Efficient Synthesis of Gradient Solid Textures

Guo-Xin Zhang [a] Yu-Kun Lai [b] Shi-Min Hu [a]

[a] *TNList, Department of Computer Science and Technology, Tsinghua University, China*

[b] *School of Computer Science and Informatics, Cardiff University, UK*

Abstract

Solid textures require large storage and are computationally expensive to synthesize. In this paper, we propose a novel solid representation called *gradient solids* to compactly represent solid textures, including a tricubic interpolation scheme of colors and gradients for smooth variation and a region-based approach for representing sharp boundaries. We further propose a novel approach to *directly* synthesize gradient solid textures from exemplars. Compared to existing methods, our approach avoids the expensive step of synthesizing the complete solid textures at voxel level and produces optimized solid textures using our representation. This avoids significant amount of unnecessary computation and storage involved in the voxel-level synthesis while producing solid textures with comparable quality to the state of the art. The algorithm is much faster than existing approaches for solid texture synthesis and makes it feasible to synthesize high-resolution solid textures in full. We also propose a novel application — instant editing propagation on full solids.

Key words: solid textures, synthesis, vector representation, gradient, vectorization, distance fields, tricubic interpolation, editing propagation, real-time rendering, 2D exemplars

1 Introduction

Textures are essentially important for current rendering techniques as they bring in richness without involving overly complicated geometry. Most previous work on texture synthesis focuses on synthesizing 2D textures, which require texture mapping with almost unavoidable distortions when they are applied to 3D objects. Solid textures represent color (or other attributes) over 3D space, providing an alternative

Email addresses: zgx.net@gmail.com (Guo-Xin Zhang), Yukun.Lai@cs.cardiff.ac.uk (Yu-Kun Lai), shimin@tsinghua.edu.cn (Shi-Min Hu).

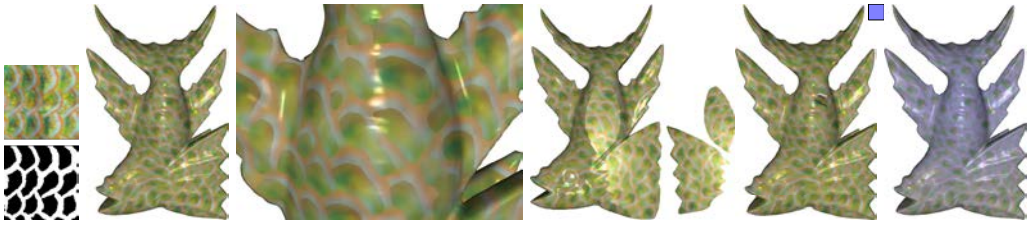

Fig. 1. High-resolution gradient solid texture synthesis and editing. From left to right: the input exemplar, the synthesized gradient solid texture following a given directional field, a closeup, internal slices and instant editing (user interaction and the output). Part of the figure was previously published in [43] and is republished with permission by Springer.

approach to 2D textures that avoids complicated texture mapping and allows real solid objects to be represented with consistent textures both on the surface and in the interiors alike.

Due to the extra dimension, solid textures represented as attributes sampled at regular 3D voxel grids are extremely expensive to synthesize and store. To provide sufficient resolution in practice, a typical solution is to synthesize only a small cube (e.g. 128^3), and tile the cube to cover the 3D space. However, tiling may cause visual repetition (see Fig. 8). While repetitions could be alleviated with some rotations, they cannot be eliminated completely when the volumes are sliced with certain planes. Further it is possible only when the solid textures have no interaction with the underlying objects, and thus cannot respect any model features or user design intentions. To address this, previous approaches [4,42] synthesize solid textures on demand; however, handling high-resolution solid textures is still expensive in both computation and storage.

Inspired by image vectorization, for pixels (or voxels) with dominantly smooth color variations (within each homogeneous region), vectorized graphics provide significant advantages such as being compact, resolution independent and easy-to-edit. The possibility and effectiveness of vectorizing solid textures have been recently studied in [33]. This work is essentially a 3D generalization of image vectorization, which requires voxel-level (raster) solid textures as input and inherits similar advantages over traditional raster solid textures. It remains computationally costly and involves large intermediate storage for raster solid textures to synthesize high resolution solid textures with a nonhomogeneous spatial distribution (e.g. [42]).

This paper is an extended version of the conference paper [43] with substantially extended technical details, experimental results, evaluation and applications including solid vectorization and instant editing. In this paper, instead of first synthesizing the full voxel solid textures before vectorizing them [33], we propose a novel approach to directly synthesize vectorized solid textures from exemplars. Inspired by gradient meshes in image vectorization [29], we propose a novel *gradient solid* representation that uses a tricubic interpolation scheme for smooth color variations

within a region, and a region-based approach to represent sharp boundaries with separated colors. This representation is compact, more regular than Radial Basis Functions (RBFs) [33] and thus particularly suitable for real-time rendering and efficient solid texture synthesis. Our approach can be used to vectorize input solids, which is over 100 times faster than [33] and leads to reduced approximation errors in most practical cases, as shown later by extensive comparative experiments. As discussed later in the paper, while the proposed representation is not suitable for all textures, it is sufficient to represent a variety of practical solid textures in high quality, in particular those having dominantly smooth color variations within each homogeneous region.

We further treat solid texture synthesis as an *optimization* process of *control points* of gradient solids to produce synthesized solids with similar sectional images as given exemplars. Compared with traditional solid texture synthesis, we have *far less* control points than voxels, leading to a much more efficient algorithm. While we solve both bitmap solid synthesis and solid vectorization together and produce solid textures with comparable quality as the state of the art, it is over 10 times faster than existing synthesis methods.

The main contributions of this paper are:

- A new *gradient solid* representation with regular structure that is compact, resolution-independent and capable of representing smooth solids and solids with separable regions.
- A novel optimization-based algorithm for *direct* synthesis of high quality solid textures vectorizing high resolution solids which is efficient both in computation and storage.
- We also propose a novel application — *instant solid editing*, as demonstrated in the paper.

To the best of our knowledge, this is the first algorithm that synthesizes vector solid textures directly from exemplars, allowing high resolution, potentially spatially nonhomogeneous solid textures to be synthesized *in full*. Thanks to the new compact representation, solid textures can be directly synthesized in this representation, significantly reducing the computational and memory costs. Our representation also allows instant editing without resorting to time-consuming conversion between vector and raster solids. Both of these would be difficult to achieve, if possible, by previous methods. This addresses major drawbacks of using solid textures in practical applications, namely large storage requirements and synthesis time. Various techniques have also been developed to effectively improve the quality or reduce the computational cost.

A typical example of high-resolution gradient solid texture synthesis and editing is given in Fig. 1. In Sec. 2, we review prior work in texture synthesis and vectorization. Our vector solid representation is described in Sec. 3 and the algorithm details

given in Sec. 4. Experimental results, applications and discussions are presented in Sec. 5 and finally concluding remarks are given in Sec. 6.

2 Related Work

Our work is closely related to example based texture synthesis and vector images/textures.

Solid Texture Synthesis Texture synthesis has been an active research direction in computer graphics for many years. Please refer to [35] for a comprehensive survey of example-based 2D texture synthesis and [28] for a recent survey of solid texture synthesis from 2D exemplars.

Early work on solid texture synthesis focuses on procedural approaches [26,27]. Since rules are used to generate solid textures, very little storage is needed. Procedural solid textures can be generated in real-time [2]. However, only restricted classes of textures can be effectively synthesized and it is inconvenient to tune the parameters. Exemplar-based approaches do not suffer from these problems, and thus received more attention. 2D exemplar images are popular due to their wide availability. Wei [34] extends non-parametric 2D texture synthesis algorithms to synthesize solid textures. An improved algorithm is proposed in [13] to generate solid textures based on texture optimization [14] and histogram matching [8]. Further extended work [3] considers k-coherent search and combined position and index histograms to improve the results. To synthesize high resolution solid textures, Dong et al. [4] propose an efficient synthesis-on-demand algorithm based on deterministic synthesis of certain windows from the whole space [16] necessary for rendering, based on the fact that only 2D slices are needed at a time for normal displays. This work is extended in [42] that introduces user-provided tensor fields as guidance for solid texture synthesis. This approach allows synthesizing solid textures with nonhomogeneous spatial distributions, thus cannot be achieved by tiling small fixed cubes.

