# Automatic Quality Measurement and Parameter Selection for Example-based Texture Synthesis

L. F. Laursen & L. H. Clemmensen & J. A. Bærentzen &
T. Igarashi & B. K. Ersbøll

Kongens Lyngby 2012

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

**Abstract**

Texture synthesis algorithms have been researched extensively in the past decade. However, most synthesis algorithms are governed by a set of parameters and produce different results depending on which parameter settings are chosen in conjunction with an exemplar used as a basis for synthesis. So far, automatically selecting parameters suitable for synthesis has been a relatively unexplored topic. In effect, this makes texture synthesis supervised rather than fully automatic.

In this technical paper, we propose automatic parameter optimization methods for example based texture synthesis. We cover research to directly estimate specific texture synthesis parameters, such as patch size and iteration convergence, based on input textures. We also examine various similarity measures and evaluate their effectiveness. The goal for each measure is to properly evaluate how well the resulting synthesis compares to the original input. A good similarity measure will enable the search for the optimal texture synthesis parameters by maximizing the quality of the synthesis as a function of parameters.

We apply presented methods to a state of the art texture synthesis algorithm, namely the one proposed by Kopf et al [14]. It is easy to find a set of exemplars for which there is no single optimal set of settings. The results show a promising foundation for further research in establishing an automated optimal synthesis for a multitude of textures.

# 1 Introduction

Textures are commonly used in computer graphics to enhance the appearance of a scene. Despite the abundance of online texture repositories [1, 5, 3, 6, 4, 2], the acquisition of new textures still poses challenges. Example-based Texture Synthesis mitigates this issue by artificially creating new textures from a small input example.

Like most other texture synthesis algorithms, example-based texture synthesis requires manual parameter tweaking to obtain the optimal result in the shortest amount of time. The proper settings aid the algorithms in detecting and recreating the structure present in the examples provided.

We examine two general approaches to improving a state of the art texture syn-

thesis algorithm presented by Kwatra et al. [15] and further refined by Kopf et al. [14]. The texture synthesis algorithm works by minimizing an energy function describing the difference between the input texture(s) (exemplars) provided by the user, and the texture being synthesized. This is effectively done by finding matching texture patches (neighborhoods) and iteratively altering the synthesized patches to look more like the input exemplar(s).

In our first approach we examine methods with which to automatically estimate the optimal synthesis parameters by examining the input exemplar. In the second, we evaluate methods with which to provide a qualitative measure for the resulting synthesized texture. A reliable qualitative measurement would allow us to maximize the quality of the synthesis as a function of the input parameters. We propose a heuristic with which to automate the tweaking of the synthesis parameters based on the qualitative measurement.

The structure of this report is as follows: Related work to both presented approaches is discussed in section 2. The texture synthesis algorithm these approaches are applied to, as well as its associated parameters, is detailed in section 3 and 3.1, respectively. Work involving the direct estimation of parameters is presented in section 4, while section 5 covers the indirect estimation of parameters via a qualitative measure. Preliminary results of the direct method is presented in section 4, while more comprehensive results from the indirect method is detailed in section 6. Current limitations of both approaches is presented in 7. We conclude our findings in section 8 and discuss future work in section 9.

# 2 Related Work

We first present work related to the texture synthesis algorithm to which we apply automated parameter selection. We then detail work specifically related to the direct and indirect parameter optimization approaches.

Additionally, a number of papers exist that present altered and enhanced versions of existing texture synthesis methods, for the purposes of accelerating them. Although this is somewhat removed from the topic of automatic parameter selection, it shares a similar goal. The accceleration of the texture synthesis algorithm while attempting to maintain a quality result. We note a few publications presenting the aforementioned type of research.

## 2.1 Texture Synthesis

A large body of work within texture synthesis research [27] has led to the algorithm presented by Kopf et al. [14].

Texture synthesis algorithms have evolved over the past decade from being parametric [12] to non-parametric [9], pixel [29] and patch-based [16], to optimization-based methods [15, 14]. As previously noted, the publications have focused on either presenting a new and different approach, or evolving an existing method to produce better results. A recent publication [26] has even focused on reverse texture synthesis, which compacts an existing texture down to a smaller representation, from which a new texture is more easily synthesized.

We present results of automatic parameter selection conducted on the optimization-based approach described by Wexler et al. [30], applied by Kwatra et al. [15] and further refined by Kopf et al. [14].

## 2.2 Direct Parameter Optimization

To the best of our knowledge, no paper exists with the explicit goal of optimizing the given parameters of a texture synthesis algorithm, apart from the paper presenting the algorithm itself or iterative work upon the same. This is not especially surprising, given that the type of parameters eligible for optimization depend entirely on the synthesis algorithm itself. In this report, we focus on examining exemplar based texture optimization.

A parameter suitable for direct optimization is the size of the aforementioned neighborhoods used during synthesis. Section 4 provides a more detailed explanation to this effect. Briefly, the optimal neighborhood size is the smallest possible size, while still encompassing all unique structures captured in a texture. Hong et al. present a novel method with which to estimate texture scale [13] and apply it to set of highly periodic brodatz textures [8].

## 2.3 Indirect Parameter Optimization

Similar to direct parameter optimization, to the best of our knowledge, no paper exists that specifically investigates the impact of varying parameters used during texture synthesis, for the purposes of making the algorithm fully automatic.

However, within the broader spectrum of general computer science research, automatic parameter tuning based on an algorithms final result is common.

The automatic tuning of parameters with regards to a quality measure is sometimes called a metaheuristic, and is a subfield of stochastic optimization. A large body of work exists within this field dating back to the early 1950s. Luke and Talbi each provide a perspective over the current state of these types of algorithms as well as implementation based examples [19, 25]. Nanono et al. provide multiple concrete applications of parameter tuning in modern computer science problems [22].

## 2.4 Accelerating Texture Synthesis

Because texture synthesis is a computationally demanding task, it is only natural that research into improving performance or alleviating the calculatory burden exists. Lefebvre and Hoppe [18] extend Wei and Levoy's 2D synthesis approach [28] by parallelizing it and implementing it on modern GPU hardware.

Manke and Wünsche [20] extend Lefebvre and Hoppes approach allowing it to synthesize solid textures while executing on a modern GPU. Their paper provides a thorough explanation and a speculative GPU performance forecast, but their implementation is limited to a software prototype running in `C++`.

Dong et al. [10] present a novel method that synthesizes solid textures to cover the surface region of a given mesh. Their approach yields impressive results at high speeds, but due to their reliance on precomputed seamlessly interconnected neighborhoods, the algorithm can introduce a bias during synthesis, eliminating potentially significant features.

Recently, Barnes et al. have presented an algorithm that significantly increases the speed of finding the best approximate match for a patch in a given texture [7].

# 3 Texture Optimization

The synthesis algorithm we test our methods on was originally presented in Kopf et al.'s paper [14], and more thoroughly detailed in a related research paper [17]. In this paper we will restrict our explanation of the algorithm to

the portions where we deviate from the aforementioned descriptions, as well as portions related to the synthesis parameters which we tweak and analyze.

