

High quality solid texture synthesis using position and index histogram matching

Jiating Chen, Bin Wang

▶ **To cite this version:**

Jiating Chen, Bin Wang. High quality solid texture synthesis using position and index histogram matching. The Visual Computer, 2010, 26 (4), pp.253-262. 10.1007/s00371-009-0408-3 . inria-00517935

HAL Id: inria-00517935
https://inria.hal.science/inria-00517935v1

Submitted on 16 Sep 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization

High quality solid texture synthesis using position and index histogram matching

Jiating Chen · Bin Wang

Abstract The synthesis quality is one of the most important aspects in solid texture synthesis algorithms. In recent years several methods are proposed to generate high quality solid textures. However, these existing methods often suffer from the synthesis artifacts such as blurring, missing texture structures, introducing aberrant voxel colors, and so on. In this paper, we introduce a novel algorithm for synthesizing high quality solid textures from 2D exemplars. We first analyze the relevant factors for further improvements of the synthesis quality, and then adopt an optimization framework with the k-coherence search and the discrete solver for solid texture synthesis. The texture optimization approach is integrated with two new kinds of histogram matching methods, position and index histogram matching, which effectively cause the global statistics of the synthesized solid textures to match those of the exemplars. Experimental results show that our algorithm outperforms or at least is comparable to the previous solid texture synthesis algorithms in terms of the synthesis quality.

Keywords Solid texture · Texture synthesis · Position histogram matching · Index histogram matching

This paper is based on *Solid Texture Synthesis using Position Histogram Matching*, by Jiating Chen and Bin Wang, which appeared in the Proceedings of IEEE CAD/Graphics 2009.

J. Chen (✉) · B. Wang
School of Software, Tsinghua University, Beijing 100084, China
e-mail: chenjt04@gmail.com

B. Wang
e-mail: wangbins@tsinghua.edu.cn

J. Chen
Department of Computer Science and Technology,
Tsinghua University, Beijing 100084, China

J. Chen · B. Wang
Key Laboratory for Information System Security,
Ministry of Education, Beijing 100084, China

J. Chen · B. Wang
Tsinghua National Laboratory for Information Science and
Technology, Beijing 100084, China

1 Introduction

Texture mapping has been recognized as an important tool in modeling surface details for rendering photorealistic graphic scenes without explicit modeling for geometry or material properties. However, defining a distortion-free and discontinuity-free mapping is challenging and sometimes even impossible for some complex objects. In contrast, when applying a solid texture onto an object, it allows carving the object out a block of the texture volumetric data, avoiding the problems of distortion and discontinuity. Furthermore, once a solid texture is available, it can be used to texture arbitrary objects not only on object surface, but also throughout the entire object volume.

In the past, procedural techniques and image-based techniques have been developed to produce solid textures. Procedural approaches [3] are very efficient since they require little storage. Nonetheless, only a limited range of textures can be modeled, and it is often difficult to be understood and controlled. A wider variety of solid textures have been produced by recent work on solid texture synthesis from 2D exemplars [8, 13, 16], the goal of which is to generate a volume where arbitrary 2D slice looks visually similar to the exemplar. However, some of these techniques exhibit certain disadvantages such as expensive computation or complex algorithms. Most of all, they fail to synthesize high quality

results, such as blurring, missing texture structures, introducing aberrant voxel colors, and so on. Recently, several algorithms were proposed to generate high quality solid textures [2, 9]. However, the quality problems mentioned above have not been solved yet.

Our algorithm focuses on generating high quality solid textures from input samples. It is considered that high quality solid textures should be similar to the exemplars on arbitrary slices through the entire volume. We analyze the relevant factors for further improvements of the synthesis quality, and find that directly copying pixels from the exemplar to the solid voxels rather than blending, can avoid the problems such as blurry blending and introducing aberrant voxel colors. Furthermore, all these pixels being copied equiprobably from the exemplar can preserve most of the texture structures well. Once these two requirements are satisfied, high quality solid results can be obtained.

Based on an optimization framework with the k-coherence search [14] and the discrete solver [6], our approach aims at introducing the position and index histogram matching methods in the re-weighting scheme, which ensure that all the voxel colors are copied equiprobably from the exemplars, thus both neighborhood matching and histogram matching are achieved. Since we use the k-coherence search in the nearest neighborhood search phase, the computation time is greatly shortened. As expected, experimental results show that our algorithm outperforms or at least is comparable to the previous solid texture synthesis algorithms in terms of the synthesis quality.

2 Previous work

Recent years have witnessed significant progress in example-based texture synthesis algorithms. In general, they can be classified as local ones [4, 5, 10, 15] and global ones [11, 12]. Local texture synthesis algorithms synthesize an output texture by pixels [4, 15] or by patches [5, 10]. Global ones evolve the output texture as a whole, and refine the entire texture results progressively.

Some per-pixel algorithms achieve quality and speed improvements based on the k-coherence search [14], which provides an efficient trade-off in both speed and quality. The k-coherence algorithm is divided into two phases: analysis and synthesis. During analysis phase, the algorithm builds a similarity-set for each input texel. During synthesis phase, the algorithm copies pixel, both the colors and the source pixel position, from the input to the output. Since the nearest neighborhood search is limited to a candidate-set, which is built by taking the union of all pre-computed similarity-sets of the neighborhood texels for each output texel during synthesis, a shorter constant time complexity is reached.