Alternative approaches for solid texture synthesis exist. Jagnow et al. [10,11] propose an algorithm based on stereological analysis which provides more precise modeling of solid textures. Du et al. [5] synthesize solid textures by analyzing the shapes and colors of particles from 2D exemplars and appropriately placing particles to form consistent sectional images as the exemplars. This is conceptually similar to salient structural element analysis in 2D texture synthesis [24]. The method is particularly suitable for semi-regular solid texture synthesis. However, these approaches only work for restricted types of solid textures with well separable pieces. Lapped textures have been extended to synthesize 3D volumetric textures [30]. 3D volumetric exemplars instead of 2D image exemplars are needed as input. Solid texture synthesis has also been used for other applications. Ma et al. [21] use simi-

lar techniques for motion synthesis.

Unlike previous methods, our approach directly synthesizes *gradient solid textures* from 2D exemplars. This provides the benefits from both procedural and exemplar-based approaches: the representation is more compact and high resolution solid textures can be synthesized *in full* efficiently. The algorithm is flexible to synthesize various solid textures using 2D exemplars and follow given tensor fields if specified by the user. The whole solid textures need only to be synthesized once which reduces overall computation.

Vector Images and Vector Solid Textures Different from raster images, vector graphics use geometric primitives along with attributes such as colors and their gradients to represent the images. Due to the advantages of vector graphics, plenty of work recently focuses on generating vector representations from raster images. Recent work proposes automatic or semi-automatic approaches to high-quality image vectorization using quadrilateral gradient meshes [29,15] or curvilinear triangle meshes for better feature alignment [37]. Diffusion curves [23] model vector images as a collection of color diffusion around curves. Some works consider combining raster images with extra geometric primitives [1,32,25] to obtain benefits such as improved editing and resizing.

Vector graphics have recently been generalized to solid textures [33,31]. Compared to raster solids, vector solids have the advantages of compact storage and efficient rendering. Wang et al. [33] propose an automatic approach to vectorize given solid textures using a RBF-based representation. However, this approach relies on raster solids as input, thus an expensive raster solid texture synthesis algorithm [13] needs to be performed first if only 2D exemplars are given as input. Diffusion surfaces [31], a generalization from diffusion curves [23], was used to represent vector solids; their focus however is user design of solids rather than automatic generation.

Vector representation is loosely related to volume compression techniques (e.g. [22,41]) as both consider more compact representations than raster solids. The focus of vector representation however aims at creating compact and resolution-independent representation suitable for graphics applications that produce visually similar and pleasing results even when magnified while the purpose of volume compression techniques is to reconstruct large volumes as close as possible to the original even under significant compression. Research work on volume compression tends to use blocks and block-based coding which leads to less smooth reconstruction.

We propose a novel algorithm that synthesizes gradient solids directly from 2D exemplars, bypassing intermediate bitmap solid synthesis and subsequent bitmap-to-vector conversion, leading to an efficient algorithm in both computation and storage that produces high quality solid textures. The representation although with a somewhat different aim may be useful for certain volume compression applications.

3 Gradient Solid Representation

We give details of the gradient solid representation, allowing efficient representation of smooth regions and regions with boundaries.

3.1 Representing Smooth Regions

We first consider representing regions with smoothly varying colors. We use an $n \times n \times n$ grid of control points with axes u, v, w to represent the solid textures. At each control point (i, j, k), we store a feature vector $\mathbf{f}$ including r, g, b color components and additional feature channels such as the signed distance measuring the distance as well as inside/outside to some surfaces that separate the volume into two sides. This is useful for better structure preservation [17] as well as region separation. The latter use will be detailed in the next subsection. In addition, the gradients of $\mathbf{f}$, i.e. $\frac{d\mathbf{f}}{du}, \frac{d\mathbf{f}}{dv}, \frac{d\mathbf{f}}{dw}$ are also stored allowing flexible control of variations in 3D space. 3D tricubic interpolation with gradients [7,18] is used to obtain the feature vector $\tilde{\mathbf{f}}$ for any voxel inside the grid. Similar tricubic interpolation has been used in isosurface extraction from volumetric data for visualization [12]. Assume that $p = 1, 2, \ldots, 8$ represents the 8 control points in the cube that covers the voxel and assume second or higher order derivatives of $\mathbf{f}$ to be zero, $\tilde{\mathbf{f}}$ at parameter (u, v, w) $(0 \leq u, v, w \leq 1)$ can be evaluated as

$$\tilde{\mathbf{f}}(u, v, w) = \sum_{i,j,k=0}^{3} \mathbf{a}_{ijk} u^i v^j w^k. \tag{1}$$

The coefficients $\mathbf{a}_{ijk}$ are determined by setting the interpolated function to give identical values, gradients and some selected higher order derivatives as stored values at each corner of the cube. The higher order constraints are selected to be isotropic (consistent with different axes) and introduced to ensure uniqueness of the solution. As proved in [18], all the 64 coefficient vectors $\mathbf{a}_{ijk}$ are weighted sums of 32-dimensional vectors $V = (\cdots \mathbf{f}^{(p)}, \frac{d\mathbf{f}^{(p)}}{du}, \frac{d\mathbf{f}^{(p)}}{dv}, \frac{d\mathbf{f}^{(p)}}{dw} \cdots)$ and the interpolation is C^1 continuity not only at cube corners but also over the whole volume.

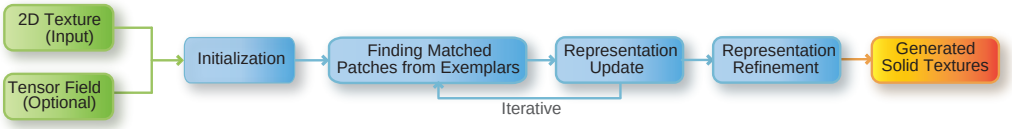

Fig. 2. Algorithm pipeline of gradient solid texture synthesis. The figure was previously published in [43] and is republished with permission by Springer.

The geometric positions of control points in our representation are fixed, however, these points still carry other attributes such as color and gradients which control

the appearance of the solids. Assuming the displacement between adjacent control points is d, the geometric position of the control point (i, j, k) is (id, jd, kd). The displacement determines the number of voxels located within each cube of the control grid. Larger d leads to more compression while smaller d implies better capture of details. In all of our experiments we use $d = 4$ which means that the number of control points is roughly $\frac{1}{64} = 1.56\%$ of voxels.

This simple representation has several significant advantages. For any fixed point with known parameter (u, v, w), since $u^i v^j w^k$ can be pre-computed, the expensive evaluation of Eqn. 1 can be reduced to a weighted sum of elements in V. In practice, we pre-compute these coefficients for a regular grid with 33^3 samples in each cube, with interval at $\frac{1}{8}$ voxel for accuracy. A fixed look-up table irrelevant to the input is pre-computed and stored, with $33^3 \times 32$ entries (about 4.4MB), and the interpolated feature at any space position can be computed as a linear combination of V with these prebuilt weights.

The interpolation is achieved in rendering via GPU acceleration, as detailed in Sec. 5.3. This allows efficient evaluation, particularly important as solid textures are computationally intensive. The look-up table does not need to be stored and is calculated on the fly. It is of fixed size even for very large volumes (equivalent to e.g. 512^3 or 1024^3) and in such cases becomes negligible. Compared with the RBF-based representation [33], we have regular structures suitable for texture synthesis. As demonstrated in Figs. 11 and 12, our local interpolation representation has much better color reproduction. There is no need to store the positions of control points, which further saves storage. The regularity also helps efficient direct solid texture synthesis and supports other applications such as instant editing propagation, as detailed later.

