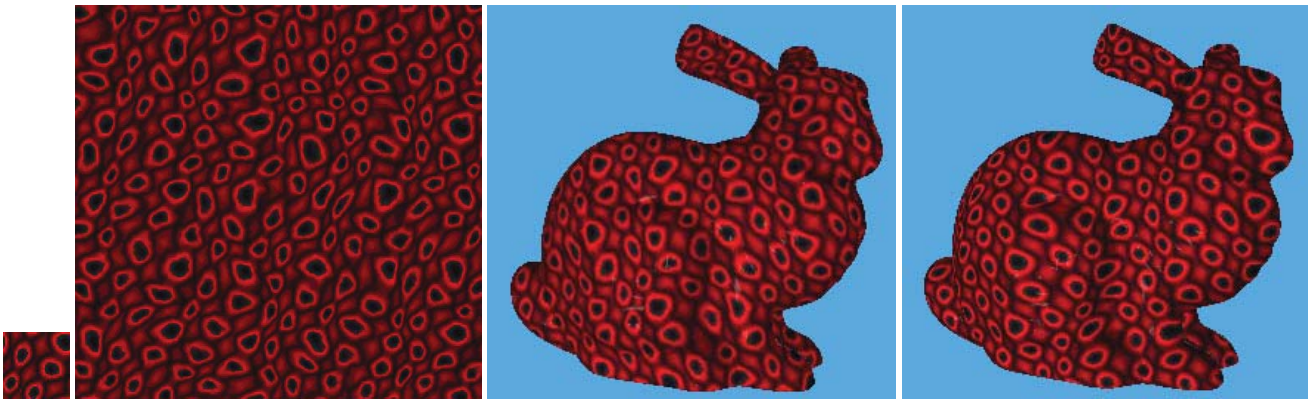
Simple and Efficient Example-based Texture Synthesis Using Tiling and Deformation

[1,2]Kan Chen* [1]Henry Johan [1,2]Wolfgang Mueller-Wittig

[1]Nanyang Technological University [2]Fraunhofer IDM@NTU



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

Figure 1: *(a) Input example texture (64×64). (b) Synthesized 2D texture (490×446). (c) Mapping the synthesized 2D texture (b) on Bunny. (d) Synthesized 3D solid texture (based on (a)) on Bunny (no texture parameterization needed for Bunny).*

Abstract

In computer graphics, textures represent the detail appearance of the surface of objects, such as colors and patterns. Example-based texture synthesis is to construct a larger visual pattern from a small example texture image. In this paper, we present a simple and efficient method which can synthesize a large scale texture in real-time based on a given example texture by simply tiling and deforming the example texture. Different from most of the existing techniques, our method does not perform search operation and it can compute texture values at any given points (random access). In addition, our method requires small storage which is only to store one example texture. Our method is suitable for synthesizing irregular and near-stochastic texture. We also propose methods to efficiently synthesize and map 3D solid textures on 3D meshes.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: Texture synthesis, tiling, deformation

1 Introduction

In computer graphics, texturing is a conventional way to add details to the surface of objects, but manually creating such textures requires tedious effort. To overcome this issue, example-based texture synthesis methods were proposed to synthesize a larger texture based on a small input example texture. Existing example-based texture synthesis methods, in general, have high computational cost because these methods usually involve intensive matching and copying pixels/patches. Furthermore, example-based methods usually offer only limited resolution and lack of random accessibility [Wei et al. 2009]. Another way for texture synthesis is Procedural texturing, which generates synthetic texture images by computing function values instead of copying from examples [Ebert et al. 2002]. However, it is in general difficult to design a procedure for reproducing a given specific pattern.

In this paper, we propose a simple and efficient method to synthesize a texture based on an input example texture in real-time. Our method has the advantages of the two approaches mentioned above. Specifically, it consists of the following features:
(1) The proposed method is simple and easy to implement (with only a few lines of code, see Appendix 1) and it is able to achieve comparable results to the prior work, especially for irregular and near-stochastic example textures. Moreover, pattern continuity of the example input texture is also preserved in the synthesized texture.
(2) Our method is efficient, it has no time-consuming search operation, thus it can synthesize textures in real-time. As a result, users can on the fly evaluate and control the synthesis results.
(3) Our method can synthesize textures directly on meshes or points with texture coordinates. Extending our basic idea, we also propose methods to efficiently synthesize and map 3D solid textures on 3D meshes.
(4) Our method is memory efficient and it only requires small storage. Only the example texture needs to be stored in memory.
(5) Our method allows random access. As such, textures can be computed at any given coordinates.

