ARTICLE IN PRESS

Graphical Models 73 (2010) 57–71

Contents lists available at ScienceDirect

Graphical Models

journal homepage: www.elsevier.com/locate/gmod

Sketch guided solid texturing

Guo-Xin Zhang [a], Song-Pei Du [a], Yu-Kun Lai [b], Tianyun Ni [c], Shi-Min Hu [a,*]

[a] Tsinghua University, China
[b] Cardiff University, UK
[c] NVIDIA, USA

ARTICLE INFO

Article history:
Received 1 August 2010
Received in revised form 22 October 2010
Accepted 28 October 2010
Available online xxxx

Keywords:
Solid texture
Texture synthesis
Sketch

ABSTRACT

Compared to 2D textures, solid textures can represent not only the bounding surfaces, but also their interiors. Existing solid texture synthesis methods pay little attention to the generation of conforming textures that capture geometric structures or reflect the artists' design intentions. In this paper, we propose a novel approach to synthesizing solid textures using 2D exemplars. The generated textures locally agree with a tensor field derived from user sketching curves. We use a deterministic approach and only a small portion of the voxels needs to be synthesized on demand. Correction is fundamental in deterministic texture synthesis. We propose a history windows representation, which is general enough to unifiedly represent various previous correction schemes, and a dual grid scheme based on it to significantly reduce the dependent voxels while still producing high quality results. Experiments demonstrate that our method produces significantly improved solid textures with a small amount of user interaction.

1. Introduction

Textures play an essential role in current rendering techniques. Conceptually speaking, a texture is a function represented by a regular array of discrete samples, which defines some rendering attribute (e.g. color) on a surface. In addition to the sampled data, there must also exist a mapping between the texture's sampling space and the surface of the object which is to be textured. Most research on texture synthesis to date has been based on two-dimensional textures. However, 2D textures require complex mapping and suffer nearly unavoidable distortion when applied to objects in 3D space. Solid textures, which are sampled in three dimensions, offer an attractive alternative to 2D textures. Since the texture values are sampled on a 3D grid, they can be mapped to 3D geometry without the need for extra parametrization. Based on the similarity with physical solid objects, solid textures also allow the objects to undergo physical operations. Consistent textures are produced from the whole volume instead of only the bounding surface.

Consistency with significant geometric structures or features is crucial if textures are to reveal rather than obscure the shape of the model [30]. It is also widely known that texture synthesis over surfaces and texture mapping can benefit from controllable, well designed vector fields [23,28,31,5]. However, solid texture synthesis methods developed so far have paid little attention to the alignment of textures with the general tensor field or with the overall shape of objects. This is not particularly noticeable if the textures are isotropic, but for anisotropic textures, it is often desirable that textures follow natural or designed direction fields on the surface and in the interior. Furthermore, many real objects contain their own internal 3D structures, which should be reproduced if textures are synthesized on such objects. For example, as shown in Fig. 1, a rattan fertility object typically follows its overall shape on the surface and is consistently woven inside. To obtain realistic results

* Corresponding author.
E-mail addresses: zgx.net@gmail.com (G.-X. Zhang), dsp05@mails.tsinghua.edu.cn (S.-P. Du), Yukun.Lai@cs.cardiff.ac.uk (Y.-K. Lai), tianyun06@gmail.com (T. Ni), shimin@tsinghua.edu.cn (S.-M. Hu).

1524-0703/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.gmod.2010.10.006

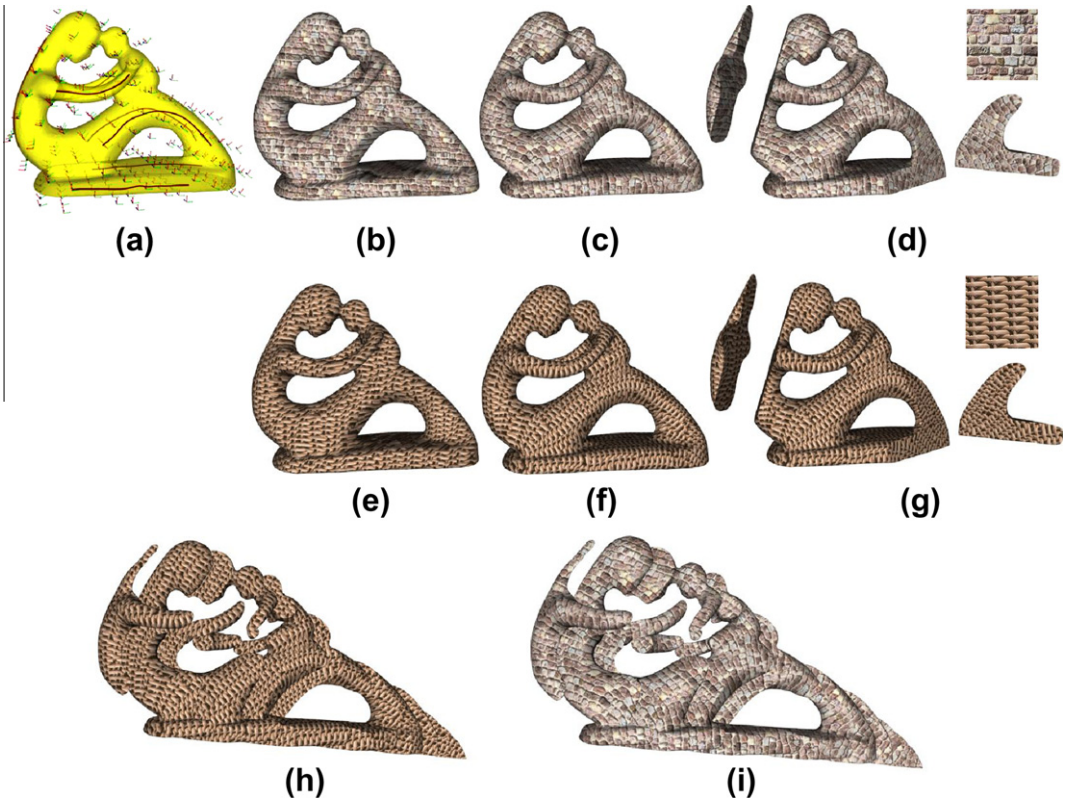

Fig. 1. Sketch guided solid texture synthesis on the 'fertility' model. Compared with the results with a trivial field (b and e), using a few user sketched curves (a), produced solid textures (c and f) are well aligned with the overall shape. Interior structures of (c and f) are revealed in (d and g) (h and i).

in the virtual world, such structures should be reproduced during solid texture synthesis process.

Similar to [10,2], our method synthesizes solid textures from 2D exemplars, since they are much easier to obtain. In order to synthesize solid textures that capture the overall shape and internal structure of objects, we propose a novel method with the following contributions.

First, we achieve significant improvement in appearance and consistency by synthesizing solid textures from 2D exemplars with non-trivial tensor fields. Synthesized solid textures following significant features can often be more impressive. To compactly reproduce real textured solids by synthesis, exemplar images with particular orientation (rather than canonical axes as previous methods do) of the solid can often be much more representative. This flexibility further allows artists to control the synthesized solid textures and effectively express their design intentions. Although automatic detection of guiding feature curves are possible, we believe certain amount of user interaction is in fact a more flexible solution as user controlled design may follow other artistic consideration (see the example in Fig. 14). Our method generates user-guided solid textures by constructing smooth 3D tensor fields based on interactive user sketching and then synthesizing solid textures following the sketch derived tensor field.

Second, based on deterministic parallel synthesis paradigm [2,14], we propose a novel algorithm to significantly improve the efficiency through a general history windows representation combined with a dual grid based correction scheme. Compared to two-dimensional image textures, solid textures can often be overwhelming to compute and store. In contrast to traditional solid textures, those following 3D tensor fields cannot be produced by synthesizing in a cube and repeatedly translating the cube to cover the whole space, as the tensor field will not usually be consistent between translated cubes. To solve this problem, we also use a synthesis-on-demand approach to synthesize only the visible portion of overall voxels. This does require that the synthesis process should be deterministic. However, compared to 2D texture synthesis, the determinism will introduce much more computation and storage overhead in 3D texture synthesis. Some common techniques such as subpass will also introduce significantly more overhead than its 2D counterpart. In addition, for more structured, anisotropic exemplars as used in this work, a larger neighborhood and/or more passes (subpasses) of synthesis are needed to produce high quality results. Large numbers of dependent voxels need to be synthesized which makes it not only expensive to compute but also prohibitive to store on commodity PCs using previous methods. As demonstrated later by both complexity analysis and experiments, our proposed approach significantly reduces the number of voxels needed to be synthesized while preserving the visual quality of the synthesized results, leading to lower storage costs and much more efficient computation.

1.1. Related work

Texture synthesis has been an active research direction in computer graphics for many years. A complete survey of texture synthesis algorithms is beyond the scope of the paper. Please refer to Wei et al. [26] for an excellent recent survey of example based texture synthesis, and Pietroni [20] for a thorough and up-to-date survey of solid texture synthesis. Here we briefly review most relevant research works.

