and the overlapping region, regardless of the

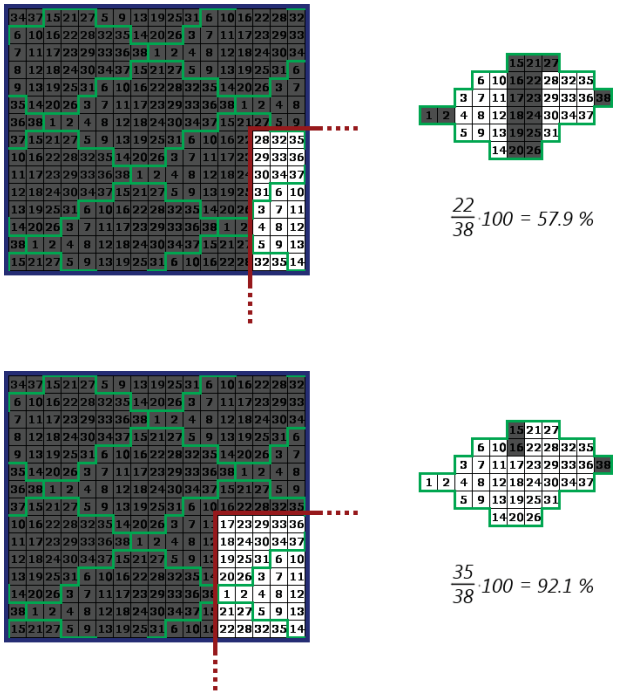


Figure 3.7: The overlappings on the left correspond to the computation of $\gamma(12, 8)$ and $\gamma(14, 7)$ respectively. On the right, the representation shows that the upper overlapping contains 22 out of 38 unique identifiers and the lower ovelapping 35 out of 38.

alignment. The particular identifiers included in the overlapping region may change depending on the relative position, but the size of the set of unique identifiers remains constant. So, for example, a rectangular 3×8 overlapping as in the upper row in Figure 3.7 with the defined tile will always contain 22 unique identifiers no matter if it is on the top-left, top-right, bottom-left or bottom-right corners of the input space (correspondig to $\gamma(-14, -7), \gamma(14, -7), \gamma(-14, 7), \gamma(14, 7)$ respectively).

Therefore, to calculate the percentage of the tile included in an overlapping for a candidate translation vectors pair $\{\mathbf{v_1}, \mathbf{v_2}\}$, we can proceed as follows. First, identify each pixel in the overlapping with its regular $\mathbf{x} = [x, y]^T$ coordinates. Then, *wrap* (see below) the $\mathbf{x}$ coordinates to the first tile defined by $\{\mathbf{v_1}, \mathbf{v_2}\}$ so that the wrapped coordinates $\tilde{\mathbf{x}} = [\tilde{x}, \tilde{y}]^T$ of every pixel at the same relative position within a tile are equal. And finally, find out how many different wrapped coordinates $\tilde{\mathbf{x}}$ there are compared to the maximum possible. Note that if $\mathbf{v_1} = [v_{1x}, v_{1y}]^T$ and $\mathbf{v_2} = [v_{2x}, v_{2y}]^T$ have integer coordinates, the maximum possible different wrapped coordinates (i.e. the number of pixels in one tile) is:

$$\left| \det \left(\begin{bmatrix} v_{1x} & v_{2x} \\ v_{1y} & v_{2y} \end{bmatrix} \right) \right| = |v_{1x} v_{2y} - v_{2x} v_{1y}| \tag{3.25}$$

where $|.|$ is the modulus or absolute value operator and $\det(.)$ stands for determinant.

To understand what the *wrapping* operation does, consider the analogy with a 1D-periodic signal with period T. If a given position x is *wrapped* to $\tilde{x}$, it means that

$$x = kT + \tilde{x} \tag{3.26}$$
$$\tilde{x} = x - kT \tag{3.27}$$

with $k \in \mathbb{Z}$ so that $\tilde{x} \in [0, T)$. Then,

$$\tilde{x} = x - \left\lfloor \frac{x}{T} \right\rfloor T \tag{3.28}$$

where $\lfloor . \rfloor$ is the round-down (or floor) operator.

Analogously, in the context of a 2D-periodic signal with translation vectors $\{\mathbf{v_1}, \mathbf{v_2}\}$, the wrapped coordinates $\tilde{\mathbf{x}}$ of a position $\mathbf{x}$ can be obtained as follows:

$$\tilde{\mathbf{x}} = \mathbf{x} - Q \left\lfloor Q^{-1}\mathbf{x} \right\rfloor \tag{3.29}$$

where

$$Q = \begin{bmatrix} \mathbf{v_1} & \mathbf{v_2} \end{bmatrix} = \begin{bmatrix} v_{1x} & v_{2x} \\ v_{1y} & v_{2y} \end{bmatrix} \tag{3.30}$$

Note that the wrapped coordinates are always in the *first* tile, i.e. in the parallelogram $(0,0), \mathbf{v_1}, \mathbf{v_1} + \mathbf{v_2}, \mathbf{v_2}$.

3.4 Conclusion

The computation described in the previous Section 3.3.1 allows us to determine which multiples of each candidate pair of vectors should compose the sum of the goodness evaluation in Equation (3.22). This ultimately leads to the estimation of the translation vectors of the texture under analysis as the candidate pair with the highest score. Then, the analysis stage is finished and the detected lattice is represented with the estimated pair of vectors, which along with the sample image are the input of the subsequent synthesis stage explained in the following Chapter 4.

4 Synthesis

The objective of example-based texture synthesis is to generate an arbitrarily sized image that faithfully reproduces the texture of a relatively small sample image. The procedure is said to be example-based if it composes the output only from extracted pieces of the input sample in contrast to methods that infer a statistical model for the input texture. This section describes two example-based synthesis methods focused on the reproduction of near-regular textures.

The type of near-regular textures addressed in this thesis are textures that have a recognizable structure (i.e. repetition pattern) mixed up with characteristic stochastical disturbances that cannot be reproduced by simple tiling. On the other hand, local-statistics example-based methods that are not aware of the underlying regular structure of the input texture fail to fairly reproduce that repetition pattern.

In section 3, we have shown how the regular structure of a near-regular texture sample image can be analyzed to obtain the two independent translation vectors that describe the underlying repetition pattern. Now, we can make use of this information to synthesize an arbitrarily sized image that respects the same regular structure of the texture sample.

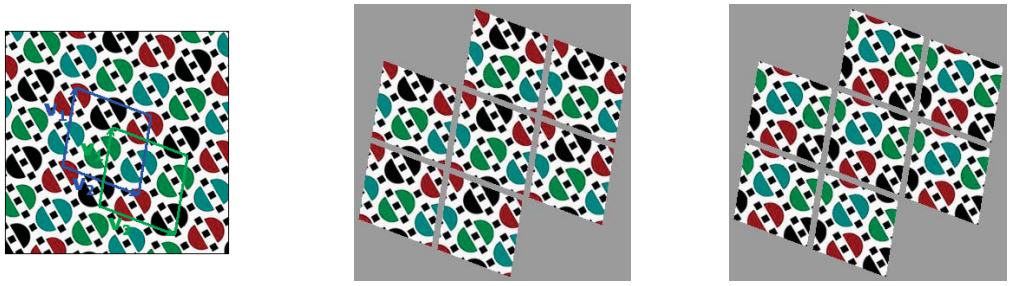
First, we describe how the two independent translation vectors found in the analysis step define the *tile* of the underlying regular structure in section 4.1. In section 4.1.1, we present a method to search for the best self-similar piece of texture with the shape of the tile that produces the most seamless tiling, inpired by [DED05]. This fairly reproduces the regular structure of the texture to be synthesized, but pays no attention to the slight yet characteristic stocastic deviations that usually make near-regular textures look natural. Nonetheless, it serves us to illustrate the improvement that we get with the constrained *random sampling and gap filling* technique that is presented in section 4.2. Proper comparisons between the two outputs are shown in section 5.

The *constrained random sampling and gap filling* technique that is presented in section 4.2 exploits the information about the regular structure of the input texture that is obtained in the analysis step to ensure its preservation in the synthesized texture, but introduces random deviations from strict regularity to make the output look more natural. It is mainly subdivided in two substeps: constrained *random sampling* and constrained *gap filling*. Both of them are described in detail in sections 4.2.1 and 4.2.2.

4.1 Best Self-similar Tile Repetition

Recall that a 2D-periodic signal follows the rule:

$$f(\mathbf{x}) = f(\mathbf{x} + a\mathbf{v_1} + b\mathbf{v_2}) \tag{4.1}$$



(a) Input texture sample. **(b)** Repetition of the tile in blue in (a). **(c)** Repetition of the tile in green in (a).

Figure 4.1: Any portion of the signal with the shape and size of the *tile* reproduces the original signal when repeated.

where $\mathbf{x} = [x, y]^T$, a, b are integers and the independent vectors $\mathbf{v_1}, \mathbf{v_2}$ are the shortest possible *translation vectors* and define the *tile* of the signal.

We call the *tile* of a 2D-periodic signal the parallelogram that has the translation vectors $\mathbf{v_1}, \mathbf{v_2}$ as its non-parallel sides (see Figure 3.3). The tile in the 2-dimensional space is the counterpart of the 1-dimensional period. In the 1-dimensional space, any interval of length T of a periodic signal $f(x)$ with T its period is representative of the signal. That means that any array of size T of the form

$$[f(x_{off}), f(x_{off}+1), \ldots, f(x_{off}+T-2), f(x_{off}+T-1)]$$

can be repeatedly concatenated to faithfully reproduce the signal to any length regardless of the offset x_{off}. Anagously, in two dimensions, a tile-shaped piece is enough to synthesize an arbitrarily sized portion of a 2-dimensional signal by repeatedly pasting the extracted tile-shaped piece in both the directions of the translation vectors, thus joining opposite sides of the parallelogram.

This is illustrated in Figure 4.1. It shows two examples of viable pieces (i.e. with the shape and size of the tile) that can reproduce the input sample of a regular texture. By repeatedly pasting together copies of the extracted tiles,[1] we can generate an arbitrarily sized output with the same regular structure. In the examples, some space is left between neighboring copies of the extracted tile to illustrate the composition.

As the input texture deviates from strict regularity, it is probable that direct tiling of a single piece of texture creates seams between adjacent copies pasted together in the output. Inspired by [DED05], we propose here a way to minimize this effect. The basic idea is to examine every possible tile-shaped portion of the input and select the best self-similar tile, i.e the tile-shaped piece of texture whose opposite sides are the most similar possible to each other so that the transition from the end of one tile copy to the begining of the next is smooth. The main difference to [DED05] is that they suppose that the input texture sample is properly aligned and only consider rectangles while we

[1] We sometimes overuse the term *tile* to refer to a tile-shaped piece of texture.

(a) (b)

Figure 4.2: (a) Input texture sample with a mesh generated by the translation vectors found in the analysis step in red. (b) Detail of the shape of the tile.

(a) Top and bottom $\mathbf{v_1}$-long borders of one possible tile.

(b) Detail of the mask for each $\mathbf{v_1}$-long border. Anchor point in red.

(c) Left and right $\mathbf{v_2}$-long borders of one possible tile.

(d) Detail of the mask for each $\mathbf{v_2}$-long border. Anchor point in red.

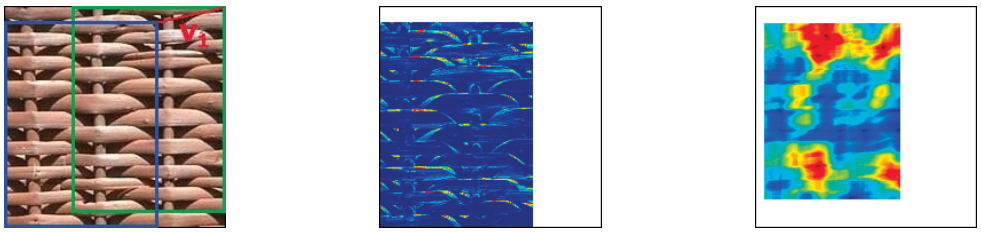
Figure 4.3: Example of borders at tile edges. Opposite borders of every possible tile-shaped piece of input texture are compared to each other to find the best self-seamless tile.

accept any parallelogram as the tile shape. Moreover, their measure of self-similarity for a given tile is based on the first derivative at tile junctions approximated as the squared color difference between pixels at opposite sides (between pixels at the first column and pixels at the last column, and between pixels at the first row and pixels at the last row of the candidate rectangular tile). We evaluate direct squared color differences instead, as detailed in the following section 4.1.1.

4.1.1 Best Self-Similar Tile Search

Consider the input texture sample in Figure 4.2. The analysis step has estimated two independent vectors $\mathbf{v_1}, \mathbf{v_2}$ as the shortest translation vectors. They generate the mesh depicted in Figure 4.2a. The tile shape is represented in Figure 4.2b, where $\mathbf{v_2}$ defines the length of the left and right sides and $\mathbf{v_1}$ defines the length of the top and bottom sides.

To evaluate the self-similarity of a given tile, we compare a border centered on the left side with a same-shaped border centered on the right side, and a border centered on the top side with a same-shaped border centered on the bottom side (see Figure 4.3). The measure of similary between opposite borders is given by their accumulated squared color differences. Every pixel in the left border is compared to the corresponding pixel



(a) (b) (c)

Figure 4.4: (a) Every pixel $\mathbf{x} = [x, y]^T$ is related to the pixel $\mathbf{x}+\mathbf{v_1}$. (b) Squared color difference between every pair of pixels distant to each other in $\mathbf{v_1}$. (c) The integration in (b) of a border-shaped patch (see fig. 4.3d) gives the accumulated squared color differences between opposite $\mathbf{v_2}$-long sides of every possible tile.

in the right border, their squared color difference is obtained and the accumulation of all the squared color differences in the border gives the left-right similarity measure. An analogous computation gives the top-bottom similarity measure. Then, the sum of both measures gives the final accumulated squared color differences between opposite borders. The lower this values is, the more self-similar the tile under analysis is.