*e-mail: {kchen1,henryjohan,askwmwittig}@ntu.edu.sg

2 Related Work

2.1 Basic techniques for example-based texture synthesis

Example-based texture synthesis has been widely investigated (refer to the survey in [Wei et al. 2009]). The basic techniques are:
(1) **Pixel-based synthesis:** The texture is synthesized by sequentially fetching each pixel's color from the best matching neighborhood from the example texture [Efros and Leung 1999][Wei and Levoy 2000].
(2) **Patch-based synthesis:** The texture is synthesized by assembling patches rather than pixels from the input texture [Praun et al. 2000][Efros and Freeman 2001][Liang et al. 2001][Lefebvre and Neyret 2003][Kwatra et al. 2003].
(3) **Texture optimization:** This approach combines the properties of pixel and patch based algorithms. The texture synthesis problem is formulated as an optimization problem which is solved by minimization of a energy function [Kwatra et al. 2005][Han et al. 2006][Kopf et al. 2007].
(4) **Other methods:** There are also methods based on statistical analysis [Heeger and Bergen 1995][Portilla and Simoncelli 2000], coherence search [Ashikhmin 2001][Tong et al. 2002] and high resolution texture synthesis [Lefebvre and Hoppe 2005] [Han et al. 2008]. These algorithms require iteratively searching and copying pixels/patches or an optimization solver. Thus, they usually need high computational cost. Since in general, they require synthesized neighbor pixels/patches, they lack of random accessibility.

2.2 Tiling and deformation techniques for example-based texture synthesis

Besides the above basic algorithms, there are some techniques based on tiling and deformation:
(1) **Tiling:** Tile-based methods are very popular and have several applications in Computer Graphics [Lefebvre and Neyret 2003][Fu and Leung 2005][Lefebvre 2008]. Stam first introduced using Wang Tiles for texture synthesis in [Stam 1997]. Various tiling-based methods have also been applied in example-based texture synthesis in [Cohen et al. 2003][Wei 2004][Ng et al. 2005][Lagae and Dutre 2006]. To make tileable texture, there are many methods, such as [Wei and Levoy 2000].
(2) **Deformation:** There are several works using deformation for texture synthesis [Liu and Lin 2003][Liu et al. 2004][Wu and Yu 2004][Shen et al. 2006][Shen et al. 2007]. These methods, however, usually require the analysis of structural or layering information of the example texture and re-composition of textures. Existing deformation based methods such as [Liu et al. 2004] also exhibit some distortions in the results. Based on observing the results in previous work, we believe that some distortions can represent the variations of features in some types of textures. Thus, we developed a deformation-based method for texture synthesis.

2.3 Procedural texture synthesis

Procedural texture synthesis is a major texture synthesis method [Ebert et al. 2002]. It uses specific algorithms to generate the appearance of natural surfaces such as wood, caustic, and marble [Perlin 1985]. Procedural texture synthesis methods usually require little storage, easy to compute, random accessible and thus can generate texture color at any given coordinates. However, it is difficult to design a procedural algorithm which can reproduce a given pattern. Our proposed method computes the color on a point based on only its coordinates, and only sampling operation is involved, thus it has the same advantages as procedural methods.

2.4 Solid texture synthesis

The main advantages of solid texture are: it does not require texture parameterization on 3D meshes and it allows volumetric texturing, such as the interior of wood materials. Many methods have been proposed for solid texture synthesis (refer to the survey in [Pietroni et al. 2010]). The existing methods [Wei 2003][Jagnow et al. 2004][Kopf et al. 2007][Wang et al. 2010] still require iteratively finding and copying pixels/patches.

3 Example-based 2D Texture Synthesis

We observe that in many cases, especially for irregular and near-stochastic example textures, we can generate a variety of similar patterns by just deforming the input example texture. Based on this finding, our texture synthesis method consists of the following three main steps (Figure 2):
(1) **Tiling and grids:** We tile the input example texture, and name it as tiled example texture. The same tiling arrangement is also defined on the synthesized texture so that there is a one-to-one correspondence between the tiles in the tiled example texture and the synthesized texture. We then define 2D grids on the tiles and thus we have a one-to-one correspondence between the grids.
(2) **Deformation:** The grid cell's corners of the tiled example texture are displaced using noise. This means a grid cell in the synthesized texture corresponds to a deformed grid cell in the tiled example texture.
(3) **Mapping and re-sampling:** A point inside a grid cell of the synthesized texture is mapped to a point in the corresponding deformed grid cell in the tiled example texture using bilinear interpolation. Its texture value is sampled from the corresponding point in the tiled example texture. Thus, our method is based on inverse warping.

3.1 Tiling and grids

If the input texture is not tileable, we can use existing methods such as [Wei and Levoy 2000] to create a tileable texture. Another method is to offset the texture, wrapping around by half, and blend the seams introduced from offsetting using the clone stamp tool of Adobe©Photoshop©[Adobe©2010] (Figure 3). Then we tile the tileable texture. After that 2D grids are defined, the size of each 2D grid cell (s_x, s_y) can be set by user, usually it is set based on the size of the pattern in the input texture. For each point (i, j), we can compute its coordinates within its grid cell as (u, v), $0 \leq u, v \leq 1$ (Figure 2). Note that in terms of implementation, we do not need to explicitly store the tiled example texture and grids in memory, we only need to store the example texture.