In short, the texture optimization algorithm attempts to minimize an energy function describing the difference between the input exemplar and the texture being synthesized. A simplified version of the function detailed by Kopf et al. is

$$E(N_s, N_e) = \sum_{i=1}^{n_s} \|N_{s,i} - N_{e,best}\|^r. \quad (1)$$

The input parameters ($N_s$) and ($N_e$) represent the texture being synthesized and the input exemplar respectively. To measure the energy difference between $N_s$ and $N_e$, small texture patches (usually 8 by 8 pixels) are extracted and compared. The total number of patches (a.k.a neighborhoods) from $N_s$ is denoted $n_s$. For each of the neighborhoods extracted from the synthesized texture $N_{s,i}$, its corresponding best match (measured via $L_2$ norm distance) is subtracted ($N_{e,best}$). The sum of differences describe the energy difference between the two textures. Setting the exponent $r = 0.8$ in the energy function keeps it more robust against outliers [14, 15].

The synthesis algorithm progresses through several levels of detail, starting with a coarse `32x32` resolution synthesis texture, comprised of random samples from the input exemplar. For each extracted synthesis neighborhood, the approximate best matching neighborhood is found. Finally, every pixel is updated based on those best matching neighborhoods. The process is comparable to an expectation maximization algorithm. The best matches are found, the whole texture is improved, and finally the process repeats itself.

The neighborhoods extracted from the synthesized texture lie on a sparse grid spaced 2 pixels apart as shown in Figure 1, where as neighborhoods extracted from an exemplar lie on a densely populated grid. In the dense grid, each pixel can be thought of as representing a single neighborhood.

Since the exemplar used to synthesize a new texture usually contains color, each vectorized neighborhood is comprised of 192 values, consisting of three color channels for each of the `8x8` pixels in the neighborhood. Finding the best matching exemplar neighborhood for each synthesized neighborhood is a computationally expensive task in such a high dimensional room, so prior to finding matches, the dimensionality of the neighborhoods is reduced using principal component analysis (PCA) to a state where 95% of their variance is still retained ($\sigma = 0.95$). To further increase the speed of searching for each neighborhoods nearest neighbor, an approximate nearest neighbor algorithm is utilized from the ANN Library [21]. It requires a distance parameter $\varepsilon$ to be set, which is

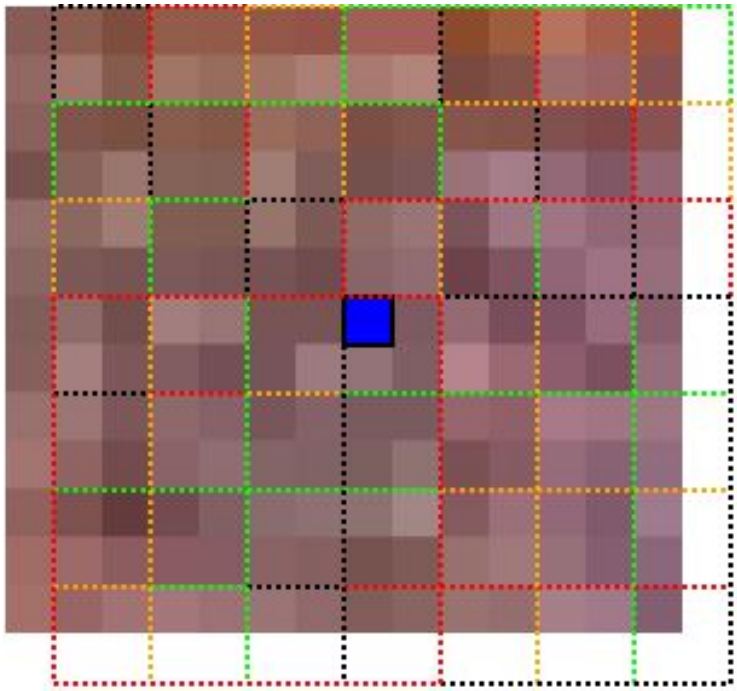

Figure 1: Density of extracted neighborhoods in the synthesis texture. Visualized are all the `8x8` neighborhoods which the blue highlighted pixel is a member of.

usually set to 2 [14], guarenteeing that all found neighborhoods lie no further than $1 + \varepsilon$, times the optimal match distance, away.

Once the approximate best neighborhoods have been located, the algorithm employs a clustering approach proposed by Wexler et al. [30] to reduce the number of contributing neighborhoods to a single pixel. This speeds up convergence by removing outliers and constrains the neighborhoods, that each pixel is a part of, to a group that contribute a similar color.

Additionally, the texture optimization approach employs a histogram weighing scheme that prunes contributions which overshoot the current amount of color, in the respective channel, for the pixel being updated. This ensures overall global correspondence to the input exemplar, while the neighborhood matching serves to increase local spatial correspondence.

In some cases it is beneficial, and occasionally a requirement, to attach a feature map (as an additional channel) in order to produce a satisfactory result, when

synthesizing a troublesome texture. The feature map basically assigns a weight to each pixel, indicating its importance with regards to the textures structure. This weight can be adjusted as necessary, if the algorithm is having trouble reproducing the pattern found within the texture.

## 3.1 Synthesis Parameters

Texture optimization provides multiple adjustable parameters. This section reiterates the ones we believe are most significant, along with a brief description. How each parameter is affected during direct and indirect optimization is explained in sections 4 and 5, respectively.

- **Neighborhood size** — The size of the texture patches compared in between the input exemplars and the texture being synthesized. Kopf et al. define the default Neighborhood size as 8 by 8 pixel.
- **Neighborhood grid density/sparsity** — The density/sparsity of neighborhoods extracted from the exemplar and synthesis textures. Kopf et al. extract neighborhoods from a dense grid on exemplars, and from a sparser grid (2 pixels apart) on the synthesized texture.
- **Neighborhood dimension reduction ($\sigma$)** — The method and severity with which the dimensionality, of the vectorized neighborhoods, is reduced. In the method proposed by Kopf et al., the variance is reduced to 95% by using PCA.
- **Convergence** — The method with which convergence of the synthesized texture is determined. Kopf et al. use a set number of iterations for each level of detail (Johannes Kopf, personal communication, March 10, 2010).
- **ANN Distance ($\varepsilon$)** — A parameter defining that the best matching exemplar neighborhood, found for a given synthesized neighborhood, is guaranteed to be no further than $1 + \varepsilon$ times the distance to the actual closest exemplar neighborhood.
- **Histogram matching weight adjustment** — When a new color is determined for a given synthesis texture pixel, each contributing color is compared with the already existing color contribution in the whole synthesis texture. If the amount of color in the synthesized texture is higher than that of the exemplar texture, then the contribution is punished as detailed by Kopf et al. [14]. This value is further amplified by a static weight parameter. Setting this parameter value to zero will nullify the effects of histogram matching completely.

- **Clustering algorithm** — The clustering algorithm intended to accelerate convergence and remove outlying contributors. A meanshift algorithm is employed by Kopf et al. with a number of threshold parameters.

- **Feature map channel** — The feature map represents an individual weight parameter for each pixel (or voxel) in the input exemplar(s). The static weight associated with the feature map can be adjusted as needed for the synthesis algorithm to converge successfully.

# 4 Direct Parameter Selection

Most of the parameters associated with the texture optimization algorithm presented by Kopf et al. [14] affect unique portions of the algorithm itself. We apply methods specifically aimed at automatically selecting the individual parameters presented below.