Let us start with the $\mathbf{v_2}$-long left and right sides of the parallelogram. Note that pixels in the left border are always related to their right-border-counterpart pixel by a $\mathbf{v_1}$ displacement. Therefore, to compute the left-right similarity measure for every possible tile in the input, we precompute the squared color difference between each pixel and its $\mathbf{v_1}$-distant counterpart:

$$\sum_{i=R,G,B} [f_i(\mathbf{x}) - f_i(\mathbf{x} + \mathbf{v_1})]^2 \tag{4.2}$$

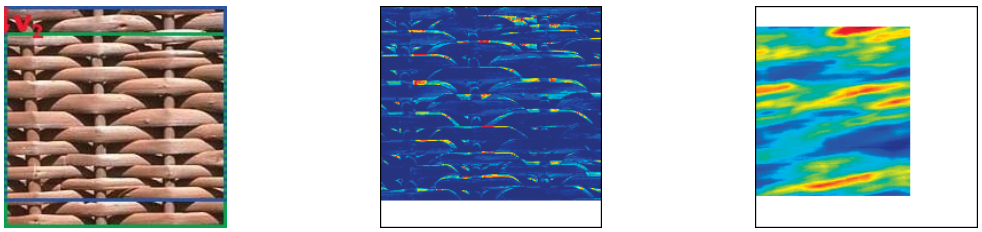
and then integrate a patch of the shape of the defined border over the result.

Figure 4.4 illustrates this process. First, every pixel $\mathbf{x} = [x, y]^T$ is related to the pixel $\mathbf{x} + \mathbf{v_1}$ and their squared color difference is computed, i.e. we precompute the squared color differences between pixels of the subimage marked in blue and pixels of the subimage marked in green in Figure 4.4a thus obtaining Figure 4.4b. Then, an integration over the intermediate result of a patch of the shape and size of the desired border (see Figure 4.3d) gives the final comparison between opposite $\mathbf{v_2}$-long borders in Figure 4.4c. Note that every pixel in Figure 4.4c represents the left-right similarity of a tile with the top-left corner at the position of that pixel. Tiles whose borders are outside the size of the input have undefined self-similarity (white).

The top and bottom $\mathbf{v_1}$-long sides of the tile are analogously compared. Figure 4.5 shows the analogous process.

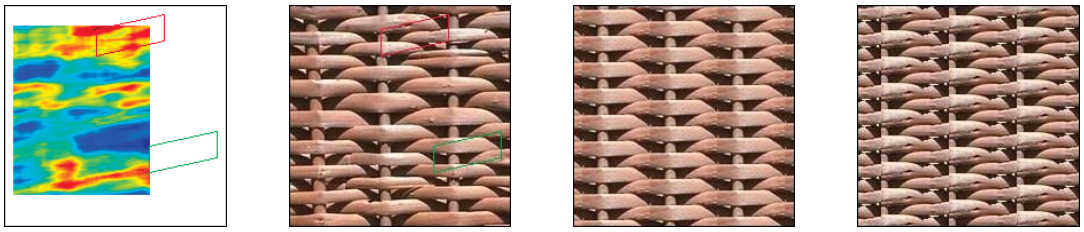
Finally, we sum both results to get a final rating of how self-similar every possible tile is. The tile with the lowest total accumulated squared color differences between opposite borders is the best self-similar tile.

Figure 4.6 shows the final best and worst self-similar tiles for the conducting example. The top-left corner is the anchor point of the tiles, so the worst tile is at the highest (dark red) point of the measure and the best tile at the lowest (dark blue). Note how



(a) (b) (c)

Figure 4.5: (a) Every pixel $\mathbf{x} = [x, y]^T$ is related to the pixel $\mathbf{x}+\mathbf{v_2}$. (b) Squared color difference between every pair of pixels distant to each other in $\mathbf{v_2}$. (c) The integration in (b) of a border-shaped patch (see fig. 4.3b) gives the accumulated squared color differences between opposite $\mathbf{v_1}$-long sides of every possible tile.



(a) (b) (c) (d)

Figure 4.6: (a) Final accumulated squared color differences between opposite borders of every possible tile with the best tile in green and the worst tile in red. (b) Input texture sample with the best tile in green and the worst tile in red. (c) The best tile present a smooth transition at tile junctions. (d) The worst tile presents visible seams at tile junctions.

the best tile presents smooth transitions between contiguous tile copies, whereas the worst tile has clear discontinuities at tile junctions.

4.2 Constrained Random Sampling and Gap Filling

In this section, we present a near-regular texture synthesis technique that we call *constrained random sampling and gap filling*. The intention is to synthesize a near-regular texture of an arbitrary size from a previously analyzed (as explained in section 3) sample image. The previous analysis step is exploited to preserve the repetition pattern of the near-regular texture sample and at the same time, contrary to the method described in section 4.1, the output preserves the characteristic random irregularities of the input texture to make the result look more natural.

This synthesis method is mainly subdivided in two substeps. The *constrained random sampling* substep ensures that stocastic deviations from regularity are introduced in the output by filling half the output space with randomly selected pixels from the input. Then, the *constrained gap filling* completes the synthesis process reducing visible transitions that might have appeared if the procedure were completely random. Both

substeps are fully described in sections 4.2.1 and 4.2.2 respectively.

Before explaining the synthesis process in detail, we will first make some definitions.

Our synthesis approach splits the output space in square-shaped *blocks* of a constant size.[2] Each of these blocks will be filled with a piece of texture or *patch* from the input sample at the end. We usually call *patch* any piece of texture extracted from the input sample during the synthesis process.

It is also worth mentioning that a regular texture can be seen as a 2D-periodic discrete signal and, if we recall that a 2D-periodic discrete signal satisfies

$$f[\mathbf{x}] = f[\mathbf{x} + a\mathbf{v_1} + b\mathbf{v_2}] \tag{4.3}$$

where $\mathbf{v_1}, \mathbf{v_2}$ are its shortest translation vectors and a, b are integers, it is straightforward that a given pixel in a regular texture sample image is repeated at every point at a distance multiple of the translation vectors. Moreover, any group of pixels of the sample image is repeated every multiple of the translation vectors. However, we actually deal with non-ideally regular textures but near-regular textures that have a regular structure but also some stochastic irregularities. Therefore, if the regular structure were separable form the irregularities, the regular part (more energetic) would be repeated every multiple of the translation vectors, but the irregular part might differ. So we say that a group of pixels has a *similar* group of pixels at every multiple of the translation vectors, because their most energetic part is equal but they have little random disturbances that may be different.

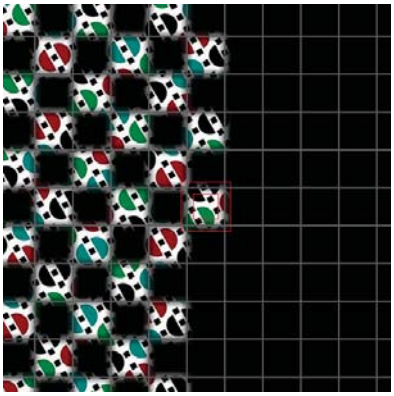
Furthermore, the output image is initially empty and usually bigger than the input sample. So, when we say that a *patch* is *similar* to an empty *block*, it means that the patch is similar to what we would find in the input at the position of the block if the sample image were big enough.

Note the difference in sampling from the previous method (see section 4.1). While in the previous section we extract a piece of texture from the input with the shape and size of the tile, here, the extracted pieces are always squared-shaped regardless of the actual shape of the tile. The size of the squared blocks/patches can be any as long as it remains constant during the synthesis process. The tile (i.e. the translation vectors) serves here as a guidance for the sampling, i.e. to constrain the sampling so that every block is filled with one of its *similar* patches. This is explained in detail in the following subsections.

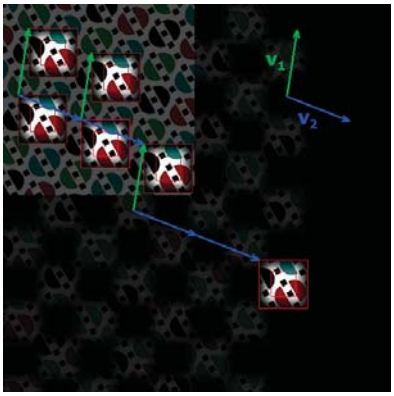
4.2.1 Constrained Random Sampling

First of all, we divide the output space in square-shaped blocks of a constant size. For simplicity, if we want to synthesize an output image of size $N_{out} \times M_{out}$ with blocks of side b, we will synthesize an output of size $\left\lceil \frac{N_{out}}{b} \right\rceil b \times \left\lceil \frac{M_{out}}{b} \right\rceil b$ and then crop the result to the originally desired size. Each of these blocks will be filled with a patch of the same

[2] We leave the discussion about the election of an appropriate block size for the evaluation chapter (see section 5)



(a) Synthesis in progress. The grid in gray illustrates the division of the output in blocks. The current block is marked with a red frame.



(b) The distance between the candidate (i.e. *similar* to the current block) patches and the current location is a multiple of the translation vectors $\mathbf{v_1}, \mathbf{v_2}$.

Figure 4.7: Example of *random sampling* in progress. The borders around the blocks are shown to emphasize that only patches whose borders do not lie outside the input image can be candidates. Input image size: 256x256 px. Block size: 50x50 px. Border width: 2x8 px.

size and shape from the input sample in two different steps. We explain here the first of these steps which we call *constrained random sampling*.

The first step in our synthesis process consists in filling every second block of the output (in a chess-like fashion) with a randomly selected patch from the input.

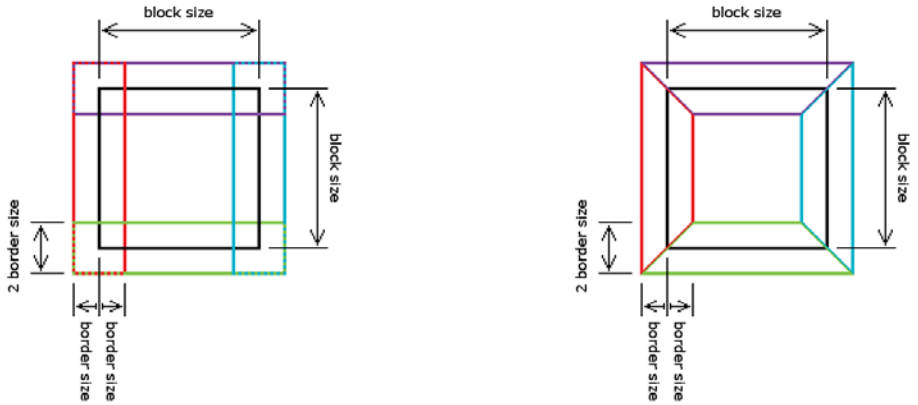
For the texture synthesis process, we only have at our disposal a texture sample of a limited size. However, in the analysis step, we have learned that our texture presents a regular pattern that is described by two independent translation vectors $\mathbf{v_1}, \mathbf{v_2}$ that we have estimated. In near-regular textures, this regular structure appears diffused with stocastic deviations from regularity, but still the estimated translation vectors tell us where to find *similar* pieces of texture.

For each second block, patches from the input texture that are *similar* to the current block are preselected. Then, we randomly choose one among these preselected patches to finally fill the current block, and proceed to the next block. We repeat the random assignment of patches to blocks until every second block of the output is covered.

Figure 4.7a shows the synthesis process at the time one of these randomly selected patches is pasted into the current block. Note that we enforce some overlapping between neighboring blocks. We need this for the next step, which is described in section 4.2.2.

Patches that are *similar* to the current block can be found at integer multiples of the translation vectors, as defined in Section 4.2. Figure 4.7b illustrates this relationship. Note that the top-left corner is taken as the anchor point of the patch/blocks.

Once we have covered all the output space and randomly filled half the output, we preceed to the *constrained gap filling* substep, which is described in the following section.



(a) Rectangular-shaped borders. **(b)** Wedge-shaped borders.

Figure 4.8: Comparison between border shapes. While rectangular borders overlap each other, wedge-shaped borders do not.

4.2.2 Constrained Gap Filling

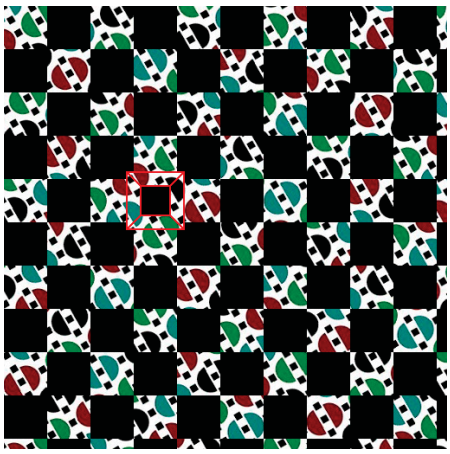
At this point, every second block of the output has been filled with a randomly selected patch from the input (see Figure 4.7), thus leaving empty blocks (gaps) sorrounded by already synthesized blocks. Again, these resulting gaps will be filled with patches from the input, but now, for each block, we will choose the patch that best matches the borders with its already synthesized sorrounding neighbors.

We consider borders centered on the edge between blocks, hence the border is always an even number of pixels width. This means that the sorrounding borders of a gap would overlap each other if they had straightforward rectangular shapes, so we consider wedge-shaped borders instead. We prefer this to an alternative blending between overlapping borders, as the latter would cause blurring and detail loss. Figure 4.8 shows these different border shapes. It can be seen that, contrary to rectangular borders, wedge-shaped borders do not overlap each other.

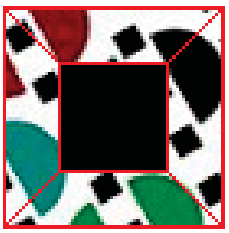
For the synthesis of each gap, we know where in the input sample the source of each of the sorrounding blocks is. Therefore, as in section 4.2.1 we ensured that only patches with borders inside the input image were selected, we can compose a full border for the gap by extracting the corresponding wedge-shaped border of every sorrounding neighbor. Figure 4.9 shows an example of the composition of this full border as well as a representation of its position in the output. We will fill the gap with the patch from the texture sample that best matches this full border.

We propose two ways of measuring how good a border matches a patch of the input.

The first alternative is to make use of the combined GNCC presented in sections 3.1 and 3.2. In this case, we consider the border as a template to be correlated with our sample image. Although it is the same problem enunciation as an ordinary NCC, the problem now is that our template has a hole in the middle. However, the formulation of the GNCC presented in equation (3.3) helps us to overcome this problem. The only



(a) Output after *random sampling*. Current gap and sorrounding borders in bright red.