3.2 Deformation

In this step, the position of the four corners (x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3) of each grid cell g in the tiled example texture are displaced. We want to have some continuity in the displacement and thus we determine the amount of displacement using Perlin Noise $n \in [-1, 1]$. We adopt Perlin noise since it meets the following requirements and has been widely used to model natural effects. (1) Noise values can be evaluated at any given coordinates. (2) Noise values change smoothly, so smooth deformation can be achieved. (3) Noise values can be evaluated very fast. (4) Little memory consumptions. The noise are computed for horizontal and vertical components: (n_x, n_y). The new positions of the four corners after displacement $(x_0^{'}, y_0^{'})$, $(x_1^{'}, y_1^{'})$, $(x_2^{'}, y_2^{'})$, $(x_3^{'}, y_3^{'})$ (Figure 2) are computed as follows.

$$(x^{'}, y^{'}) = (x, y) + (n_x \lambda_x s_x, n_y \lambda_y s_y), \quad (1)$$

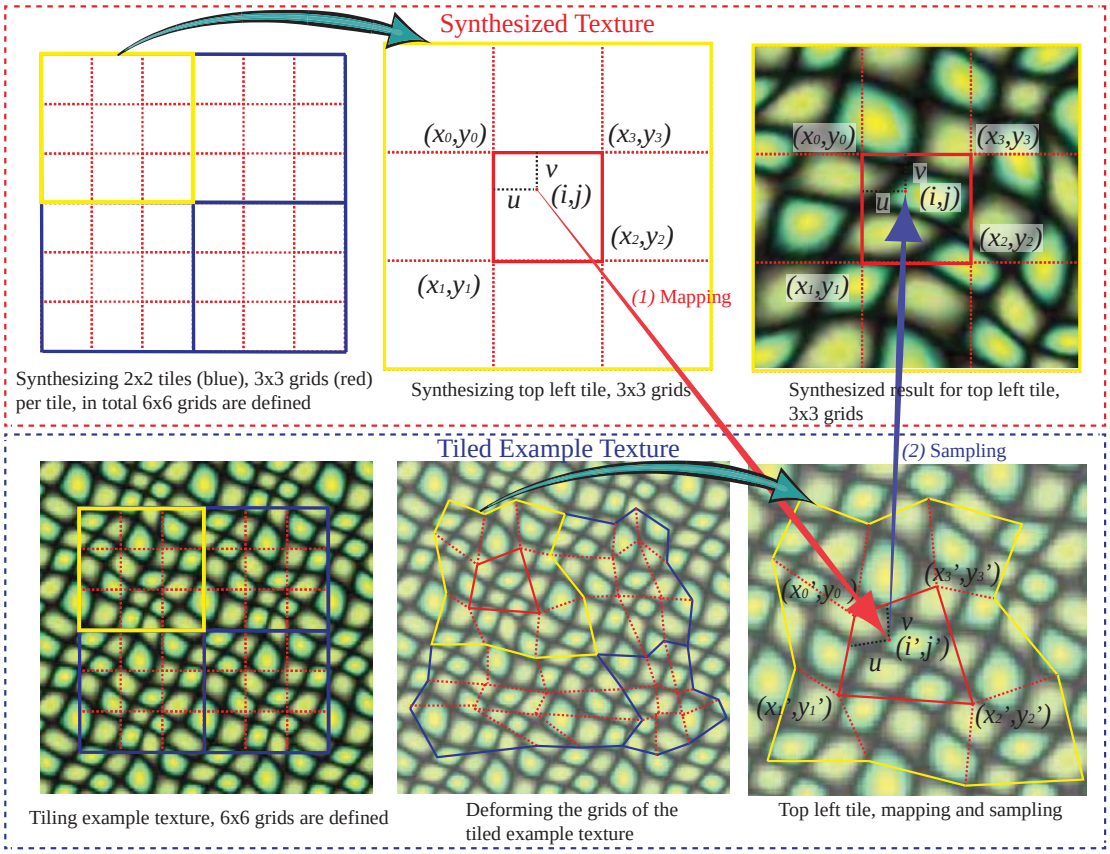


Figure 2: *Steps of the proposed example-based texture synthesis. Inverse warping is employed. (1) For point (i, j) in the synthesized texture, compute its corresponding point $(i^{'}, j^{'})$ in the tiled example texture. (2) Sample the color at point $(i^{'}, j^{'})$ in the tiled example texture as the color of point (i, j) in the synthesized texture.*

Figure 3: *Manually creating tileable texture. (a) Non-tileable texture. (b) Offsetting the texture, wrapping around by half. (c) Blending the seams.*

where λ_x and λ_x are noise factors that control the amount of displacement with respect to the grid cell's size. A threshold can be set to restrict the amount of deformation in the horizontal and vertical directions. Since our method can synthesize texture in real-time, user can change the values of parameters on the fly to adjust the synthesis results.

3.3 Mapping and re-sampling

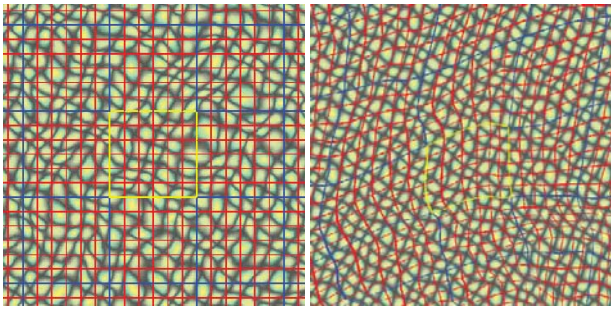
In this step, for point (i, j) in the synthesized texture, compute its corresponding point $(i^{'}, j^{'})$ in the tiled example texture using bilinear interpolation:

$$\begin{aligned}(i^{'}, j^{'}) &= (1-v)\Big((1-u)(x_0^{'}, y_0^{'}) + u(x_3^{'}, y_3^{'})\Big) \\ &+ v\Big((1-u)(x_1^{'}, y_1^{'}) + u(x_2^{'}, y_2^{'})\Big). \end{aligned} \quad (2)$$

We sample the value at $(i^{'}, j^{'})$ at the tiled example texture as the value of point (i, j) (Figures 2 and 4).