- **Neighborhood size** — Kopf et al. [14] note that the use of histogram matching allowed for the use of small neighborhoods (`8x8`), while still recreating the features of the original input exemplar. While this is true, our empirical testing revealed improved results with certain 2D textures using a larger neighborhood size, as visualized in figure 10.

  The optimal neighborhood size is undoubtedly dependent on the textures used as input. Recreating the features found in these textures, is a question of scale. A larger neighborhood is more suitable to properly recreate the features of a texture with a larger scale, where as a smaller neighborhood is suitable to a texture with a smaller scale, as visualized in figure 2.

- **Convergence** — As previously mentioned, Kopf et al. use a set number of iterations for each level of detail (Johannes Kopf, personal communication, March 10, 2010).

  Through empirical testing, we have determined that certain texture will converge much faster than others during synthesis. An approach measuring convergence would avoid wasting computational power by stopping the synthesis after an acceptable result has been achieved.

Optimization of the remaining parameters is beyond the scope of this report.

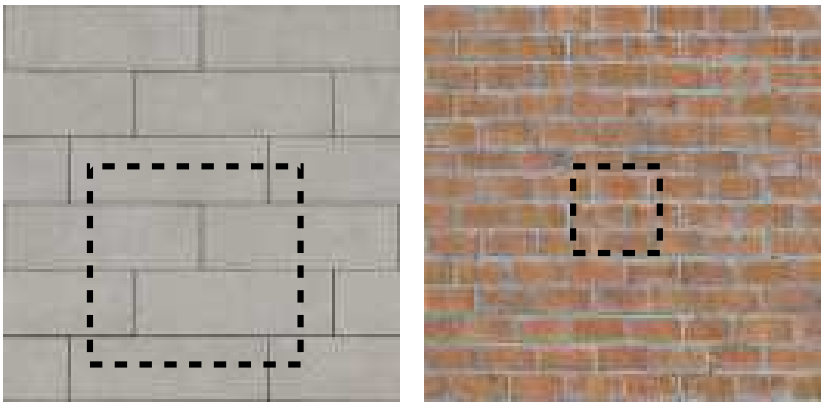

Figure 2: An approximation of the ideal neighborhood size given two differently scaled textures. On the left, the brick wall with the much bigger scale

## 4.1 Neighborhood size estimation

As previously mentioned in section 2, Hong et al. [13] present a novel method with which to estimate texture scale, using the following scale descriptor applied on each neighborhood $N_e$:

$$\inf_r D(N_{e,r}, N_{en,r}) - \alpha H(N_{e,r}) + \beta r(x). \tag{2}$$

The equation minimizes the energy measured by three terms, using the variables $N_{e,r}$, $N_{en,r}$, and $r$. The current neighborhood with a "radius" $r$ (height and width), is denoted $N_{e,r}$. It's surrounding neighbors is denoted $N_{en,r}$. The first term measures differences between the two regions either via Kullback-Leibler or Wasserstein distance. In this report we focus on the Wasserstein distance, as it yielded more reliable results during testing. The second term compensates for comparing homogenous patches, by reducing the energy proportionally to the amount of entropy measured in the neighborhood patches. Finally, the last term ensures that the equation favors patches with as small a size as possible. The weight parameters $\alpha$ and $\beta$ are set to 0.001 and 0.1, respectively, as suggested by Hong et al. [13].

We applied Hong et al.s method to one of the brodatz textures [8], as well as a number of textures used by Kopf et al. [14], visualized in figure 3. Table 1 display the measured mean and median for each of the textures.

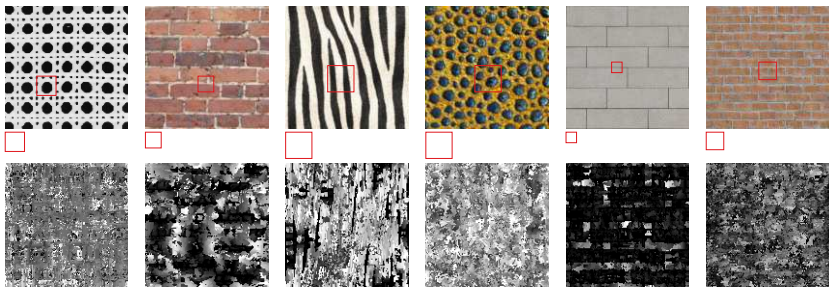


Figure 3: The top row shows the original textures, along with the median neighborhood estimated from every single pixel as a red box in the texture. The same neighborhood size is also visualized immediately below the texture. The bottom row is a scale map representation of each of the textures, obtained by applying Hong et al.s energy equation 2, where each pixel is given an intensity matching the estimated best neighborhood size surrounding that pixel. The more intense the pixel, the larger the estimated neighborhood for that location.

| Texture | Mean | Median |
|---|---|---|
| Brodatz (a) | 22.3423 | 21 |
| Brick wall (b) | 17.6359 | 15 |
| Zebra stripe (c) | 24.3031 | 27 |
| Animal Skin (d) | 26.5022 | 27 |
| Big Brick (e) | 10.7212 | 9 |
| Small Brick (f) | 17.8322 | 27 |

Table 1: The mean and median of the scales estimated for every pixel in figure 3.

The results do not always correlate directly with the size of the structural elements in the textures. The most striking example of this is the estimation of scale for the big bricks (e) and the smaller bricks (f). This is likely the cause of the homogeneous nature of the large gray areas contained within the brick texture which lead to a high similarity in the first term of equation 2.

Although the method delivers some results similar to our own estimation of optimal neighborhood size, there are also notable failure cases, such as the big bricks (e) and the smaller bricks (f).

We consider two alternative methods of estimating texture scale, based on an analysis of the relationship in between the neighborhoods:

### 4.1.1 Neighborhood Clustering

Texture Optimization [14] uses clustering as a means to speed up convergence, by discarding contributions which are not a part of the dominant cluster while updating a single pixel. As previously shown in figure 1, several neighborhoods contribute to a single pixel, and mean-shift clustering ensures that only contributions from the main cluster is retained.

This type of clustering approach could also be used as a scale descriptor. By expressing each neighborhood of a texture as a point in a high dimensional space, it may be possible to estimate scale based on this distribution. For each neighborhood size, meanshift clustering is applied to determine the dominant cluster. The optimal neighborhood size would have the biggest cluster.

There are issues that require further attention while implementing this approach:

- **Neighborhood Size** — As the size of the neighborhoods increase in an attempt to find the biggest cluster, so does the number of dimensions that each neighborhood "point" lies on. Principal component analysis is a useful tool in both reducing calculatory complexity as well as limiting the number of dimensions for each "point". However, it would still be necessary to include a weight parameter for the purposes of counter balancing this increased difficulty in clustering due to the higher number of dimensions.
- **Meanshift Threshold** — The meanshift clustering algorithms works with thresholding values that would need to be adjusted empirically to determine an optimal setting for the majority of textures. Since the point is to automatically estimate parameters, and not replace these with other parameters, the thresholds should either be established automatically or perform well for all textures of a given type.
- **Mahalanobis distance** — The distance between neighborhoods in the high dimensional space could potentially be better estimated using the mahalanobis distance, instead of the euclidean distance.