3.2 Representing Region Boundaries

If the given texture only contains gradual change of colors, the representation described in Sec. 3.1 is sufficient (e.g. Fig. 11). If the texture contains sharp boundaries that need to be preserved, a feature mask image is often used in texture synthesis as an additional component (other than color) to better preserve structures. Similar to previous work both in 2D and 3D textures [17,33], we assume regions can be separated using a binary mask. To represent the boundary in the solid textures, we also use a signed distance field stored at the *same* regular $n \times n \times n$ grid. We store both the signed distance D and its gradients $\frac{dD}{du}$, $\frac{dD}{dv}$ and $\frac{dD}{dw}$ and use the same tricubic interpolation as in Sec. 3.1 to calculate the interpolated signed distance $\tilde{D}$ at each voxel. The sign of $\tilde{D}$ indicates which side of the regions in the binary mask this voxel belongs to. Different from [33], gradients are stored in addition to the distance, and thus we process the distance field consistently with colors and represent region boundaries with flexibility. For each control point that is adjacent to at least

one cube with both positive and negative distances, other than the distance component where one version is sufficient, two feature vectors $\mathbf{f}^P$ (positive distance) and $\mathbf{f}^N$ (negative distance) and their gradients are stored. Any voxel with positive (or negative) distance will be evaluated using the same interpolation in Sec. 3.1 but with $\mathbf{f}^P$ (or $\mathbf{f}^N$) and their gradients instead. This guarantees C^1 smoothness within each region while also allowing sharp boundaries to be produced between regions. Our gradient solid representation is easy to evaluate but also sufficient to represent various solid textures, as demonstrated in Sec. 5. Although the representation is more restrictive than gradient meshes in that control points are located at fixed positions, it allows more efficient evaluation and synthesis. The representation still bears major properties of traditional vector representation such as being resolution independent and more compact than raster solids.

4 Gradient Solid Texture Synthesis

Our algorithm synthesizes gradient solid textures directly from 2D exemplars, which may include optional binary masks (if sharp boundaries exist between regions). In addition, a smooth tensor field may be given to specify the local coordinate systems the exemplar images align with [42]. We use an optimization based approach to synthesize gradient solid textures, with local patches aligned to the field if given.

4.1 Algorithm overview

The algorithm pipeline is summarized in Fig. 2, which involves several key steps: initialization, iterative optimization and final gradient solid refinement. Our gradient solid representation is first initialized based on the input exemplar. The synthesis is then carried out using a multi-resolution approach from coarse to fine. At each level, an optimization based approach is used that first identifies similar patches from the exemplar that best matches the current gradient solid and the gradient solid is then updated based on the samples in the patches. An approximate but sufficient fast evaluation of the vector solid representation is used in the intermediate stages. In the last stage, accurate evaluation as given in Eqn. 1 is used to optimize the control points for the best approximation. We will also discuss techniques to ensure efficiency in both computation and storage. If the binary mask is given, we pre-compute a signed distance field for the image with the absolute value at each pixel being the distance to the region boundary and different signs (positive or negative) for different regions. This signed distance is considered as an extra component in the feature vector $\mathbf{f}$ [17].

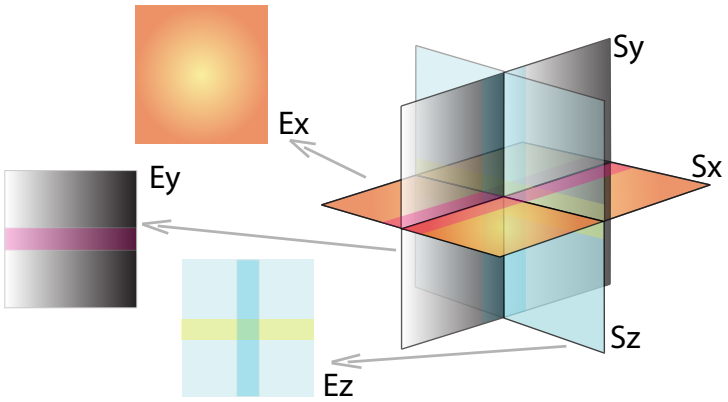

Fig. 3. Illustration of crossbars.

4.2 Initialization

We simply start from a randomized initialization. For each control point, we randomly select a pixel from the exemplar image, and assign the feature vector at the pixel to the control point. All the gradients are initialized to zero.

4.3 Optimization-based Synthesis

Optimization is the key step in our gradient solid texture synthesis pipeline. It involves iterations of two alternating steps, namely choosing optimal patches from exemplars that best match the current representation and updating the representation to better approximate the exemplar patches. Unlike traditional texture optimization [14,13], we optimize the feature vectors in the control points of the gradient solids, a much more compact representation than voxels. New challenges exist due to the different nature of the representation which we will address with various technical solutions. We apply N_O iterations for each synthesis level, and use a modified coarse-to-fine strategy detailed in Sec. 4.3.3. $N_O = 3$ is sufficient and used for all the experiments in the paper.

4.3.1 Finding matched patches from exemplars

We first identify those local patches from the exemplars that best match the current gradient solid. These patches will then be used to improve the representation. Since gradient solids have much sparser control points than voxels, we randomly choose a small number N_C of check points within each cube of the grid ($N_C = 3$ provides a good balance and is used for all the examples in the paper). At each check point, we sample three orthogonal planes each with $N \times N$ samples (denoted as $\mathbf{s}_x$, $\mathbf{s}_y$ and $\mathbf{s}_z$ respectively) which are evaluated based on our representation (as illustrated in Fig. 3). A fast approximate evaluation is used in intermediate synthesis to

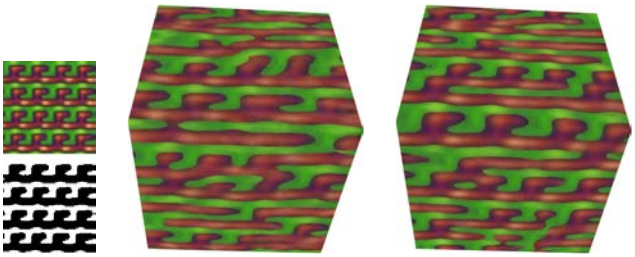

Fig. 4. Results without (left) and with (right) crossbar matching. The figure was previously published in [43] and is republished with permission by Springer.

significantly improve the performance without visually degrading the quality (see Sec. 4.3.4).

We then find three local patches from exemplars that best match these sampled patches. If all the three slices are equally important, we use three independent searches as [13]. Many practical solid textures are anisotropic and it is not possible to keep all three slices well matched with a single exemplar image. In such cases, it is known that matching two slices instead of three may lead to better results [13]. We propose a new approach that takes crossbar consistency into account, which works best when two slices are matched. Crossbars are those voxels shared by two or three slices (see Fig. 3) and inconsistent crossbars may result from independent best searches. For computational efficiency, we first search for the patch E_x from exemplars that best matches $\mathbf{s}_x$, as usual. We then search for the patch E_y that best matches $\mathbf{s}_y$ from a set of N_1 candidates with the most consistent crossbar voxels as E_x. If three slices are matched, we similarly search for the best match E_z of $\mathbf{s}_z$ from a set of N_2 candidates with the most consistent crossbars as E_x and E_y. $N_1 = 20$ and $N_2 = 50$ are used for all the experiments in the paper. This leads to improved synthesis results with better structure preservation, which shows the importance of crossbar consistency, as demonstrated in Fig. 4. While crossbar matching has been used in correction-based synthesis [4], using this in optimization based synthesis is new.

To speed up the computation, a PCA projection of the matching vectors is used [9], which effectively reduces dimensions from hundreds to 10-20 while keeping most of the energy. After this, the searches can be effectively accelerated with ANN approximate nearest neighbor library.