One problem encountered frequently in the optimization-based texture synthesis [11] is excessive blurry blending, which is usually introduced by the least square solver, and may even lead to an undesirable local minima problem. Discrete solver is proposed by Han et al. [6] to address the blur issue. Instead of blending pixel colors, it selects color directly from the candidate-sets to update the result pixels, avoiding the blurry blending problem and retaining the input position information of each output texel, which is available for the k-coherence search.

Example-based 2D texture synthesis methods have been extended to synthesize solid texture. Wei [16] adapted 2D neighborhood matching synthesis schemes to 3D volumes, but most of the time got unfavorable results, which are rather blurring and fail to preserve most texture structures. Qin and Yang [13] generated solid textures by capturing the co-occurrences of grayscale levels in the neighborhoods of 2D images. Since the color channels are closely correlated, independent channel synthesis usually leads to visual artifacts, such as color smearing and bad texture structures. Jagnow et al. [8] proposed a solid texture synthesis method based on stereology techniques, in which large texture structures are effectively preserved by modeling for different particle shapes. However, it tends to omit a number of fine structures due to the segmentation, and the colors in the result are often slightly different from the exemplar. Most of all, it is specialized for a class of materials and does not support arbitrary examples. Kopf et al. [9] extended 2D texture optimization techniques to synthesize solid textures, by introducing a color histogram matching approach to further improve the synthesized results. Dong et al. [2] generated solid textures by limiting the synthesis domain to a subset of the voxels around the object surface and performing a parallel spatially deterministic synthesis algorithm on GPU. Though both of them produce impressive results in many cases, some obvious problems have not been alleviated. For example, most fine grain details are missing, and a few distinct texture structures cannot be well preserved. These quality problems are illustrated in Fig. 1.

3 Overview

Our algorithm aims at generating high quality solid textures from 2D exemplars. Inspired by texture optimization approach [9, 11], we cast our synthesis process as an optimization problem. The interesting task here is that why we introduce the position and index histogram matching methods, and how to synthesize high quality solid textures by using these two simple but effective methods.

In our approach, all the voxel colors are supposed to be copied directly from the exemplars. Hence, it is necessary to

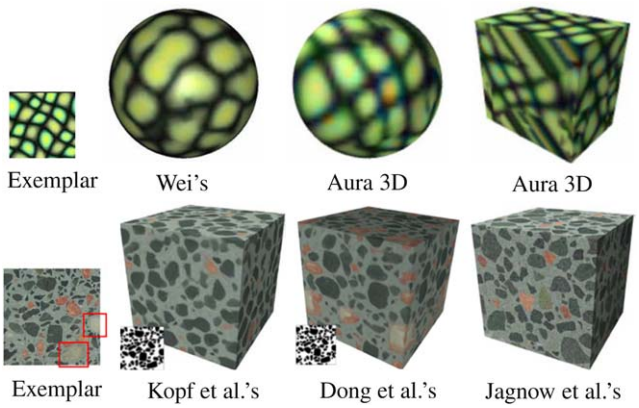


Fig. 1 Some quality problems for the synthesized results generated by previous methods. Wei's and Aura 3D results are rather blurring and fail to preserve texture structures. For Jagnow et al., Kopf et al. and Dong et al.'s results, though large texture structures are preserved, a number of fine grain details and even some obvious structures enclosed by the *red panes* are missing in the synthesized results

keep the source position information for each output voxel during synthesis process. In optimization phase, we adopt the discrete solver [6] instead of the least square solver. During the copy operation, we copy not only the pixel color value, but also the position of each copied pixel, ensuring that k-coherence acceleration is available in search phase. It is noted that color histogram matching is not always efficient in improving the synthesis performance, especially for texture structures with similar colors. By taking advantage of the source position information, we propose two new kinds of histogram matching methods, position and index histogram matching, which compute the histograms of the source position information for all the synthesis voxels and the nearest neighborhood indices for the neighborhoods of the voxels. Both texture structures and color histograms are effectively preserved synchronously by using the position and index histogram matching methods.

4 High quality solid texture synthesis

Similarly to the approach of Kopf et al. [9], the energy function that measures the differences between the solid texture and the exemplar is defined as

$$E(s,\{e\}) = \sum_{v} \sum_{i\in\{x,y,z\}} \|s_{v,i} - e_{v,i}\|^{\gamma} = \sum_{v} \sum_{i\in\{x,y,z\}} \sum_{u\in N_i(v)} w_{v,i,u}\|s_{v,i,u} - e_{v,i,u}\|^2 \quad (1)$$

Here e denotes the input exemplar, s denotes the synthesized solid, s_v refers to a single voxel, $s_{v,x}$, $s_{v,y}$ and $s_{v,z}$ are the neighborhoods of v in the slices orthogonal to the x, y and z axes, $N_i(v)$ denotes the neighborhood of the voxel v in the slice perpendicular to the ith axis, and the factor $\gamma = 0.8$. More specific issues are discussed in [9].