**Neighborhood Tree Structure** An alternate approach of examining the relationship in between neighborhoods is by building a tree structure representing nearest neighbors. Starting with a random neighborhood from the input exemplar, we form a new cluster consisting of that one member. Each cluster is represented by a single neighborhood, i.e. an average of all the existing neighborhoods in that cluster. We then continuously add the remaining neighborhoods

to the tree structure. If the new neighborhood is within a certain threshold distance of the representative neighborhood of a cluster, it is added to the same cluster. Otherwise, it will form its own new unique cluster.

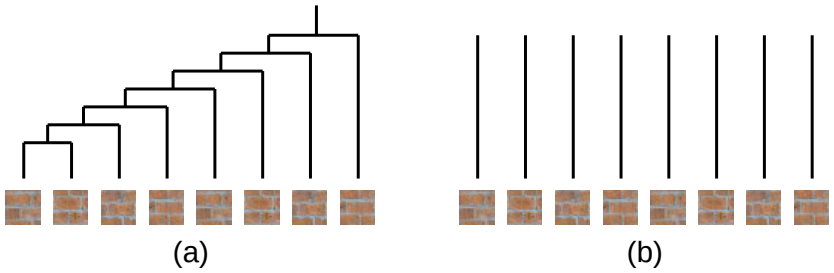


Figure 4: (a) The ideal tree structure where each new neighborhood is added to the already existing cluster. (b) The worst case scenario where each neighborhood forms its own cluster.

Figure 4 shows both the ideal, and worst case scenario of the tree structure. As with the clustering method, there are a few issues that require further attention:

- **Cluster Representative Neighborhood** — The best method of representing a cluster should be further investigated. One option would be to pick the neighborhood closest to all other neighborhoods in the entire cluster. An alternate option would be to simply average all the neighborhoods in the cluster together and use that as a representative. Although the latter option would likely cause unwanted blurring and should be carefully considered.
- **Distance Threshold** — The optimal threshold for determining if a neighborhood is close enough to a cluster to become a member should be empirically tested.
- **Mahalanobis distance** — Just like with the neighborhood clustering algorithm, the mahalanobis distance could also be applied here when introducing new neighborhoods to existing clusters.

## 4.2 Convergence estimation

As previously noted, Kopf et al. rely on a fixed set of iterations for each level of detail, during texture synthesis. We experimented with both the $L_1$- and $L_2$-Norm applied in conjunction with the energy function (in equation 1), as a useful measurement for synthesis convergence.

We were unable to achieve acceptable results using a fixed and/or dynamic threshold. A solution might be using a probabilistic metaheuristic to provide a good approximation to the global optimum, such as simulated annealing [31].

# 5 Indirect Parameter Selection

The core of our indirect parameter optimization method is straightforward. We attempt to establish an objective measurement of texture quality. Assuming that this objective measurements correctly identifies the best synthesized texture among several candidates, then determining the optimal parameters for a specific texture can be solved using a pure brute force method. If we also assume that the parameters are at least moderately orthogonal, we can approach the problem in a linear fashion. By only varying a single parameter, we can determine its optimal setting. Applying our objective measurements on each synthesized result while varying one parameter, we determine which setting produces the best result, for that parameter. We continue until we've determined the optimal setting for each parameter.

We break down the indirect method into five separate steps and detail in each how we mitigate the complexity arguably without reducing the quality of the results. For the benefit of the reader, we list the steps in an abbreviated form below:

1. Select parameters to optimize
2. Select similarity measures
3. Determine parameter bounds using similarity measures
4. Evaluate results of varying parameters within bounds using similarity measures
5. Perform automated texture synthesis

These steps are thoroughly detailed in identically ordered subsections below.

## 5.1 Tested Synthesis Parameters

Ideally we would like to measure the effects on all parameters involved in the texture optimization process. However, certain parameters are arguably harder

to optimize, and the complexity of determining optimal parameters can increase exponentially the more parameters are involved. Below we detail which parameters we vary, and which remain static.

1. Fixed

   - **Neighborhood Grid Density/Sparsity** — Although we firmly believe that certain textures could easily produce an acceptable result with a much sparser set of neighborhoods, we've chosen to constrain the complexity of our analysis, and keep the recommended neighborhood sparsity on the synthesized texture as recommended by Kopf et al.
   - **Convergence** — Similar to Kopf et al. we rely on a fixed set of iterations per detail scale to achieve a successful synthesis result. We found that the algorithm almost always converged when using 100, 30, and 10 iterations for the 32, 64, and 128 pixel resolution scale respectively.
   - **Feature map** — Originally introduced by Wu and Yu [32], feature maps aid patch-based texture synthesis methods in reproducing the structure found in the original input exemplar. All original feature maps require user input and are created artificially. We've chosen to focus on textures that do not require feature maps in order to produce acceptable results.
   - **Clustering Algorithm and associated Parameters** - Kopf et al. note in their paper [14] that purely averaging all the contributing colors for a single pixel might produce blurry results. While this sometimes occurred during our testing procedure, it is our impression that blurring, during 2D synthesis, only happened when there was no way for the algorithm to satisfactorily converge, in a particular region. Meanshift clustering on the other hand would force convergence and cause an unsightly seam to appear in its place.

     However, we did find that if histogram matching was not applied, simply averaging all contributors would occasionally cause the synthesis algorithm to yield unsatisfactory results.

     We also applied K-Means as an alternate clustering algorithm, and found that it produced results slightly less favorable when compared to Meanshift clustering, but had a much faster runtime. Almost comparable to simple averaging. Since the best textures were achieved using straight averaging of all contributors in conjunction with histogram matching, we utilize it exclusively during our testing procedure.

- **Histogram Weight Adjustment** — Originally intended as a non-static parameter, testing revealed that the best results were consistently achieved with a fixed value. Since there is no computational gain from varying the parameter (except for turning it off), we're keeping it static. Table 2 notes which settings were tested, as well as the permanent setting chosen.

2. Variable

- **Neighborhood Size** — Kopf et al. suggest using 8 by 8 pixel sized neighborhoods during the synthesis process. While determining the upper and lower bounds for this parameter we found this neighborhood size to often be the threshold for where a number of textures started converging properly.
- **Neighborhood Dimension Reduction ($\sigma$)** — Retaining 95% of the original textures variance is suggested by Kopf et al. and works well for all tested textures. A further reduction in retained variance sometimes shows little to no visual artifacts, where as other textures will immediatly cease to produce an acceptable result.
- **ANN Distance ($\varepsilon$)** — Kopf et al. suggests setting $\varepsilon = 2.0$ during synthesis. A higher setting leads to less exact/faster neighborhood matching, whereas a lower setting will generally be slower and produce more exact matches. A side-effect of a low setting (2.0) is that the algorithm occasionally synthesizes a near exact replica of the input exemplar, albeit with a vertical and horizontal offset. An example of this can be seen in Figure 5.

## 5.2 Texture Similarity Measurements

Determining the quality of a synthesized texture might seem trivial at first, since we are so used to comparing and evaluating what we see as humans. In fact, stating that people are living, breathing pattern recognition machines wouldn't be far from the truth. But in addition to a subjective evaluation, we are interested in obtaining quantifiable objective measurements. Below we detail which similarity measures we test to help us determine how successful a synthesized texture is in recapturing the original textures variety and likeness.