4.3.2 Representation update

Each matched patch at every check point gives $N \times N$ samples, which will be used to update the gradient solid representation. To efficiently collect samples, we conceptually build a bucket for each voxel in the grid that holds all the samples located in the voxel. After considering check points in all the cubes, each bucket may end up with none or a few samples. For buckets with more than one samples, in order to determine the feature vector, simply averaging all the samples in the

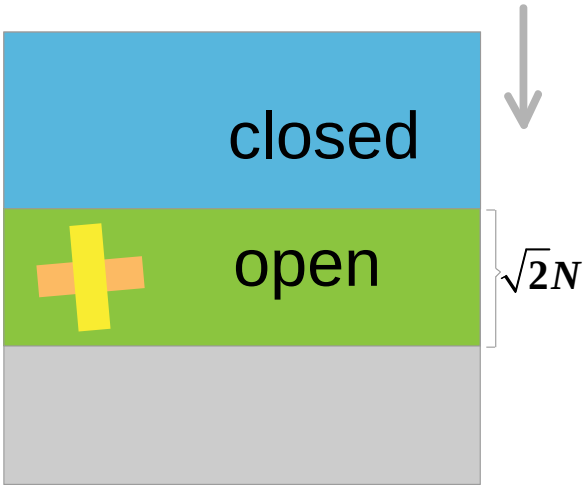

Fig. 5. Illustration of bucket reuse.

bucket tends to produce blurred voxels. Previous methods [36,13] use mean shift clustering to avoid blurring, which is expensive as all the samples in the buckets need to be preserved and clustering algorithms need to be performed many times. We propose two novel approaches to significantly improve the efficiency.

Quantization. To avoid blurring without storing all the samples in each bucket, we propose a novel approach based on vector quantization. We preprocess the given exemplar to quantize colors of all the pixels into N_T clusters. A small number of N_T (e.g. 12) is sufficient for practical textures. For the texture with a binary mask, we start from two clusters for both positive and negative regions, and iteratively allocate the new cluster to regions with most significant average quantization error, until all the N_T clusters are allocated. We use a two-pass approach in the synthesis. In the first pass, for every bucket, only the number of samples belonging to each cluster is recorded. In the second pass, we compute the average feature vector only for those samples belonging to the two *dominant* clusters (with maximum counts in the first pass). Since the dominant clusters are known before the second pass, whenever a sample is generated we test if it should be included for averaging. Only the sum and the number of samples need to be kept which significantly saves the storage. This avoids using the computationally expensive clustering algorithm for each voxel but also significantly reduces blurring, as demonstrated in various results in Sec. 5. Using quantization in the finest level is sufficient, according to our experience.

Bucket reuse. Although conceptually the number of buckets is the same as the number of voxels, i.e. $O(n^3)$, we can significantly reduce the memory requirement by bucket reuse. We update our representation in the 3D scanline order of control points. Depending on the template size N, check points more than $\frac{\sqrt{2}}{2}N$ voxels away will not produce any sample in the current bucket, where $\frac{N}{2}$ is half the template size and $\sqrt{2}$ is introduced due to rotation. As illustrated in Fig. 5 for a 2D

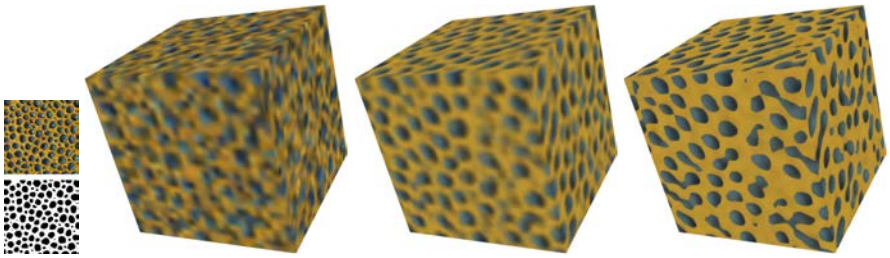

Fig. 6. Comparison of results using direct upscaling (left) and our algorithm without (middle) and with (right) region separation.

illustration, we keep track of two references in the dominant dimension (one of the three dimensions that can be chosen arbitrarily) that mark the boundaries of the open region (where new samples will be generated) and the closed region (where no more samples will be produced and we can safely update the gradient solid representation). In case the two-pass algorithm in quantization is used, this buffer needs to be doubled i.e. up to $2\sqrt{2}N$ span in the dominant dimension is sufficient, or the memory cost is $O(n^2N)$. This is because either pass has an affected region as we discussed and the second pass relies on the results collected from the first pass. The required buffering space does not increase with more synthesis iterations as buckets are cleared after each synthesis iteration and no further propagation as in [4] happens. Since $N \ll n$ and often constant for various examples, this effectively saves the storage by reducing the complexity from n^3 to n^2, without any extra recomputation. This is possible, because after each iteration of optimization, only a very compact gradient solid representation is kept, while traditional solid texture synthesis requires the whole dense volume to be accessible. By using this technique, we can synthesize gradient solid textures corresponding to 1024^3 voxels within $2GB$ memory, even less than storing the voxels alone.

After obtaining the average feature vector for any bucket with at least one sample, we assign each non-empty bucket to the closest control point. The feature vector as well as gradients of the control point are updated by minimizing the fitting error in the least-squares sense. For a particular control point, assume s buckets are related with relative coordinates du_t, dv_t, dw_t and feature vector $\mathbf{f}_t$ ($1 \leq t \leq s$), we find $\mathbf{f}_c$, $\frac{\mathbf{f}_c}{du}$, $\frac{\mathbf{f}_c}{dv}$, $\frac{\mathbf{f}_c}{dw}$ that minimizes

$$E_C = \sum_{t=1}^{s} \left\| \mathbf{f}_c + \frac{\mathbf{f}_c}{du} du_t + \frac{\mathbf{f}_c}{dv} dv_t + \frac{\mathbf{f}_c}{dw} dw_t - \mathbf{f}_t \right\|^2 . \tag{2}$$

This can be considered as a local first-order Taylor expansion of our representation which can be efficiently solved by small linear systems. This approximation is sufficient for intermediate computation and we optionally use the accurate evaluation in the final stage.

4.3.3 *Multi-resolution synthesis*

To capture features at multiple scales, a multi-resolution approach is also used in our algorithm. However, since a sparse control grid is used, reducing the resolution is not feasible as it would be too coarse in low resolutions to effectively capture details. Instead, inspired by fractional sampling [16], in each successive coarser level we keep the resolution of the control grid *unchanged* and double the spacing between sample pixels in exemplar image and voxels in the 3D space. From coarse to fine we use three levels of synthesis with $N = 9, 11, 21$ respectively. The finest level uses a significantly larger neighborhood in order to cover a few control points at minimum in our sparse representation.

4.3.4 *Fast approximate evaluation*

Our gradient solid representation is relatively easy to evaluate; however, in the solid texture synthesis process, many evaluations are needed. We suggest two approximations for improved performance. In the intermediate synthesis process, instead of evaluating the accurate values at each sample point, we use a first-order Taylor expansion as an approximation. For any point p whose closest control point is c with feature vector $\mathbf{f}_c$ and its gradients $\frac{\mathbf{f}_c}{du}$, $\frac{\mathbf{f}_c}{dv}$, $\frac{\mathbf{f}_c}{dw}$, the approximate feature vector at p with relative coordinates du_p, dv_p, dw_p, can be evaluated as $\mathbf{f}_p = \mathbf{f}_c + \frac{\mathbf{f}_c}{du}du_p + \frac{\mathbf{f}_c}{dv}dv_p + \frac{\mathbf{f}_c}{dw}dw_p$. This approximation does not ensure smoothness, but only involves 3 multiplications and 3 additions for each component of the feature vector, thus only takes about $1/10$ of the computation of a full evaluation. In the iterative synthesis, another approximation is to ignore the region-based calculation given in Sec. 3.2 (as if there is no separated region as in the single channel case such as Fig. 6(middle)). This may mix up voxels in different regions within *the same cube* and leads to visual degradation of the final results; the impact on the intermediate synthesis however is negligible as it is restricted to a couple of voxels due to the cube size.

4.4 *Gradient solid representation refinement*

As the final step, we further optimize the gradient solid representation to better represent the synthesized gradient solids.

