Recent Developments in Example-based Texture Synthesis for Graphics Rendering

Carsten Rudolph[1]

Abstract: Textures are essential to allow real-time rendering of computer graphics. While they are easy to use and widely supported by current graphic frameworks, their creation still involves a large amount of manual work. Within the recent years algorithms have been developed that help artists automating common tasks in texture creation. Those tasks can involve texture creation, manipulation and fitting them to certain constraints like tiling, size and spatial uniformity. Example-based Texture Synthesis describes the creation of arbitrarily large textures from a typically smaller and limited exemplar whilst maintaining certain user-defined constraints. In [We09] the authors presented a tutorial-fashioned introduction towards Texture Synthesis combined with a state of the art overview on current algorithms. The goal of this paper is to give a basic introduction, an overview over the recent developments in the area and a comparison of current state-of-the-art algorithms, determining up- and downsides of each of the presented approaches.

Keywords: Computer Graphics, Rendering, Textures, Texture Synthesis, Example-based Texture Synthesis

1 Introduction

In the recent years the available performance and amount of memory on graphics cards has grown considerably, leading to a more realistic look of current graphics applications such as video games, movies or simulations [NV15]. Thus the demand for high quality and resolution textures has grown equally. This typically increases the required work to create great looking textures.

Texture Synthesis aims to automate reoccurring tasks involved in texture creation and give artists a tool to create textures fitting their requirements. Example-based Texture Synthesis is a form of Texture Synthesis where a usually low-resolution exemplar is used to create a completely new texture that should resemble the exemplars global appearance while increasing its original resolution. Also certain constraints can be applied to the exemplar to create textures that can preserve spatial uniformity or make the texture tileable. There are also other constraints but those two are typically common when creating textures for computer graphics.

In 2009 Wei *et al.* [We09] wrote a good introduction into various applications of Texture Synthesis in form of a tutorial for readers who are new to the field. Following that approach, this paper wants to:

[1] Technische Universität Chemnitz, Straße der Nationen 62, 09111 Chemnitz, carsten.rudolph@s2014.tu-chemnitz.de

- Give the reader a short introduction into the topic by describing common algorithms (sections 2 and 3).
- Compare different approaches and try to name applications where one algorithm can be preferred over another one (section 3).
- Give an overview over the recent developments within the field of Texture Synthesis (section 4).

This paper only describes different non-parametric synthesis methods, which include pixel- and patch-based approaches and Texture Optimization. Those are typically used for static, non-volumetric textures, which are most common in the field of real-time rendering. For a detailed insight into other applications, like synthesis of animated or solid textures or real-time Texture Synthesis, the reader is advised to start with the work of Wei *et al.* [We09].

2 Texture Synthesis

Textures are 2- or 3-dimensional sets of values, used to model geometric features that would be to expensive to calculate them each on their own. They can contain different surface information that get interpreted during the rendering process. Typical texture-encoded surface data are normals, depth and color information, but they are not limited to those examples.

Due to the infinite variety of surfaces in the real world, each texture is different. The main question for analytically dealing with textures is to find an appropriate representation for the content of the texture. Section 2.1 tries to implement a general-purpose definition of the appearance of single characteristics within a single texture.

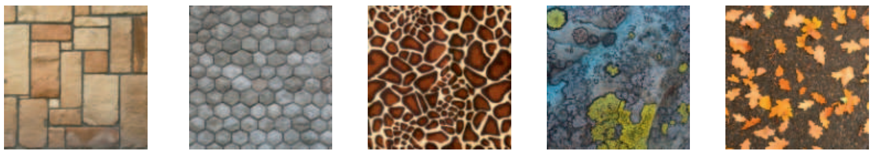
However, different algorithms might choose different descriptions of textures. Section 2.2 shows the basic principles of different common algorithms and their approaches.

2.1 Texture Classes

The distribution of single characteristics of a certain texture highly influences its global and local visual appearance. In order to compare and test algorithms it is important to classify their input. The authors of [Ka15] have chosen to use the following notion to classify their textures:

- **structured** a set of differently scaled, but regularly shaped features (Figure 1a).
- **regular** a set of features which are roughly of same size and shape (Figure 1b).
- **cellular** a set of features with varying size and shape, but clearly delimited towards their neighbors (Figure 1c).