1.1.1. Example based image texture synthesis

Most previous research work in texture synthesis focuses on image textures. Various example based methods have been proposed using pixel-based synthesis [4,27,1,9] or patch-based synthesis [3,13]. Further improvements have been made in recent years, such as iterative texture optimization in [12] for high quality synthesis and parallel, deterministic synthesis [14] which allows windows from an infinite texture image to be synthesized deterministically. This idea is particularly useful since computing and storing solid textures is much more expensive. Structured textures are challenging to fully reproduce. Improvements can be made by using feature matching and alignment based on structural similarity [29] or converting the image to appearance space to capture more essential structures [15].

1.1.2. Image texture synthesis with guidance of direction fields

Direct texture synthesis over surfaces reduces the artifacts of texture mapping. Direction fields are often used to guide the synthesis process, especially for anisotropic textures. Praun et al. [21] texture surfaces by repeatedly pasting patches and using alpha blending to hide the seams. Various planar texture synthesis methods have been extended to surfaces with appropriately constructed neighborhoods [23,28,31,11]. Recently [30] further emphasizes aligning synthesized textures with surface features and promotes feature-to-feature correspondence.

1.1.3. Solid texture synthesis

Early works of solid texture synthesis focus on procedural approaches [18,19], i.e. using rules to simulate solid textures. This requires very little storage to store the rules, but is restricted in expressibility and unintuitive to control. To address such issues, most recent methods use exemplars to guide the synthesis. Early work in [6,7] estimates parametric models from 2D exemplar images. Wei [25] first extends non-parametric 2D texture synthesis algorithms to synthesize solid textures. An improved algorithm is proposed in [10] using both texture optimization [12] and histogram matching [8]. A more efficient GPU-based implementation was proposed in [2] to synthesize and store pixels when necessary, using a deterministic approach. All of the methods discussed so far assume that solid textures are well aligned with world coordinate system, which limits their ability to capture arbitrary geometric structures and design intentions. The idea of vector representation has been introduced recently [24], as a way to improve storage compactness and provide nice features such as resolution independence.

Lapped textures have been extended to synthesize 3D volumetric textures [22], however, they require 3D volumetric exemplars instead of 2D image exemplars as input. Although 3D exemplars can be first synthesized from 2D exemplars, such precomputation leads to unnecessary higher dimension in local neighborhood matching thus slows down the computation. Seams between patches cannot be fully hidden; this becomes more noticeable for structured textures as used in the paper. Owada et al. [17] propose a system for interactive volume painting, which also utilizes user sketches and geometry to guide the volume synthesis process. Their focus however, is on the user interface, while the synthesis method is simple and does not reproduce structured textures.

The work in [16] uses techniques similar to solid texture synthesis for motion synthesis. Due to the different nature of the problem, their work focuses on synthesizing motion vectors based on 3D vector fields and uses a much coarse grid. To the best of our knowledge, this is the first work to synthesize tensor field guided solid textures from 2D exemplars (in a Lazy manner). Novel algorithms are proposed to address the practical difficulties in this more flexible formulation.

The overall algorithm is presented in Section 2 followed by detailed discussions of key technical components: 3D tensor field generation in Section 3, solid texture synthesis w.r.t. tensor fields in Section 4 and efficient deterministic synthesis in Section 5. Experimental results are presented in Section 6, and finally conclusions and future work are given in Section 7.

2. Overview

The overall algorithm pipeline is presented in Fig. 2. We first construct a smooth 3D tensor field both on the surface and in the interior of the model. An intuitive sketching interface is provided to allow users to incrementally draw sketching curves and tensor fields agreeing with such curves in the least-squares sense are computed using harmonic interpolation of quaternions. This process is sufficiently efficient to provide interactive feedback.

Following the well-known algorithm structure [14], our synthesis process is carried out in a multi-resolution 3D pyramid with L levels using a deterministic approach. We start from a coarse grid in the base level which is first initialized. Two operations, namely upsampling and correction, are applied to obtain the synthesis results in the next finer level.

We adapt the concept of 'triple' in [2] and briefly describe some notations. A *coordinate triple* for some voxel refers to a triple of coordinates in three 2D exemplar images I_{xy}, I_{yz} and I_{zx} corresponding to the planes xy, yz, zx where x, y, z agree with local tensor field. Assume N is the template size in the current level, for an arbitrary coordinate triple, three intersecting $N \times N$ neighborhoods are determined. Pairwise intersections of such neighborhoods result in three intersected crossbars. Not all the coordinate triples are proper. *Candidate triples* refer to such "compatible" coordinate triples with relatively consistent color along the intersected crossbars (small matching errors)

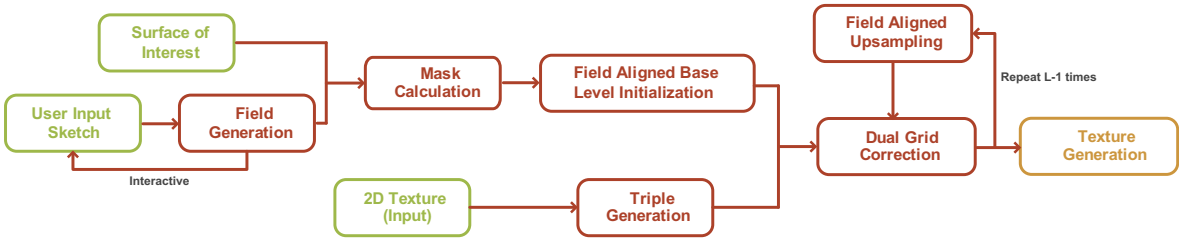

Fig. 2. Algorithm pipeline.

and spatially consistent with at least one neighboring pixel. For every pixel p in the input 2D exemplar, at most N_T best matched candidate triples are precomputed. $N_T = 12$ is used for all the examples in the paper. For each pixel p in the input 2D exemplar E, the first candidate triple of p defines 3 interleaved $N \times N$ neighborhoods. One of the intersected stick with length N is orthogonal with E, which can be used to thicken E according to the stick to form a *candidate slab*.

In the initialization stage, an initial coordinate triple is assigned to each coarse level voxel. To compute from a coarser level of the pyramid to a finer level, upsampling interpolates for each voxel the coordinate triple in the finer level from that of the coarser level. Correction is the key step of the synthesis algorithm. The coordinate triple (and color) at a voxel is updated, according to its local neighborhood matched against the given exemplars. For efficient implementation, the local coherence idea can be utilized to obtain a relatively small set of possible candidates, using the candidate triples from neighboring voxels. To further improve the results, multiple correction passes can be performed. A subpass correction scheme can also be employed, which separates the whole volume into blocks of s^3 size, where s is the size in all the x, y and z directions. s^3 subpasses are performed in each correction pass. In each subpass, $\frac{1}{s^3}$ of overall voxels with the same relative location in each block are updated. Later subpasses can utilize newly corrected voxels from previous subpasses, accelerating the convergence. These techniques were originally proposed in the settings of 2D texture synthesis [14] and directly generalized for solid texture synthesis [2]. The correction output is locally deterministic; only those voxels sufficiently close to the one being considered will be relevant, making the algorithm highly parallel. This allows synthesizing voxels on demand while keeping consistency. In a departure from previous techniques, we employ new strategies respecting the local tensor field in the major steps of the synthesis process.

Correction is the most important step in the synthesis process. High quality results can be obtained by performing multiple correction passes. Unlike the image texture synthesis, a thin layer close to some given surface is often requested for a solid texture to be synthesized. Therefore, more correction passes or subpass scheme requires much more voxels to be synthesized. We develop a new general representation called *history windows*, by which different correction schemes can be designed to better balance the competing demands of computational complexity and output quality. Based on the general representation, we design a *dual-grid correction* scheme, which effectively restricts the propagation of the dependent voxel set using "barriers" and ensures synthesis quality using dual arrangements of such barriers. A mask of dependent voxels based on the correction scheme is computed and solid textures are synthesized only within the mask.

3. Smooth 3D tensor field generation

Our method allows synthesized solid textures to be guided by a sparse set of user sketches. To control the synthesis process in local regions, a smooth 3D tensor field is first computed. The rotation component of the tensor field at each point v contains three orthonormal vectors, forming a right-hand local frame. Taking the trivial global frame as a reference, the local tensor field at v can be denoted by the rotation from the global frame to the local frame. The rotation can be well represented by a unit quaternion $\mathbf{q}(v)$. Unit quaternion $\mathbf{q}(w,x,y,z)$ has four components, where $w = \cos(\frac{\alpha}{2})$. $\mathbf{q}$ represents the counterclockwise rotation by angle α around the axis (x,y,z). Quaternion interpolation has also been used in [32] for surface deformation propagation.

We propose to use a sketch-based approach to interactively design a smooth 3D tensor field for a given model. The x, y and z directions of the local tensor field determines the three sectional planes in space with consistent textures as exemplar images.

In our interface, users are allowed to incrementally draw a few guiding curves on the surface or in the interior of the volume. Local frames on such curves are first determined. The 3D tensor field in the volume is then obtained by interpolating the specified local frames in the least-squares sense. This is very efficient since only a few sparse linear systems need to be solved, allowing interactive feedback. An example is shown in Fig. 3. A smooth tensor field is first computed based on a few sketch curves rendered in yellow (a). The tensor field is then interactively updated when new curves are drawn (rendered in red) (b). The field is used in later synthesis stages to produce field aligned solid textures (c and d).