Region separation. For solids with smooth variation of colors (e.g. Fig. 11), our algorithm does not require a binary mask as input and can effectively reproduce the solids with a single region. For solids with sharp region boundaries that need to be preserved, we differentiate regions with positive and negative signed distances for the computation of control point parameters described in Sec. 4.3.2. For each control point, we compute positive parameters ($\mathbf{f}^P$ and gradients) using samples with positive signed distance. Similarly, samples with negative signed distance con-

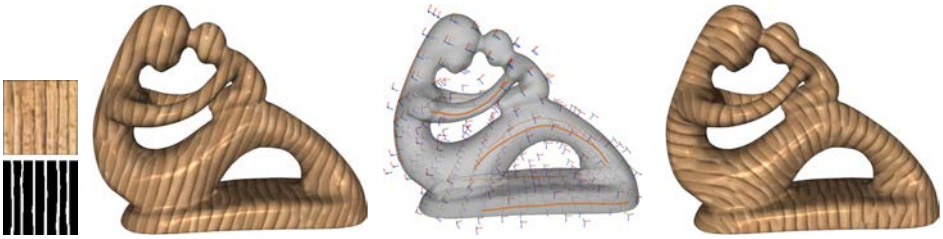

Fig. 7. Synthesized object without (a) and with the field (c); the field is given in (b).

tribute to negative parameters ($\mathbf{f}^N$ and gradients). To improve reliability in the fitting of boundary control points, we propagate boundary samples (with neighboring samples having different signs of distance) to the near neighboring space, similar to dilation in mathematical morphology. This mainly ensures cubes near region boundaries have sufficient samples to make fitting reliable.

Control point optimization. To further improve the quality, instead of fitting with first order approximation, we can also minimize the fitting error of all the samples between the sample values and those interpolated using Eqn. 1. For a sample point with sampled feature vector $\hat{\mathbf{f}}_i$ located in the cube c_i with corner control points collected as V_i and parameter (u_i, v_i, w_i), the evaluated feature vectors $\tilde{\mathbf{f}}$ are linear functions of V_i, denoted as $\mathbf{f}(V_i; u_i, v_i, w_i)$. We minimize the following quadratic energy

$$\bar{E}_C = \sum_{i=1}^{N_S} \|\tilde{\mathbf{f}}_i - \hat{\mathbf{f}}_i\|^2 = \sum_{i=1}^{N_S} \|\mathbf{f}(V_i; u_i, v_i, w_i) - \hat{\mathbf{f}}_i\|^2, \tag{3}$$

where N_S is the number of sample points. Minimization of $\bar{E}_C$ leads to a sparse linear system. As we have a good estimation from the previous approximation, the linear system can be effectively solved in a few iterations. As demonstrated in Table 1, control point optimization reduces the approximation error but also takes some extra time. Our method without this optimization is sufficiently good in many cases so it is considered as an option to tradeoff quality with speed.

4.5 Instant solid editing

Editing propagation often takes a sparse set of user input as constraints and extends this to similar regions to avoid otherwise labor-intensive procedures. Editing propagation has been studied for image/video processing (e.g. [39,19]), Bidirectional Texture Functions editing (e.g. [40]) etc. Similarly, 3D solids are expensive to store, and also time-consuming to edit. We achieve instant solid editing by adapting a recent development [19] in images and videos. Alternative approaches for texture editing may involve texture classification (e.g. [38]) to identify similar patterns. In this work, we restrict the propagation based on color similarity and location closeness, which is much more efficient thus suitable for solid textures, and more robust as no classification is needed. A typical scenario for this editing is

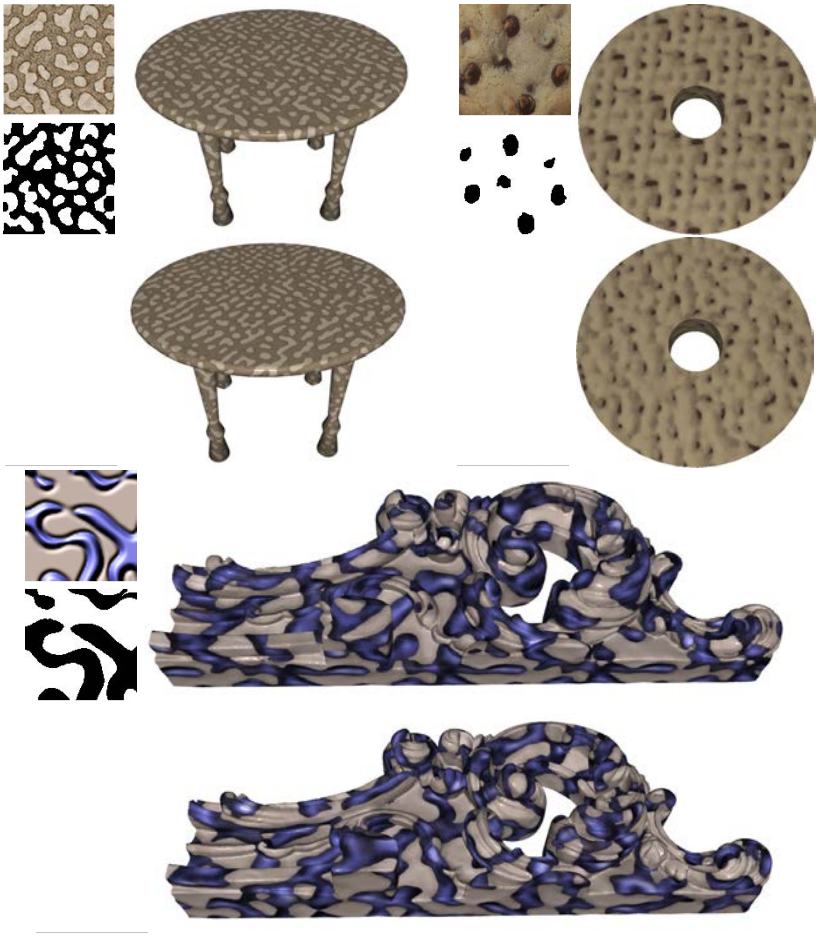

Fig. 8. Synthesized solids without fields. First row: tiled low resolution (128^3) solids. Second row: high resolution (512^3) solids.

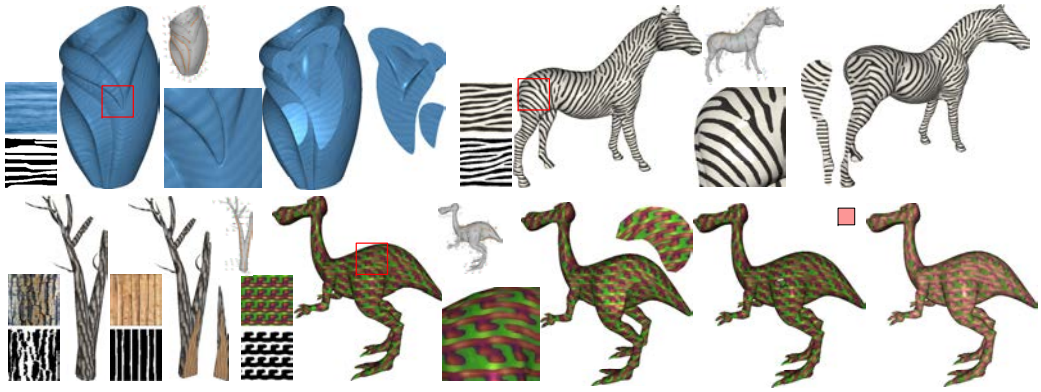

Fig. 9. Synthesized high-resolution solids (about 512 samples in the longest dimension) following given directional fields with our algorithm: 'vase', 'horse', 'tree' and 'dinopet' with synthesized solids, close-ups and internal slices. 'Dinopet' is turned pink with instant editing. Part of the figure was previously published in [43] and is republished with permission by Springer.

the user first draws a few strokes with different intensities indicating how strong