(b) Detail of the current gap and its full border. The wedges from each separate source are enclosed in bright red.

Figure 4.9: Example of *gap filling* in progress. Block size: 50x50 px. Border width: 16 px.

adaptation we need to do is to substract to $\mathcal{S}_f$, $\mathcal{S}_{f^2}$, $\mathcal{S}_g$, $\mathcal{S}_{g^2}$, and $\mathcal{N}$ the integration corresponding to the hole analogously to equations 3.6,3.8,3.9. The unnormalized cross-correlation $\mathcal{S}_{fg}$ remains the same if we fill the hole with zeros. Note that, as we are only interested in those values of the correlation where the border (template) fully overlaps the input, $\mathcal{S}_g$, $\mathcal{S}_{g^2}$ and $\mathcal{N}$ are constant. The patch with the highest correlation is then chosen as the best matching patch.

The second alternative is to evaluate the squared color differences for each postion within the input:

$$\chi(\mathbf{x}') = \sum_{\mathbf{x}} \sum_{i=R,G,B} [f_i(\mathbf{x}) - g_i(\mathbf{x} - \mathbf{x}')]^2 \tag{4.4}$$

where f is the input sample, g is the composed border and the outer sum is over the overlapping. Note that

$$\sum_{\mathbf{x}} [f_i(\mathbf{x}) - g_i(\mathbf{x} - \mathbf{x}')]^2 = \mathcal{S}_{f_i^2} - 2\mathcal{S}_{f_i g_i} + \mathcal{S}_{g_i^2} \tag{4.5}$$

so it can also be computed in the frequency domain (see equation (3.3)). In this case, the best match is that with the smallest χ.

Both alternatives produce similarly good results. In general, it can be said that the normalized cross-correlation may adjust sharp edges more accurately in some cases (e.g. lines between bricks in a brik wall) while the color differences also pays attention to the overall brightness and contrast. This may sometimes make the GNCC result look slightly more blocky specially when the input sample was taken with non-uniform

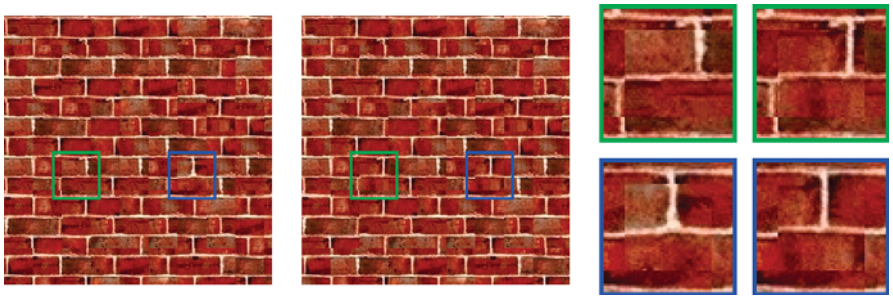


Figure 4.10: Differences between GNCC border matching and color differences matching. GNCC matching (on the left) can sometimes give slightly more blocky results, while color differences matching (on the right) may adjust sharp edges slightly worse.

illumination. Moreover, the color differences needs a few less linear-time operations to be computed, thus it is roughly a constant factor faster.

Figure 4.10 shows an example of the explained before. GNCC matching adjusts the lines between blocks slightly better but color differences matching preserves brightness and contrast local uniformity somewhat better. Nonetheless, both approaches show similarly good results.

Although in most cases it is enough to look for the best border-matching piece of texture from the input without further restrictions, sometimes this does not ensure that the global structure of the texture is preserved. To avoid this possible structure misreproduction, we limit the search space as in the previous section, but with a looser restriction. In section 4.2.1, we considered only those patches as candidates whose distance from the current position is a multiple of the translation vectors $\mathbf{v_1}, \mathbf{v_2}$. Now, we allow a certain tolerance around those candidates, i.e. patches that are a certain configurable number of pixels apart from the former candidates are also taken into account. A tolerance of ± 3 pixels has been enough in all studied cases. This ensures structure preservation and allows little adaptative adjustments that may be required in some cases (e.g. when the translation vectors are not completely accurate. See section 3.3).

Figure 4.11 shows an example of synthesis after the *Gap Filling* step and the result obtained with the *Best Self-similar Tile Repetition* method described in section 4.1. It can be observed that the previous *Random Sampling* step has effectively introduced stochastic irregularities in the output without introducing visible repetitions and making the result look more natural than if produced by the *Best Self-similar Tile Repetition* method. Nonetheless, some visible transitions between neighboring blocks appear. The next step helps smooth this visible transitions and obtain the final result.

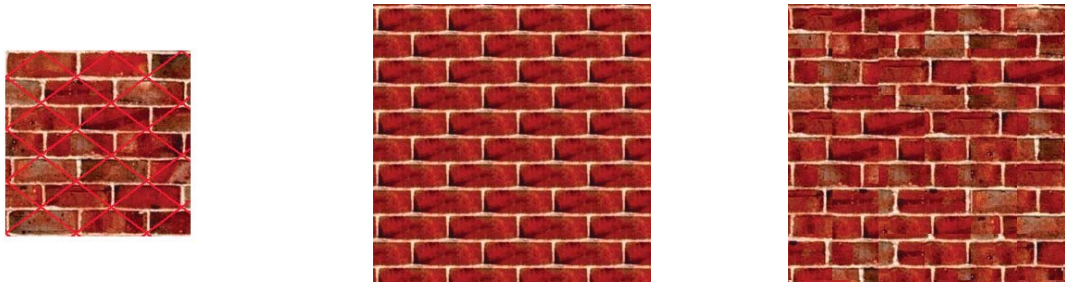


Figure 4.11: Result after *Gap Filling* compared to *Best Self-similar Tile Repetition*. Left: input texture with its lattice superimposed as detected by the analysis step (see section 3). Center: result with *Best Self-similar Tile Repetition*. Right: result after *Gap Filling* step.

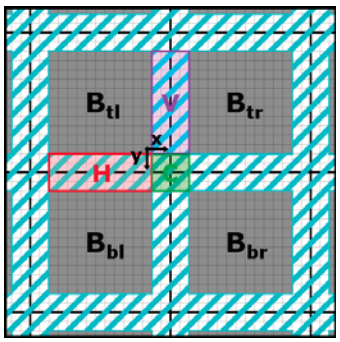


Figure 4.12: Blending sketch.

4.2.3 Final Composition and Blending

At the end of the previous step we have all the blocks in which we had split the output space assigned to a patch from the input. As a result, we already have a complete synthesized output image of the desired size. However, the synthesized texture may still present visible discontinuities at block junctions due to the stochastic nature of the input texture. We propose here a linear blending of overlapping borders around blocks as a way to effectively reduce this effect.

Figure 4.12 illustrates the blending composition at edges between neighboring blocks. The blending is applied to the area in stripes, smoothing the black edges between blocks. There are 3 different cases of combination in the blending area for wich we give the mathematical description:

Vertical edge (V) where one block on the left (B_l) meets its right neighbor (B_r):

$$\alpha_x B_l + (1 - \alpha_x) B_r \tag{4.6}$$

Horizontal edge (H) where one block on the top (B_t) meets its neighbor below (B_b):

$$\alpha_y B_t + (1 - \alpha_y) B_b \tag{4.7}$$

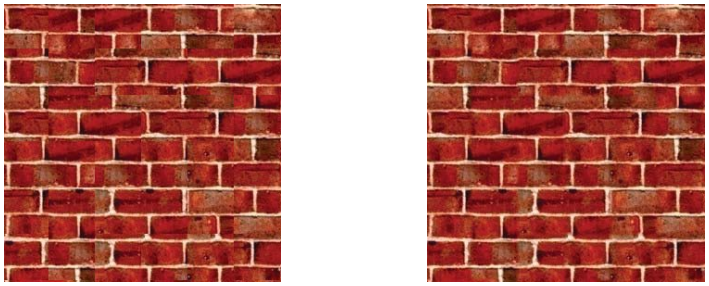


Figure 4.13: Final synthesis result. Left: output without smoothing of edges between blocks. Right: final result after transitions have been smoothed.

Corner (C) in which 4 different blocks (top-left B_{tl}, top-right B_{tr}, bottom-left B_{bl} and bottom-right B_{br}) intervene:

$$\alpha_x\alpha_y B_{tl} + (1-\alpha_x)\alpha_y B_{tr} + \alpha_x(1-\alpha_y)B_{bl} + (1-\alpha_x)(1-\alpha_y)B_{br} \tag{4.8}$$

The previous formulas define the linear combination that is applied to each RGB pixel in the blending area of the participating blocks where

$$\alpha_x = 1 - \frac{x}{w+1} \tag{4.9}$$

$$\alpha_y = 1 - \frac{y}{w+1} \tag{4.10}$$

and where w is the width of the blending area and the reference point $x = 0, y = 0$ is situated at the last bottom-right pixel of the top-left block that is not in the blending area.

The final result effectively smooths the visible transitions and enhances the previous output. Figure 4.13 presents the final smoothed result compared to the output obtained just after *Gap Filling* for the example input image of Figure 4.11.

4.3 Conclusion

Section 4.1 describes a procedure to find within the input image the tile-shaped patch that better reproduces the regular part of the input texture. It is straightforward that this gives optimum results if the input texture is strictly regular or the irregularities are caused by noise in the capture of the texture sample and its reproduction is to be avoided.

We do not further discuss the performance of the best self-similar tile repetition method proposed in Section 4.1. Instead, we use its output to illustrate that textures with subtle yet characteristic deviations from regularity (i.e. near-regular textures) are more faithfully reproduced with our main synthesis method described in Section 4.2.

This and other advantages of our proposed technique over existing methods along with other considerations and possible disadvantages are presented in the following Chapter 5.

5 Results and Evaluation

In this section, we will evaluate the performance of our texture synthesis technique. First, we will concentrate on the analysis stage of our algorithm, presenting example results and comparing and discussing other existing approaches in section 5.1. Then, we will discuss existing trade-offs in the election of block/border sizes and compare final synthesized results with other existing methods in section 5.2.

5.1 Analysis Evaluation

We have run our fully automatic regular structure detector with several texture samples (see figures 5.2 to 5.11).

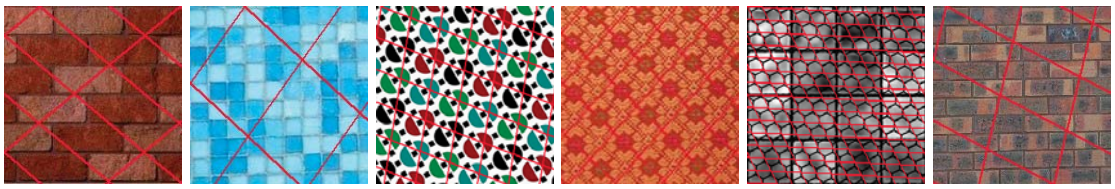
A part from some rare cases, our approach works fine with all examples that we have tried. We see that our analysis method has some advantages over other existing approaches. The main advantages of our regular structure detector is that it estimates the *tile* correctly in most cases and with no user assistance at all. Other existing methods require an important amount of user intervention [LCT04, LHW$^+$04, LHW$^+$06].

Liu et al. [LCT04] presented an automatic method based on the selection of peaks in the (unnormalized) autocorrelation by thresholding their regions of dominance and then estimating the shortest translation vectors with a Hough transform approach. The use of a (more robust) normalized autocorrelation in our approach makes lattice-related maxima stand out from other spurious maxima and allows our simpler approximation. Moreover, Liu et al. exploit intensity images whereas we exploit all three RGB channels, thus introducing sensitivity to color in case of similar intensities.

Nicoll et al. [NMMK05] performed a fractional Fourier analysis to isolate the regular part from the irregularities and generate a *Fractional Fourier Texture Mask* to constrain the synthesis. The main advantage of their method is that it works even if the input sample only contains a single tile. However, a number of different tiles are needed to reproduce the stochastic nature of the irregularities. In addition, the frequencies are extracted with an intensity threshold that is dependent on each particual case and needs especific tunning and the generated texture masks have degeneration problems if the extracted frequencies are not completely accurate. In our experience, the degeneration occurs in most cases when the output size is big compared to the input sample.

Although the smaller tile was estimated correctly in most cases, we have detected that the procedure has problems in some very specific cases (see Figure 5.1):

- The texture is composed of basic elements of irregularly varying color but arranged regularly ((a) and (b) in Figure 5.1). If texture elements are uniformly colored, this may cause our analysis procedure to detect a bigger yet still valid tile.



(a) (b) (c) (d) (e) (f)

Figure 5.1: Examples of cases where our lattice detector fails to estimate correctly.

- The texture is composed of basic elements of regularly varying color and arranged regularly ((c) and (d) in Figure 5.1). The color variation may not lower the spurious maxima enough and mislead the estimation to a smaller tile, especially if it is combined with geometric distortion.
- The texture is a combination of various (near-)regular textures of different lattices with no common multiple in the input sample ((e) in Figure 5.1).
- The texture is highly geometrically distorted or the irregularities are similarly energetic to the regular structure ((f) in Figure 5.1).

5.2 Synthesis Evaluation

The evaluation of the synthesis step is dedicated to the *Costrained Random Sampling and Gap Filling* technique, and results obtained by *Best Self-similar Tile Repetition* are used to illustrate how the introduction of random irregularities improves the output.

The synthesis stage has a few parameters that need to be configured in advance (see section 4):

- The size of the blocks in which the output space is split.
- The width of the boundary that forms the neighborhood of the gaps.
- The width of the blending area at the edges between blocks.

Although configurable, our results were obtained with the same configuration execept for some cases were a wider blending area was needed to better reduce visible transitions between neighboring blocks.

Synthesis tests show that a trade-off exists in the election of an appropriate block size. Larger block sizes make the synthesis faster and are likely to produce nicer results in general, because they create less transitions. However, larger blocks also reduce the randomness of the irregularities and may cause noticeable repetition, something that may be caused by a sample containing too few tiles as well. All presented results over $128px \times 128px$ texture sample images were obtained with $40px \times 40px$ blocks to produce $256px \times 256px$ synthesized outputs. Regardless of the block size, the process successfully preserves the regular pattern of the input texture as long as the lattice was detected

correctly. Compared to direct tiling of a representative tile, the ensured randomness of the output makes the result look far more natural even when the irregularities create visible seams.