$s_v^0 \leftarrow$ random pixel from e, $\forall v \in s$
for resolution $l = 0$ to L **do**
  if $l > 0$ **then**
    $s_v^0 \leftarrow$ upsample from s_v at $l-1$
  end if
  for iteration $n = 0$ to N **do**
    /Index histogram matching and k-coherence search are used in search phase./
    for all $v \in s, i \in x, y, z$ **do**
      $e_{v,i}^{n+1} \leftarrow$ nearest neighborhood of $s_{v,i}^n$ in e
    end for
    /Position histogram matching and discrete solver are used in optimization phase./
    $s_v^{n+1} \leftarrow argmin_s E(s,\{e\})$
    if $n == N-1$ **then**
      $s_v \leftarrow s_v^n$
    end if
  end for
end for

Fig. 2 Our solid texture synthesis algorithm

Our algorithm is multi-resolution and computes the output solid textures from the lowest resolution to the highest resolution. It is achieved by solving (1). Before synthesis process, our algorithm requires a preprocessing for computing a k-coherence similarity-set for each input pixel. The synthesis process begins with the lowest resolution volume where the initial value of each voxel is randomly chosen from the exemplar. When synthesizing a higher resolution, we first initialize it by upsampling from the already synthesized lower resolution result, and then perform synthesis on this level in an iterative way, alternating between the optimization phase and the search phase, until the synthesized solid achieves good quality. Figure 2 provides the pseudocode for our solid texture synthesis algorithm.

4.1 Factors for quality improvements

As is well known, an important requirement for solid texture synthesis is that the Markov Random Field (MRF) assumptions should be satisfied. In other words, the solid texture is supposed to be similar to the exemplar on arbitrary slices through the volume. And in order to preserve the global statistics, several parametric approaches such as histogram matching [1, 7] are introduced. But they are still insufficient for synthesizing high quality results. One common problem for previous solid texture synthesis algorithms is blurry blending, which sometimes introduces aberrant voxel colors and leads to serious visual artifacts. As noted by Han et al. [6], directly copying pixels from the exemplar to the

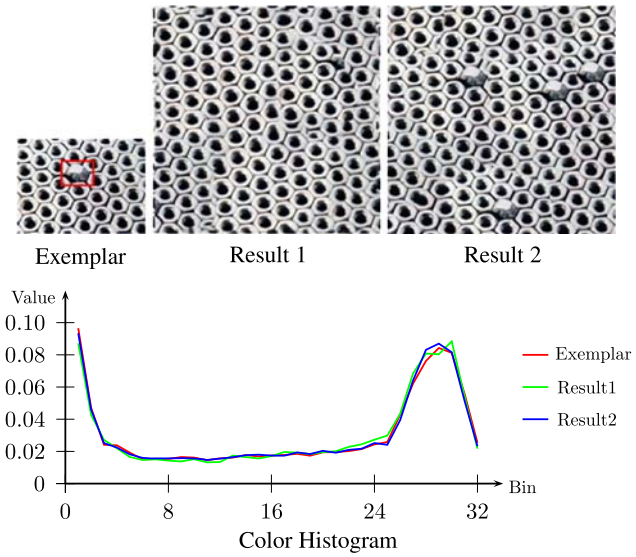


Fig. 3 Two different synthesized results and their color histograms compared with the exemplar. Result 1 is synthesized with color histogram matching, while Result 2 is synthesized with the position and index histogram matching. Since the color histograms of the results are in line with that of the exemplar, color histogram matching is insufficient to re-weight the $w_{u,i,v}$ in this case

voxels can eliminate this problem. Another problem is that some unique texture structures in the exemplar are missing in the result. For example, in Fig. 3, an obvious structure enclosed by the red pane is missing in Result 1 compared with the exemplar. Kopf et al. [9] tried to preserve them by color histogram matching, which is proved to be insufficient in this case. Therefore, we consider that all the pixels in the exemplar should have the same probability to appear in the result, which would adequately preserve most of the texture structures in the result. Once these factors mentioned above are satisfied, most of the texture structures, including large texture structures and fine grain texture details, can be well preserved, reaching further improvements in synthesis quality.

4.2 Optimization phase

Kopf et al. [9] adopted the least square solver to minimize the texture energy. The updated voxel is defined as

$$s_v = \frac{\sum_{i\in\{x,y,z\}} \sum_{u\in N_i(v)} w_{u,i,v} e_{u,i,v}}{\sum_{i\in\{x,y,z\}} \sum_{u\in N_i(v)} w_{u,i,v}} \tag{2}$$

Instead, we use the discrete solver [6]. For each updated voxel, the pixel in the set $\{s(v)\} = \{e_{u,i,v} | i \in \{x, y, z\}, u \in N_i(v)\}$ that mostly reduces the energy function is chosen for the updated voxel. In practice, we first calculate a prospective value s_v using (2), and then select a texel $e_{u,i,v}$ from the set $\{s(v)\}$ that is most similar to s_v for the updated voxel. Since each voxel is copied directly from the input exemplar, the blur issue can be avoided.

4.3 Search phase

We adopt the k-coherence search in search phase. A candidate-set is built for each output voxel by taking the union of all similarity-sets of the neighborhood pixels, and then the best match for each voxel neighborhood is obtained by searching through the candidate-sets. Initial experiments show that it is still time consuming for the neighborhood comparison in a full dimensional space with 8×8 neighborhood. A good trade-off between the quality and speed for the nearest neighborhood search is achieved by reducing the full dimensionality from 192 to about 20 dimensions with a Principal Component Analysis (PCA) algorithm.