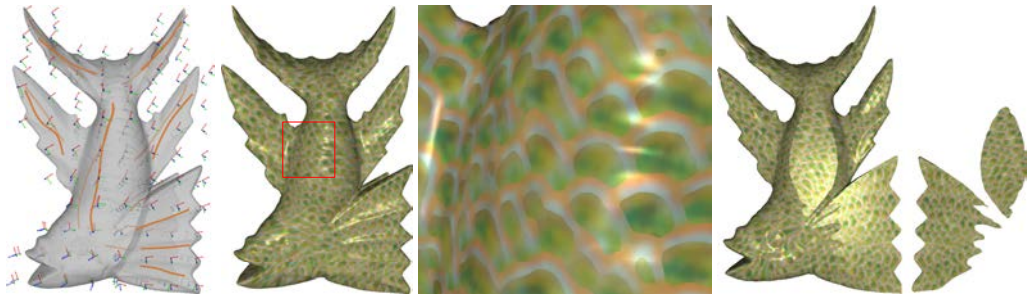

Fig. 10. Synthesized high-resolution solid texture (1024 samples in the longest dimension) with a field. (a) input user specified tensor field; (b) synthesized solid texture; (c) close up (d) internal slices.

the selected voxels will be affected by the editing. The user then selects a reference color, and voxels will be affected based on similarities in the position and the appearance (color) to those with user specifications. While the editing in [19] is generally efficient, dealing with large volumes is still relatively slow. Worse still, if the volume is in some vector representation, naive application of this method will involve converting to raster representation before editing and back to vector representation afterwards. We show as follows that our adaptation of the editing algorithm is instant with virtually *equivalent* solution; this cannot be achieved with Wang's representation.

For each control point i with color $\mathbf{c}_i = (r, g, b)^T$ and position $\mathbf{p}_i = (x, y, z)^T$, we need to know the influence h_i. This is effectively modeled as m RBFs, the centers of which are randomly selected from the stroke voxels

$$h_i = \sum_{k=1}^{m} \omega_k h_{i,k} = \sum_{k=1}^{m} \omega_k \exp\left\{-\alpha(\beta|\mathbf{p}_i - \bar{\mathbf{p}}_k|^2 + |\mathbf{c}_i - \bar{\mathbf{c}}_k|^2)\right\}, \tag{4}$$

where $\bar{\mathbf{p}}_k$ and $\bar{\mathbf{c}}_k$ are the position and color of k-th stroke voxel selected as a RBF center. ω_k, restricted to be non-negative, can be obtained by solving a linear programming problem that minimizes the strength deviation for user specified voxels [19]. Parameters α and β control the propagation and $\alpha = 10^{-4}$, $\beta = 0.1$ work well in many cases. Assuming the reference color is $\mathbf{c}_{ref}$, to compute the edited gradient solids, $\mathbf{c}_i$ and $\frac{d\mathbf{c}_i}{d\mathbf{p}_i}$ need to be updated for each control point, which can be effectively calculated as follows. We define $\mathbf{c}'_i = (1 - h_i)\mathbf{c}_i + h_i \cdot \mathbf{c}_{ref}$, and thus we have

$$\frac{d\mathbf{c}'_i}{d\mathbf{p}_i} = (1 - h_i)\frac{d\mathbf{c_i}}{d\mathbf{p}_i} - (\mathbf{c}_i - \mathbf{c}_{ref})(\frac{dh_i}{d\mathbf{p}_i})^T, \tag{5}$$

where

$$\frac{dh_i}{d\mathbf{p}_i} = -\sum_{k=1}^{m} \omega_k 2\alpha h_{i,k}\{\beta(\mathbf{p}_i - \bar{\mathbf{p}}_k) + \left(\frac{d\mathbf{c}_i}{d\mathbf{p}_i}\right)^T (\mathbf{c}_i - \bar{\mathbf{c}}_k)\}. \tag{6}$$

The editing is demonstrated in Fig. 1 to turn a fish purple instantly. A few strokes on the fish object are drawn to indicate the effect of change and a purplish color

sample is chosen (as in the box). Another example is in Fig. 9 where 'dinopet' is turned pink instantly with a few strokes and a pink reference color (as in the box). The editing algorithm only takes about 0.1 seconds, providing instant feedback on such large volumes. Comparatively, direct application on raster solid of equivalent resolution takes about 1 second and naive implementation on vector solids takes a few minutes.

5 Results and Discussions

Our algorithm is useful for either direct synthesis of solid textures, or vectorizing input solids. We carried out our experiments on a computer with 2×2.26GHz quad-core CPU and NVIDIA GTS 450 GPU. Our algorithm involves a few parameters for various stages of the pipeline. We used the following settings for experiments in the paper: the grid size $d = 4$, the number of iterations $N_O = 3$, the number of checkpoints $N_C = 3$, the number of quantization clusters $N_T = 12$, the number of crossbar matching candidates $N_1 = 20$, $N_2 = 50$, neighborhood size for different levels $N = 9, 11, 21$ and editing propagation parameters $\alpha = 10^{-4}$, $\beta = 0.1$.

5.1 Solid texture synthesis

Our algorithm directly synthesizes more compact and resolution-independent gradient solid textures from 2D exemplars. Solids with comparable quality to the state of the art can be synthesized, as shown in Figs. 1, 4-8, 10. As for other CPU-based algorithms that focus on synthesizing full solids of a 128^3 cube, the typical reported times have been tens of minutes, e.g. [13] uses 10-90 minutes (without tensor fields) and [21] (a CPU-based implementation similar to [4] with direction fields considered) reported about 30 minutes with a single core. Our results are vector solid textures which are resolution-independent. For simplicity, we consider solid textures with equivalent detail resolution to raster solid textures when certain resolution is referred to in the following discussion. Our current implementation, after about 10 seconds preprocessing of the input exemplar (which is the same for arbitrarily sized output volumes), takes only 13 seconds. Even counting the different performance of CPUs, our algorithm is over 10 times faster. Due to the compactness in representation and the technique for memory reuse, we can synthesize high-resolution solid textures in full. 512^3 solids can be synthesized within 15 minutes (Fig. 8). Other examples throughout the paper with about 512 samples in the longest dimension take 3-7 minutes, while the example in Fig. 10 at resolution of 1024 takes 35 minutes and within 900MB memory. Region separation is not needed if the input texture does not contain sharp boundaries, as the 'vase' and 'tree' examples in Fig. 9. In these examples, the binary mask is used only as part of the feature vector, not for region separation. The 'tree' example shows that our synthesis algorithm can be

example	error			time		
	Wang's	ours w/o opt.	ours w/ opt.	Wang's	ours w/o opt.	ours w/ opt.
Fig. 11 ('caustics')	6.14	2.62	1.50	8 min 25 sec	1.08 sec	5.10 sec
Fig. 12 ('balls')	8.59	5.83	4.21	24 min 21 sec	2.67 sec	11.46 sec

Table 1
Statistics of solid vectorization results.

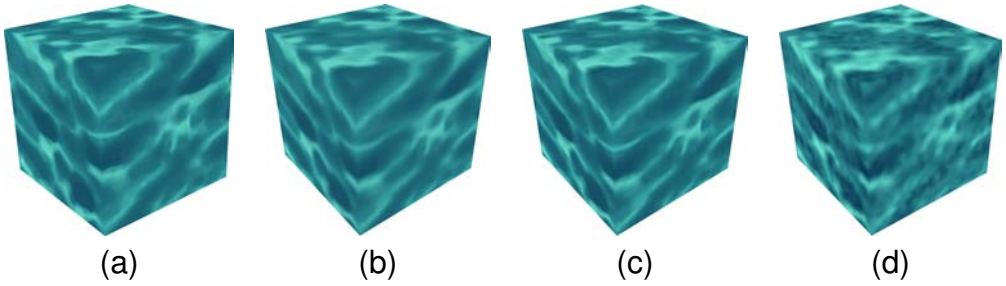

Fig. 11. Solid vectorization of the input volume 'caustic' without binary mask. (a) input volume; (b)(c) our results without and with further optimization; (d) result using [33].

generalized to synthesize solids with different exemplars covering different spaces, mimicking the real structure of a tree.