Regarding the width of the neighborhood border width for the gap filling substep, a constant value of $10px$ width has been used in all presented results. A thiner border is more tolerant to local deformations, but, as it is more sensitive to local irregularities, it may adjust the dominant global structure worse.

The width of the blending area has remained a constant value of $4px$ width except for few specific cases for which extra blending has smoothed residual visible transitions between blocks.

Figures 5.2 to 5.7 show that our synthesis technique outperforms other existing patch-based approaches when applyed to NRT Type I (see Figure 1.2). Our results are presented along with the output of direct tiling and the output of an implementation of IQ [EF01] as a representative of existing patch-based techniques. The IQ algorithm worked with an equivalent configuration of $50px \times 50px$ blocks and $10px$-width overlappings. All RSGF results enhance the appearance of direct tiling and have better (or equal) quality than IQ results.

We observe that IQ (with constant configuration and no specific tuning) may sometimes suffer from a phenomenon analogous to the *garbage accumulation* of pixel-based synthesis algorithms. The sequencial synthesis implies that the selection of every new patch is dependent on what has been previously synthesized. This may cause visible repetitions in the output or boundary missmatch propagation.

Our method does neither suffer from *garbage accumulation* that pixel-based methods report [EL99, WL00] nor from any kind of accumulated misalignment or missmatch propagation that other patch-based techniques [LLX+01, EF01, KSE+03] may cause. The fact that each second block is filled with a "fresh" aligned patch in the *constrained random sampling* substep, makes each boundary matching independent from the others (*gap filling*). In this way, the maximum disagreement between neighboring blocks is limited regardless of the output size in our technique.

Another important advantage of breaking the dependency on what has been previously synthesized is that the stochastic nature of the irregularities is better reproduced in the output. Our *constrained random sampling* step makes sure that "rich" randomness is introduced in the synthesized output. Figures 5.8 and 5.9 show examples where our technique produces more random irregularities than IQ.

We have observed that the output can sometimes look blocky. This mainly happens when the input texture has non-uniform lighting or each composition unit (each brick in a brick wall or tile in a tiled floor) of the texture takes a different color with no particular rule. This blockiness can be effectively reduced if the width of the blending area is increased. Figure 5.12 shows how the output is effectively improved when changing from a $4px$-width blending area to a $28px$-width blending area.

Non-sequencial synthesis has also some disadvantages. The main disadvantage is that it is somtimes more prone to produce visible boundary missmatch artifacts (discontinuities at block junctions). This is usually caused by an input texture that presents important geometric distorsions or by little disagreements between especially noticeable features, such as sharp edges between bricks in a brick wall. IQ's sequential synthesis might avoid this artifacts at the price of breaking the global repetition pattern. Figure 5.10 shows examples where our method has more visible boundary missmatch artifacts than IQ. Note that the IQ breaks the structure one long, one short of the grayscale brick wall, produces a long brick in the color brick wall and does not conserve the window size in the windows texture.

We have also observed that some discontinuities at block junctions are created because the actual translation vectors of the texture have non-integer coordinates (subpixel level) but we estimate them as the pixelwise position of the corresponding peaks in the autocorrelation. This makes the location of similar patches not completely accurate. Other especially noticeable disagreements are mainly caused by irregular geometric deformations or non-uniform lighting in the input. Figure 5.11 shows some of these cases.

Compared to texture synthesis approaches that are also specialized in near-regular textures, our method has also some advantages. The main advantage is the high amount of automatism that we achieve. In most cases, we get high quality results fully automatically, i.e. with constant (default) configuration. A part from this important advantage, we believe our method overcomes other weaknesses of some existing techniques.

Fractional Fourier Texture Mask [NMMK05] report degeneration problems if the frequencies are extracted inaccurately. In our experience, the bigger the output is (compared to the input sample), the more unlikely is that the estimated frequencies are accurate enough to avoid degeneration problems. Degeneration of any kind is absolutely impossible in our approach. In our case, the expected local quality is uniform across the output space due to the independence of each random block/gap from the others. In addition, the use of masks to preserve the regularity is less tolerant to geometric deformations than the only-boundary check of our method (*gap filling* substep).

Liu et al. [LTL05] estimate the underlying regular lattice of the texture using the method in [LCT04]. Then, they proceed in the lattice direction to fill the image with rectangles containing a complete tile. They split the input in disjoint contiguous tiles and select the alignment of the division manually. The size of our sampling unit is not related to the tile of the texture. This allows the creation of "new" tiles composed of pieces of different tiles if the synthesis blocks are smaller than the tile, thus resulting in a richer (more random irregularities) output. On the other hand, if the tile is small, a block size containing several tiles proportionally accelerates the synthesis.

Other methods not specialized in near-regular textures are focused on preserving local properties and generally fail to faithfully reproduce the global structure of the input texture [EL99, WL00, EF01, LYS01, Ash01, KSE+03]. We have also tested our synthesis approach with some far from regular and some completely stochastic textures

skiping the analysis step and taking $\mathbf{v_1} = [1, 0]^T$, $\mathbf{v_2} = [0, 1]^T$, thus unconstraining the selection of patches. The results are as good or better than with existing patch-based methods in many cases, but the sequential pasting of other techniques seems to be more appropriate for some non-completely stochastic (irregular) textures. Results are shown in figures 5.13 to 5.17.

Finally, our procedure also has some advantages regarding synthesis speed. The most interesting advantage to us is that the two substeps of the synthesis are inherently parallelizable. The random election of the patch to fill each block in the first substep is independent of the others. The same happens in the second substep: once every second block has been synthesized, the patch that better agrees with the sorrounding neighbors of each resulting gap is independent of the others. Moreover, the nearest neighbor search that determines which patch agrees better with the sorrounding blocks of each gap could be made as in [LYS01] thus achieving real-time synthesis.

Figure 5.2: Results: NRT Type I textures are better synthesized with our method than with other existing patch-based approaches - Part I. In columns from left to right: input texture sample with the detected lattice superimposed in red, output of best self-similar tile repetition, our main synthesis technique (RSGF) and an implementation of IQ [EF01].

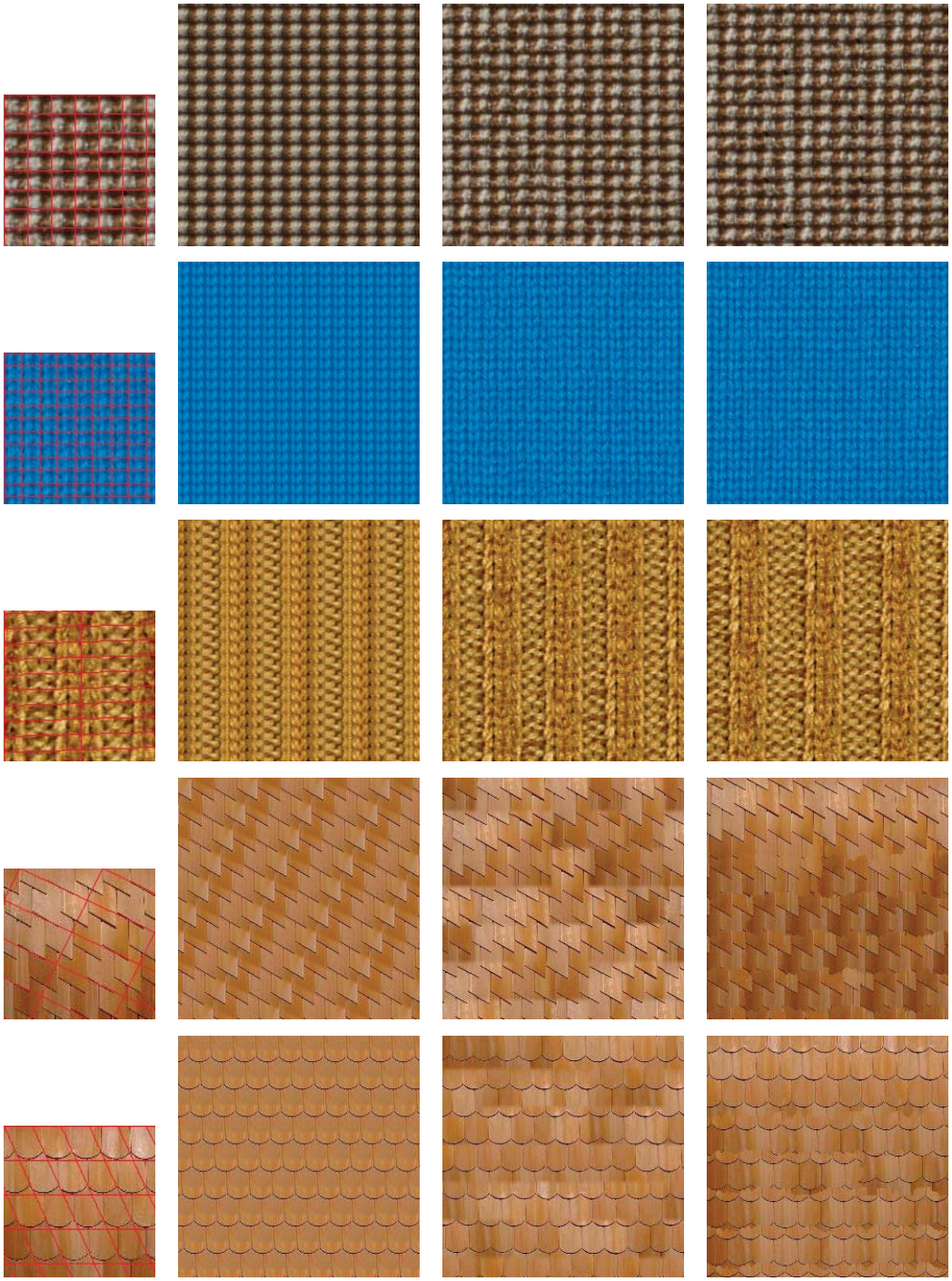


Figure 5.3: Results: NRT Type I textures are better synthesized with our method than with other existing patch-based approaches - Part II. In columns from left to right: input texture sample with the detected lattice superimposed in red, output of best self-similar tile repetition, our main synthesis technique (RSGF) and an implementation of IQ [EF01].

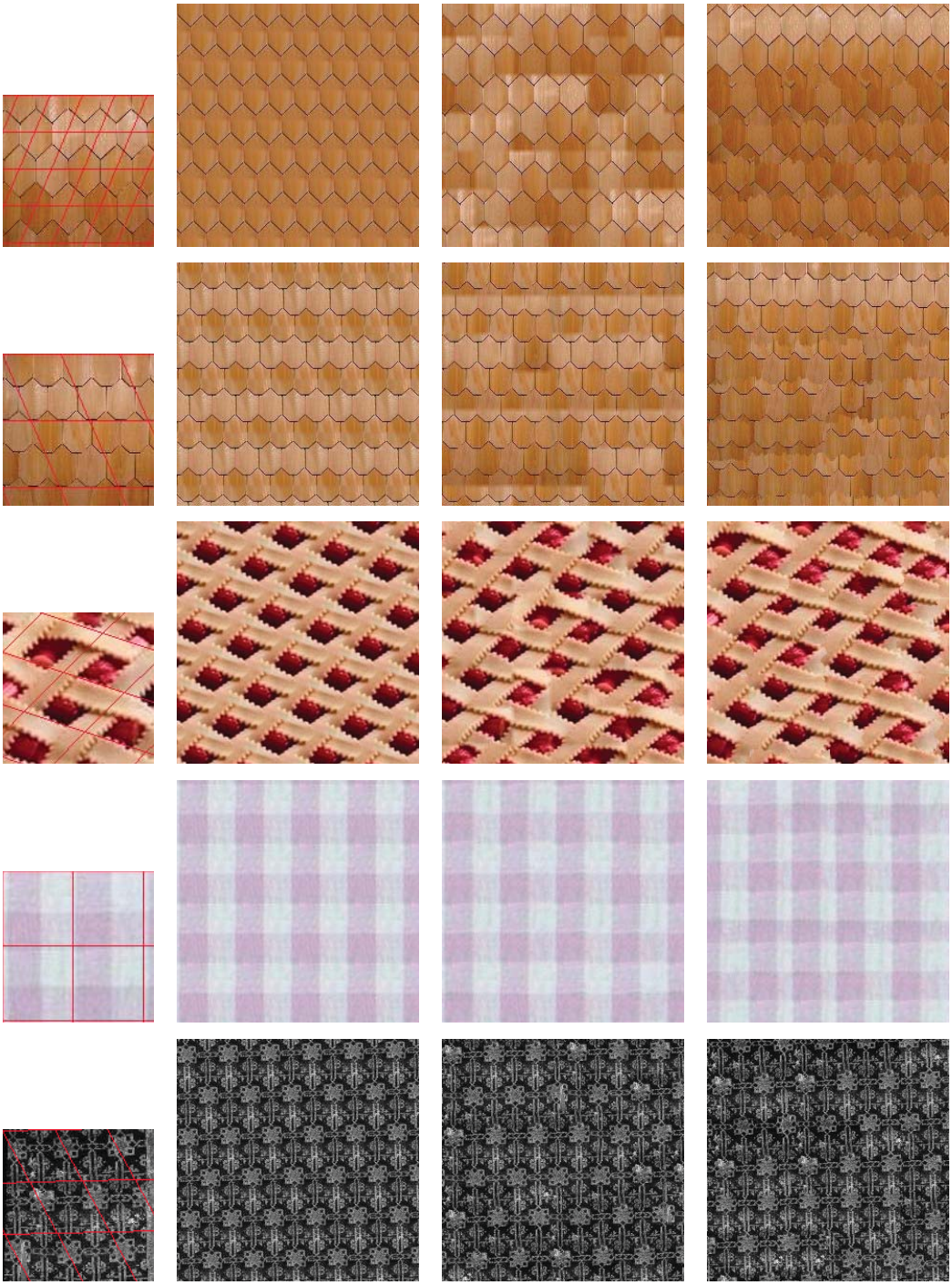


Figure 5.4: Results: NRT Type I textures are better synthesized with our method than with other existing patch-based approaches - Part III. In columns from left to right: input texture sample with the detected lattice superimposed in red, output of best self-similar tile repetition, our main synthesis technique (RSGF) and an implementation of IQ [EF01].