4.4 Position and index histogram matching

Kopf et al. [9] preserved global statistics by using a re-weighting scheme with color histogram matching in the following way:

$$w'_{u,i,v} = \frac{w_{u,i,v}}{1 + \sum_{j=1}^{k} \max[0, H_{s,j}(b_j(e_{u,i,v})) - H_{e,j}(b_j(e_{u,i,v}))]} \tag{3}$$

$H_{s,j}$ and $H_{e,j}$ denote the jth histogram of the synthesized result and the exemplar. $H(b)$ denotes the value of bin b in a histogram, and $b_j(c)$ specifies the bin containing color c in the histogram $H_{s,j}$ and $H_{e,j}$. There are two conspicuous limitations existing in color histogram matching. Firstly, it works only for color but not for general structure information, as demonstrated in Fig. 3. Both color histograms of Results 1 and 2 keep in step with that of the exemplar, but some obvious texture structures are missing in Result 1. Secondly, it even fails to preserve color histograms sometimes. For example in Fig. 1, the texture structures of white color enclosed by red panes are missing in Kopf et al.'s result. One possible reason is that texture channels are improperly decorrelated. For instance, the weight of a texel $A(r, g, b)$ expected to be kept in the re-weighting scheme, is reduced actually according to (3) because one or two bins among $\{b_r(A), b_g(A), b_b(A)\}$ in the result histogram have a larger count than those in the exemplar histogram.

4.4.1 Position histogram matching

We first give a primary illustration on the notion of the position histogram. It is defined as a 2D grid, with the same size as the exemplar. Each of the grid unit records the frequency, appearing in the result volume, of the corresponding pixel in the exemplar, as illustrated in Fig. 4. In the position histogram, the frequency grows with the increase of brightness. In order to distinguish the place where the frequency is 0, we

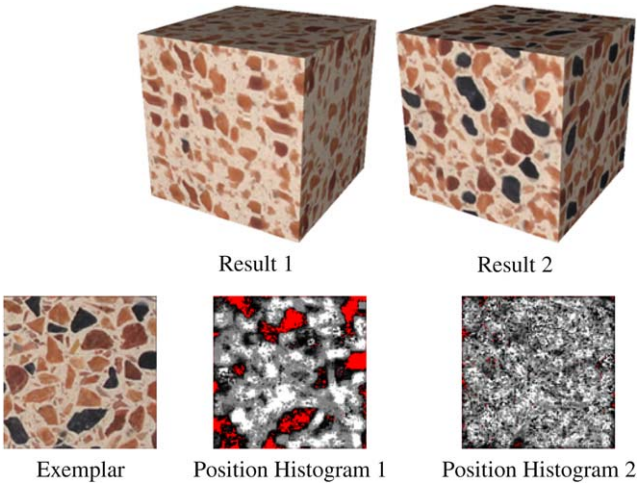


Fig. 4 Position histograms for two different solid textures. The histogram value is 0 in the *red parts*, and grows with the increase of brightness in the *gray parts*

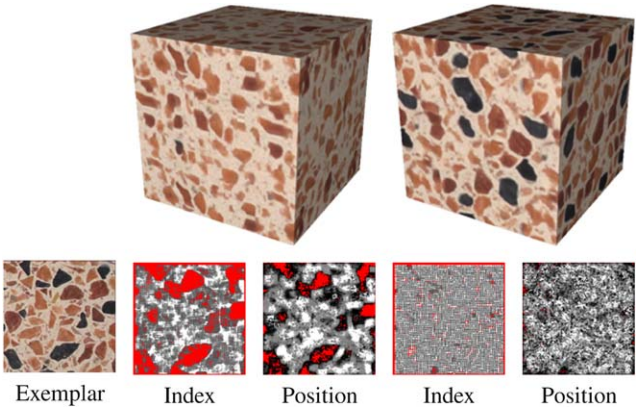


Fig. 5 Two kinds of histograms: index histogram (columns 2 and 4) and position histogram (columns 3 and 5). It can be observed that there is a significantly positive correlation between the index histogram and the position histogram

paint it red. In other words, in the red part the corresponding pixels in the exemplar do not appear in the result. In Fig. 4 the black texture structures corresponding to the red part of the position histogram are missing in the synthesized result for Result 1. For Result 2, the frequency of most part of the position histogram is approximately the same. Therefore, almost all the texture structures in the exemplar can be found in the result volume.

Our main purpose of introducing position histogram matching is to ensure that most of the pixels in the exemplar can be found in the result, and they are required to equiprobably appear in the result. Once position histogram matching is satisfied, color histogram matching is satisfied spontaneously.

Similarly to (3), we modify the weight as

$$w'_{u,i,v} = \frac{w_{u,i,v}}{1 + \max[0, H(p(e_{u,i,v})) - \theta]} \tag{4}$$

Here $p(e_{u,i,v})$ refers to the position of the pixel $e_{u,i,v}$, H is the position histogram, $H(p)$ is the value of position p in histogram H, and θ is the histogram value when all the pixels from the exemplar completely evenly appear in the result. Intuitively speaking, when $e_{u,i,v}$ has a large weight w, it is very likely to choose $e_{u,i,v}$ or a texel similar to $e_{u,i,v}$ for the updated voxel. Therefore, if the histogram has a smaller count than θ, the weight should be kept. In the contrary case that $H(p) > \theta$, we should reduce the weight in order to make it harder to choose pixel p for the updated voxel, or else it would converge to the same position, preventing the pixels equiprobably appearing in the result.