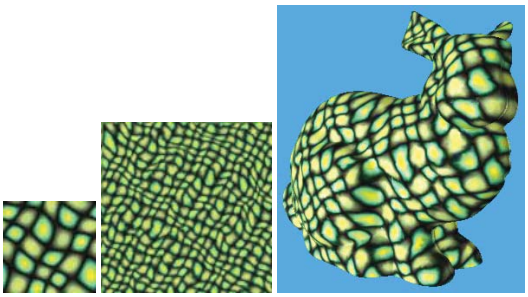
3.4 Implementation

We implement our method in GPU, the main steps contains only a few lines of HLSL shader code (refer to Appendix 1). A 2D texture can be synthesized in real-time (refer to video). Similarly, we can also directly synthesize 2D texture on the surface of a 3D mesh with texture coordinates (Figure 5). Only the 2D example texture needs to be stored. The deformation grids and sampling techniques in Section 3 are computed on the fly in the pixel shader. Only one sampling operation to the example texture is required per pixel. To reduce the computational time, we can pre-compute the Perlin noise values and store them as a texture. In this case, another noise texture is maintained. To reduce aliasing artifacts, it is sufficient to use texture filtering methods such as mipmapping and anisotropic filtering in shaders, since in general the deformation is not large.



(a) (b)

Figure 4: *Mapping and re-sampling implementation. Red: one grid cell. Blue: one tile. Yellow: corresponding tile. (a) Synthesized texture with grids. (b) Tiled example texture with deformed grids, sampling from the tiled example texture.*



(a) (b) (c)

Figure 5: *(a) Input 2D example texture (64×64). (b) Synthesized 2D texture (256×256). (c) Synthesized 2D texture on Bunny with uv-coordinates.*

4 Example-based 3D Solid Texture Synthesis

The proposed method can be extended from the 2D case in Section 3 to synthesize 3D solid texture.

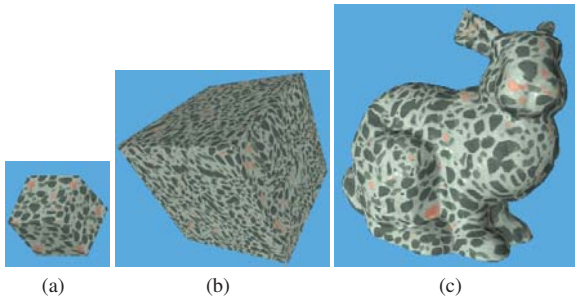
(1) **Method 1:** The input is a 3D example solid texture. We apply tiling, define 3D grids, 3D deformation of grid cell's corners, and re-sampling based on 3D positions of points following the algorithm in 2D case (Figure 6).

(2) **Method 2:** The input is a 2D example texture. An initial solid texture is created by stacking the input 2D example texture then applying the algorithm in Method 1. Figure 7 shows an example of stacking the input example texture in Z direction. Comparing with Method 1, in terms of storage, only one 2D image is maintained. This method is simple, however, on faces parallel to the stacking direction, the features of the example texture cannot be well synthesized (Figures 7(b) and 8(c)). This is because originally there are no features from the example texture but only lines on these faces parallel to the stacking direction in the initial solid texture (Figure 7(a)). In our experiments, we observe that this method of only stacking 2D example texture in one direction usually can achieve visually plausible results for example textures with irregular or near-stochastic directional patterns, such as marble and wood textures (Figures 8(a),(b)).

To synthesize features from the example texture on all face orientations, our idea is to stack the input 2D example texture in three directions (X, Y and Z directions), that is we synthesize three solid textures and blend them to create the final solid texture. The blending is based on the surface's normal of a point on mesh where the final synthesized solid texture is applied to. The normal of one point on the mesh is $\vec{n}$ and $\vec{x}, \vec{y}, \vec{z}$ refers to the X, Y and Z directions. Assume w_x, w_y, w_z are the blending weights for the solid textures by stacking the 2D exemplar in the three respective directions. If the point's normal is closer to direction i, w_i should be larger and thus we use the following weight function:

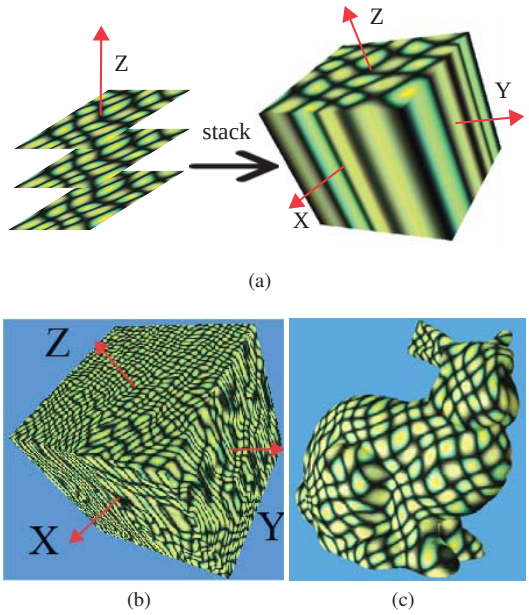
$$w_i = (\vec{n} \cdot \vec{i})^{2p}, \; i \in x, y, z, \tag{3}$$

where p is the parameter to control the normal directions which are affected by each solid texture. Lower p value means each solid texture affects more normal directions. We then normalize the weights w_i and compute the color of the point as the weighted sum of the values sampled at three solid textures. Note that we do not need to explicitly generate the three solid textures and thus only one 2D example texture needs to be stored and three texture sampling operations are performed. Figure 8(d) shows a synthesized result of the stacking in three directions method.