For curves drawn on the surface, a local frame (without orientation) can be obtained from the tangent direction, outward normal and their cross product. If the curve is drawn within the volume, a reference sectional surface (e.g. a plane) is first specified and the curve is assumed to lie on the reference surface whose normal is served instead of the bounding surface normal. Various examples

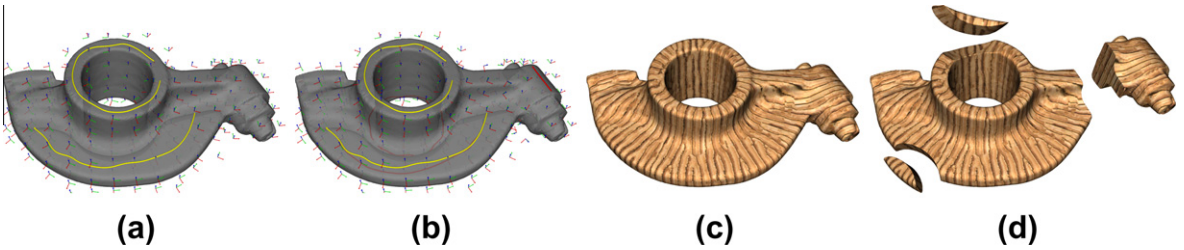

Fig. 3. Sketch guided 3D tensor field generation. Smooth tensor fields are interactively obtained while users adding sketches (a and b). Solid textures are consistently produced w.r.t. the field on the bounding surface (c) or in the interior on demand (d).

are shown in Fig. 12 where user sketches and corresponding smooth tensor fields are presented. Since texture exemplars may be different in different section planes, users are also allowed to assign labels x, y and z to different principal directions of the tensor field at one point of each sketch curve and this assignment is propagated to the whole curve.

When a new curve is drawn, the tensor field is adjusted to take into account its effect. Based on the sketching order, all the curves form a sequence $C=\{C_1, C_2, \ldots, C_n\}$. As described above, when C_i is sketched, we first estimate the local frames along the curve (represented as unit quaternions for efficiency). To smoothly interpolate the tensor field, we interpolate the quaternions by solving the following harmonic equations

$$\Delta\mathbf{q}(v_i) = \sum_{v_j \in N(v_i)} \omega(i,j)(\mathbf{q}(v_i) - \mathbf{q}(v_j)) = 0. \quad (1)$$

Δ is the Laplacian operator defined on the volumetric grid. $N(v_i)$ is the set of voxels in the 1-ring neighborhood of a voxel v_i. The weight $\omega(i,j)$ for voxel v_j and v_i can be chosen simply as $\frac{1}{d_i}$, where $d_i = \|N(v_i)\|$ is the degree of vertex v_i. In practice, we solve Eq. (1) in the least-squares sense, taking the guiding curves as soft constraints, i.e. minimizing the following quadratic energy

$$\|\mathbf{LQ}\|^2 + \sum_{v \in C'} w^2 \|\mathbf{q}(v) - \bar{\mathbf{q}}(v)\|^2, \quad (2)$$

where $\mathbf{Q}$ denotes the vector composed of the quaternions of all voxels, $\mathbf{L}$ is the Laplacian matrix, constituting connectivity and coefficients derived from Eq. (1). C' includes voxels on all the sketched curves. $\mathbf{q} \in \mathbf{Q}$ is the quaternion at a specific voxel and $\bar{q}(v)$ is the local frame derived from sketches. w is the weight that balances the relative importance of smoothness and boundary constraints. $w = 100$ works well in practice and is used for all the examples in the paper.

If the interpolated $\mathbf{q}$ is non-zero, we normalize it to have unit length, thus corresponds to a valid rotation. In case this becomes zero, singularities may appear. Such occurrence is rare in practice and does not significantly affect the synthesis results, as demonstrated with the results in Fig. 10. An arbitrary neighboring direction is simply chosen to replace such singularities.

In practice, a few initial sketch curves are often sufficient to establish a smooth tensor field globally aligned with user intentions. Users are allowed to specify a new curve C_i for local refinement. Conceptually the current tensor field will be rotated a bit such that one direction $\mathbf{q}_{C_i}$ out of the six possibilities in the tensor field namely $\bar{\mathbf{q}} = \{\mathbf{q}_x, -\mathbf{q}_x, \mathbf{q}_y, -\mathbf{q}_y, \mathbf{q}_z, -\mathbf{q}_z\}$ which leads to the minimal alignment error with the tangent vector t along the curve C_i will be identified and aligned with t using appropriate rotation. Local frames obtained through rotation form part of the boundary conditions as before. Unit quaternions are efficient to represent tensor fields, but have an ambiguity that $\mathbf{q}$ is the same as $-\mathbf{q}$. We use a simple and effective solution to this issue. For the first point on the first curve C_1, we arbitrarily choose $\mathbf{q}$, e.g. with $w \geqslant 0$. We can determine $\mathbf{q}$ on the curve C_1 based on continuity along the curve. For points on later curves C_i, the sign of $\mathbf{q}$ can be chosen based on the current tensor field $\bar{\mathbf{q}}$ such that $\|\mathbf{q} - \bar{\mathbf{q}}\|$ is minimized.

Most previous efforts on field design focus on vector fields (typically over manifold surfaces). Takayama et al. [22] proposed a sketch-based approach which first optimizes Laplacian energies independently for each coordinate component and orthogonalizes such directions only when necessary. Since our synthesis algorithm always requires orthogonal tensor fields, quaternion interpolation is more efficient as only four independent linear systems are involved (rather than nine) and no postprocessing is needed to ensure orthogonality. Our interface is also different from the depth field based approach and probably more suitable for our scenario since the symmetric assumption of textures does not exist in general in our cases. To control solid texture synthesis, rotation fields are often sufficient. Our method can be extended to use more generalized tensor fields with varied scalings in space. This can be implemented by specifying scalings at a few sparse 3D positions by the user, and similarly estimating the scaling at each point of the 3D space using harmonic interpolation.

4. Solid texture synthesis

To synthesize solid textures guided by tensor fields, we extend the deterministic synthesis approach to deal with tensor fields in all the major steps of the algorithm. To measure the similarity between two pixels, we use L_1 distance between a vector comprising the red, green, and blue[1] components of the pixels. For exemplar image

[1] For interpretation of color in all figures, the reader is referred to the web version of this article.

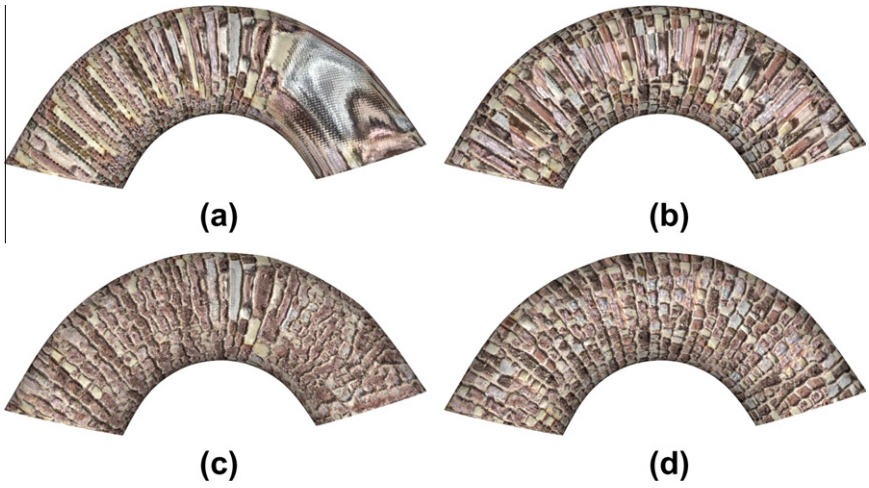

Fig. 4. Distortions of the initial solid texture (a) can be greatly reduced by subpart initialization (b). This also leads to improved synthesis result from (c) to (d).

textures with high regularity (e.g. bricks), a further feature component is introduced, using high intensity to indicate feature curves. Such feature curves in the input image may be obtained automatically, or specified by the user [29].

4.1. Initialization

The synthesis process starts from a low level L_{start} of the 3D volume pyramid. The initialization of L_{start} is very important, since a good initialization can better preserve the structure of the input image and improve convergence of the synthesis process.

Solid texture synthesis amounts to computing a triple coordinate for each grid point. To obtain an initial guess, we use the candidate slabs. First we tile candidate slabs to cover the 3D space to get a volume texture V, so each voxel $v(i,j,k)$ (pixel coordinate) in V corresponds to a coordinate triple denoted by $T(v)$. Since we now have a tensor field, for v in L_{start}, with local tensor field $\mathbf{q}_v$, the warped coordinate triple is $T([\mathbf{q}_v^{-1} v])$, where $[\cdot]$ means the closest integer point. If the derivative of the local tensor field is significantly large or the voxel v is far from the origin, the initial solid texture obtained in this manner may be highly distorted. We can divide L_{start} to a few $K \times K \times K$ subparts (here K refers to pixel resolution, and $K = 64$ is used for our experiments) and initialize each part respectively. Although this may cause discontinuities between different parts, as an initial value, it does provide a good approximation to the synthesized volume and leads to improved synthesis results, as shown in Fig. 4.