- **semi-structured/stochastic** a random background with randomly spread features of roughly same size and shape (Figure 1e).
- **large-/small-scaled features** a random background with some features that are are not regularly spread over the surface and do not necessarily have the same size nor shape (Figure 1d).



(a) structured (b) regular (c) cellular (d) features (e) stochastic

Fig. 1: Examples of texture classes. The textures are classified in the order they appear. [Ka15]

Classifying textures is a good way to make assertions about how good an algorithm works. To be general-purpose an algorithm should produce good quality outputs for a variety of exemplars from each of those classes.

2.2 Algorithm Classes

Texture Synthesis has a broad field of application, with each one having certain requirements for algorithms. In the field of non-parametric Texture Synthesis there are three major principles to classify algorithms [We09]:

- **Pixel-based** algorithms synthesize their output in a pixel-wise manner. This means, they interpret input textures as a set of pixels and synthesize the result pixel-by-pixel. Each algorithm step synthesizes a new pixel within the output.
- **Patch-based** algorithms increase the unit of interpretation to whole patches instead of pixels. Those are typically larger features that are copied into the output texture where they may be manipulated later on. The main difference between those algorithms is how they fill up the remaining space in a way that all features are well-distributed without any gaps, repetitions or artifacts.
- **Texture Optimization** is a technique proposed from [Kw05a] which aims to combine both previously mentioned approaches by synthesizing the texture per-pixel. Unlike pure pixel-based approaches it considers all of the pixels and interprets the mismatches between input-/output pixels as an energy function. This function can be minimized in order to achieve an output that looks similar to the input. The energy function and it’s solver determines the quality of the output.

The first two principles mainly differ in the size of the unit of interpretation. Whilst pixel-based algorithms interpret textures as plain pixel-sets without any sense of what the texture actually represents, patch-based algorithms try to analyze the texture first to find out logically connected areas (so called patches), thus increasing the complexity of such algorithms. The differences are explained in detail within the sections 3.1 and 3.2.

There are other principles for other applications, like solid or liquid Texture Synthesis ([Kw07], [Yu09]), and there are also approaches using existing principles in real-time applications ([Le08], [WL02]).

3 Previous Work

Many state-of-the-art Texture Synthesis algorithms rely on the ability to choose a synthesis unit (pixel or patch) from the exemplar that get's copied into the synthesis result during a synthesis step. A convenient method to do this are Marov Random Fields (MRF). Efros *et al.* [Ef99] were one of the first who utilized this mathematical tool for Texture Synthesis. Their algorithm is described in detail in section 3.1.

In patch-based approaches the synthesis process is typically split into two phases: An *analysis phase* that extracts characteristics of the sample image and the actual *synthesis phase* that combines those characteristics to create a new image. During the synthesis phase the algorithms are using local properties of the sample patches and global functions like uniqueness constraints to spread patches equally and prevent obvious artifacts like heavy repetition. As an example authors of *Kaspar et al.* [Ka15] are utilizing the analysis phase to create so called ”guidance channels”, which describe homogeneous areas of the sample and their relation towards their environment. In synthesis stage those channels are used along functions like a *global uniqueness constraint* to distribute patches evenly. Their method is described in detail in section 4.2.

3.1 Pixel-based Approaches

As mentioned before the algorithm described in [Ef99] was one of the first to use Markov Random Fields to describe the spatial neighborhood of pixels. The approach is simple and elegant and thus a good starting point for readers who are not familiar with other algorithms.

The basic idea of the algorithm is to initialize the synthesis result with a random field from the exemplar that gets copied to the output. All pixels neighboring the seed are synthesized in a greedy manner, selecting the next pixels to synthesize in a inside-out fashion. The neighborhood of the current pixel to synthesize is described by an Markov Random Field, which can be seen as a window around the pixel. The size of the MRF is the only user-defined parameter the algorithm takes and should resemble the size of the largest patch inside the exemplar to prevent sampling artifacts.