- **Subjective comparison** — While objective measurements aim to automatically quantify our synthesized results, they cannot capture the human impression given by the resulting texture. What we perceive remains a

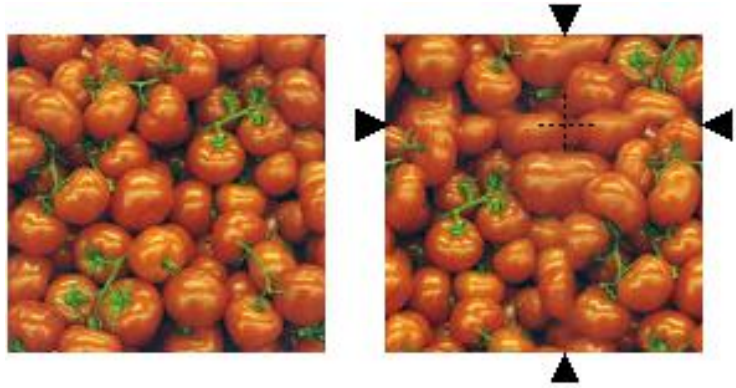

Figure 5: On the left, the original tomato exemplar used as input. On the right, is the resulting synthesized 2D texture using the parameters as suggested by Kopf et al. Notice that the original exemplar has been reproduced in its entirety with the original edges pointed out by the arrows.

cornerstone of graphics development and as such, the subjective impression cannot be disregarded. We therefore perform a subjective evaluation of the results that the synthesis process yields, in addition to the objective measurements.

- **Reverse neighborhood look-up comparison** — During the synthesis process, the approximate best matching exemplar neighborhood is found for each synthesized neighborhood. Treating each neighborhood as a 192 dimensional vector and calculating the distance using the $L_2$ Norm yields an objective measurement of how much the textures differ within that neighborhood region. It seems like an ideal method when applied to all neighborhoods in order to get a sense of how well the resulting synthesized texture turned out.

  Unfortunately, this is not the case. Matching each synthesized neighborhoods to its best matching exemplar, as done during texture synthesis, can yield a result indicating high similarity, even if the whole synthesized texture only resembles a small portion from the input exemplar. Since we want to punish textures for not making full use of the variance provided by the input exemplar, we instead match each exemplar neighborhood to it's best matching synthesized neighborhood on a dense grid and calculate the $L_2$ as follows

$$Diff(N_e, N_s) = \frac{\sum_{i=1}^{n_e}(N_{e,i} - N_{s,best})^2}{n_e}. \tag{3}$$

  The total number of neighborhoods derived from the exemplar texture

is denoted $n_e$ and a single exemplar neighborhood and its best matching synthesized neighborhood is denoted by $N_{e,i}$ and $N_{s,best}$ respectively. Note the differences between this equation (3), and the equation used during synthesis (1):

- For each extracted **exemplar** neighborhood, the best **synthesized** neighborhood is found.
- The actual best match is located during comparison (i.e. $\varepsilon = 0$).
- No dimensionality reduction is performed (i.e $\sigma = 1.0$).
- Neighborhoods are sized `10x10`, and are extracted from a dense grid on **both** the synthesized and exemplar texture.

Texture optimization actively minimizes the difference between synthesis and exemplar neighborhoods, so there exists a direct coupling between the process of obtaining a synthesized texture and this particular measurement of its objective quality. We perform the comparison using `10x10` sized neighborhoods, as opposed to the default `8x8` setting as suggested by Kopf et al.

Each channel is scaled to fit between 0..1, and the end result is divided by the number of neighborhoods extracted from the exemplar. The potential range for this objective measurement therefore spans between 0 and 192.

This approach is similar to the bidirectional similarity measure presented by Simakov et al. [24].

- **Crude Reverse neighborhood look-up comparison** — The *reverse neighborhood look-up comparison* test (*rnlc* test) is computationally expensive. It sacrifices no quality, and will give the best indication of whether the test itself produces viable results. This crude version of the same test sacrifices some quality in the hopes of achieving comparable results to the *rnlc* test, at much faster speeds. Compared to the *rnlc* test it retains 95% variance (as opposed to 100%), and locates approximate best matches ($\varepsilon = 2.0$), similar to the settings used during synthesis.

- **Global histogram difference** — To ensure a global texture correspondence, we calculate the histogram difference between the input exemplar and the synthesized result via the following equation:

$$HistDiff(T_e, T_s) = \sum_{c=1}^{j} \sum_{b=1}^{k} \| \frac{T_{e,b,c}}{v_e} - \frac{T_{s,b,c}}{v_s} \|. \quad (4)$$

The variable $j$ denotes the number of channels being synthesized, as determined by the input exemplar. All of our exemplars are RGB colored

and therefore $j = 3$. The total number of contributing pixels from the exemplar and synthesized texture is denoted by $v_e$ and $v_s$ respectively. The number of bins used to compare histograms is denoted $k$. This similarity measure, like the *reverse neighborhood look-up comparison*, is coupled to the synthesis process. Contributing pixels that do not conform to the exemplars histogram are pruned during synthesis. To minimize the coupling we set $k$ to a maximum of 255 bins, instead of 16 used during actual synthesis.

If the histograms from the exemplar and the synthesized result are identical, eqn. 4 will yield a zero sum. The most opposite textures (a white and a black one) would yield a maximum of $j * 2$ in difference.

## 5.3 Determining parameter bounds

As previously mentioned, we attempt to solve the problem of optimizing our parameters using linear brute force. The approach is simple, yet functional. However, it is far too computationally expensive to be viable. There are several parameters involved and some of them have potentially infinite settings. By only varying our parameters within fixed upper and lower bounds, we can optimize the approach arguably without losing any significant resulting quality.

When reducing our parameter search space, we use an extended set of textures, shown in Figure 7. Most textures are from the same set which Kopf et al. [14] used to produce solid textures with. The remaining few textures originate from the CGTexture repository [1].

We synthesize results for each permutation of parameter settings to visually determine the bounds for the parameters (detailed in Section 5.1). In other words, a single test consists of fixing every parameter to the defaults as defined by Kopf et al. and varying one parameter within a heuristically determined range. To avoid outliers we synthesize a minimum of four textures and visually compare these to the original input texture.

A number of parameters have a natural upper bound where neither quality nor precision is sacrificed, at the expense of computational complexity. For example, setting the ANN distance ($\varepsilon$) to 0 ensures that the best matching neighborhood is always found, or keeping 100% of the variance ($\sigma$) during the principle component analysis leads to no reduction in quality. We define this as the upper bound for these types of parameters. The lower bound is found by incrementally sacrificing more and more quality until the synthesis algorithm only produces indiscriminant noise for the majority of test textures. Figure 6

shows the synthesis results for three different textures retaining a continously lower amount of variance, while applying PCA. Note that while all textures degrade in quality, some degrade much faster than others depending on the complexity of structure found in the original texture.