4.2. Upsampling

Due to the use of the local tensor field, simply upsampling the coordinates from the parent level is not suitable. As shown in Eq. (3), $v(i,j,k)$ is a point in the synthesized level L_l, and its parent is $v_p\left(\left\lfloor\frac{i}{2}\right\rfloor, \left\lfloor\frac{j}{2}\right\rfloor, \left\lfloor\frac{k}{2}\right\rfloor\right)$ with texture coordinate triple $T(v_p)$ and rotation $\mathbf{q}_{v_p}$, then the coordinate triple for the child v in level L_l is given as

$$T(v)^l = T(v_p)^{l-1} + \mathbf{q}_{v_p}^{-1} t_v, \tag{3}$$

where t_v equals to the offset between v and its "upsampled" parent, which is $h_l(i \bmod 2, j \bmod 2, k \bmod 2)$, and the add operation between a triple $T(T_{xy}, T_{yz}, T_{zx})$ and a vector in 3D $d(d_x, d_y, d_z)$ is defined as

$$T + d = (T_{xy} + (d_x, d_y), T_{yz} + (d_y, d_z), T_{zx} + (d_z, d_x)), \tag{4}$$

where T_{xy}, T_{yz} and T_{zx} refer to the three 2D coordinates of the coordinate triple T respectively (see Fig. 5a).

4.2.1. Gaussian image stack

Instead of using a Gaussian image pyramid, we employ the Gaussian image stack [14] in our synthesis process. This uses Gaussian filtering without downsampling to replace the traditional Gaussian pyramid. This essentially allows "fractional sampling" in a Gaussian image pyramid, improving the sampling accuracy. Let h_l denote the regular spacing of exemplar images in level l ($l = 1, 2, \ldots, L$), $h_l = 2^{L-l}$.

4.3. Correction

Correction replaces the coordinate triple associated with each voxel by one best matching its neighborhood. For a voxel v, we gather the voxels in 3 $N \times N$ neighborhoods, and use these neighborhoods to search the best matching candidate. There are two main differences from the previous method:

(1) The 3 $N \times N$ neighborhoods should be aligned with the local tensor field, not the trivial $N \times N$ neighborhoods in the global coordinate. For a voxel v with local tensor field $\mathbf{q}_v$, the 3 $N \times N$ neighborhoods are:

$$N_{xy} = \{v + \mathbf{q}_v \cdot (x, y, 0) | -N_h \leqslant x, y \leqslant N_h\}$$
$$N_{yz} = \{v + \mathbf{q}_v \cdot (0, y, z) | -N_h \leqslant y, z \leqslant N_h\}$$
$$N_{zx} = \{v + \mathbf{q}_v \cdot (x, 0, z) | -N_h \leqslant x, z \leqslant N_h\}$$

where $N_h = \frac{N-1}{2}$. We use trilinear interpolation in coarse levels and nearest neighbor in finer levels to approximate

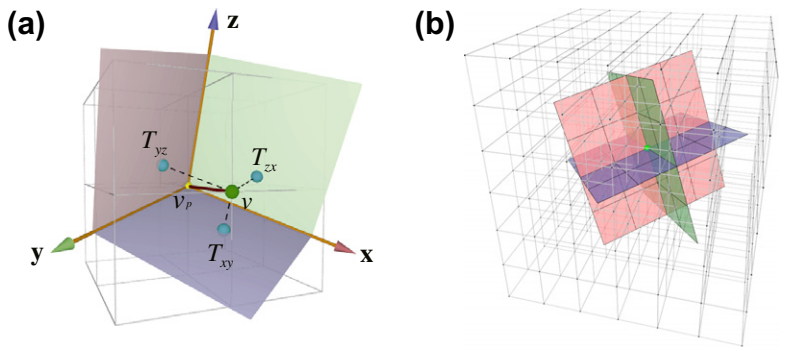

Fig. 5. Illustrations for (a) upsampling w.r.t. local tensor field and (b) the local neighborhood.

the color of voxels in the three neighborhoods. This better balances performance and quality, since the quality is more affected with coarse level synthesis while the computational cost is mainly determined by the fine level synthesis. The neighborhood w.r.t. local fields is shown in Fig. 5b. For fine levels, the tensor field at the center voxel is used to sample the local neighborhood. For coarse levels, a numerical integration along the directions is used instead to improve the sampling accuracy, similar in spirit to the surface case in [11].

(2) In the searching phase, we first gather a set of candidate triples for each voxel. For some voxel v, considering the local tensor field $\mathbf{q}_v$, we denote $P_{xy}(d)$, $P_{yz}(d)$, $P_{zx}(d)$ the projection of a vector d on planes xy, yz, zx of the local frame $\mathbf{q}_v$, respectively. Our candidate set $\mathcal{S}(v)$ is different from [2] and is composed of

$$\mathcal{S}(v) = \mathcal{S}_{xy}(v) \cup \mathcal{S}_{yz}(v) \cup \mathcal{S}_{xz}(v)$$
$$\mathcal{S}_{xy}(v) = \{{}^{xy}C^k([T_{xy}(v+d) - h_l P_{xy}(d)]) | d = (d_x, d_y, d_z)\},$$
$$\mathcal{S}_{yz}(v) = \{{}^{yz}C^k([T_{yz}(v+d) - h_l P_{yz}(d)]) | d = (d_x, d_y, d_z)\},$$
$$\mathcal{S}_{zx}(v) = \{{}^{zx}C^k([T_{zx}(v+d) - h_l P_{zx}(d)]) | d = (d_x, d_y, d_z)\},$$
$$d_x, d_y, d_z \in \{-1, 0, 1\}, k = 1, 2, \ldots$$

where k iterates over precomputed candidate triples at each exemplar pixel, ${}^{xy}C^k$, ${}^{yz}C^k$, ${}^{zx}C^k$ represent the kth candidate triple for a pixel in I_{xy}, I_{yz}, I_{zx}, $[\cdot]$ refers to the closest pixel, and T_{xy}, T_{yz}, T_{zx} are three 2D coordinates of the current coordinate triple at a voxel, as defined in Section 4.2. In our implementation, we have found a reduced space with d set to $(d_x, d_y, 0)$, $(0, d_y, d_z)$, $(d_x, 0, d_z)$ for S_{xy}, S_{yz} and S_{zx} respectively is sufficient to produce good results. All the neighboring pixels in three image planes w.r.t. the local tensor field are collected. We accelerate the searching phase using a PCA projection to reduce the data to about 20 dimensions per plane, without significant degradation of the output quality.

4.3.1. Correction subpass

We similarly use subpass scheme (with $s^3 = 8$) for better convergence. However, unlike in 2D case where only a small amount of extra cost is needed, on-demand synthesis for solid textures is significantly different since some particular surfaces are often requested, which requires synthesizing a thin layer around the surface. The thickness of the layer is approximately proportional to the number of subpasses, so is the computation overhead. Our dual-grid correction scheme reduces the dependency chain further than what was possible with the standard correction scheme used in [2]. This issue and our solution will be addressed in detail in the next section.

5. Spatial determinism

Since we synthesize solid textures on demand, spatial determinism is an important requirement. Determinism can be achieved by synthesizing all the voxels in each pyramid level that will influence the correction passes of the desired part. This is not a problem for image texture synthesis; for solid textures, however, the total number of synthesized voxels grows quickly with increased parameters. For level l in a total of L levels, relevant parameters include the size of local neighborhood N_l, the number of correction passes P_l and the number of subpasses s_l^3. We define the impact distance of level l $\mathbb{L}_l$ as the largest distance of dependent voxels away from the voxel in question, which satisfies $\mathbb{L}_l = \frac{P_l N_l s_l^3}{2}$ and the overall impact distance $\mathbb{L} = \sum \mathbb{L}_l = \sum \frac{P_l N_l s_l^3}{2}$. Although, in practice the interested part of the solid is usually some 2D surfaces the total number still grows quickly with increased parameters. This does restrict us from using larger parameters to get high quality results.

5.1. Observation and assumption

Using larger values of L, N_l, P_l and s_l^3 can greatly improve the quality, and the resulting texture can better preserve global statistics as well as local similarity and continuity. Notice that the overall appearance of the synthesized volume is mainly determined by the coarse level of the volume pyramid, while the correction passes in finer grid can only improve the local continuity. So it is reasonable to regard that the coordinate of a voxel in finer levels is influenced only by a limited size of neighborhood, even if more times of correction passes are performed. In fact, it

is not necessary to keep such a large dependent set as in [2].

Based on this assumption, to fully control the size of the dependent set, we propose a novel representation called *history windows*. History windows describe the local dependency for on-demand synthesis, and is used in the correction passes. Previously used correction schemes, including the correction pass and subpass schemes can be treated as special cases of the new representation. This also allows further flexible schemes to be designed, balancing between the computational cost and the quality of results, while keeping the desirable spatial determinism property. We further propose a novel dual grid correction scheme using this general representation which is very efficient and achieves high quality synthesis for on-demand solid texture synthesis.

5.2. History window

In the correction pass, as mentioned in Section 4, we use 3 $N \times N$ neighborhoods determined with the local tensor field to search for a best matching candidate triple. We associate every voxel with a history window which specifies how to choose the color of voxels in the local neighborhood. For simplicity, we will explain the history window in 2D case, while the extension to 3D is trivial.