The neighborhood gets compared to the neighborhood of all pixels from the exemplar, resulting in a set of similar candidates for the synthesis. If multiple candidates are found, a random one will be chosen and copied to the output, resulting in a new neighborhood for the next pixel. The algorithm continues with a pixel that has a neighborhood with as many already synthesized pixels as possible. Figure 2 shows a selection of three pixels that are candidates for copying into the output.

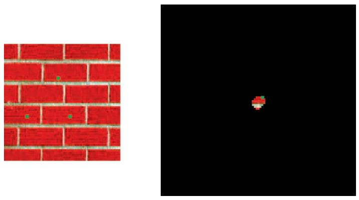
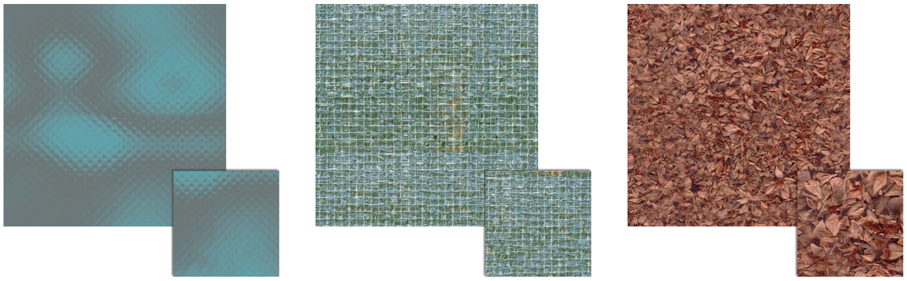


Fig. 2: Candidates for the synthesized pixel feature similar spatial neighborhoods. If multiple candidates exist in the exemplar, a random match is picked.



(a) Glass (b) Wired glass (c) Oregano

Fig. 3: Synthesis results based on the implementation of [Ef99]. The larger images are the input samples. The results were kept at low resolution for performance reasons.

Sample image	Input resolution	Output resolution	MRF size	approx. Time
Glass	512x512	256x256	20	8 days
Wired glass	512x512	256x256	30	10 days
Oregano	1024x1024	256x256	30	18 days

Tab. 1: The performance of the algorithm depends heavily on the resolution of the input and output image, just as the size chosen for the MRF window. The table shows that the algorithm is not suited for today's typical texture resolutions, even if the quality is very high if the MRF size is chosen cautiously.

Besides it's simplicity, the algorithm has the disadvantage that it can be inefficiently slow, depending on the size of the output texture and the size of the MRF. Also the pixel-by-pixel approach prevents the algorithm from taking great advantage of parallelization, because the output of one synthesis step directly influences the neighborhood of pixels for available for the next synthesis steps. The overall performance of the algorithm depends on the following factors: sample size, MRF size and output size. Each algorithm step synthesizes a new pixel in the output image, so the number of steps equals the number of pixels inside the output image. During each step the whole sample image is scanned, so the number of processed pixels equals approximately the number of input pixels times the numbers of pixels withing the MRF (even if not all pixels are considered during the MRF analysis). The overall performance can be described by $\mathscr{O}(p_{in} * p_{out} * s_{MRF})$.

For visualization purposes we benchmarked a set of sample images from different texture classes with different resolutions. The results were achieved on an Intel® Core^{TM} i7-5820 @ 3.30GHz. Note that the process only was able to utilize one core. Figure 3 shows the results created from the test run. Table 1 shows the parameters used for the synthesis process and the time it took to create the results. The output resolution has been chosen smaller than the input resolution, because the time it takes to create an 512 square pixel output from an 256 square pixel input does not differ from a 512 square pixel input with a 256 square pixel output, as shown above. Whilst the synthesis of the images in figure 4 (with an sample size of 90 square pixels) only took a few hours to complete, the time consumption raises much higher when synthesizing textures from or for today's typical resolutions of 512 square pixels or above.