(a) (b) (c)

Figure 6: *Method 1. (a) Input 3D example solid texture (64×64×64). (b) Synthesized solid texture (256×256×256). (c) Synthesized solid texture colors on Bunny based on (a).*



(a)

(b) (c)

Figure 7: *(a) Stacking the input 2D example texture to create an initial solid texture. (b) Synthesized solid texture by tiling, deformation and re-sampling. (c) Synthesized solid texture on Bunny (stacking in one direction).*

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

Figure 8: *(a)(b) Synthesized textures by stacking in one direction. (c) Synthesized texture by stacking in one direction: on faces which are almost parallel to the stacking direction, features are not well synthesized. (d) Blending three synthesized solid textures based on stacking in three directions: features from the example texture can be better synthesized on all face orientations.*

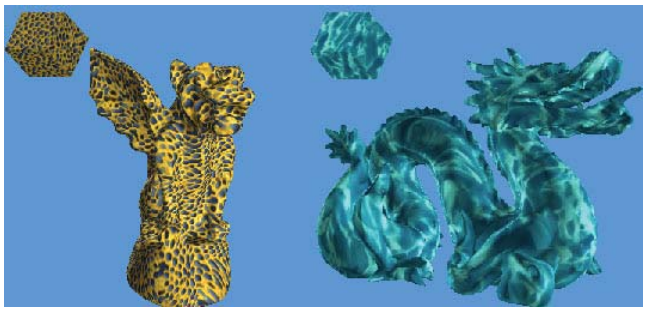


Figure 10: *Synthesized solid textures based on an example 3D solid texture (Method 1) on different objects.*

5 Results and Discussions

Figures 1, 5, 6, 9, 10 and 11 show various texture synthesis results using our proposed method. We implemented our method on a laptop with Intel Core i7 M620 @2.67Ghz CPU, 4.0GB Memory and Nvidia NVS3100M GPU on the DirectX 9.0c platform. With precomputed Perlin noise, 2D textures can be synthesized at 150FPS, and 2D or 3D textures (Methods 1 and 2) can be directly synthesized on a mesh (80000 faces) at 120FPS. The frame rate drops to half if we compute Perlin noise in real-time in GPU. In terms of storage, only one 2D (or 3D) example texture is maintained in memory. We tested our method using the texture categorization based on [Liu et al. 2004] (Figure 9). We do not show the results of regular textures since it is sufficient to synthesize them by tiling the input example. From the experiments, we found that the details and curvature of a 3D mesh usually help to enhance the visual impression of the synthesized textures. Please see the video and supplemental material for more results.

Comparison with other 2D texture synthesis methods Our approach provides a simple and efficient solution to 2D texture synthesis. No iteratively finding and copying pixels/patches are required. The results are comparable to the existing methods. Based on the results in the papers and websites of [Efros and Leung 1999][Wei and Levoy 2000][Efros and Freeman 2001][Ashikhmin 2001] and [Cohen et al. 2003], we found that some unnatural distortion artifacts and discontinuities are noticeable. Compared with [Liu et al. 2004] and [Kwatra et al. 2005], our method can achieve results with similar quality. Tiling based methods [Cohen et al. 2003][Wei 2004] and [Ng et al. 2005] are generally based on creating Wang tiles to reduce the repetition artifacts. However in the results, some pattern discontinuities are noticeable. On the other hand, pattern continuity of the example input texture is preserved in our results. Deformation based methods, such as [Liu et al. 2004] can generate good quality results, but it requires texture analysis and user assistance for identifying and adjustment of lattices (1 to 20 minutes based on their paper). On the other hand, our method can generate comparable quality and it only requires sampling operations, very little user assistance, it is simpler and very suitable for real-time applications. In addition, we can compute the texture color at any given coordinates, there is no restriction on the size of the synthesized texture.