5.2.1. 2D History window

We first introduce some notations. For a pixel p in the synthesized pyramid, we use subscript to represent the pyramid level l. Suppose we perform P_l correction passes at every pyramid level, each pixel p has a sequence of $P_l + 1$ versions. Thus p_l^i refers to the pixel of p at level l after the ith correction pass, p_l^0 means the upsampled pixel from level $l-1$ without any correction. We call i the version of p_l^i. In the ith correction pass for 2D texture synthesis, we gather an $N \times N$ neighborhood for every pixel p^{i-1} to find the most well matched neighborhood, then get the updated pixel p^i, that is, pixels of version $i-1$ in p^{i-1}'s neighborhood are used in the correction pass. We use an $N \times N$ window to record the version of pixels in p^{i-1}'s neighborhood, which is used in the ith correction pass. This $N \times N$ window is called the *history window* of p^{i-1} and is denoted as $H(p^{i-1})$. Simple correction without subpass implies that each element in $H(p^{i-1})$ has the value $i-1$; such H is also used in the first subpass in the $s^2 = 4$ subpasses. The history windows of the 2nd, 3rd and 4th subpasses are illustrated in Fig. 6a–c.

5.2.2. History window with 3D tensor field

In the 3D case, we use a $D \times D \times D$ 3D window H^3 to record the version of voxels in the $D \times D \times D$ neighborhood. As described in Section 4, the 3 $N \times N$ neighborhoods are selected to align with the 3D tensor field, computed using the nearest neighbor of the voxels in the pyramid volume. The version of voxels used in the interpolation is determined by H^3. D should be chosen larger than N, to deal with local rotations. In practice we choose $D = \lceil N \times \sqrt{2} \rceil$. This guarantees that all the relevant neighborhood voxels are covered in the history window.

5.3. Dual-grid correction

The history window is a general representation which allows construction of arbitrarily complicated update schemes. According to the observation in Section 5.1, a voxel is highly related to voxels in its local neighborhood, so we can safely set some barriers to limit the growth of the dependent set. Again for simplicity, the basic idea is first described in 2D case; all the techniques can be easily extended to 3D space.

For some level l of the pyramid, we associate each pixel p with a history window $H(p)$. If a pixel q is in p's $N \times N$ neighborhood, the version for q in $H(p)$ is denoted as $H(p, q)$. The size of the dependent set is proportional to Ps^3 thus grows quickly. By proper assignment of H_p, we can limit the size of the dependent set under a constant value independent of P and s. In our implementation, H_p is selected as follows: we divide the whole image into several $M \times M$ subparts, for every pixel q in p's $N \times N$ neighborhood, if p and q belong to the same subpart, then $H(p,q)$ equals to the latest version of q, otherwise $H(p,q)$ equals to 0. We call this scheme (primary) grid correction, as illustrated in Fig. 6d and Fig. 7 (left).

This strategy can produce high quality results for voxels near the center of each subpart, but there may exist some artifacts (discontinuity) near the subpart boundaries. To handle this problem, we introduce a dual-grid correction scheme, which uses grid of the same dimension but replaces each center of box with a vertex and each vertex with a box centered at the vertex, as illustrated in Fig. 7 (right). For each level of synthesis with N_l passes, we first perform $\lceil \frac{N_l}{2} \rceil$ (primary) grid correction steps followed by $\lfloor \frac{N_l}{2} \rfloor$ passes of dual-grid correction. Note that the initial version the dual-grid correction used is the final version by the grid correction.

5.3.1. Subpass for grid correction

Using history windows, we can easily perform correction subpasses based on the current dual-grid correction scheme. To simplify the description, we first introduce intersection operator for history windows. Suppose $H_1(p)$ and $H_2(p)$ are two history windows with the same size, their intersection denoted by $H = H_1 \cap H_2$ can be defined as $H(p,q) = \min(H_1(p,q), H_2(p,q))$. Thus the version value in H is the minimum of the version values at the corresponding position. It is clear that the size of dependent set using H is no larger than that of using H_1 or H_2. Suppose the history windows used in the two steps are H_g and H_{dg} respectively. The history windows used for traditional subpasses are denoted as H_s, and are illustrated in Fig. 6 for 2D case with $s^2 = 4$. We can then use $H_g \cap H_s$ and $H_{dg} \cap H_s$ to perform subpass in dual grid correction.

5.3.2. Complexity analysis

We use the impact distance described previously to estimate the effects of dual-grid correction. Assume that similar notations are used, except for that we use P_l and P'_l to represent the number of grid and dual-grid corrections, s_l and s'_l to represent the number of grid and dual grid subpasses. The impact distance for grid correction of

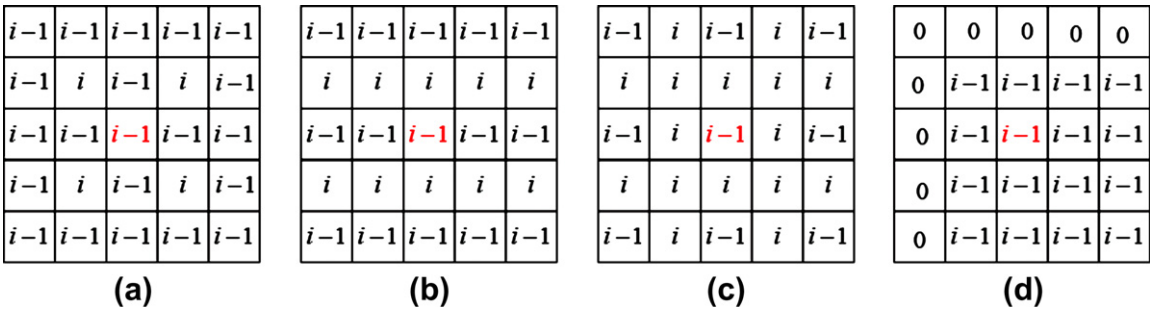

Fig. 6. History windows for different correction schemes. (a–c) history windows used in 2nd, 3rd and 4th subpasses for $s^2 = 4$; (d) a constrained scheme using version 0 to stop the growth of the dependent set.

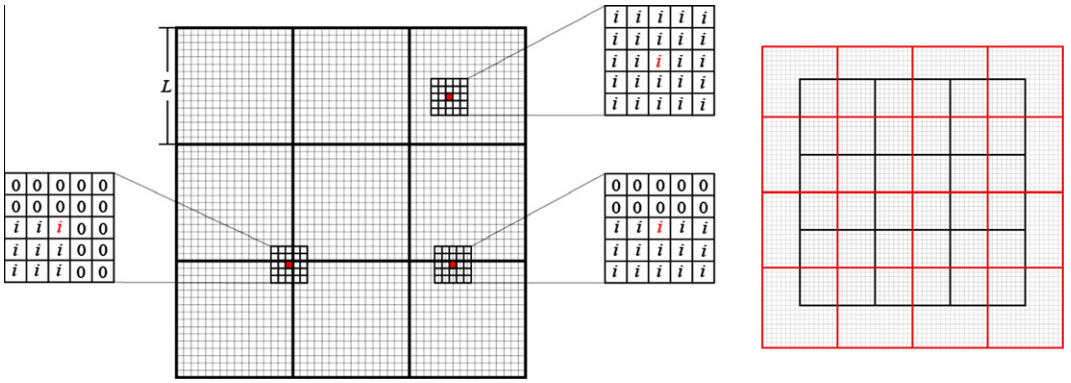

Fig. 7. Left: grid correction with related history windows. Right: dual grid arrangement in red following primary grid in black for improved continuity.

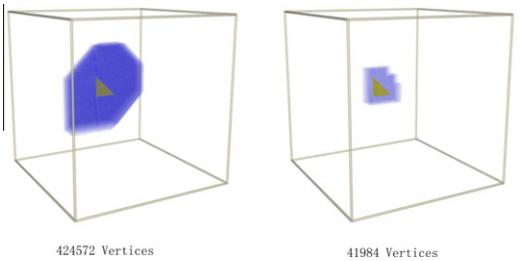

Fig. 8. The dependent set (blue voxels) for synthesizing a triangular object, $N = 5, P = 3, s^3 = 8$, normal correction (left), and dual-grid correction (right).

level l satisfies $\mathbb{L}_{l,1} = \min\left(\frac{P_l N_l s_l^3}{2}, M_l + \frac{N_l}{2}\right)$, and that for dual-grid correction has $\mathbb{L}_{l,2} = \min\left(\frac{P'_l N_l s_l'^3}{2}, M_l + \frac{N_l}{2}\right)$. The overall impact distance $\mathbb{L} = \sum(\mathbb{L}_{l,1} + \mathbb{L}_{l,2})$. When M is significantly large, the complexity is the same. If an appropriate M is chosen, the complexity can be well bounded by the M and N, significantly reducing the computational and storage cost. We found that $M = 10 - 15$ shows good balance between quality and performance.