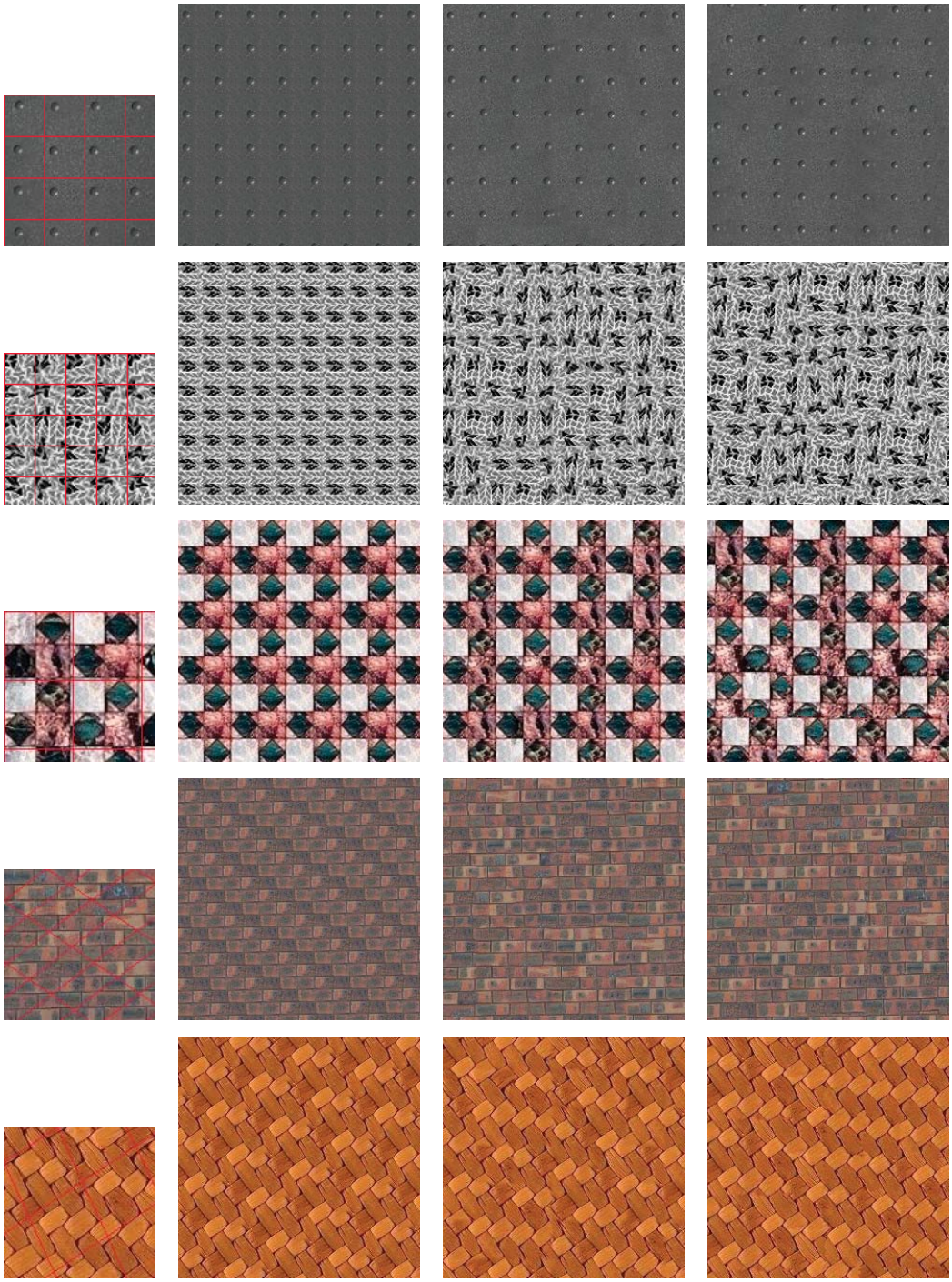


Figure 5.5: Results: NRT Type I textures are better synthesized with our method than with other existing patch-based approaches - Part IV. In columns from left to right: input texture sample with the detected lattice superimposed in red, output of best self-similar tile repetition, our main synthesis technique (RSGF) and an implementation of IQ [EF01].

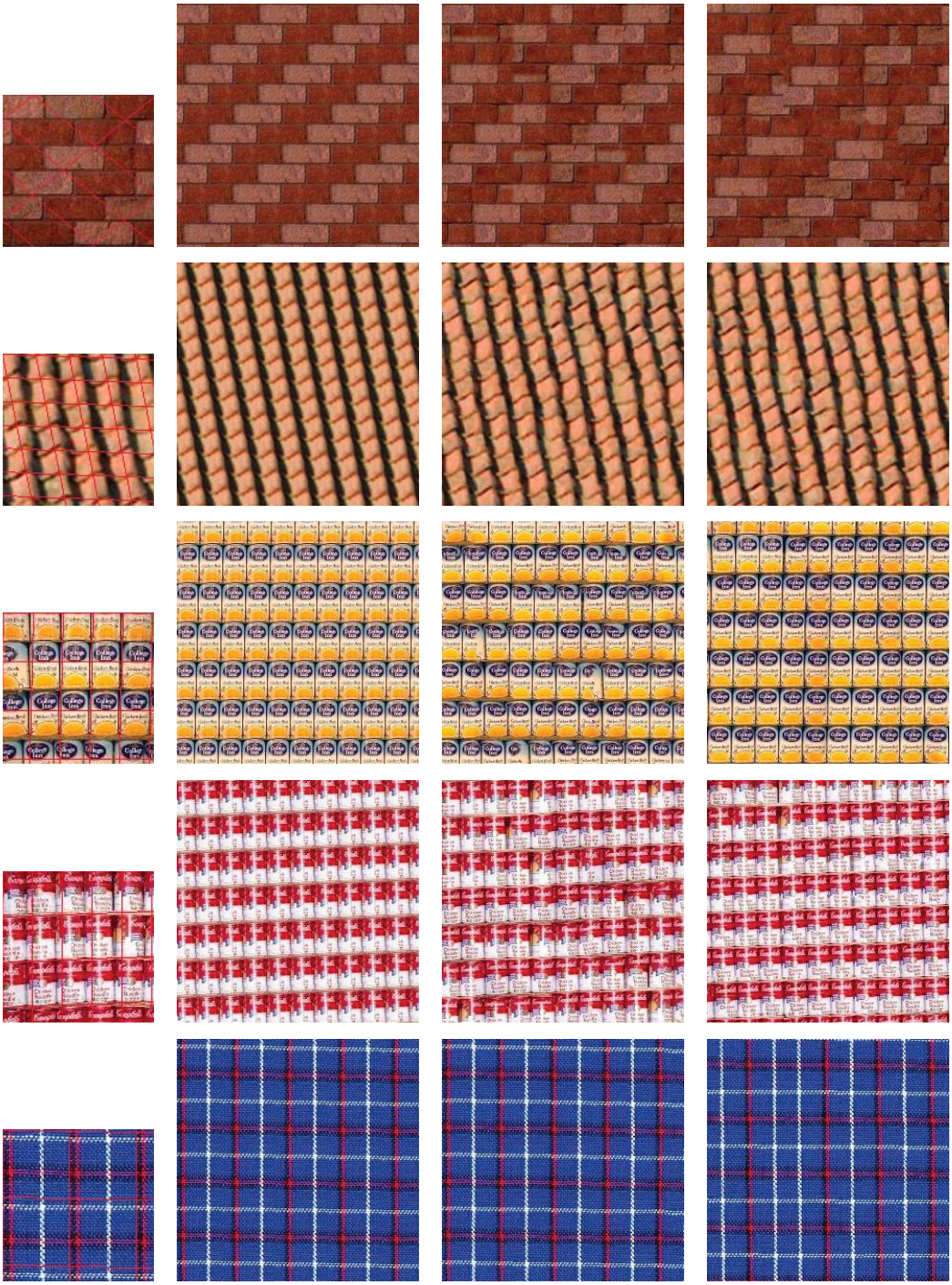


Figure 5.6: Results: NRT Type I textures are better synthesized with our method than with other existing patch-based approaches - Part V. In columns from left to right: input texture sample with the detected lattice superimposed in red, output of best self-similar tile repetition, our main synthesis technique (RSGF) and an implementation of IQ [EF01].

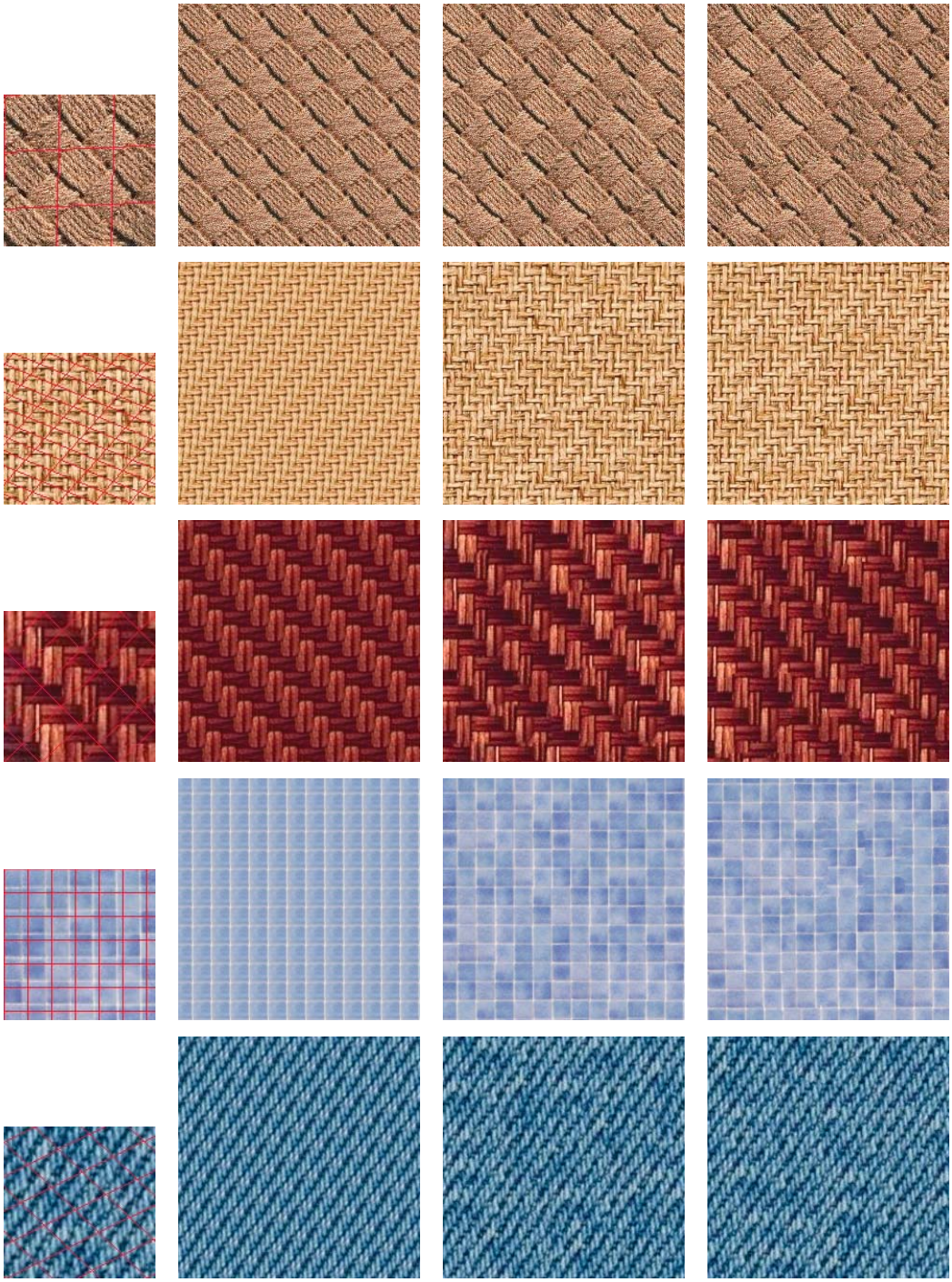


Figure 5.7: Results: NRT Type I textures are better synthesized with our method than with other existing patch-based approaches - Part VI. In columns from left to right: input texture sample with the detected lattice superimposed in red, output of best self-similar tile repetition, our main synthesis technique (RSGF) and an implementation of IQ [EF01].