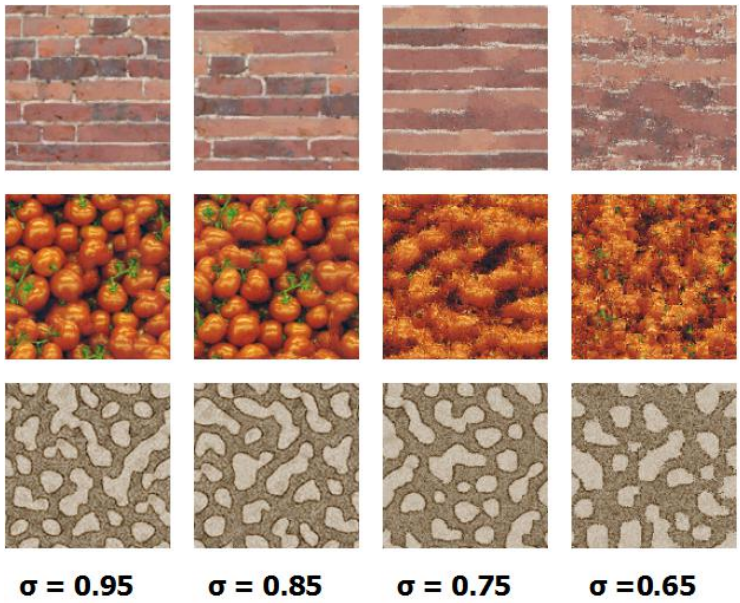


Figure 6: Synthesized results using three different exemplars. From left to right, each column shows the end result produced while retaining less variance ($\sigma$).

## 5.4 Evaluate objective measurement

Once the upper and lower and lower bounds have been determined, we continue our testing with a smaller set of textures (shown in Figure 7), due to computational complexity. For each of the four textures, we synthesize a set of 10 textures for each permutation of settings within the bounds determined by the previous tests. During these tests we apply our objective measurements to each of the synthesized results to determine how well they reflect the quality of the texture when compared to the visual impression.

Figure 7: The fourteen different input exemplars we use to determine the various bounds of the synthesis parameters. The blue highlighted exemplars on the left are also used to measure the efficiency of our objective measurements.

| Parameter | Preliminary Test Settings | Upper bound | Lower bound |
|---|---|---|---|
| NB Size | 2x2, 3x3, ... 15x15, 16x16, 20x20, 30x30 | 16x16 | 5x5 |
| Dimension Reduction ($\sigma$) | 0.95, 0.90, ... 0.70, 0.65 | 0.95 | 0.75 |
| ANN Distance ($\varepsilon$) | 1.5, 2, 3, 4, 8, 16, 28 | 1.5 | 12 |
| Hist Punishment | 0, 0.5, 1, 2, 4, 8, 16, 32, 64, 128, 1000 | 128.0 | 128.0 |

Table 2: The parameters and their settings used during preliminary testing to determine upper (best quality) and lower bounds (worst quality).

### 5.5 Automated texture synthesis

Finally, we use the objective measurements to perform a small set of automatic synthesis runs using the objective measurements as guide for our algorithm to determine the optimal setting for each parameter.

## 6 Indirect Parameter Selection Results

Having selected parameters and similarity measures as detailed in Section 5, we visually determined the upper and lower bounds as listed in Table 2, using the extended set of textures shown in Figure 7.

After determining these bounds, we proceed to test our similarity measures as explained in Section 5.4. The similarity measures are tested by completing a

total of 10 synthesized textures per settings permutation using a reduced set of textures, as shown in Figure 7. The results indicate that the *global histogram difference* measurement does not correlate well with our visual impression of the quality. It's not entirely surprising given that it measures global color correspondence and not structural similarity. A concrete example of lacking visual correlation is shown in Figure 8. Using small neighborhoods (`2x2` and `3x3`) during synthesis measure as being better than using larger neighborhoods (`4x4`, `5x5` and `6x6`), when using *global histogram difference* as a similarity measure. It's clear that its due to the green tomato stalks missing from these results. Using bigger neighborhoods, they reappear and the *global histogram difference* measurement confirms that these results are better.

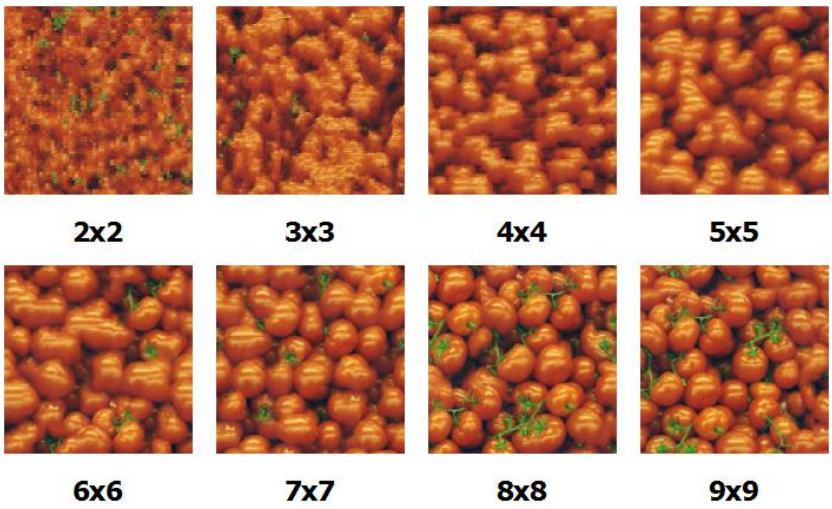


Figure 8: A synthesized result for different sizes of neighborhoods, using the tomato picture as input exemplar.

Consequently, the *global histogram difference* measurement cannot be used to singlehandedly determine if a result turns out well or not. The *Reverse neighborhood look-up comparison* on the other hand correlates well with our visual impression of the synthesized results. Figure 9 shows a general tendency of an improved result as the approximate nearest neighbor distance is reduced, or a higher variance is retained. It's worth noting that the brown dirt texture is the least affected by a reduction in variance, which the results correlate with. Figure 10 shows the results of applying the optimally detected neighborhood size along with the default $\varepsilon = 2.0$ and $\sigma = 0.95$ settings.

The *rnlc* test takes near a minute to complete with a mean completion time of 42.8 seconds and a std. deviation of 26.6 seconds. Fortunately, we found

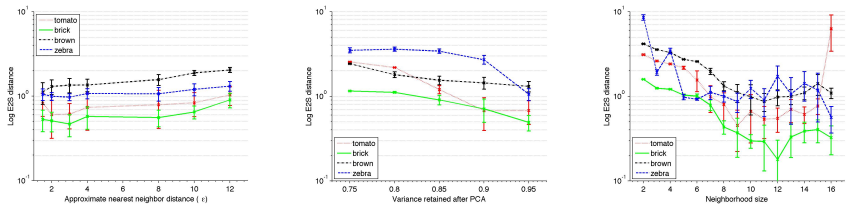


Figure 9: Three plots showing the results of the reverse neighborhood look-up comparison while varying the parameters specified in Section 5.1. Each plot shows the average result of 10 tests including std. deviation. The line colors refer to which texture being synthesized, as detailed in the legend.

that the crude *rnlc* test performs equally well when looking at each texture individually. Specifically, the objectively measured results closely resemble those of the regular *rnlc* test, except for being consistently better or worse. The mean completion time for the crude *rnlc* test is 4.6 seconds with a std. deviation of 1.9 seconds. Approximately 10 times faster.

Treating each parameter as orthogonal and performing a linear search through the parameter settings space occasionally causes the algorithm to select parameters which are not ideal, resulting in a sub optimal synthesized texture. To mitigate this issue, we've chosen an order by which the parameters are tuned, and once an optimal value for a parameter is found, we retain it when tweaking the remaining parameter. Figure 11 shows a comparison between a standard and optimized synthesis result using the four highlighted textures from Figure 7 as input.