4.4.2 Index histogram matching

Here the index histogram records the frequency of the nearest neighborhood index corresponding to the neighborhood in the exemplar for all the neighborhoods in the result volume. Each neighborhood in the exemplar is represented by its center pixel. Hence it is easy for each of the grid unit to record the corresponding index frequency. This kind of histogram is also very useful. We notice that the local minima problem for the optimization-based approaches is often incurred when most nearest neighborhood indices are gathered to the same part in the exemplar. In Fig. 5, we also find that when most indices gather to some main parts, the structures corresponding to the red part in position histogram are missing in the result. Only if the indices distribute equiprobably in the exemplar, most texture structures can be well preserved. Therefore, we use the index histogram matching to restrict most of the nearest neighborhood indices to distribute equiprobably throughout the exemplar.

In practice, index histogram matching is used to restrict the nearest neighborhood search. During the search phase, we modify the distance between two neighborhoods as

$$d = w_d \cdot \|s_{v,i} - e_{v,i}\|^2 \tag{5}$$

And the weight w_d is defined as

$$w_d = 1 + \max[0, H(i(e_{v,i})) - \phi] \tag{6}$$

Here $i(e_{v,i})$ refers to the nearest neighborhood index corresponding to the pixel $e_{v,i}$, $H(i)$ is the histogram value of the index i, and ϕ is the histogram value when all the indices completely equiprobably distribute in the exemplar. When the histogram value $H(i(e_{v,i}))$ is larger than ϕ, we increase the distance d, making it harder to choose $e_{v,i}$ as the nearest neighborhood index for $s_{v,i}$. Otherwise, the index would converge to the same position in the exemplar, failing to preserve the texture structures in the result.

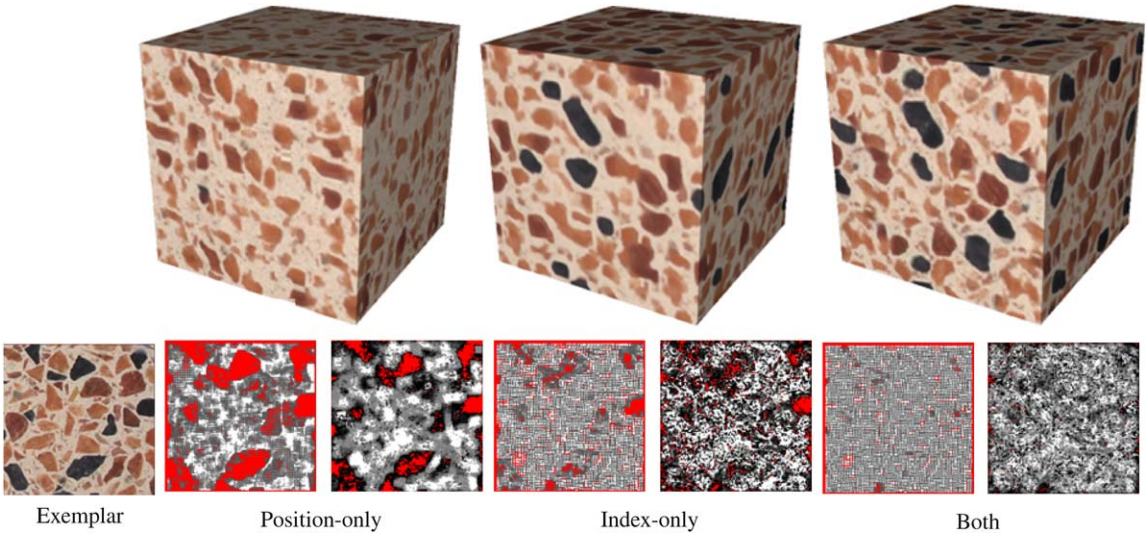


Fig. 6 Why we need these two histogram matching methods: without index histogram matching, the nearest neighborhood indices would gather to some parts in the exemplar easily; and without position histogram matching, some fine grain details may be missing due to the color average. High quality results could be generated when both of them are used jointly

4.4.3 Benefits of position and index histogram matching

Both position and index histogram matching methods depend on each other. We illustrate the necessity of both of them for our histogram matching in Fig. 6. Without the index histogram matching, the indices usually converge to some certain parts in the exemplar, and the pixels are also picked from those parts even using position histogram matching. And though the nearest neighborhood indices distribute equiprobably in the exemplar by using index histogram matching, some fine grain details are missing without using position histogram matching. Figure 6 demonstrates that the index histogram matching is more powerful than the position histogram matching, and high quality synthesized results are obtained when both of them are used jointly.

We notice that the index histogram matching is also applicable for most other texture synthesis methods in improving the synthesis quality, since it only needs to record the position of the nearest neighborhood index in the exemplar. Once it is used in the optimization-based methods, most nearest neighborhood indices distribute equiprobably in the exemplar. Hence the undesirable local minima problem can be effectively avoided. Our position and index histogram matching methods also allow synthesis controlling. For example, some texture structures are required to appear more in the result. In this case, θ or ϕ is not constant. $\theta(p)$ or $\phi(p)$ is different at different positions. The texture structures from the position p appear more in the synthesized result, with the increase of $\theta(p)$ or $\phi(p)$.