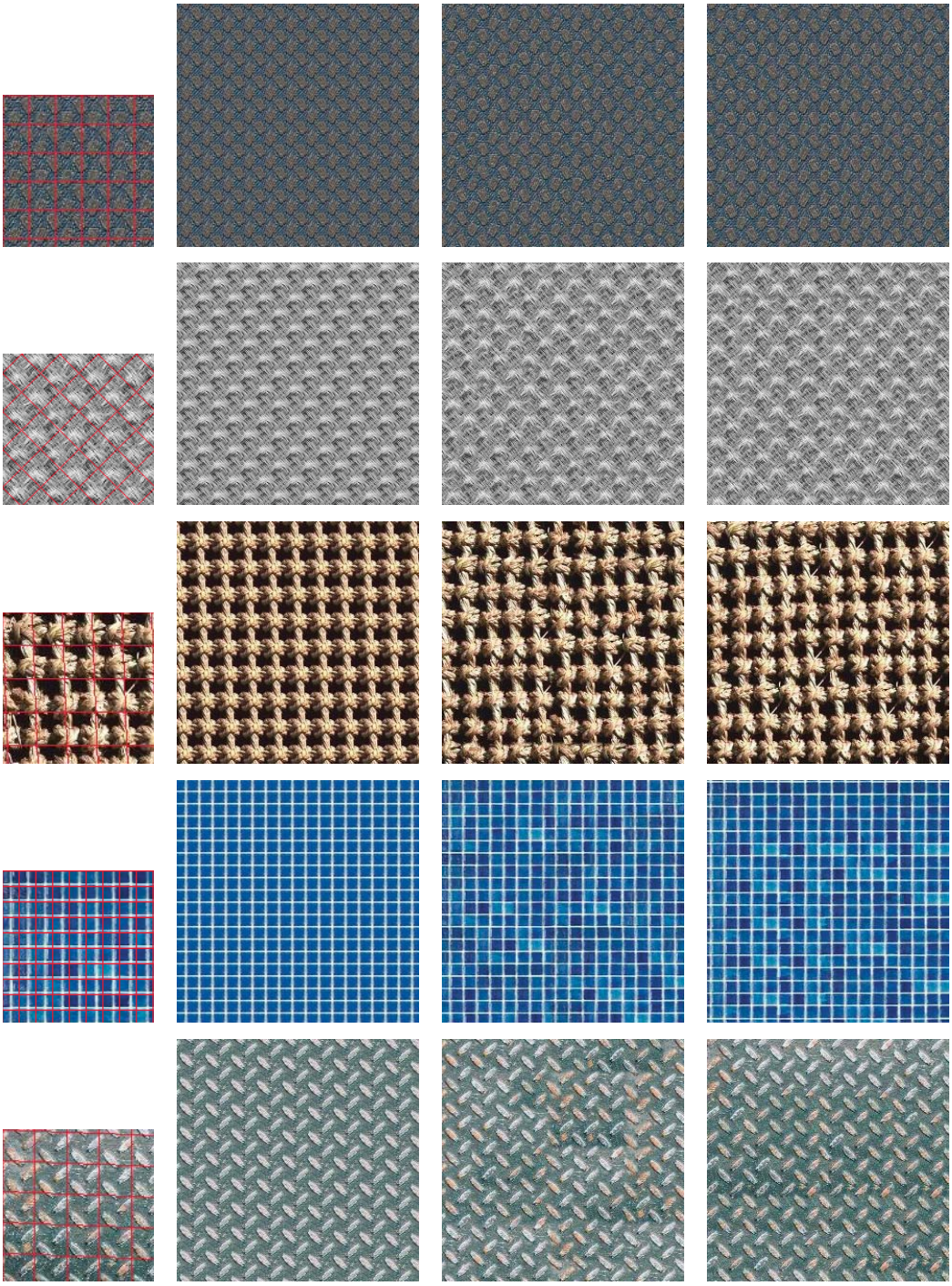


Figure 5.8: Results: our method ensures the output "rich" random irregularities - Part I. In columns from left to right: input texture sample with the detected lattice superimposed in red, output of best self-similar tile repetition, our main synthesis technique (RSGF) and an implementation of IQ [EF01].

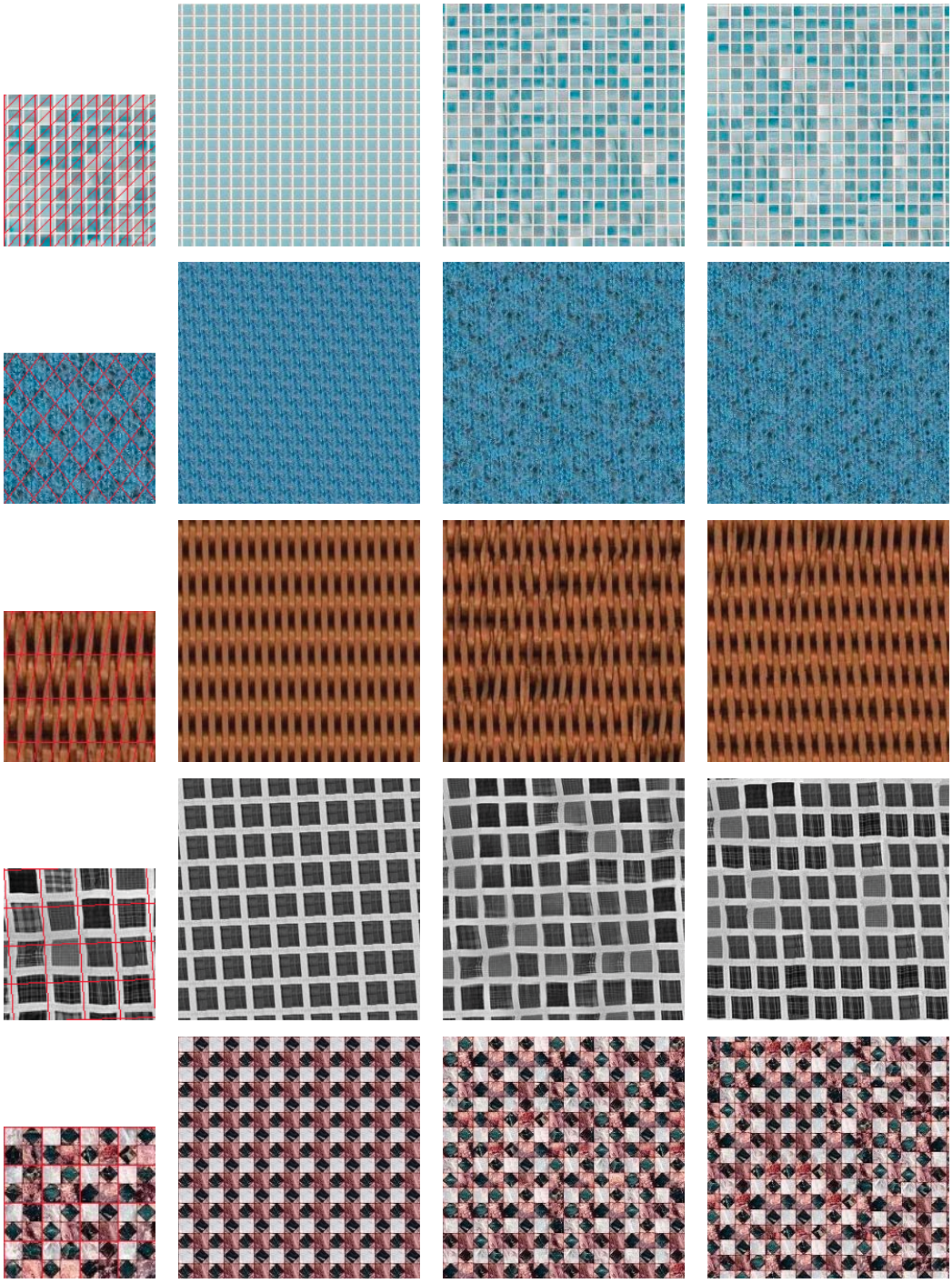


Figure 5.9: Results: our method ensures the output "rich" random irregularities - Part II. In columns from left to right: input texture sample with the detected lattice superimposed in red, output of best self-similar tile repetition, our main synthesis technique (RSGF) and an implementation of IQ [EF01].

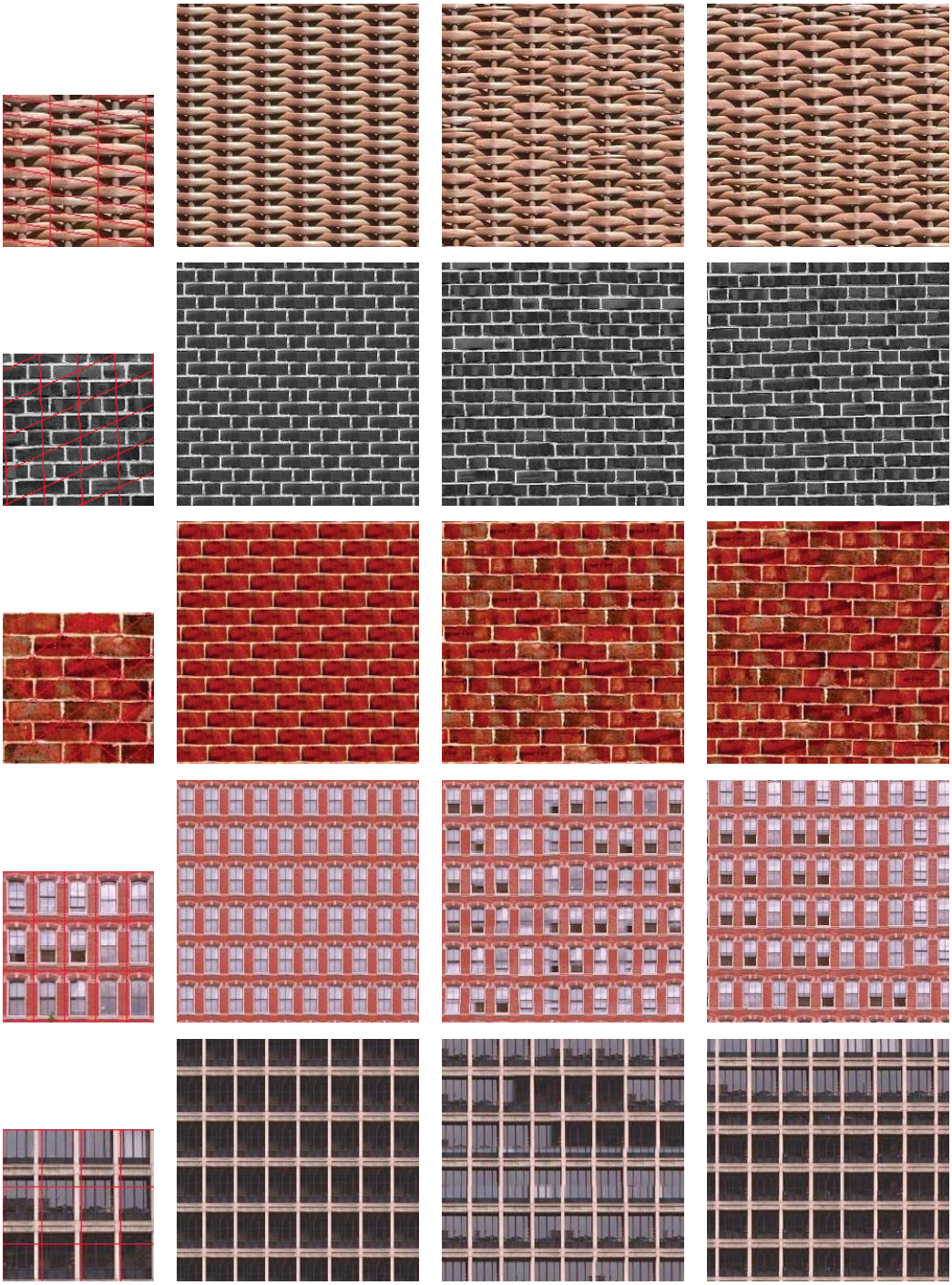


Figure 5.10: Results: in some cases, the price of ensured regularity reproduction is the appearance of boundary missmatch artifacts. In columns from left to right: input texture sample with the detected lattice superimposed in red, output of best self-similar tile repetition, our main synthesis technique (RSGF) and an implementation of IQ [EF01].

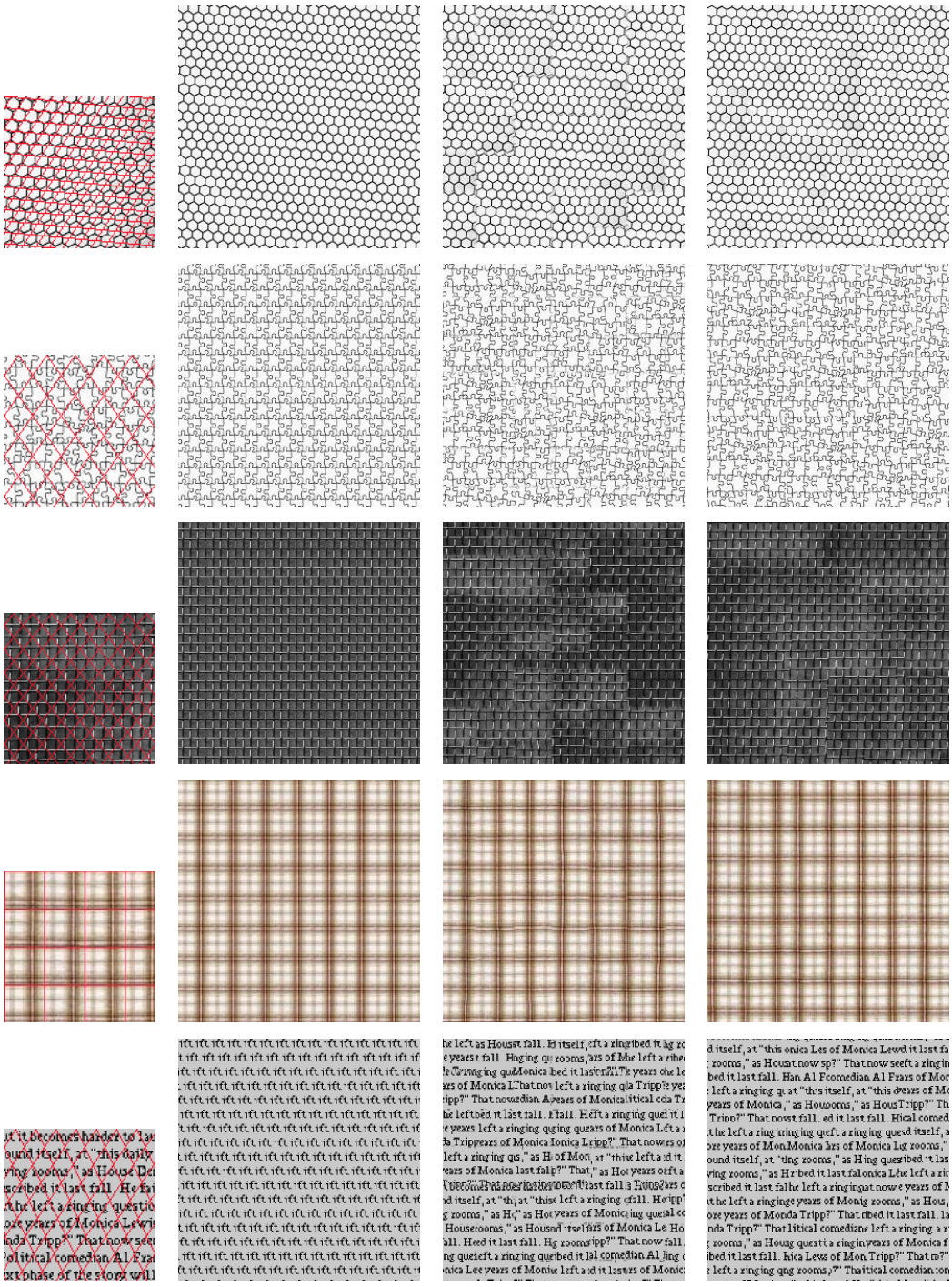


Figure 5.11: Results: if the tile of the texture has non-integer coordinates or the texture sample presents geometric distortions, visible boundary missmatch artifacts may appear. In columns from left to right: input texture sample with the detected lattice superimposed in red, output of best self-similar tile repetition, our main synthesis technique (RSGF) and an implementation of IQ [EF01].