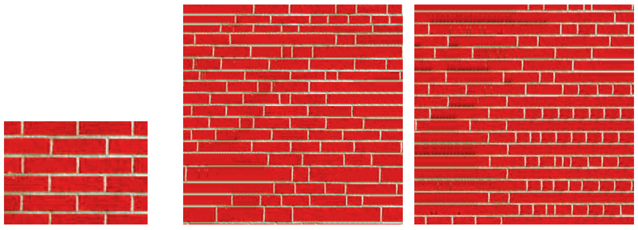

(a) Input sample (b) 10px MRF size (c) 30px MRF size

Fig. 4: An sample input is synthesized with two different MRF sizes: The lower the MRF size is, the greater the *randomness* gets in terms of patch distribution.

In terms of quality, the local neighborhood approach may result in increased noise or garbage regions, especially if the MRF size has been chosen to low. Figure 4 shows the impact of different MRF sizes. In both cases the window was not large enough to capture global neighborhood information (like the mortar stripes around a whole brick in this example). As a rule of thumb the MRF should be large enough to capture the largest structure inside the texture.

To address the performance issue *Wei et al.* [WL00] are adopting the previously described approach and replace the full MRF with a fixed neighborhood and the inside-out synthesis fashion with a scaline-based approach. The fixed neighborhood enables the algorithm to take advantage of tree-based algorithms, like *tree-based vector quantification (TSVQ)*, as the authors suggest. Later authors based their research on this method and combined it with other tree-structures, like *kd-trees* (*Kwatra et al.* [Kw05b]) or *k-coherence* (*Tong et al.* [To02]), which is the best in terms of quality and performance according to [We09].

3.2 Patch-based Approaches

Patch-based Texture Synthesis describes a technique that extends pixel-based approaches by using a different interpretation of synthesis units. Instead of single pixel, whole patches of different size get copied to the output. Naturally this cannot be done in an optimal manner, because it is not guaranteed that a patch that exactly fits a hole inside the output can be found in the exemplar. Another challenge of those approaches is to uniformly seed patches from the exemplar in the synthesis result to prevent unnatural looking repetitions or artifacts. Therefor different algorithms exist, that ultimately differ in the way how they combine patches together to create a naturally-looking result.

Praun et al. [PFH00] described an algorithm that simply overwrites new patches over existing ones, generating surprisingly good results for stochastic textures, but does not work conveniently for structured textures. Therefor *Lian et al.* [Li01] described an algorithm that blends both patches together, which can cause blurry regions, while *Efros et al.* [EF01] use dynamic programming techniques to find an optimal path through both patches and cut them there.

3.3 Texture Optimization

Texture Optimization is an approach, proposed by *Kwatra et al.* [Kw05b], that combines the advantages of both, pixel- and patch-based approaches. Their implementation performs the actual synthesis step in units of pixels, but unlike traditional pixel-based approaches, Texture Optimization considers all of the pixels and generates an energy function from the mismatches between their values in the exemplar and the output. The interesting thing is that this (quadratic) function directly appears to resemble the quality of the output: optimizing (e.g. minimizing) the function leads to better quality results.

4 Recent Developments

In the past years many scientists researched problems with current Texture Synthesis algorithms, mainly trying to improve their performance and the quality of their results. This section goes a little bit more into detail on the *PatchMatch* ([Ba09], [Ba10]) algorithm, which was a significant milestone for Texture Optimization algorithms. *Image Melding* ([Da12]) further improved it and *Kaspar et al.* [Ka15] utilized both methods to present a *Self-Tuning Texture Optimization algorithm*.

4.1 PatchMatch and Image Melding

Previous algorithms mostly derived the nearest-neighbor search problem from [Ef99]: the more pixels the input and output image has, the more time it takes to search the whole input for fitting synthesis candidates in each synthesis iteration. *Ashikhmin* [As01] showed that significant performance gains can be achieved by limiting the search space in the exemplar to a patch around the source neighbors of the neighboring pixels of the target, thus exploiting *local coherence*. *Barnes et al.* [Ba09] based their *PatchMatch*-algorithm on that coherence assumption. Also they prototyped a GPU-based implementation of their nearest-neighbor search algorithm, generating a even better speedup.