The runtime of automatically tuning the parameters is highly dependent on the complexity of the texture used as an input exemplar. Using the four textures shown in Figure 11, the whole process took process, including synthesizing the final result, lasted between one and four hours, depending on the texture.

## 6.1 Summary

The extensive testing described in the previous sections have resulted in the development of the following heuristic, for automatically adjusting parameters of example-based texture synthesis algorithms.

1. Isolate parameters intended for optimization, and use the crude *Reverse neighborhood look-up comparison* test, as described in Section 5.2, on a

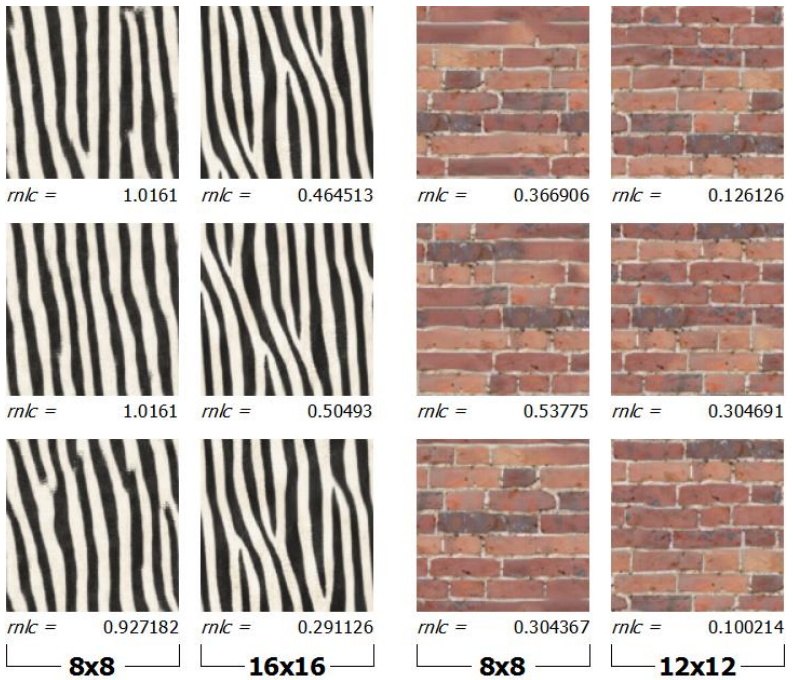


Figure 10: Synthesized results using the zebra (left) and brick (right) textures as input exemplars. The first/third column show three different results of using the parameters suggested by Kopf et al. (neighborhoods sized `8x8`) and the second/fourth column shows the results using the objectively measured optimal neighborhood sizes (`16x16` and `12x12`). Below each result is the qualitative measurement calculated via the *rnlc* test.

wide range of settings to determine the upper and lower bounds for each. In the case of texture optimization, the determined detailed in Table 2 work well.

2. Apply a metaheuristic, like linear search, and determine the optimal setting using at least 10 measurements via the crude *rnlc* test. Optimize parameters in the order of approximate nearest neighbor distance, neighborhood size, and PCA variance retention. Once an optimal parameter has been determined, retain it while optimizing the remaining parameters.

3. Synthesized texture using the determined optimal settings.

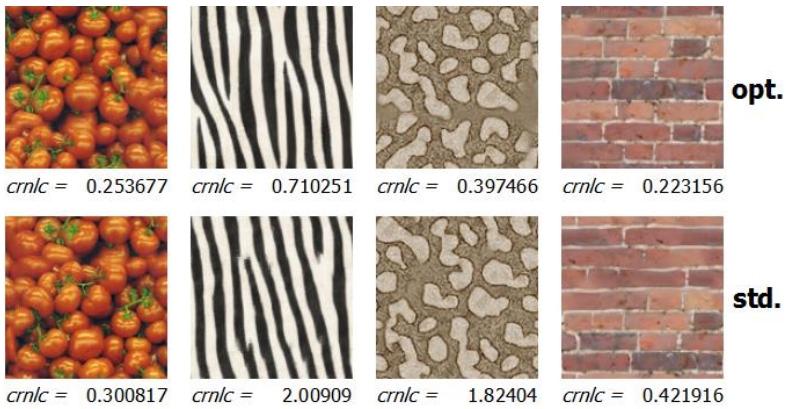


Figure 11: Comparison of the four sample textures synthesized using automatically detected optimal settings (top row) and standardized settings used by Kopf et al. (bottom row). The quality as measured by the crude *rnlc* test is listed below each result.

# 7 Limitations

Automatic parameter selection is no small task, and in some ways, this technical report only scratches the surface. The following section sums up the achievements of our efforts, while this section details short-comings we have identified with either method.

## 7.1 Direct Parameter Selection

Preliminary findings using direct parameter optimization indicate that the work by Hong et al. [13] produces suboptimal results in certain cases. Although the work shows promise, we question whether a probability density distribution is a sufficiently accurate representation of the different neighborhoods when determining scale. All spatial information is discarded in this representation which is limiting.

The alternate methods are in the preliminary stages, and therefore not viable for a proper evaluation of limitations. However, it is clear that both alternatives have the potential to introduce further parameters into the process, which would undo the purpose of this research. It is important that these methods either function well with a set of static parameters, or handle are capable of self-estimating them.

## 7.2 Indirect Parameter Selection

- **Objective Quality Measurement** — The crude *Reverse neighborhood look-up comparison* generally delivers favorable results. There are however some failure cases, such as the brown textured result in figure 11 having a much lower score than the standardized version, despite being more blurry. Additionally, the measured quality of a texture is not monotone and contains local minima.

  Using our determined upper and lower bounds lessens the risk of the algorithm falling into a local minima, but does not eliminate it entirely. It is also important to note that multiple measurements are required for the crude *Reverse neighborhood look-up comparison* test to deliver a consistent result. Looking at the results in Figure 9, its clear that the measurements can deviate considerably, and to compensate we would recommend measuring the result more than ten times. Since the runtime of the algorithm is already 'high' and further tests would only exacerbate the situation, it would be advantageous to apply a more advanced metaheuristic, such as simulated annealing.

  The high number of measurements required for a reliable estimation, causes the algorithm to run for several hours. Implementing Barnes et al.s algorithm [7] should cause a considerable speed up overall.

  Finally, it would also be advantageous to extend the similarity measure to incorporate detection for visual elements we find unappealing, such as blurring or hard edges. In its current state, the *Reverse neighborhood look-up comparison* would highly favor a result that is strikingly similar to the original input exemplar (such as the result shown in Figure 5).

- **Parameter Order Bias** — Since we've selected a fixed order in which we optimize our chosen parameters, as detailed in Section 6.1, we introduce a potential bias into the system.

- **Global Histogram Difference Measurement** — This measurement has some short comings. One of the most critical ones being, a case where one neighborhood is just a single shader darker than the neighborhood it is currently being compared to. Currently, the estimate would rate the two neighborhoods as very different, although they would optically be quite similar.

  Reducing the number of bins would alleviate the problem. Another solution would be to apply a histogram less prone to overall shifts in intensity, such as earth movers distance [23].