6. Results

We demonstrate solid texture synthesis results for a wide variety of models using our algorithm. Like many algorithms for texture synthesis, some parameters need to be chosen for the system to work, however, a fixed set of parameters works reasonably well in most majority of experiments. 2D exemplars used in these examples are given in Fig. 16. The masks used to enhance feature matching for textures in the second row are given in the third row. In most examples, we synthesize four levels with the object embedded in a volume whose edges are longer than those of the bounding box to reduce boundary effects(the longest edge of the volume is about 20–30% longer than that of the bounding box) and discretized to about 800–1024 in its longest dimension. A local neighborhood size $N = 7$ is used for the coarsest level and $N = 9$ for other levels. For the coarsest level, 3–5 correction passes are used without barriers for high quality results with acceptable cost due to the coarse grid. For other levels, dual-grid correction scheme with 2 (primary and dual) passes of corrections are usually performed at each level, and $s^3 = 8$ subpasses are usually applied to improve the synthesis quality. The experiments were carried out on a desktop computer with 2 × Quad Core 2.27 GHz CPU. Since the tensor field is usually smooth, we found a grid with the shortest dimension discretized to 20–30 is sufficient for our experiments. And in coarser levels, the field is further smoothed and downsampled to reduce discontinuity. The computation of the field takes less than 1 s each time when a new curve is drawn. This allows tensor fields to be modified interactively by designers. The tensor field only takes a small amount of memory to store. In the synthesis process, a

simple trilinear interpolation is used to obtain the tensor field at each voxel from eight corner points of the coarse grid covering the voxel. This is simply applied to each component of the quaternion followed by a normalization to obtain unit quaternion. The synthesis process takes about 10–25 min. We cache synthesized solids for the two coarser levels as well as those generated by the model surface. Benefitting from the use of dual-grid correction, the synthesis process for a new cut is usually 10–20 s, allowing interactive operations to be performed on the synthesized solids. These examples take longer time than the performance reported in [2] because the following reasons. Our exemplar textures are more structural and the human visual system tends to be more sensitive to well structured images. To capture such structures, we use larger neighborhood size and more correction passes. Fig. 9 shows the results with $N = 5$ and other parameters unchanged. Significantly degraded results are produced for either the trivial or given tensor fields. The quality becomes even poorer when non-trivial tensor fields are applied since the initialization will be much worse. Furthermore, the tensor field needs to be accessed and involved in various computations, leading to longer per-voxel synthesis time. Current implementation is CPU-based; since our algorithm is highly parallel, we expect a GPU-based implementation to potentially improve the performance significantly. Similar to previous works, often a single exemplar image is sufficient for synthesis purpose, as we have done for most examples in the paper.

Fig. 8 demonstrates the effectiveness of the grid correction algorithm. Much fewer voxels are dependent when a triangular object is to be synthesized. As analyzed theoretically in Section 5.3, our method is effective in reducing the number of dependent voxels and corrections. A detailed comparison for a surface in Fig. 11 is listed in Table 1. Using dual-grid correction, the dependent voxels are reduced by about 80% and an over 4 times speedup is obtained. Another example is the vase in Fig. 12 listed in Table 2. For the last two levels that need recomputation for different slices, only about $\frac{1}{5}$ voxels or comparisons are needed, indicating a great saving of both time and memory, as the number of voxels is proportional to the memory cost and the number of comparisons is proportional to the correction time. Complicated examples similar to those presented in this paper may easily require more memory than typically available in the computer (such as 4 GB) without dual-grid correction. Although as a deterministic algorithm, it is possible to partition the space and synthesize textures piece by piece, significant amount of duplicated computation would be needed, leading to even slower computation. Although our dual-grid correction significantly reduces the memory and time cost (with the same parameters), similar, visually pleasing results are generally produced. An example is given in Fig. 11 where (e) shows the close-up image of our method while (f) is the result obtained without dual-grid correction, keeping parameters unchanged.

An example of sketch guided solid texture synthesis is shown in Fig. 1. A few sketches are drawn and used to produce a smooth 3D tensor field (a). Three orthonormal directions are rendered in red, green and blue respectively. Solid textures synthesized using a trivial field [10,2] are shown in (b and e). Using the tensor field, results produced with our method follow the overall shapes much better (c and f). Due the deterministic nature, internal structures of the produced solids can be synthesized on demand (d and g). Our method can produce reasonable synthesized results at tensor field singularities, as demonstrated in the wood textured plate in Fig. 10. Another example in Fig. 11 synthesizes zebra textures over a horse model. Our sketch guided result looks much more realistic to mimic a zebra. Even if this may not be an exact reproduce of real zebra, this shows the possibility and effectiveness of controlling the synthesized solids. It can be easily extended to synthesize different solid textures in different portions of the volume.

More solid texture synthesis examples are given in Fig. 12. The first row shows user sketches and corresponding 3D tensor fields for ‘vase’, ‘dancer’ and ‘dinosaur’ models. Synthesized solid textures are shown in the second row and internal structures are given in the third row. In these examples, solid textures often look better if they follow overall shapes or features of the objects. A particular example is shown in the fourth row where different textures are synthesized over solid ‘GM2010’ characters. By drawing simple sketches, the synthesized textures follow and

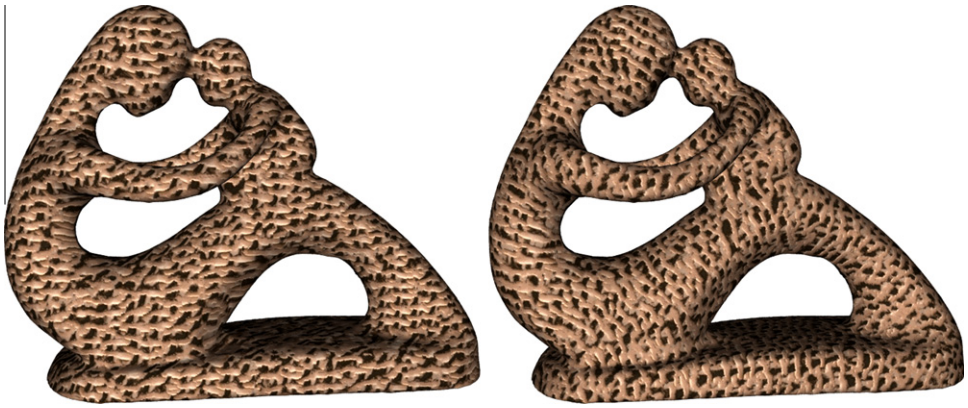

Fig. 9. Synthesized solid textures using a small neighborhood size without (left) or with (right) the given tensor field.

Table 1
Performance comparisons for a surface of the example in Fig. 11 with[1] and without[2] dual-grid correction (voxels including those from direct upsampling).

Level	Voxels[1]	Corrections[1]	Time (s)[1]	Voxels[2]	Corrections[2]	Time (s)[2]
3	1,510,190	1,588,346	17.2	12,011,476	11,334,457	114.8
4	2,484,514	2,171,045	20.4	12,851,888	9,567,335	90.8

Table 2
Performance comparisons for a surface of the vase example in Fig. 12 with[1] and without[2] dual-grid correction (voxels including those from direct upsampling).

Level	Voxels[1]	Comparisons[1]	Voxels[2]	Comparisons[2]
3	2,187,491	658,099,709	11,116,855	3,514,865,117
4	3,445,625	808,436,278	17,730,291	3,601,332,702

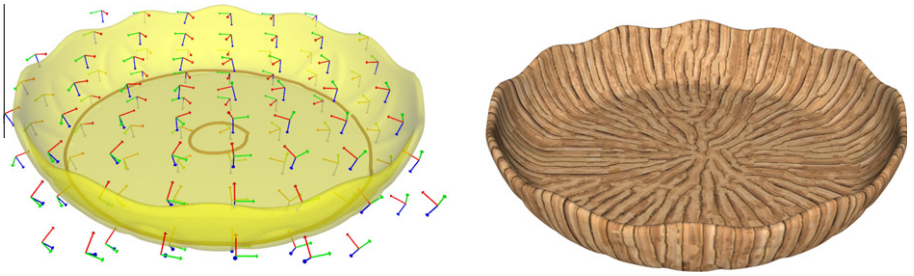

Fig. 10. Synthesized solid textures with singularity.

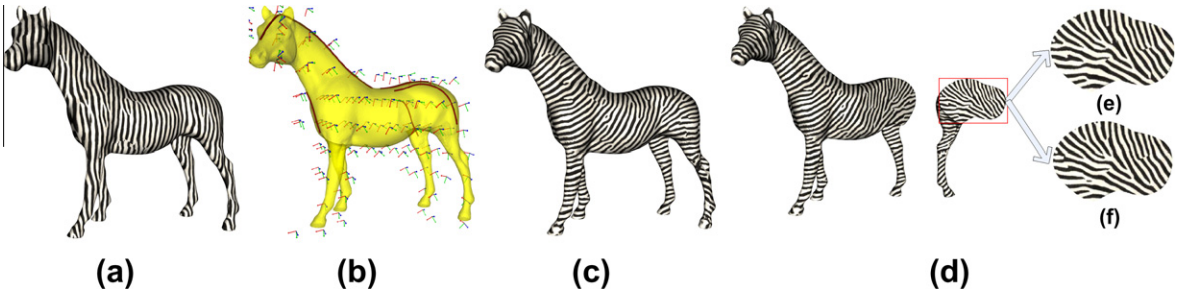

Fig. 11. A horse model with zebra textures. Significantly more realistic result (c) than trivial field (a) can be obtained with a few user sketches (b); internal structures are shown in (d); (e) close-up of the result of our method; (f) the result without dual-grid correction (using the same parameters).

emphasize the shape of each character. In addition to drawing curves on the surface, our sketching interface also allows drawing curves on some reference surface within the volume. An example is shown in Fig. 13 where the overall shape of the bridge is well represented using a single curve on a sectional plane inside the volume and the resulting solid texture reasonably follows the geometry. Solid textures may also follow artistically designed tensor fields. Our sketch based interface makes it very efficient and intuitive to put in the designers' intentions and produce solid textures accordingly (Fig. 14).