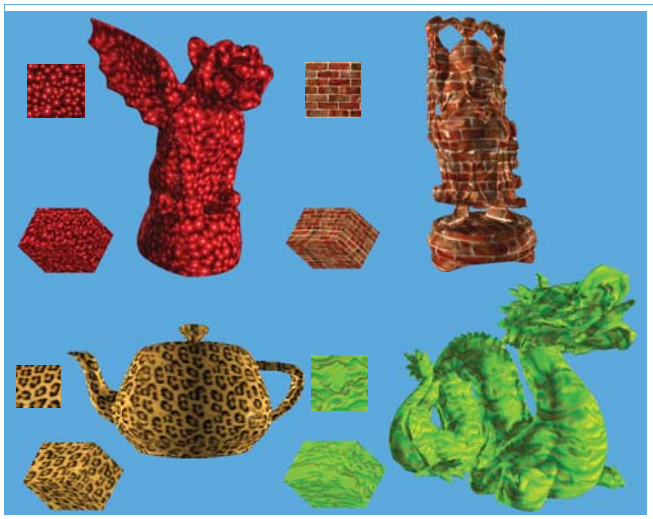
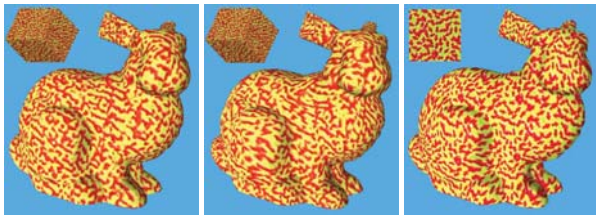


Figure 11: *Synthesized solid textures based on stacking a 2D texture (Method 2) on different objects.*



(a) (b) (c)

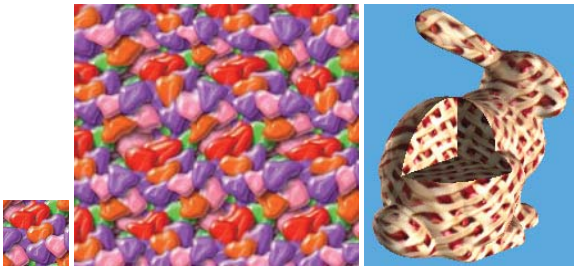
Figure 12: *Synthesized solid texture on Bunny. (a) Prior work [Kopf et al. 2007]. (b) Method 1. (c) Method 2.*

Comparison with other solid texture synthesis method Our method can synthesize results comparable to [Kopf et al. 2007] (Figure 12). However our method is simpler and more efficient, it has no time-consuming searching and copying operations. Furthermore, our stacking based approach only needs to store one 2D example texture in memory.

Limitations Our technique may perform less well for the type of texture with too much color variations, since such color patterns may become visually repetitive in our results (Figure 13(a)). Our method is also not suitable for textures with regular shapes such as straight lines and smooth curves since deformation will deform these features. However, for irregular and near-stochastic textures, our method is able to generate good quality results. To reduce unwanted stretching/distortion artifacts, we can perform deformation only in selected directions. We also provide parameters to limit the amount of deformation.

For method 2 of the example-based solid texture synthesis, three solid textures are blended. For faces whose normals are parallel

to direction i, ($i \in$ X,Y,Z), the solid textures stacking in directions other than i, do not contribute to the color of these faces which means no blending is performed for these faces. This will cause discontinuity along the shared edge of two faces whose normals are parallel to two of X, Y and Z axes (Figure 13(b)). Furthermore, the simple linear blending method also introduces some multiple exposure artifacts at the blending area. In our experiments, we observe that such artifacts will be much less noticeable for the type of texture with less color variations. However, since method 2 does not require texture parametrization for 3D meshes and it only needs to store one example texture in memory, it is very practical for solid texturing 3D meshes and very suitable for real-time applications.



(a) (b)

Figure 13: *Limitations of our method. (a) Synthesizing texture with many color variations. Left: Input example. Right: Synthesized 2D texture. (b) Synthesized 3D solid texture on Bunny based on stacking the input example texture.*

6 Conclusions and Future Work

In this paper, we have proposed a simple and efficient method for texture synthesis that does not require iteratively finding and copying pixels/patches. The proposed method is based on tiling, deformation, and re-sampling techniques and is suitable for real-time synthesis of 2D textures and 3D solid textures. Our method only requires small storage and it allows random access. As shown, our results are comparable to the state-of-the-art texture synthesis methods. We have also presented a method to synthesize a 3D solid texture based on a 2D example texture. As future work, we would like to apply this method to synthesize bump and displacement maps. We also would like to reduce the unwanted repetitive effect such as the one in Figure 13(a) using some post processing or recoloring techniques. Another interesting future work is to let users to control the placements of certain features.

Acknowledgments

We gratefully thank the reviewers for their constructive comments. This research was supported in part by Fraunhofer IDM@NTU, which is funded by the National Research Foundation and managed through the multi-agency Interactive & Digital Media Programme Office hosted by Media Development Authority of Singapore. The input example textures are from [Wei and Levoy 2000][Lin et al. 2004][Ng et al. 2005][Kopf et al. 2007]. The 3D models are from Stanford 3D Scanning Repository.

References

ADOBE©. 2010. *Photoshop©CS5.1*. Adobe System Inc.

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, I3D '01, 217–226.

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. *ACM Trans. Graph. 22*, 3 (July), 287–294.

EBERT, D. S., MUSGRAVE, F. K., PEACHEY, D., PERLIN, K., AND WORLEY, S. 2002. *Texturing and Modeling: A Procedural Approach*, 3rd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 341–346.

EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. In *Proceedings of the International Conference on Computer Vision-Volume 2*, IEEE Computer Society, Washington, DC, USA, ICCV '99, 1033–1038.

FU, C.-W., AND LEUNG, M.-K. 2005. Texture tiling on arbitrary topological surfaces using wang tiles. In *Proceedings of the Sixteenth Eurographics conference on Rendering Techniques*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, EGSR'05, 99–104.