In terms of quality, *PatchMatch* utilizes user-provided *guidance channels* to solve the problem of poor synthesis results where the boundaries of the synthesized target deliver only weak or none constrains for natural looking completion results. However, *guidance channels* in general are user-provided thus only shifting the search for fitting patches towards user suggestions. This prevents a highly or fully automated process and may even increase the overall workload for the actual artist.

The introduction of *guidance channels* as a pre-calculated map of characteristics from the input sample also firstly split up the synthesis process into two distinct stages: the *analysis phase* and the actual *synthesis phase*, which is typical for all following Texture Optimization algorithms, which later aim on (partially) automating this task.

Darabi et al. [Da12] generalized the PatchMach algorithm to be able to apply patch-based approaches to the whole family of image melding problems, where Example-based Texture Synthesis represents a special case with one input source. They introduced three major changes towards patch-based algorithms:

- Extending the degrees of freedom of the search space to handle not only the basic transformations translation, rotation and scale, but also reflection and non-uniform scale.
- Using the image gradient as an additional property to represent a patch. The gradient is stored as a *guidance channel* during analysis phase.
- Changing the interpretation of *energy* in Texture Optimization thus increasing the sharpness of the result.

During synthesis phase, which can be seen as a generalized application of *hole-fitting* or *image completion* for PatchMatch and Image Melding implementations, the algorithm searches for fitting patches within the guidance channels from the analysis phase. In order to find fitting patches it performs a search and votes for the best choice. It does so by solving a screened Poisson equation (*Poisson fusion*), instead of using a Fourier-based solver. This proved to give good quality results while still reducing the complexity.

4.2 Self-Tuning Texture Optimization

Kaspar et al. [Ka15] base their implementation on *Image Melding* by removing some of its features which are not common for two-dimensional textures, which are typically rectified and do not exhibit perspective distortions. Therefor they disabled searching among rotation and scale, improving the performance of the search process. Also they do not use some analysis filters like gain, bias adjustments or gradient channels. To increase texture sharpness, they are not using the *Poisson fusion* described in 4.1. Instead they quantize the nearest-neighbor field to integer locations to improve performance of the search process.

Besides those changes to *Image Melding* the authors also introduce several improvements over existing algorithms which they derived from a survey of common problems. Those problems arise when applying an algorithm to textures of different texture classes:

- **Structured and large featured** textures often suffer from missing non-local information which is explicitly derived from large structures. Repetitions are a typical symptom of this problem. The algorithm solves the problem by using *guidance channels* and introducing a technology to calculate them automatically.
- **Regular and stochastic** structures may not be evenly spread due to random initialization of the synthesis result. The authors introduce a new smart initialization technique that is build from random blocks, rather than pixels, which improves output quality for the mentioned classes, whilst not affecting the quality for textures without any regularity.

Additionally a *spatial uniformity constraint* ensures that the output image looks globally similar to the exemplar by keeping track and constraining the number of occurrences of certain pixels, while improving performance towards other approaches, like bidirectional similarity ([Si08]).

5 Further Research

In this paper we've presented previous and current developments in the field of non-parametric Example-based Texture Synthesis. Since computation performance, especially graphics memory on current GPU's have significantly been increased, higher resolution textures can be used to create photo-realistic virtual environments. The demand for such textures increases with the supply of computation performance, increasing both, the demand for Texture Synthesis algorithms in general to automate common manual texture editing tasks and the stress for existing algorithms. Current algorithms are delivering good results for many, but not all textures. The lack of an general-purpose solution prevents Texture Synthesis from being broadly used within the graphics industry. Also the poor performance of many of those algorithms does not allow viable interactive editing.

Researchers significantly improved existing algorithm approaches in the last years by carefully evaluating their downsides and trying to apply their benefits to other algorithms. For example pure pixel-based algorithms have been replaced by Texture Optimization, which combines the advantages of patch- and pixel-based approaches. By critically evaluating current state-of-the-art implementations, chances for a fast and quality algorithm to be developed within the next years are high.

6 Acknowledgments

This paper has been written as a student research project. The author wants to thank Tom Uhlmann, staff member of the faculty *Graphical Data Processing and Visualization* at the Chemnitz University of Technology, for the support.