Our method can be easily generalized to synthesize different solid textures in different space regions. An example is given in Fig. 15. A few user sketches are drawn to capture the directions along major branches, which are used to derive a smooth tensor field, as shown in (a). We use two different texture exemplars: a tree bark texture is used to synthesize solid textures in some space closer to the boundary surface, and a wood texture is used to synthesize the solid inside. Exemplar images and corresponding feature masks along with the synthesis result are shown in (b), cutting through the volume reveals the internal structures, as shown in (c). This example demonstrates that it is particularly important for solid textures to follow some guiding directions for it to be realistic as real objects (such as trees) often have their natural texture directions over the whole solid. Image exemplars are relatively easy to obtain. By synthesizing different space regions with different exemplars, our method can produce more realistic solids.

6.1. Limitations

Our method has a few limitations. Similar to previous methods [10,2], if the given 2D exemplars are incompatible

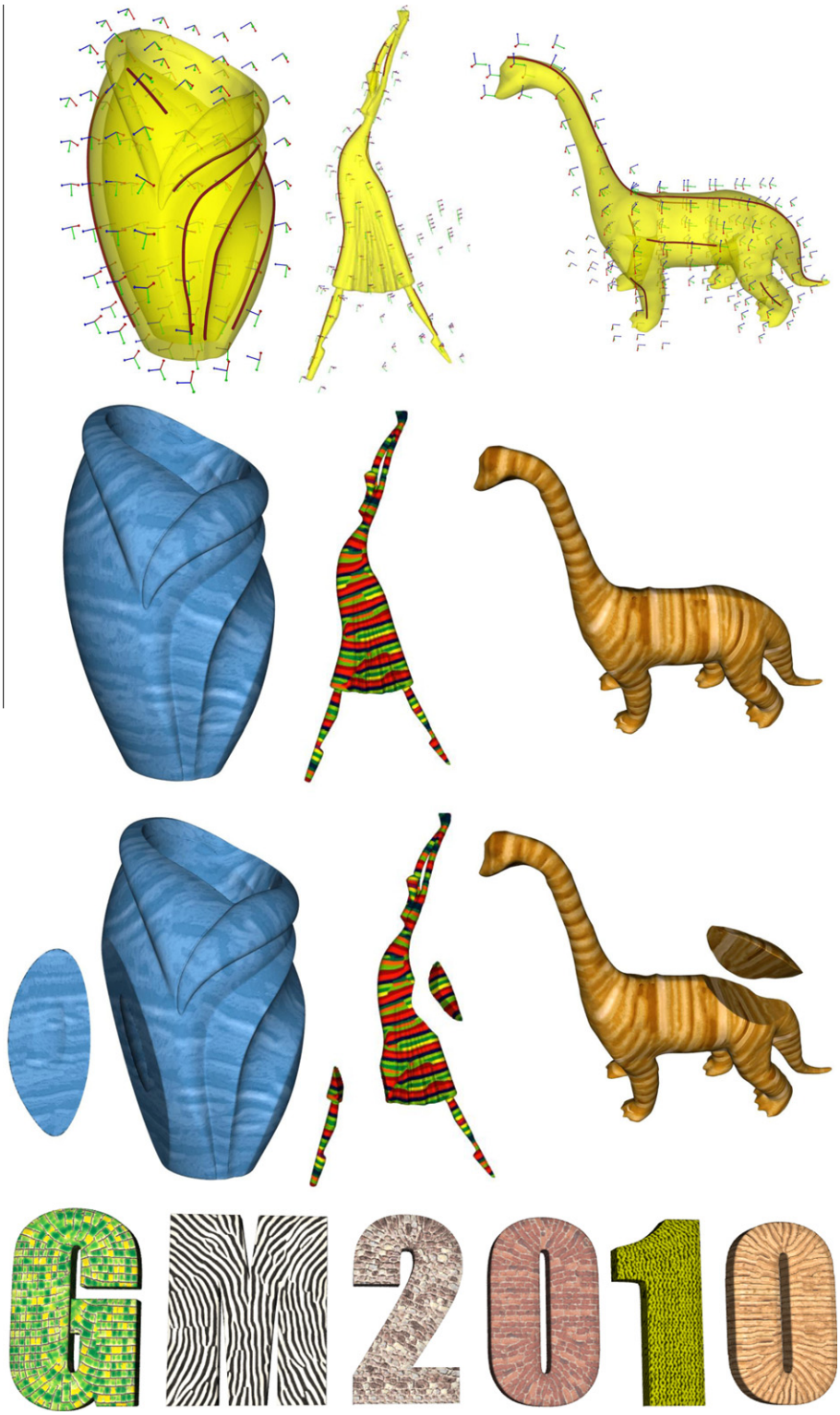

Fig. 12. Sketch-guided solid texture synthesis examples.

in 3D space, synthesis results may not be satisfactory. In most cases, the incoherence can be relieved by only consider matching errors from two sectional planes instead of three, or assigning a small weight to the third plane. However, this is still an open problem in the general sense. Our interpolated tensor field may have some singularities.

Fig. 13. Sketch-guided solid texture synthesis over 'bridge' object using sketches inside the volume.

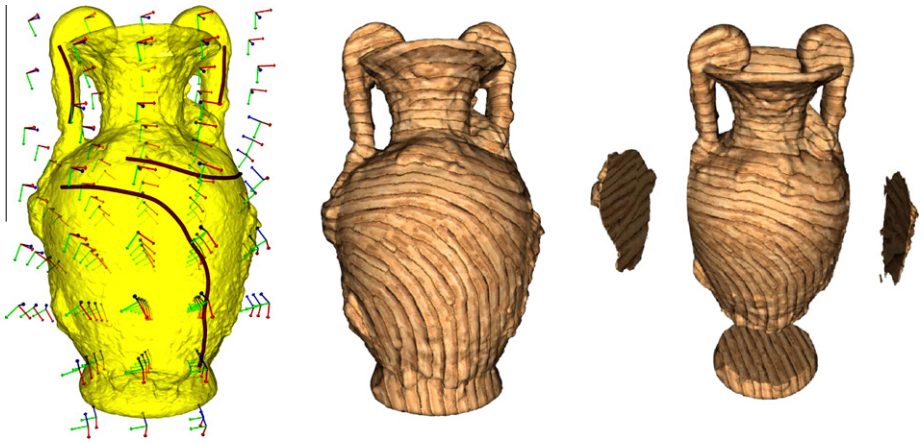

Fig. 14. Solid texture synthesis following user designs.

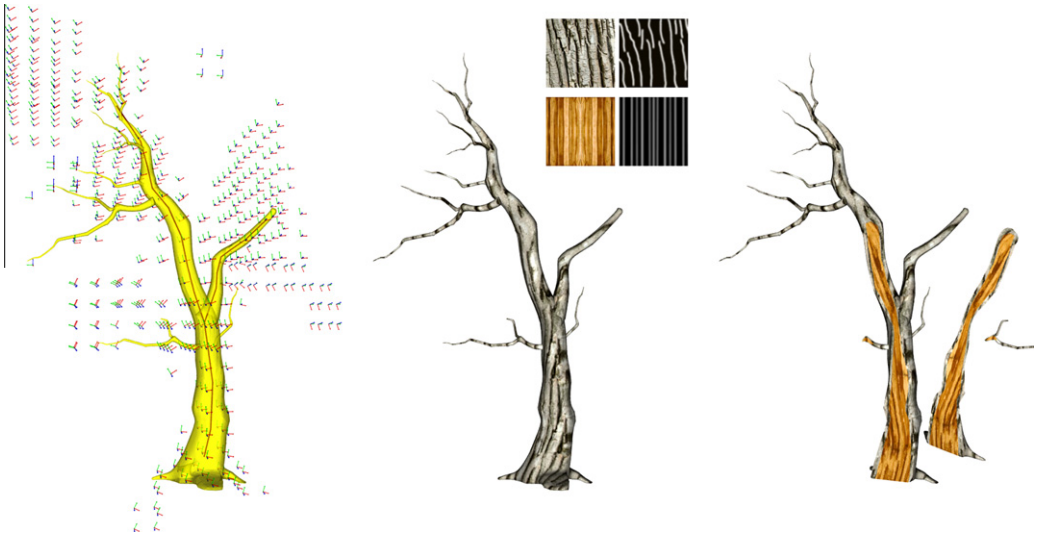

Fig. 15. Solid texture synthesis of a tree with user guidance, using multiple sets of exemplars.