5 Result and discussion

We first implement our synthesis algorithm by using a three-level synthesis pyramid, with an 8×8 neighborhood at the lower two levels and a 6×6 neighborhood at the highest level, and $k = 5$ for k-coherence search. It takes only about 5 iterations to obtain good quality results at the higher two levels. Due to the fewer iterations and k-coherence search, synthesizing a 128^3 solid texture takes less than 10 minutes on a 2.2 GHz CPU. Therefore, the synthesis time is shorter than that of Kopf et al. [9], and it is independent of the size and richness of the exemplars.

In Fig. 7, we show some of our synthesized results. It could be seen that our technique works well for a wide range of textures varying from isotropic to anisotropic, from low-frequency to high-frequency, from fine-grain-detailed to strong-large-structured, and so on. And in order to show the consistence of the interior synthesized texture, a cutting part of the bunny in Fig. 8 reveals a consistent stone texture in the interior with our approach.

Figure 9[1] shows the comparison of our method with some previous methods. It could be observed that, in comparison with Kopf et al.'s [9], our results do not suffer from color blurring, as well as most fine grain details and sparse texture

[1] In the top two rows, the *tomatoes* and *caustic* results of Kopf et al. are from http://johanneskopf.de/publications/solid/textures/index.html and then rendered by our method under the same conditions, and it is the same to Dong et al.'s result, which is from http://www.dongallen.com/lazysyn/lazysyn.html. The rest of images or synthesized results generated by the previous techniques are extracted from the corresponding papers and/or websites.



Fig. 7 Results of our high quality solid texture synthesis. Various textures are applicable, including isotropic and anisotropic, low-frequency and high-frequency, and textures with large structures and with fine grain details

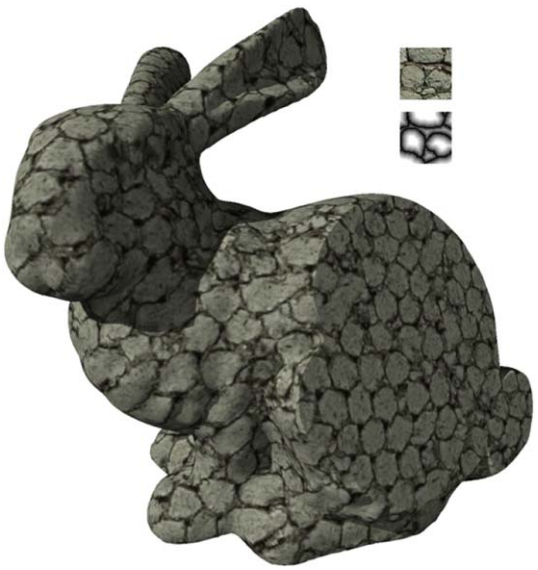


Fig. 8 An additional synthesized result. A part of the bunny is cut off to show the consistence of the interior synthesized texture

structures are well kept. More importantly, our method preserves the global statistics and global large structures better than those by Kopf et al. [9] and by Dong et al. [2]. To be highlighted, large texture structures can be efficiently preserved even without the aid of the feature maps by our method, which is demonstrated in the last row. It should be noted that while all the pixels from the exemplar are restricted to have the same probability to appear in the result, little extra effort is made to preserve texture structures. Therefore, we can conclude that our method does produce high quality solid textures for a wider range of texture exemplars better than previous methods.

During the synthesis process, it can be observed that most texture structures are coarsely synthesized at the lowest resolution, and progressively refined at the higher two levels by removing fine scale errors from the texture. Hence, the neighborhood sizes at different levels can be different. At the lowest level, the size is proportional to the feature size of the texture patterns. And at the higher two levels, we focus on refining the texture details progressively without destroying the large texture structures. Smaller neighborhood