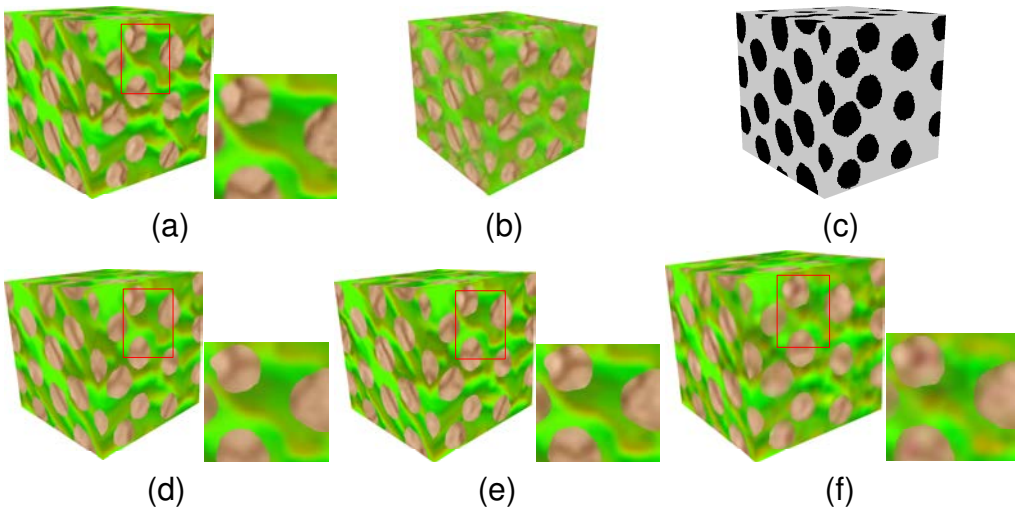

Fig. 12. Solid vectorization results with a binary mask. (a) input volume 'balls'; (b) input volume rendered in transparency; (c) input mask; (d) vectorized solid with our algorithm without optimization; (e) our result with optimization; (f) result using [33].

We demonstrate the effectiveness of our algorithm with various examples. Although our method uses a rather sparse set of control points, they are much more expressive than voxels at the same resolution. An example is given in Fig. 6. The left result is synthesized with [42] (using a proportionally downsized exemplar image as input) and looks sensible at original 32^3 resolution. We use tricubic interpolation to upscale the volume to 128^3 and clear artifacts appear indicating that 32^3 volume is not sufficient to capture the structure of the solid. Our results with

also 32^3 control points are significantly better and sharp region boundaries can be recovered with region separation. Tiling small cubes such as of 128^3 size to cover the whole space is commonly used, due to the prohibitively expensive computation with most previous algorithms. Synthesizing high resolution solids is essential to avoid visual repetition (as demonstrated by 'table', 'cake' and 'statue' in Fig. 8) or produce solids following certain direction fields (see Fig. 9). A comparison of results without or with the field is given in Fig. 7. High resolution solid textures with 512 and 1024 samples in the longest dimension are shown in Figs. 1 and 10, respectively. Note that in all the results we synthesize the full solids rather than only the visible voxels [4,42]. This is preferred since in many applications objects are synthesized once but rendered many times on lower-end systems. Our representation makes rendering algorithm both efficient and simple to implement.

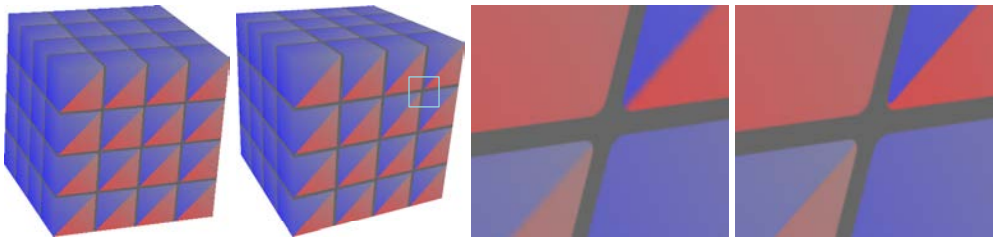

Fig. 13. An example that a single distance field is not sufficient to fully recover sharp boundaries. (a) input solid; (b) vectorized solid with a single distance field; (c) close-up of (b); (d) vectorized solid with an additional distance field to recover sharp boundaries (close-up).

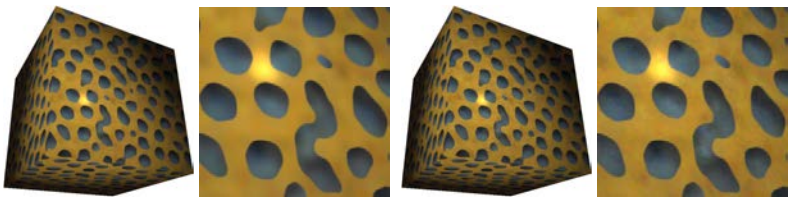

Fig. 14. Vector solid textures without (left) and with added details (right).

5.2 Vectorization of solid textures

Our approach can also be used for solid texture vectorization. In this application, we take each voxel as a sample and produce gradient solids with the method in Sec. 4.4 if optimization is used or otherwise a first order approximation as in Sec. 4.3.4. We perform comparative experiments on the same computer, using the code directly from [33]. $5,000$ RBFs are used to provide sufficient flexibility, more than most examples in [33], for fair comparison. Although our algorithm is highly parallel, we only use a single core for fair comparison. Detailed running times and fitting errors are given in Table 1. Whilst pixel-wise error measurement before and after vectorization may not be the best criterion perceptually, it is widely used in image

vectorization. For most solids suitable for vectorization, our method produces results with lower per-pixel error and avoids the spotty artifacts caused by the use of RBFs. Although RBFs seem to be more flexible, unless using a (potentially impractically) large number of RBFs for relatively complicated input, spotty artifacts are reasonable reflection of radial bases and large approximation errors result. We also experimented with varying RBFs from $3,000$ to $5,000$ but the approximation errors in our experiments only drop marginally. Wang's algorithm may also get stuck at suboptimal solutions due to the highly non-linear nature. Our vectorization does not suffer from these problems and is much simpler to optimize as only sparse linear systems need to be solved.

Our method without control point optimization is on average 500 times faster and has much reduced reconstruction error and better color reproduction than [33], as shown in Figs. 11 and 12 as well as Table 1. If the optional control point optimization is used, the error can be further reduced at a small cost. This shows that we currently achieve interactive performance for vectorization of moderate sized volumes. Direct synthesis of gradient solid textures requires many times of intermediate vectorization and evaluation and it would become impractically slow without the speedup. Since the algorithm is highly parallel, a parallel GPU-based implementation may further improve the speed.

We use regions to represent sharp boundaries (Fig. 12) but our method can deal with input solids that cannot be naturally separated into multiple regions (see Fig. 11). In this case, no binary mask image needs to be provided. A more thorough evaluation on the whole dataset provided by [13] with 21 solid textures shows that for more than 75% of the examples, especially those examples more suitable for vectorization (with lower approximation for both methods), our method outperforms [33] in fitting error (see the accompanied supplementary material for detailed statistics).

We quantize each value with 8 bits, and our representation, without further careful coding, takes only 6.5% (without region separation) or 15% (with region separation) of the voxel solids, while 17%-26% is reported in [33]. The size of the look-up table is not considered because it does not depend on input and thus does not need to be stored in external files and its size is fixed and small enough to be kept in current graphics card without any problem.

5.3 Rendering

While our current *synthesis* implementation is CPU-based, gradient solids are *rendered* efficiently with commodity GPUs. For each visible pixel, we obtain the interpolated texture coordinate using the vertex shader and evaluate the color with Eqn. 1 using the fragment shader; the colors and gradients at control points are stored as textures for efficient GPU access. The color at any continuous position is

calculated by linear interpolation of entries in the look-up table described in Sec. 3 through hardware-supported texture fetches, a commonly used technique for realtime rendering. For solid textures with binary masks, the relevant set of feature vectors is used based on the evaluated signed distance. This is both efficient and ensures accuracy as accurate values are obtained at 8^3 times higher resolution than raster solids. This linear interpolation is accurate at $\frac{1}{8}$ voxel resolution and is sufficiently close to the real function such that no visible artifact is produced, even when extremely magnified. In most practical applications, a precomputed look-up table with $\frac{1}{4}$ voxel resolution is sufficient, which leads to a look-up table with $17^3 \times 32$ entries and taking less than 0.6MB storage. To avoid jagged boundaries when gradient solids with two regions are rasterized, we use similar antialiasing technique as in [33]. The idea is for pixels close to boundaries, colors evaluated with both positive and negative regions are linearly blended.