HAN, J., ZHOU, K., WEI, L.-Y., GONG, M., BAO, H., ZHANG, X., AND GUO, B. 2006. Fast example-based surface texture synthesis via discrete optimization. *Vis. Comput. 22*, 9 (September), 918–925.

HAN, C., RISSER, E., RAMAMOORTHI, R., AND GRINSPUN, E. 2008. Multiscale texture synthesis. *ACM Trans. Graph. 27*, 3 (August), 51:1–51:8.

HEEGER, D. J., AND BERGEN, J. R. 1995. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '95, 229–238.

JAGNOW, R., DORSEY, J., AND RUSHMEIER, H. 2004. Stereological techniques for solid textures. *ACM Trans. Graph. 23*, 3 (August), 329–335.

KOPF, J., FU, C.-W., COHEN-OR, D., DEUSSEN, O., LISCHINSKI, D., AND WONG, T.-T. 2007. Solid texture synthesis from 2d exemplars. *ACM Trans. Graph. 26*, 3 (July), 2:1–2:9.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph. 22*, 3 (July), 277–286.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. *ACM Trans. Graph. 24*, 3 (July), 795–802.

LAGAE, A., AND DUTRE, P. 2006. An alternative for wang tiles: colored edges versus colored corners. *ACM Trans. Graph. 25*, 4 (October), 1442–1459.

LEFEBVRE, S., AND HOPPE, H. 2005. Parallel controllable texture synthesis. *ACM Trans. Graph. 24*, 3 (July), 777–786.

LEFEBVRE, S., AND NEYRET, F. 2003. Pattern based procedural textures. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, I3D '03, 203–212.

LEFEBVRE, S. 2008. *Filtered Tilemaps (in Shader X6)*. Shader X6. Charles River Media, 63–72.

LIANG, L., LIU, C., XU, Y.-Q., GUO, B., AND SHUM, H.-Y. 2001. Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph. 20*, 3 (July), 127–150.

LIN, W.-C., HAYS, J., WU, C., KWATRA, V., AND LIU, Y. 2004. A comparison study of four texture synthesis algorithms on near-regular textures. In *ACM SIGGRAPH 2004 Posters*, ACM, New York, NY, USA, SIGGRAPH '04, 16.

LIU, Y., AND LIN, W.-C. 2003. Deformable texture: the irregular-regular-irregular cycle. Tech. Rep. CMU-RI-TR-03-26, Robotics Institute, Pittsburgh, PA, August.

LIU, Y., LIN, W.-C., AND HAYS, J. 2004. Near-regular texture analysis and manipulation. *ACM Trans. Graph. 23*, 3 (August), 368–376.

NG, T.-Y., WEN, C., TAN, T.-S., ZHANG, X., AND KIM, Y. J. 2005. Generating an ω-tile set for texture synthesis. In *Computer Graphics International 2005*, CGI'05, 177–184.

PERLIN, K. 1985. An image synthesizer. *SIGGRAPH Comput. Graph. 19*, 3 (July), 287–296.

PIETRONI, N., CIGNONI, P., OTADUY, M., AND SCOPIGNO, R. 2010. Solid-texture synthesis: A survey. *IEEE Comput. Graph. Appl. 30*, 4 (July), 74–89.

PORTILLA, J., AND SIMONCELLI, E. P. 2000. A parametric texture model based on joint statistics of complex wavelet coefficients. *Int. J. Comput. Vision 40*, 1 (October), 49–70.

PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2000. Lapped textures. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 465–470.

SHEN, J., JIN, X., MAO, X., AND FENG, J. 2006. Completion-based texture design using deformation. *Vis. Comput. 22*, 9 (September), 936–945.

SHEN, J., JIN, X., MAO, X., AND FENG, J. 2007. Deformation-based interactive texture design using energy optimization. *Vis. Comput. 23*, 9 (August), 631–639.

STAM, J. 1997. Aperiodic texture mapping. Tech. rep., European Research Consortium for Informatics and Mathematics.

TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B., AND SHUM, H.-Y. 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Trans. Graph. 21*, 3 (July), 665–672.

WANG, L., ZHOU, K., YU, Y., AND GUO, B. 2010. Vector solid textures. *ACM Trans. Graph. 29*, 4 (July), 86:1–86:8.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 479–488.

WEI, L.-Y., LEFEBVRE, S., KWATRA, V., AND TURK, G. 2009. State of the Art in Example-based Texture Synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR*, Eurographics Association, Munich, Allemagne, 93–117.

WEI, L.-Y. 2003. Texture synthesis from multiple sources. In *ACM SIGGRAPH 2003 Sketches & Applications*, ACM, New York, NY, USA, SIGGRAPH '03, 1.

WEI, L.-Y. 2004. Tile-based texture mapping on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ACM, New York, NY, USA, HWWS '04, 55–63.

WU, Q., AND YU, Y. 2004. Feature matching and deformation for texture synthesis. *ACM Trans. Graph. 23*, 3 (August), 364–367.