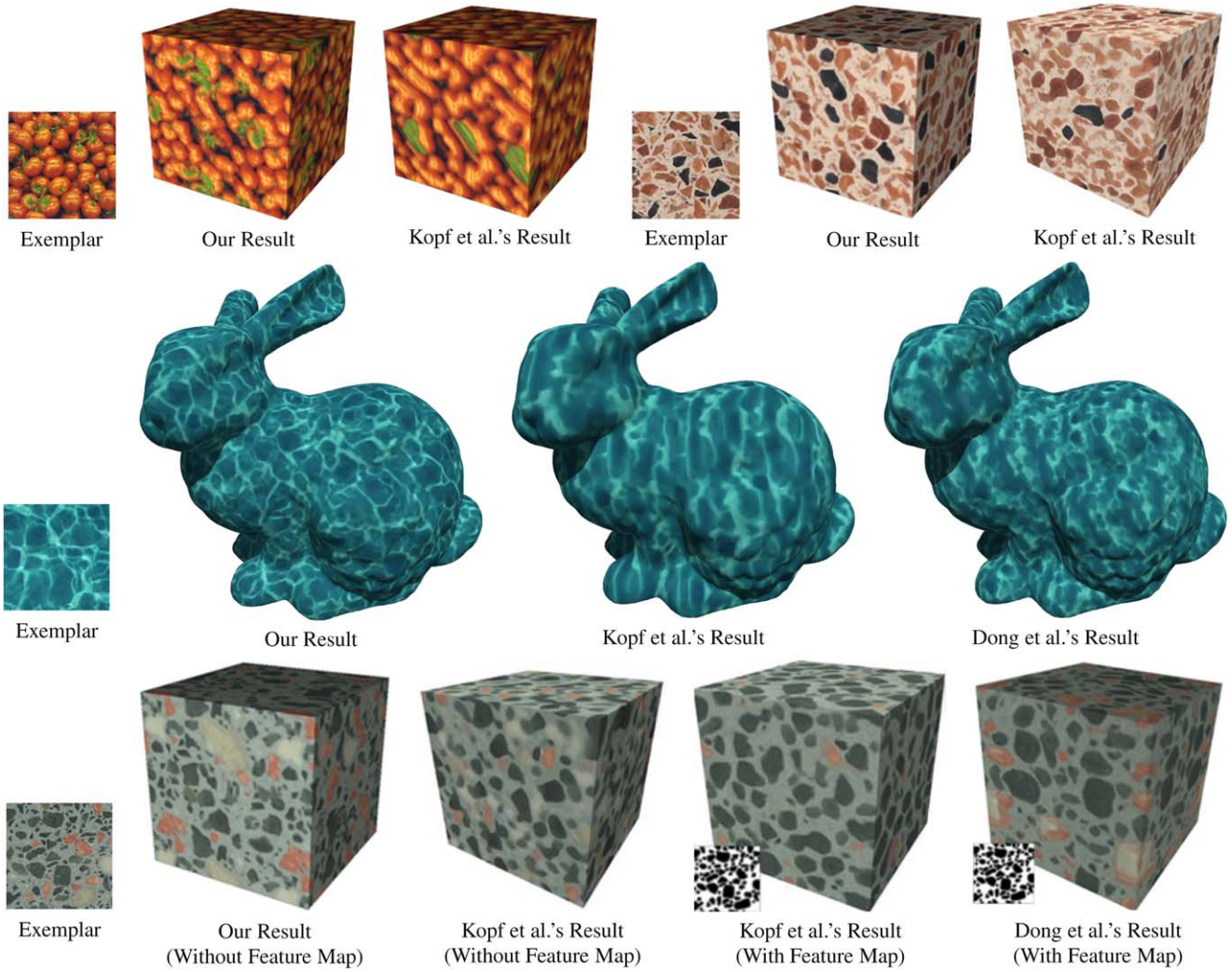


Fig. 9 Comparisons with recent methods. In the *first row*, our results do not suffer from blurring and our method preserves all kinds of texture structures better than Kopf et al.'s. In the *second row*, even the global large structures can be effectively preserved without blurring or broken structures, compared with Kopf et al. and Dong et al.'s results. And in the *last row*, texture structures are efficiently preserved even without the aid of the feature maps by our method

size is propitious to preserve the fine texture details. Specifically, we use 8×8, 6×6, and 4×4 windows for the three-level synthesis. The smaller neighborhoods cause our algorithm to be much faster. In this case, synthesizing a 128^3 solid texture often takes less than 5 minutes. And more fine texture details are preserved without destroying large texture structures. In Fig. 10, with smaller neighborhood sizes at the higher two levels, the first row shows that large texture structures are still preserved well. And the second row shows that more fine texture details are preserved.

6 Conclusions and future work

In this paper, we present a novel algorithm for synthesizing high quality solid textures from 2D exemplars. Based on the optimization framework, our algorithm enables most of the pixels in the exemplars to appear in the result volume equiprobably by using the position and index histogram matching methods. Experiments prove that our method is efficient enough to preserve not only the color histogram but also the various texture structures in the synthesized result. As noted, the local minima problem which often occurs in optimization-based approaches can be avoided by the position and index histogram matching. Experimental results demonstrate that our method outperforms or at least is comparable to the previous solid texture synthesis algorithms in terms of the synthesis quality.

For future work, besides synthesizing higher quality solid textures, we want to further improve the synthesis speed. And our method can be extended to synthesize high quality globally varying solid textures, since globally varying tex-