In practice, however, singularities are very rare and do not have significant impact on our synthesized results. It is still open to future research how to fully control the singularities in the obtained tensor field. Our 2D exemplar based solid texture synthesis assumes local isometric mapping from the exemplar image to the corresponding sectional surface (*xy*, *yz* or *zx*). It is not unusual this property conflicts with the given field (which can be arbitrary according

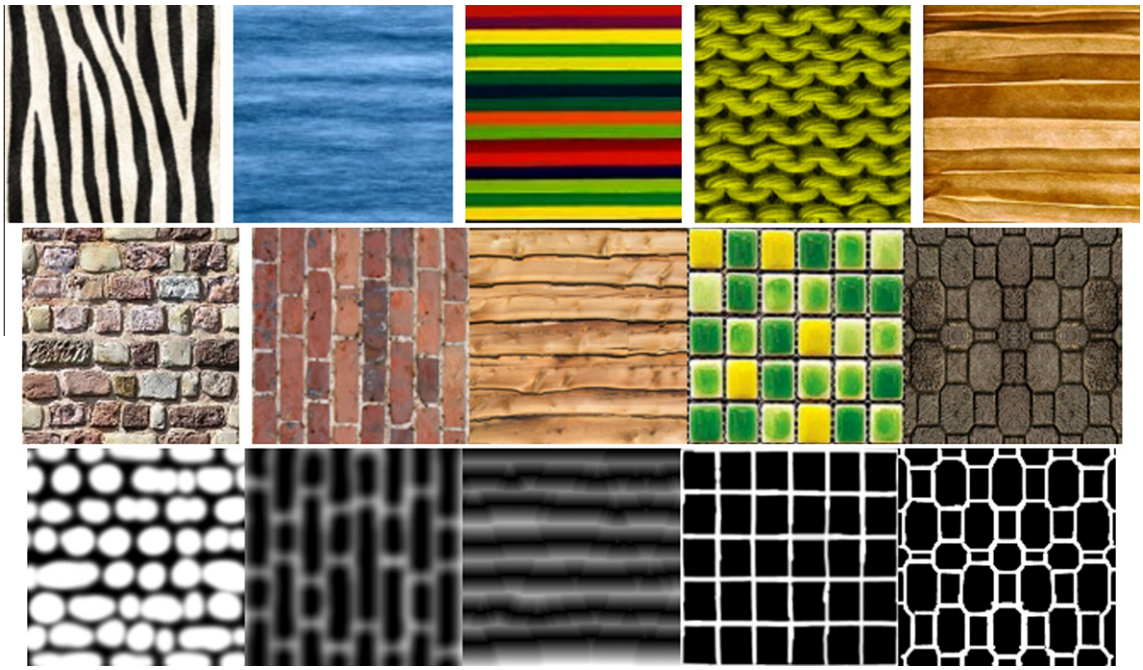

Fig. 16. Texture exemplars used throughout the paper.

to user sketches). In such cases, branching points in the synthesized textures appear for the transition purpose. Although this implies inevitable sacrifice of continuity, as shown in Fig. 14 and other examples, such occurrences of branching points may appear in real world (e.g. wood) as well and still look reasonably realistic in most cases.

7. Conclusion and future work

We have presented a novel algorithm for sketch guided solid texture synthesis from 2D exemplars. Smooth 3D tensor fields are obtained by efficiently solving a few harmonic equations w.r.t. the boundary conditions derived from user sketches. Solid texture synthesis is achieved through an on-demand approach where only relevant voxels are synthesized. Based on a new history window representation, we propose a dual-grid correction scheme, which keeps the quality of results while greatly reduces the dependent set for on-demand synthesis.

As the main part of the algorithm is fully local and parallel, perhaps the most direct extension is an efficient GPU implementation. We also intend to combine the optimization based method (such as [10]) and our correction based method, as the former has better continuity. The history window is a general representation and further correction schemes based on this can be studied in the future.

Acknowledgments

We thank all the anonymous reviewers very much for their helpful and thoughtful comments and suggestions. This work was supported by the National Basic Research Project of China (Project Number 2011CB302205), the Natural Science Foundation of China (Project Number U0735001).

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.gmod.2010.10.006.

References

[1] M. Ashikhmin, Synthesizing natural textures, in: Proceedings of the ACM Symp. Interactive 3D Graphics, 2001, pp. 217–226.
[2] Y. Dong, S. Lefebvre, X. Tong, G. Drettakis, Lazy solid texture synthesis, Comput. Graph. Forum 27 (4) (2008) 1165–1174.
[3] A. Efros, W. Freeman, Image quilting for texture synthesis and transfer, in: Proceedings of the ACM SIGGRAPH, 2001, pp. 341–346.
[4] A. Efros, T. Leung, Texture synthesis by non-parametric sampling, in: Proceedings of the ICCV, 1999, pp. 1033–1038.
[5] M. Fisher, P. Schröder, H. Hoppe, Design of tangent vector fields, ACM Trans. Graph. 26 (3) (2007) 1–9.
[6] D. Ghazanfarpour, J.-M. Dischler, Spectral analysis for automatic 3-D texture generation, Comput. Graph. 19 (3) (1995) 413–422.
[7] D. Ghazanfarpour, J.-M. Dischler, Generation of 3D texture using multiple 2D models analysis, Comput. Graph. Forum 15 (3) (1996) 311–323.
[8] D.J. Heeger, J.R. Bergen, Pyramid-based texture analysis/synthesis, in: Proceedings of the ACM SIGGRAPH, 1995, pp. 229–238, 1995.
[9] A. Hertzmann, C.E. Jacobs, N. Oliver, B. Curless, D.H. Salesin, Image analogies, in: Proceedings of the ACM SIGGRAPH, 2001, pp. 327–340.
[10] J. Kopf, C.-W. Fu, D. Cohen-Or, O. Deussen, D. Lischinski, T.-T. Wong, Solid texture synthesis from 2D exemplars, ACM Trans. Graph. 26 (3) (2007) 1–9.
[11] V. Kwatra, D. Adalsteinsson, T. Kim, N. Kwatra, M. Carlson, M. Lin, Texturing fluids, IEEE Trans. Vis. Comp. Graph. 13 (5) (2007) 939–952.
[12] V. Kwatra, I. Essa, A. Bobick, N. Kwatra, Texture optimization for example-based synthesis, ACM Trans. Graph. 24 (3) (2005) 795–802.
[13] V. Kwatra, A. Schödl, I. Essa, G. Turk, A. Bobick, Graphcut textures: image and video synthesis using graph cuts, ACM Trans. Graph. 22 (3) (2003) 277–286.

[14] S. Lefebvre, H. Hoppe, Parallel controllable texture synthesis, ACM Trans. Graph. 24 (3) (2005) 777–786.
[15] S. Lefebvre, H. Hoppe, Appearance-space texture synthesis, in: Proceedings of the ACM SIGGRAPH, 2006, pp. 541–548.
[16] C. Ma, L.-Y. Wei, B. Guo, K. Zhou, Motion field texture synthesis, ACM Trans. Graph. 28 (5) (2009) 1–9.
[17] S. Owada, T. Harada, P. Holzer, T. Igarashi, volume painter: geometry-guided volume modeling by sketching on the cross-section, in: Proceedings of the Eurographics Symposium on Sketcy-Based Interfaces and Modeling, 2008, pp. 9–16.
[18] D.R. Peachey, Solid texturing of complex surfaces, in: Proceedings of the ACM SIGGRAPH, 1985, pp. 279–286.
[19] K. Perlin, An image synthesizer, in: Proceedings of the ACM SIGGRAPH, 1985, pp. 287–296.
[20] N. Pietroni, P. Cignoni, M.A. Otaduy, R. Scopigno, Solid-texture synthesis: a survey, IEEE Comput. Graph. Appl. 30 (4) (2010) 74–89.
[21] E. Praun, A. Finkelstein, H. Hoppe, Lapped textures, in: Proceedings of the ACM SIGGRAPH, 2000, pp. 465–470.
[22] K. Takayama, M. Okabe, T. Ijiri, T. Igarashi, Lapped solid textures: filling a model with anisotropic textures, ACM Trans. Graph. 27 (3) (2008) 1–9.
[23] G. Turk, Texture synthesis on surfaces, in: Proceedings of the ACM SIGGRAPH, 2001, pp. 347–354.
[24] L. Wang, K. Zhou, Y. Yu, B. Guo, Vector solid textures, in: Proceedings of the ACM SIGGRAPH, 2010, pp. 86:1–8.
[25] L.-Y. Wei, Texture synthesis from multiple sources, in: SIGGRAPH 2003 Sketch, 2003.
[26] L.-Y. Wei, S. Lefebvre, V. Kwatra, G. Turk, State of the art in example-based texture synthesis, in: Eurographics State-of-Art Report, 2009.
[27] L.-Y. Wei, M. Levoy, Fast texture synthesis using tree-structured vector quantization, in: Proceedings of the ACM SIGGRAPH, 2000, pp. 479–488.
[28] L.-Y. Wei, M. Levoy, Texture synthesis over arbitrary manifold surfaces, in: Proceedings of the ACM SIGGRAPH, 2001, pp. 355–360.
[29] Q. Wu, Y. Yu, Feature matching and deformation for texture synthesis, ACM Trans. Graph. 23 (3) (2004) 364–367.
[30] K. Xu, D. Cohne-Or, T. Ju, L. Liu, H. Zhang, S. Zhou, Y. Xiong, Feature-aligned shape texturing, ACM Trans. Graph. 28 (5) (2009) 1–7.
[31] L. Ying, A. Hertzmann, H. Biermann, D. Zorin, Texture and shape synthesis on surfaces, in: Proceedings of the Eurographics Workshop on Rendering, 2001, pp. 301–312.
[32] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, H.-Y. Shum, Mesh editing with poisson-based gradient field manipulation, in: Proceedings of the ACM SIGGRAPH, 2004, pp. 644–651.