References

[As01] Ashikhmin, Michael: Synthesizing natural textures. In: Proceedings of the 2001 symposium on Interactive 3D graphics. ACM, pp. 217–226, 2001.

[Ba09] Barnes, Connelly; Shechtman, Eli; Finkelstein, Adam; Goldman, Dan: PatchMatch: A randomized correspondence algorithm for structural image editing. ACM Transactions on Graphics-TOG, 28(3):24, 2009.

[Ba10] Barnes, Connelly; Shechtman, Eli; Goldman, Dan B; Finkelstein, Adam: The Generalized PatchMatch Correspondence Algorithm. In: European Conference on Computer Vision. September 2010.

[Da12] Darabi, Soheil; Shechtman, Eli; Barnes, Connelly; Goldman, Dan B; Sen, Pradeep: Image melding: combining inconsistent images using patch-based synthesis. ACM Trans. Graph., 31(4):82, 2012.

[Ef99] Efros, Alexei; Leung, Thomas K et al.: Texture synthesis by non-parametric sampling. In: Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on. volume 2. IEEE, pp. 1033–1038, 1999.

[EF01] Efros, Alexei A; Freeman, William T: Image quilting for texture synthesis and transfer. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. ACM, pp. 341–346, 2001.

[Ka15] Kaspar, Alexandre; Neubert, Boris; Lischinski, Dani; Pauly, Mark; Kopf, Johannes: Self Tuning Texture Optimization. Computer Graphics Forum, 2015.

[Kw05a] Kwatra, Vivek; Essa, Irfan; Bobick, Aaron; Kwatra, Nipun: Texture Optimization for Example-based Synthesis. Technical report, Georgia Institute of Technology, 2005.

[Kw05b] Kwatra, Vivek; Essa, Irfan; Bobick, Aaron; Kwatra, Nipun: Texture optimization for example-based synthesis. In: ACM Transactions on Graphics (TOG). volume 24. ACM, pp. 795–802, 2005.

[Kw07] Kwatra, Vivek; Adalsteinsson, David; Kim, Theodore; Kwatra, Nipun; Carlson, Mark; Lin, Ming: Texturing Fluids. Technical report, Department of Computer Science, University of North Carolina at Chapel Hill, 2007.

[Le08] Lefebvre, Sylvain: Filtered Tilemaps (in Shader X6). Shader X6: Advanced Rendering Techniques, pp. 63–72, 2008.

[Li01] Liang, Lin; Liu, Ce; Xu, Ying et al.: Real-time texture synthesis using patch based samling. Microsoft Research, 2001.

[NV15] nVidia CUDA C Programming Guide.

[PFH00] Praun, Emil; Finkelstein, Adam; Hoppe, Hugues: Lapped textures. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co., pp. 465–470, 2000.

[Si08] Simakov, Denis; Caspi, Yaron; Shechtman, Eli; Irani, Michal: Summarizing visual data using bidirectional similarity. In: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on. IEEE, pp. 1–8, 2008.

[To02] Tong, Xin; Zhang, Jingdan; Liu, Ligang; Wang, Xi; Guo, Baining; Shum, Heung-Yeung: Synthesis of bidirectional texture functions on arbitrary surfaces. ACM Transactions on Graphics (TOG), 21(3):665–672, 2002.

[We09] Wei, Li-Yi; Lefebvre, Sylvain; Kwatra, Vivek; Turk, Greg: State of the Art in Example-based Texture Synthesis. In: EUROGRAPHICS 2009. 2009.

[WL00] Wei, Li-Yi; Levoy, Marc: Fast texture synthesis using tree-structured vector quantization. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co., pp. 479–488, 2000.

[WL02] Wei, Li-Yi; Levoy, Marc: Order-Independent Texture Synthesis. CoRR, abs/1406.7338, 2002.

[Yu09] Yu, Qizhi; Neyret, Fabrice; Bruneton, Eric; Holzschuch, Nicolas: Scalable real-time animation of rivers. In: Computer Graphics Forum. volume 28. Wiley Online Library, pp. 239–248, 2009.