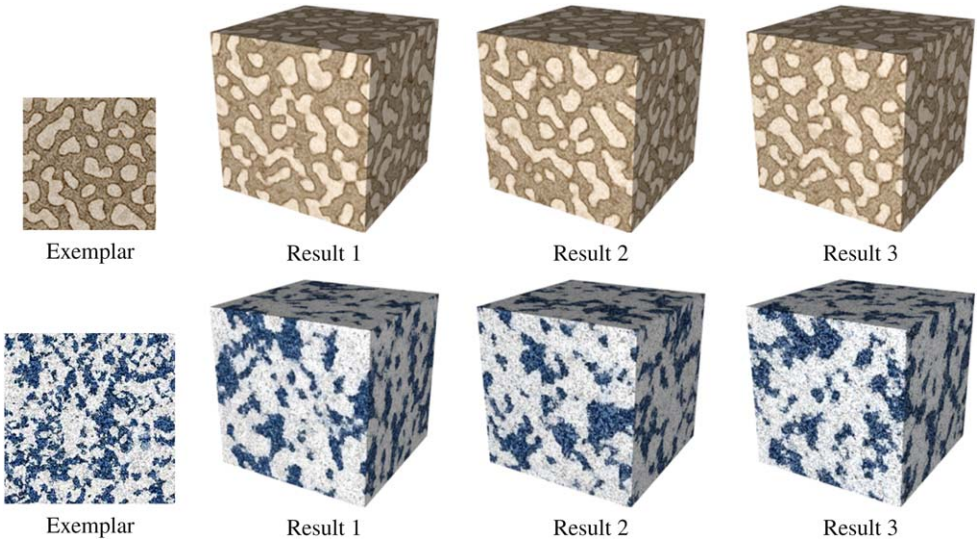


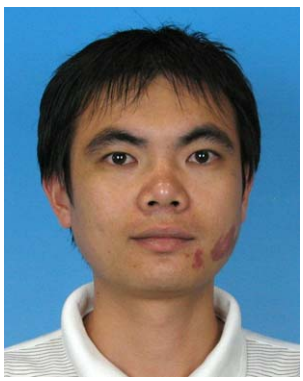
Fig. 10 Different neighborhood sizes at different levels. We use the neighborhood sizes 8×8, 8×8, 8×8 in Result 1, 8×8, 8×8, 6×6 in Result 2, and 8×8, 6×6, 4×4 in Result 3. With smaller neighborhood sizes at the higher two levels, more fine texture details are preserved without destroying large texture structures

tures [17] are becoming increasingly important in practical applications. Using existing methods to synthesize globally varying solid textures, random distribution instead of natural global distribution of the varying textures is generated in the synthesized results, leading to some visual artifacts. One possible solution is to generate a useful 3D control map, and globally varying solid textures are synthesized under the direction of the 3D control map.

Acknowledgements We would like to thank Fang Yang, Guidu Chen for help on writing, and the anonymous reviewers for their valuable suggestions and comments. This work is supported by National Science Foundation of China (Grant Nos. 90818011, 60773143 and 90715043), National High-Tech Research & Development Program of China (Grant No. 2007AA040401), and National Basic Research Program of China (Grant No. 2004CB719400).

References

1. Dischler, J.M., Ghazanfarpour, D., Freydier, R.: Anisotropic solid texture synthesis using orthogonal 2D views. Comput. Graph. Forum **17**(3), 87–96 (1998)
2. Dong, Y., Lefebvre, S., Tong, X., Drettakis, G.: Lazy solid texture synthesis. Comput. Graph. Forum **27**(4), 1165–1174 (2008)
3. Ebert, D.S., Musgrave, F.K., Peachey, D., Perlin, K., Worley, S.: Texturing and Modeling: A Procedural Approach. Morgan Kaufmann, San Mateo (2002)
4. Efros, A.A., Leung, T.K.: Texture synthesis by no-parametric sampling. In: IEEE International Conference on Computer Vision '99, pp. 1033–1038 (1999)
5. Efros, A.A., Freeman, W.T.: Image quilting for texture synthesis and transfer. In: Proceedings of SIGGRAPH 2001, pp. 341–346 (2001)
6. Han, J., Zhou, K., Wei, L.Y., Gong, M., Bao, H., Zhang, X., Guo, B.: Fast example-based surface texture synthesis via discrete optimization. Vis. Comput. **22**(9), 918–925 (2006)
7. Heeger, D.J., Bergen, J.R.: Pyramid-based texture analysis/synthesis. In: Proceedings of SIGGRAPH 1995, pp. 229–238 (1995)
8. Jagnow, R., Dorsey, J., Rushmeier, H.: Stereological techniques for solid textures. In: Proceedings of SIGGRAPH 2004, pp. 329–335 (2004)
9. Kopf, J., Fu, C.W., Cohen-Or, D., Deussen, O., Lischinski, D., Wong, T.T.: Solid texture synthesis from 2D exemplars. In: Proceedings of SIGGRAPH 2007, pp. 2.1–2.9 (2007)
10. Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A.: Graphcut textures: image and video synthesis using graph cuts. In: Proceedings of SIGGRAPH 2003, pp. 277–286 (2003)
11. Kwatra, V., Essa, I., Bobick, A., Kwatra, N.: Texture optimization for example-based synthesis. In: Proceedings of SIGGRAPH 2005, pp. 795–802 (2005)
12. Qin, X., Yang, Y.H.: Basic gray level aura matrices: theory and its application to texture synthesis. In: IEEE International Conference on Computer Vision '05, pp. 128–135 (2005)
13. Qin, X., Yang, Y.H.: Aura 3D texture. IEEE Trans. Vis. Comput. Graph. **31**(2), 379–389 (2007)
14. Tong, X., Zhang, J., Liu, L., Wang, X, Guo, B., Shum, H.Y.: Synthesis of bidirectional texture functions on arbitrary surfaces. In: Proceedings of SIGGRAPH 2002, pp. 665–672 (2002)
15. Wei, L.Y., Levoy, M.: Fast texture synthesis using tree-structured vector quantization. In: Proceedings of SIGGRAPH 2000, pp. 479–488 (2000)
16. Wei, L.Y.: Texture synthesis by fixed neighborhood searching. PhD thesis, Stanford University, (2002)
17. Wei, L.Y., Han, J., Zhou, K., Bao, H., Guo, B., Shum, H.Y.: Inverse texture synthesis. In: Proceedings of SIGGRAPH 2008, pp. 1–9 (2008)



Jiating Chen received his B.Sc. degree from School of Software, Tsinghua University in 2008. Currently he is a Ph.D. candidate in Department of Computer Science and Technology, Tsinghua University. His research interest is texture synthesis.



Bin Wang is an Assistant Professor in School of Software, Tsinghua University, China. He received his B.Sc. degree in Chemistry from Tsinghua University in 1999, and his Ph.D. in Computer Science from Tsinghua University, in 2005. His research interests include computer graphics and computer-aided design.