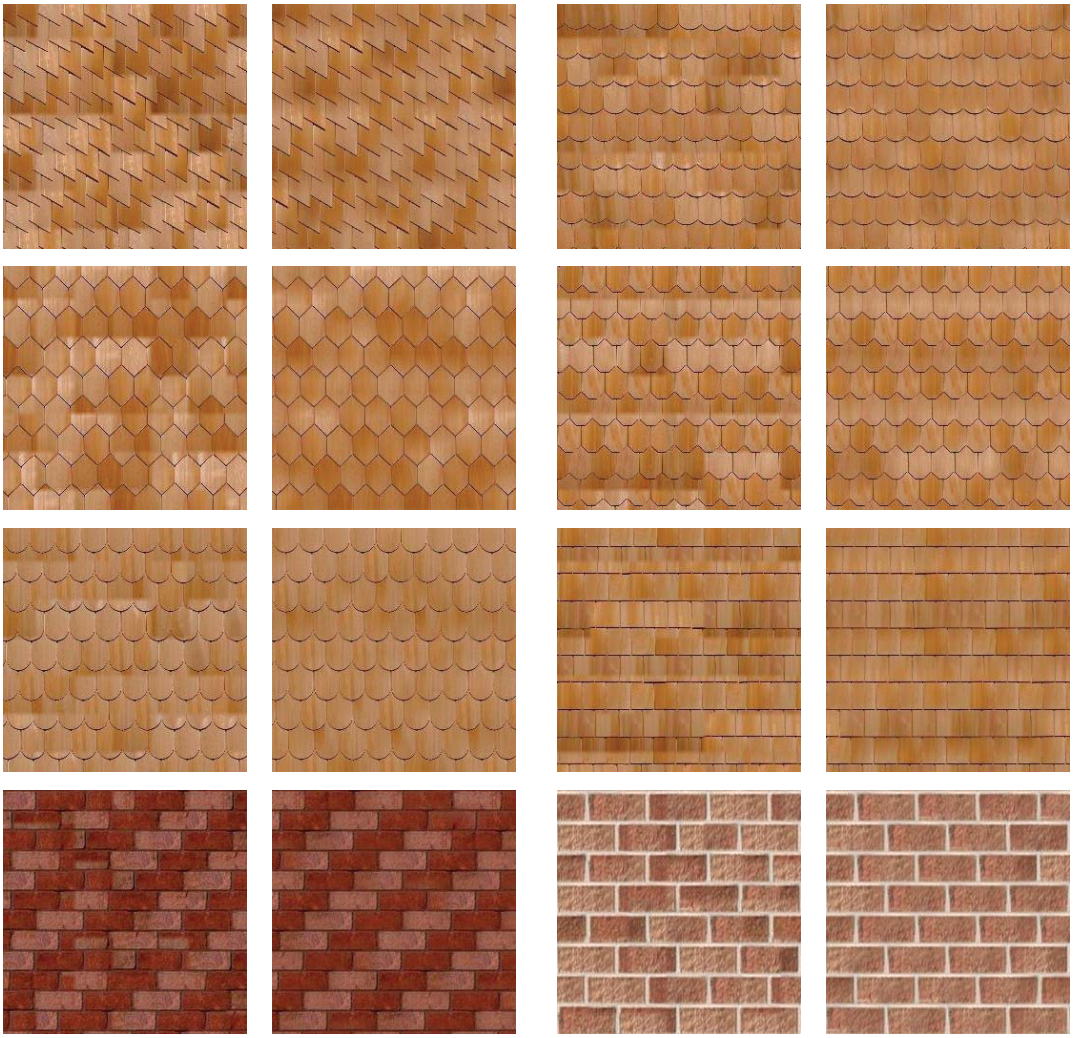


Figure 5.12: Results: blockiness appearance can be effectively reduced by applying a wider blending area between blocks. For each pair of images, execution result with a 4px-width blending area and a 28px-width blending area respectively.

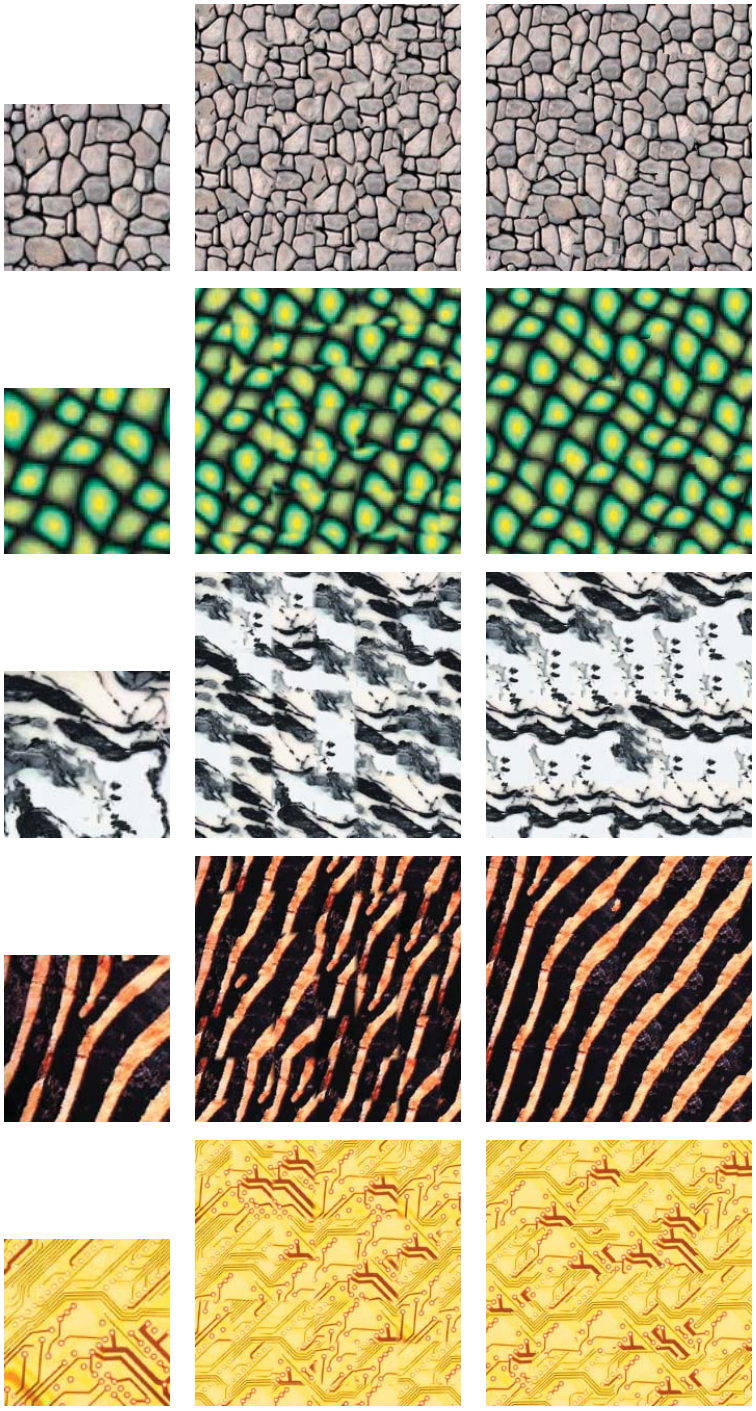


Figure 5.13: Results achieved with our method for irregular textures - Part I. In columns from left to right: input texture sample, output of our main synthesis technique (RSGF) with $\mathbf{v_1} = [1, 0]^T$, $\mathbf{v_2} = [0, 1]^T$ and of an implementation of IQ [EF01].

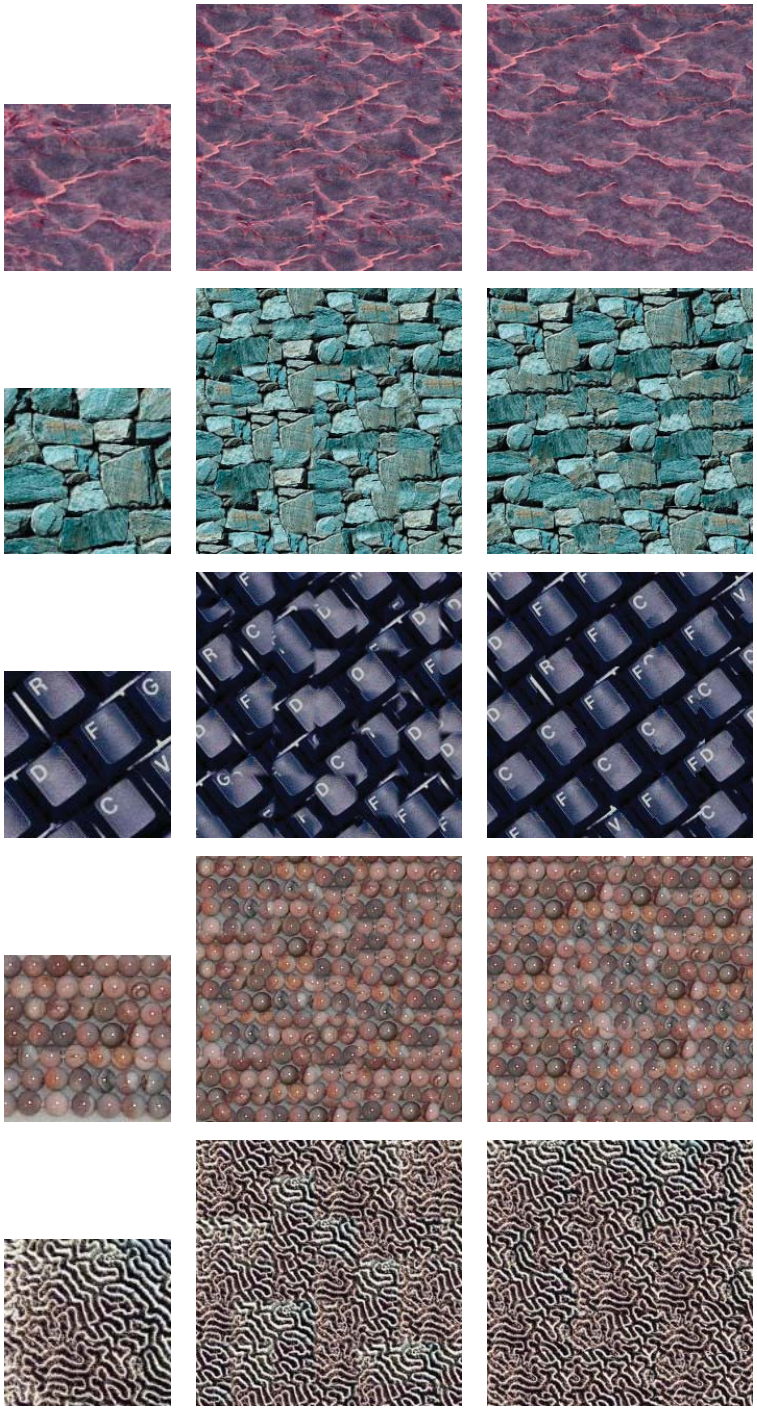


Figure 5.14: Results achieved with our method for irregular textures - Part II. In columns from left to right: input texture sample, output of our main synthesis technique (RSGF) with $\mathbf{v_1} = [1, 0]^T$, $\mathbf{v_2} = [0, 1]^T$ and of an implementation of IQ [EF01].

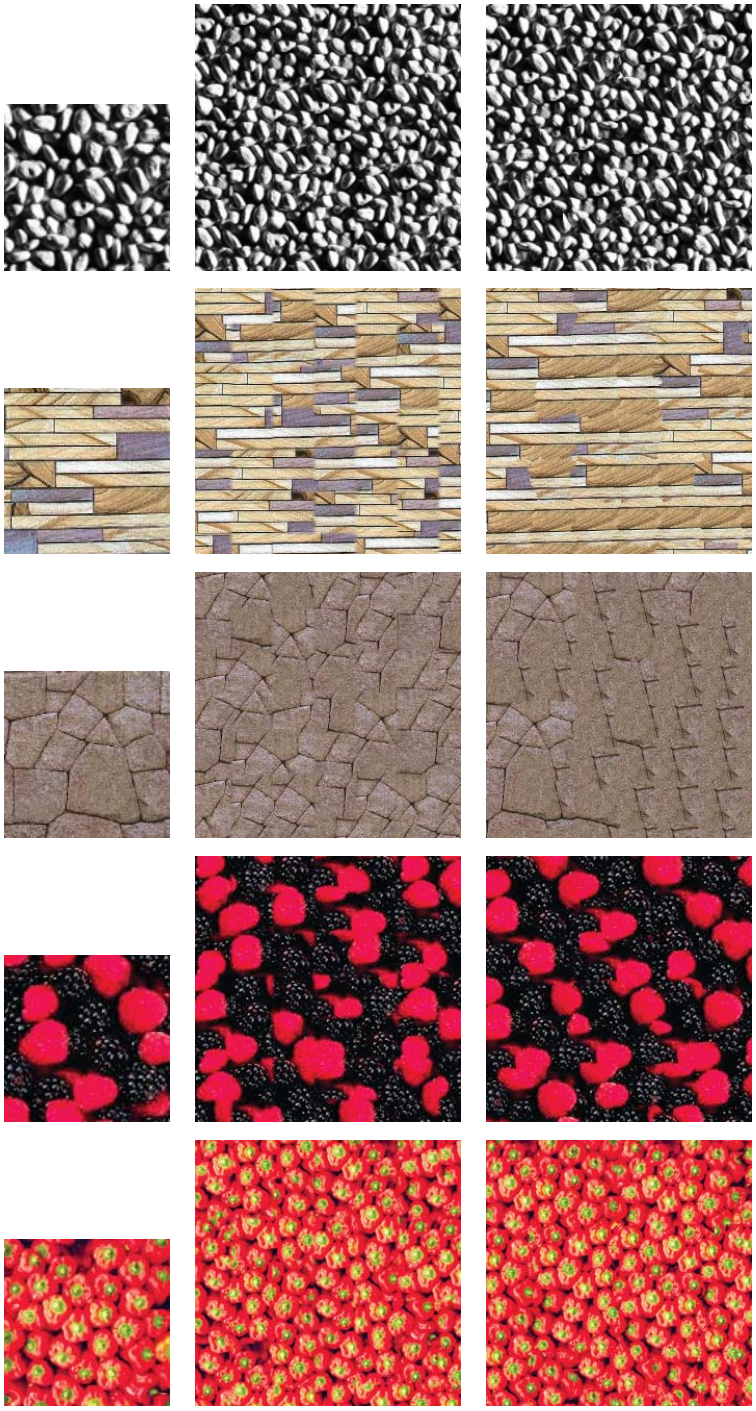


Figure 5.15: Results achieved with our method for irregular textures - Part III. In columns from left to right: input texture sample, output of our main synthesis technique (RSGF) with $\mathbf{v_1} = [1, 0]^T$, $\mathbf{v_2} = [0, 1]^T$ and of an implementation of IQ [EF01].