- **Neighborhood Size** — Our objective measurement uses a specific sized neighborhood and consequently introduces an amount of bias. This bias could be minimized by expanding the objective quality measure to include

multiple sized neighborhoods, which may be feasible given the computational optimizations provided by patchmatch [7].

- **Quality Vs. Speed** — Even with automatic parameter selection, there is still an implicit trade off between quality and speed. Allowing the user a single parameter controlling this aspect of the algorithm would be an improvement on the approach.
- **Overall speed** — The proposed method is slow when compared to manual parameter tuning. Apart from using a more advanced metaheuristic to traverse the search space, it might be possible to tune the parameters while producing a smaller, and therefore faster, texture patch (instead of the `128x128` sized result we synthesize currently). The parameters tuned while producing a smaller texture patch could serve as an initial 'good guess'. Reducing the amount of information from the input exemplar could also be achieved by first synthesizing a monochrome version.

# 8 Conclusion

We've presented analyzed and presented both direct and indirect methods for the purposes of automatic texture synthesis parameter selection. The direct methods are advantageous as they, by definition, provide a more direct route to automatically estimating parameters without the need for relying on a similarity measure, which potentially introduces further bias.

The heuristics developed as part of our indirect approach, combined with an objective similarity measure is capable of successfully synthesizing a better result than those using a standard set of parameters. The approach is not strictly limited to texture optimization (as presented in this paper), but could be applied to any exemplar-based texture synthesis algorithm.

Neither of the methods are currently without flaws. We identify the most notable limitations and note upon potential solutions. We believe that methods shows promise as a viable method for fully automating texture synthesis, and warrant further research.

# 9 Future Work

All of the limitations detailed in section 7 are ideal for future improvement, especially further work on a more accurate similarity measure as it has a significant

impact on the viability of the indirect parameter selection method.

Increasing the speed of texture synthesis and the associated parameter selection is also an attractive area for future work. A significant speed up would be achieved by implementing Barnes et al.'s patchmatch algorithm [7], with a minor quality loss. Another path of optimization would be to parallelize the complete algorithm and implement it on a GPU. Similar to the work done by Lefebvre and Hoppe [18], who extended Wei and Levoy's 2D synthesis approach [28]. The most computational heavy components of texture optimization are ideally suited to be parallelized and implemented on a GPU. Traversing a high dimensional search space has already been shown to perform much faster on a GPU by Garcia et al. [11], and since texture optimization updates each pixel from a static set of candidate neighborhoods, it could easily be computed using a GPU. As already stated by Manke and Wünsche [20], implementing histogram matching is currently the biggest issue plaguing a direct implementation of texture optimization on the GPU. Kopf et al. have already stated that the histogram must be up to date during the synthesis process. Only updating the histogram in between iterations causes the method to over/undershoot the intended goal.

A possible solution might be to randomly group pixel updates and update the histogram periodically during texture synthesis, instead of after every pixel has been updated.

## Acknowledgments

We'd like to thank Johannes Kopf for the past correspondence regarding the texture optimization algorithm. We would also like to extend our thanks to Takeo Igarashi and the University of Tokyo for their collaboration during the creation of this paper. This research was supported in part by the Danish Meat Research Institute.

# Bibliography


[1] Cgtextures. http://www.cgtextures.com/, 2011.

[2] 2textured. http://www.2textured.com/, 2012.

[3] Archive textures. http://archivetextures.net/, 2012.

[4] Grunge textures. http://www.grungetextures.com/, 2012.

[5] Mayang's free textures version 15. http://mayang.com/textures/, 2012.

[6] Texture king. http://www.textureking.com/, 2012.

[7] C. Barnes, E. Shechtman, A. Finkelstein, and D.B. Goldman. Patchmatch: a randomized correspondence algorithm for structural image editing. In *ACM Transactions on Graphics (TOG)*, volume 28, page 24. ACM, 2009.

[8] P. Brodatz. *Textures: a photographic album for artists and designers*, volume 66. Dover New York, 1966.

[9] Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 361–368, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[10] Yue Dong, Sylvain Lefebvre, Xin Tong, and George Drettakis. Lazy solid texture synthesis. In *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 2008.

[11] Vincent Garcia, Eric Debreuve, Frank Nielsen, and Michel Barlaud. k-nearest neighbor search: fast GPU-based implementations and application to high-dimensional feature matching. In *IEEE International Conference on Image Processing (ICIP)*, Hong Kong, China, September 2010.

[12] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 229–238, New York, NY, USA, 1995. ACM.

[13] B.W. Hong, S. Soatto, K. Ni, and T. Chan. The scale of a texture and its application to segmentation. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[14] Johannes Kopf, Chi-Wing Fu, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski, and Tien-Tsin Wong. Solid texture synthesis from 2d exemplars. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3):2:1–2:9, 2007.

[15] Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. Texture optimization for example-based synthesis. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 795–802, New York, NY, USA, 2005. ACM.

[16] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph.*, 22:277–286, July 2003.

[17] Lasse Laursen, Bjarne Kjær Ersbøll, and Jakob Andreas Bærentzen. Anisotropic 3d texture synthesis with application to volume rendering. In *Proceedings of WSCG'2011 (19-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2011)*, 2011.

[18] Sylvain Lefebvre and Hugues Hoppe. Parallel controllable texture synthesis. *ACM TRANSACTIONS ON GRAPHICS*, pages 777–786, 2005.

[19] Sean Luke. *Essentials of Metaheuristics*. Lulu, 2009. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[20] Felix Manke and Burkhard Wunsche. Fast three-dimensional texture synthesis. *New Zealand Computer Science Research Student Conference*, 2008.

[21] David M. Mount and Sunil Arya. Ann: A library for approximate nearest neighbor searching. http://www.cs.umd.edu/ mount/ANN/, 2010.

[22] Ken Naono, Keita Teranishi, John Cavazos, and Reiji Suda, editors. *Software Automatic Tuning: From Concepts to State-of-the-Art Results*. Springer, 1st edition. edition, 9 2010.

[23] Y. Rubner, C. Tomasi, and L.J. Guibas. A metric for distributions with applications to image databases. In *Computer Vision, 1998. Sixth International Conference on*, pages 59–66. IEEE, 1998.

[24] D. Simakov, Y. Caspi, E. Shechtman, and M. Irani. Summarizing visual data using bidirectional similarity. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[25] El-Ghazali Talbi. *Metaheuristics: From Design to Implementation (Wiley Series on Parallel and Distributed Computing)*. Wiley, 6 2009.

[26] Li-Yi Wei, Jianwei Han, Kun Zhou, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Inverse texture synthesis. *ACM Trans. Graph.*, 27:52:1–52:9, August 2008.

[27] 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.

[28] Li-Yi Wei and Marc Levoy. Order-independent texture synthesis. Earlier version is Stanford University Computer Science TR-2002-01.

[29] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[30] Yonatan Wexler, Eli Shechtman, and Michal Irani. Space-time completion of video. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(3):463–476, 2007.

[31] Wikipedia. Simulated annealing — Wikipedia, the free encyclopedia, 2012. [Online; accessed 18-April-2012].

[32] Qing Wu and Yizhou Yu. Feature matching and deformation for texture synthesis. *ACM Trans. Graph.*, 23:364–367, August 2004.