Appendix 1. Sample HLSL code

```
//For point (i, j) in the synthesized texture,
//find the four corners (xy[0] to xy[3]) of the grid cell
//that (i, j) belongs to. Grid_Size is the grid size.
xy[0] = floor(float2(i, j) / Grid_Size) * Grid_Size;
xy[1] = xy[0] + float2(0 , Grid_Size.y);
xy[2] = xy[0] + float2(Grid_Size.x, Grid_Size.y);
xy[3] = xy[0] + float2(Grid_Size.x, 0 );

//Find the uv coordinates for point (i, j).
u = (i - xy[0].x) / Grid_Size.x;
v = (j - xy[0].y) / Grid_Size.y;

//Displace the four corners (xy[0] to xy[3]) in tiled example
//texture using perlinNoise which is generated based on
//[Perlin 1985], lambda is the 2D noise factor.
for (int t = 0; t < 4; t++) {
xy_new[t] = xy[t]
+ perlinNoise(xy[t]) * lambda * Grid_Size;}

//Bilinear interpolation to compute the corresponding
//point for (i, j) in the tiled example texture.
ij_new = (1 - v) * ( (1 - u) * xy_new[0] + u * xy_new[3] )
          + v * ( (1 - u) * xy_new[1] + u * xy_new[2] );

//Sample the tiled example example texture for the value
//at ij_new and assign the value to point (i, j).
ij.color = tex2D(INPUT_EXAMPLE_TEXTURE, ij_new);
```

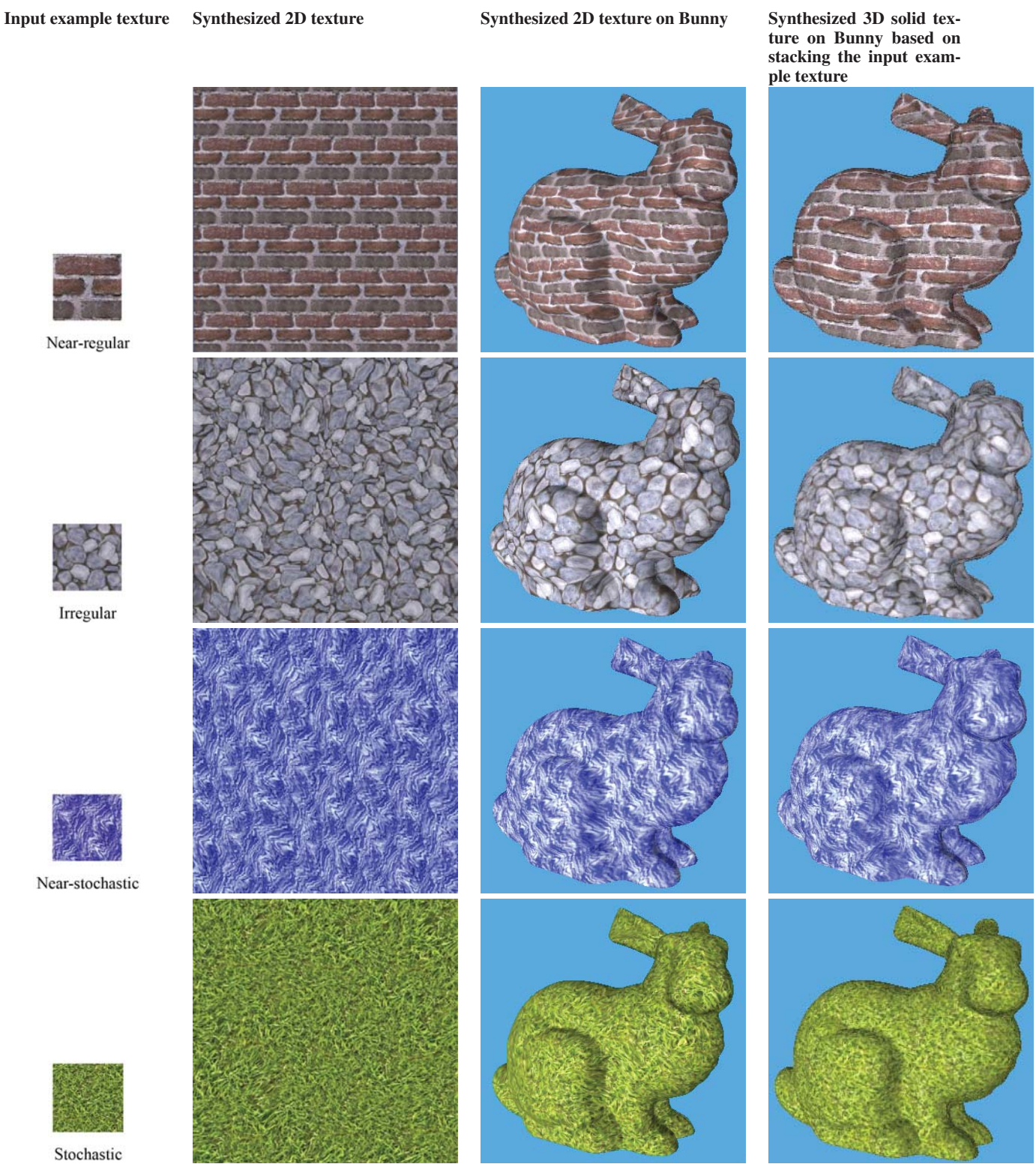


Figure 9: *Results of the proposed example-based texture synthesis.*