Figure 5.16: Results achieved with our method for irregular textures - Part IV. In columns from left to right: input texture sample, output of our main synthesis technique (RSGF) with $\mathbf{v_1} = [1, 0]^T$, $\mathbf{v_2} = [0, 1]^T$ and of an implementation of IQ [EF01].

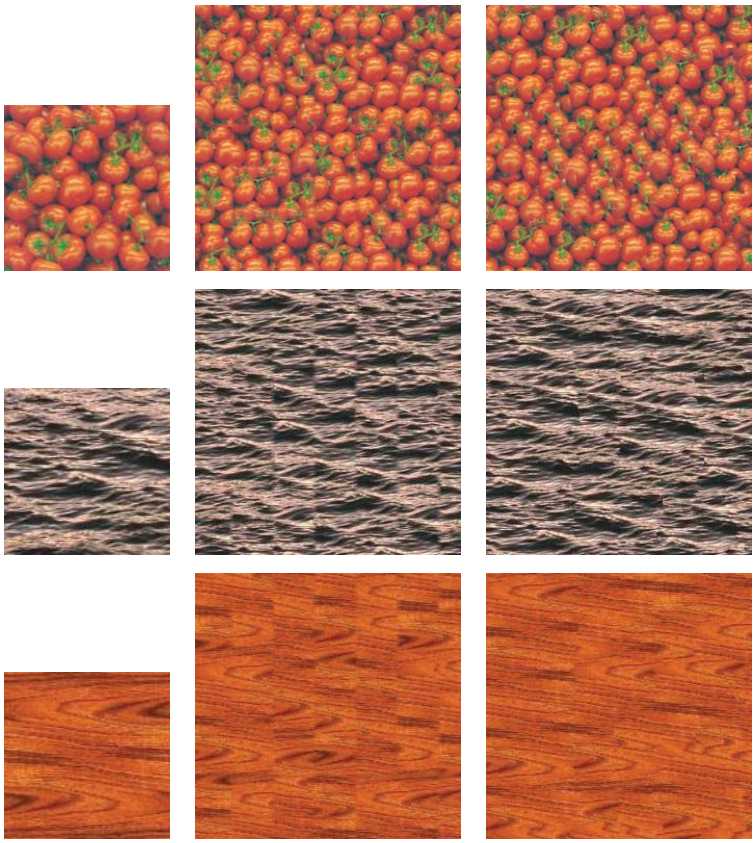


Figure 5.17: Results achieved with our method for irregular textures - Part V. In columns from left to right: input texture sample, output of our main synthesis technique (RSGF) with $\mathbf{v_1} = [1, 0]^T$, $\mathbf{v_2} = [0, 1]^T$ and of an implementation of IQ [EF01].

6 Conclusions and Future Work

In this thesis, a new approach for near-regular texture synthesis has been proposed. Two are the key observation behind this method. On the one hand, a generalized version of the normalized cross-correlation paves the way to a simpler than [LTL05] estimation of the two independent vectors defining the translational symmetry of the texture under analysis. On the other hand, once the two translation vectors are know, we know that there are examples of how the texture should look like at a distance multiple of the translation vectors for any position in the 2D space. There is no need to take a whole tile as the unit of sampling. We use the latter observation to sparsely introduce random but *suitable* patches from the texture sample in the output, thus ensuring the randomness of the characteristic irregularities of the texture. Finally, the gaps left by the previous sparse synthesis are filled with suitable best agreeing patches from the input.

Some examples have shown the high quality of the results that our method can achieve. We preserve the global structure of near-regular textures that other approaches fail to reproduce. Moreover, our sparse introduction of random patches in the output has some interesting advantages (the maximum disagreement between neighboring blocks is limited and the proces is straighforwardly parallelizable). However, it has some weaknesses that can be improved.

A goodness function based on the regions of dominance proposed in [LCT04] could be investigated to overcome some of the difficulties in the lattice estimation. Other approches for the goodness function or other methodologies to infer the shortest translation vectors from the local maxima of the normalized autocorrelation could be as well researched.

Subpixel level estimation of the translation vectors could be used to either "correct" the input to have integer translational symmetry or refine the location of the *similar* patches when composing the output. Moreover, simple tiling of a representative tile could be used to perform a prior correction of geometric irregularities of the input. Alternatively, adaptative deformations could be applied to each gap in the gap filling step to better agree with its neighbors (e.g. make lines between bricks in a brick wall coincide).

Other ways to finally smooth transitions between neighboring blocks as image quilting [EF01] or graph cuts [KSE+03] instead of linear blending could as well be explored.

Bibliography

[AMN+95] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions. Technical report, University of Maryland at College Park, College Park, MD, USA, 1995. 9

[Ash01] Michael Ashikhmin. Synthesizing Natural Textures. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, I3D '01, pages 217–226, New York, NY, USA, 2001. ACM. 8, 9, 48

[BJEYLW01] Ziv Bar-Joseph, Ran El-Yaniv, Dani Lischinski, and Michael Werman. Texture Mixing and Texture Movie Synthesis Using Statistical Learning. *IEEE Transactions on Visualization and Computer Graphics*, 7:120–135, April 2001. 5

[CSHD03] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang Tiles for Image and Texture Generation. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 287–294, New York, NY, USA, 2003. ACM. 11

[dB97] Jeremy S. de Bonet. Multiresolution Sampling Procedure for Analysis and Synthesis of Texture Images. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 361–368, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co. 5

[DED05] Khalid Djado, Richard Egli, and François Deschênes. Extraction of a Representative Tile from a Near-periodic Texture. In *Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, GRAPHITE '05, pages 331–337, New York, NY, USA, 2005. ACM. 33, 34

[EF01] Alexei A. Efros and William T. Freeman. Image Quilting for Texture Synthesis and Transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 341–346, New York, NY, USA, 2001. ACM. 10, 11, 14, 47, 48, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 61, 62, 63, 64, 65, 67

[EL99] Alexei A. Efros and Thomas K. Leung. Texture Synthesis by Non-Parametric Sampling. In *Proceedings of the International Conference on Computer Vision - Volume 2*, ICCV '99, pages 1033–, Washington, DC, USA, 1999. IEEE Computer Society. 5, 6, 16, 47, 48

[GS86] Branko Grünbaum and G C Shephard. *Tilings and Patterns*. W. H. Freeman & Co., New York, NY, USA, 1986. 11, 17

[HB95] David J. Heeger and James R. Bergen. Pyramid-based Texture Analysis/Synthesis. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 229–238, New York, NY, USA, 1995. ACM. 5

[HJO+01] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image Analogies. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 327–340, New York, NY, USA, 2001. ACM. 8, 9, 15

[KSE+03] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut Textures: Image and Video Synthesis Using Graph Cuts. *ACM Transactions on Graphics, SIGGRAPH 2003*, 22(3):277–286, July 2003. 11, 47, 48, 67

[LCT04] Yanxi Liu, Robert T. Collins, and Yanghai Tsin. A Computational Model for Periodic Pattern Perception Based on Frieze and Wallpaper Groups. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:354–371, March 2004. 14, 45, 48, 67

[Lew95] J. P. Lewis. Fast Normalized Cross-correlation. In *Vision Interface*, pages 120–123. Canadian Image Processing and Pattern Recognition Society, 1995. 17, 19, 21

[LHW+04] W.-C. Lin, J. H. Hays, C. Wu, V. Kwatra, and Y. Liu. A Comparison Study of Four Texture Synthesis Algorithms on Regular and Near-regular Textures. Technical report, School of Coumpter Science Carnegie Mellon University, 2004. 13, 45

[LHW+06] W.-C. Lin, J. H. Hays, C Wu, V. Kwatra, and Y. Liu. Quantitative Evaluation on Near Regular Texture Synthesis. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR 2006)*, volume 1, pages 427 – 434, New York, NY, USA, June 2006. 45

[LL03] Yanxi Liu and Wen-Chieh Lin. Deformable Texture: the Irregular-Regular-Irregular Cycle. Technical Report CMU-RI-TR-03-26, Robotics Institute, Pittsburgh, PA, August 2003. 14, 15

[LLH04] Yanxi Liu, Wen C. Lin, and James Hays. Near-regular Texture Analysis and Manipulation. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 368–376, New York, NY, USA, 2004. ACM. IX, 1, 2, 14, 16, 17

[LLX+01] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time Texture Synthesis by Patch-based Sampling. *ACM Transactions on Graphics*, 20:127–150, July 2001. 10, 47

[LM99] Thomas Leung and Jitendra Malik. Recognizing Surfaces Using Three-Dimensional Textons. In *Proceedings of the International Conference on Computer Vision - Volume 2*, ICCV '99, pages 1010–, Washington, DC, USA, 1999. IEEE Computer Society. 5

[LM01] Thomas Leung and Jitendra Malik. Representing and Recognizing the Visual Appearance of Materials Using Three-dimensional Textons. *International Journal of Computer Vision*, 43:29–44, June 2001. 5

[LTL05] Yanxi Liu, Yanghai Tsin, and Wen-Chieh Lin. The Promise and Perils of Near-Regular Texture. *International Journal of Computer Vision*, 62:145–159, April 2005. 1, 13, 16, 17, 48, 67

[LYS01] Xinguo Liu, Yizhou Yu, and Heung-Yeung Shum. Synthesizing Bidirectional Texture Functions for Real-world Surfaces. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 97–106, New York, NY, USA, 2001. ACM. 48, 49

[NMMK05] A. Nicoll, Jan Meseth, Gero Müller, and Reinhard Klein. Fractional Fourier Texture Masks: Guiding Near-Regular Texture Synthesis. *Computer Graphics Forum*, 24(3):569–579, 2005. 15, 16, 45, 48

[Per85] Ken Perlin. An Image Synthesizer. In *Proceedings of the 12th Annual Cnference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, pages 287–296, New York, NY, USA, 1985. ACM. 5

[PFH00] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped Textures. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 465–470, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 11, 12, 13

[PS00] Javier Portilla and Eero P. Simoncelli. A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients. *International Journal of Computer Vision*, 40:49–70, October 2000. 5

[SCA02] Cyril Soler, Marie-Paule Cani, and Alexis Angelidis. Hierarchical Pattern Mapping. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 673–680, New York, NY, USA, 2002. ACM. 11, 12, 13

[SSSE00] Arno Schödl, Richard Szeliski, David H. Salesin, and Irfan Essa. Video Textures. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 489–498,

New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 9

[SW63] Claude E. Shannon and Warren Weaver. *A Mathematical Theory of Communication*. University of Illinois Press, Champaign, IL, USA, 1963. 5

[TLR01] Yanghai Tsin, Yanxi Liu, and Visvanathan Ramesh. Texture Replacement in Real Images. In *Computer Vision and Pattern Recognition*, pages 539–544, 2001. 14

[Tur91] Greg Turk. Generating Textures on Arbitrary Surfaces Using Reaction-diffusion. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, pages 289–298, New York, NY, USA, 1991. ACM. 5

[Tur92] Greg Turk. Re-tiling Polygonal Surfaces. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '92, pages 55–64, New York, NY, USA, 1992. ACM. 12

[Tur01] Greg Turk. Texture Synthesis on Surfaces. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 347–354, New York, NY, USA, 2001. ACM. 11, 12

[TZL+02] Xin Tong, Jingdan Zhang, Ligang Liu, Xi Wang, Baining Guo, and Heung-Yeung Shum. Synthesis of Bidirectional Texture Functions on Arbitrary Surfaces. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 665–672, New York, NY, USA, 2002. ACM. 5

[WL00] Li-Yi Wei and Marc Levoy. Fast Texture Synthesis Using Tree-structured Vector Quantization. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 479–488, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 5, 6, 7, 47, 48

[WL01] Li-Yi Wei and Marc Levoy. Texture Synthesis over Arbitrary Manifold Surfaces. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 355–360, New York, NY, USA, 2001. ACM. 11, 12

[WLKT09] Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. State of the Art in Example-based Texture Synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR*. Eurographics Association, 2009. 1

[WW91] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques*. ACM, New York, NY, USA, 1991. 15

[XGS00] Ying-Qing Xu, Baining Guo, and Harry Shum. Chaos Mosaic: Fast and Memory Efficient Texture Synthesis. Technical report, Microsoft Research, April 2000. 12

[YHBZ01] Lexing Ying, Aaron Hertzmann, Henning Biermann, and Denis Zorin. Texture and Shape Synthesis on Surfaces. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 301–312, London, UK, 2001. Springer-Verlag. 11, 12

[ZG02] Steve Zelinka and Michael Garland. Towards Real-time Texture Synthesis with the Jump Map. In *Proceedings of the 13th Eurographics Workshop on Rendering Techniques*, EGRW '02, pages 99–104, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association. 5, 9

[ZGWW02] Song Chun Zhu, Cheng-en Guo, Ying Nian Wu, and Yizhou Wang. What Are Textons? In *Proceedings of the 7th European Conference on Computer Vision - Part IV*, ECCV '02, pages 793–807, London, UK, UK, 2002. Springer-Verlag. 5

[ZWM97] Song Chun Zhu, Ying Nian Wu, and David Mumford. Minimax Entropy Principle and Its Application to Texture Modeling. *Neural Computation*, 9:1627–1660, November 1997. 5

[ZWM98] Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, Random Fields and Maximum Entropy (FRAME): Towards a Unified Theory for Texture Modeling. *International Journal of Computer Vision*, 27:107–126, April 1998. 5