Our representation has similar real-time rendering performance as [33]. To make a fair comparison, in the performance measurement, we have disabled mipmapping for [33] and enabled antialiasing for both methods. For a 128^3 solid with a mesh containing $70K$ vertices rendered at 1024×768 resolution, our average frame rate is 80 fps and the rendering algorithm from [33] achieves on average 75 fps. High resolution solid textures in this paper are rendered with 30-60 fps. The slightly lower frame rates are due to the relatively complicated geometry and large textures with lower cache performance.

5.4 Discussions and Limitations

Although we can represent sharp boundaries with regions, similar to Wang et al. [33] using a single distance field we cannot in general recover sharp boundaries if more than two regions touch. An example is given in Fig. 13. The input solid (a) can be vectorized with our algorithm producing the reconstructed solid (b) with close-up (c). Sharp boundaries between triangles cannot be preserved with the single binary mask. Compared with [33], our blurring effects are much more local. If such blur is not acceptable, our algorithm needs to be modified to be augmented with another distance field to separate adjacent triangle pairs, as shown in (d) (a close-up view).

Another limitation is although our fitting error is usually lower than Wang et al. [33] for typical input, fine details within a region may not be fully reproduced; this however is a limitation for virtually all the vectorization methods. To simulate fine details of textures without excessive storage, the approximation error at any position is modeled as a Gaussian distribution. Assume for each position $\mathbf{x}$, and an arbitrary channel c (including r, g, b) with a sample pixel value $\hat{p}_c(\mathbf{x})$ and corresponding reconstructed value from the vector representation $\tilde{p}_c(\mathbf{x})$, the residual

$r_c(\mathbf{x}) = \hat{p}_c(\mathbf{x}) - \tilde{p}_c(\mathbf{x})$ is a Gaussian distribution with probability p satisfying

$$p(r_c(\mathbf{x}) = y) = G(0, \sigma_c(\mathbf{x})) = \frac{1}{\sqrt{2\pi\sigma_c(\mathbf{x})^2}} \exp\{-\frac{y^2}{2\sigma_c(\mathbf{x})^2}\}, \tag{7}$$

where $\sigma_c(\mathbf{x})$ is the standard deviation and y an arbitrary value. We optimize $\sigma_c(\mathbf{x})$ such that $p(r_c(\mathbf{x}) = \hat{p}_c(\mathbf{x}) - \tilde{p}_c(\mathbf{x}))$ is maximized, which is worked out as $\sigma_c(\mathbf{x}) = |\hat{p}_c(\mathbf{x}) - \tilde{p}_c(\mathbf{x})|$. For efficiency, σ_c is also compactly represented using our vector representation, treating as an additional channel. When rendering at any position, the residual r is randomly sampled from the distribution. To ensure consistent result, a position determined hash function as Perlin noise [27] is used. With similar lookup table based GPU acceleration, extra computation can be efficiently done, keeping the rendering algorithm realtime (current implementation with $30\% - 50\%$ of the original fps). An example is shown in Fig. 14 where richer details are recovered without losing the benefits of vector representation such as resolution independence.

As a method to produce vectorized solid textures, our method is not suitable for all textures. Even with noise modeling, for textures with large amount of high-frequency details, the method may not reproduce such textures in the synthesized solids well, as shown in Fig. 15. Nevertheless, we have demonstrated that our method works well on a variety of textures throughout the paper. Our representation is particularly suitable for solid textures having dominantly smooth color variations within each homogeneous region, as assumed by virtually all the vectorization methods. This applies also when textures contain textons at varying scales, reasonable synthesis results can be achieved as long as they don't have significant high-frequency details, as demonstrated in Fig. 8 where textures contain elementary pieces of different sizes. A regular grid is used for simplicity which may not be very efficient if the level of detail changes dramatically over the volume; alternatively, adaptive sampling may be used to alleviate this.

The instant editing algorithm in this paper does not consider the texton structures of the solid textures and thus may not provide semantically coherent editing results. Based on our general framework, this could be achieved with texton analysis and this is expected to be explored in the future.

6 Conclusions and Future Work

In this paper, we propose a novel gradient solid representation for compactly representing solids. We also propose an efficient algorithm for direct synthesis of gradient solid textures from 2D exemplars. Our algorithm is very efficient in both computation and storage, compared with previous voxel-level solid texture synthesis methods and thus allows high-resolution solid textures to be synthesized in full.

Fig. 15. Our method may not perform well on exemplar images with significant high frequency details. Left: input exemplar image; right: synthesized solid textures.

The algorithm can be generalized to take 3D solids as exemplars which will also benefit from the compactness of our representation. The representation is also potentially useful for accelerating volume processing. We have demonstrated instant editing of large volumes, and we would like to explore other applications such as efficient volumetric rendering and manipulation of (solid) textures (e.g. [6,20]) in the future. Our current implementation of the synthesis algorithm is purely CPU based. The algorithm is highly parallel and we expect to implement this on the GPU to further improve the performance. Our rendering implementation can be further augmented with mipmapping for adaptive scaling especially minification and texture composition to produce richer fractal-like boundaries, using techniques similar to those in [33]. The instant solid editing algorithm could be improved for more semantically meaningful editing by taking into account the texton structures.

Acknowledgements

This work was supported by the National Basic Research Project of China (Project Number 2012CB316400), the Natural Science Foundation of China (Project Number 61120106007 and 61170153), the National High Technology Research and Development Program of China (Project Number 2011AA010503) and the National Science and Technology Key Projects of China (2011ZX01042-001-002).

References

[1] W. Barrett and A. S. Cheney. Object-based image editing. *ACM Trans. Graph.*, 21(3):777–784, 2002.

[2] Nathan A. Carr and John C. Hart. Meshed atlases for real-time procedural solid texturing. *ACM Trans. Graph.*, 21(2):106–131, 2002.

[3] J. Chen and B. Wang. High quality solid texture synthesis using position and index histogram matching. *The Visual Computer*, 26(4):253–262, 2010.

[4] Yue Dong, Sylvain Lefebvre, Xin Tong, and George Drettakis. Lazy solid texture synthesis. *Computer Graphics Forum*, 27(4):1165–1174, 2008.

[5] Song-Pei Du, Shi-Min Hu, and Ralph R. Martin. Semi-regular solid texturing from 2D exemplars. *IEEE Trans. Vis. Comp. Graph.*, to appear.

[6] Hui Fang and John C. Hart. Textureshop: Texture synthesis as a photograph editing tool. In *ACM SIGGRAPH*, pages 354–359, 2004.

[7] James Ferguson. Multivariable curve interpolation. *J. ACM*, 11(2):221–228, 1964.

[8] D. J. Heeger and J. R. Bergen. Pyramid-based texture analysis/synthesis. In *Proc. ACM SIGGRAPH*, pages 229–238, 1995.

[9] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *Proc. ACM SIGGRAPH*, pages 327–340, 2001.

[10] R. Jagnow, J. Dorsey, and H. Rushmeier. Stereological techniques for solid textures. In *Proc. ACM SIGGRAPH*, pages 329–335, 2004.

[11] R. Jagnow, J. Dorsey, and H. Rushmeier. Evaluation of methods for approximating shapes used to synthesize 3d solid textures. *ACM Trans. Applied Perception*, 4(4):Article 24, 2008.

[12] Arie Kadosh, Daniel Cohen-Or, and Roni Yagel. Tricubic interpolation of discrete surfaces for binary volumes. *IEEE Trans. Vis. Comp. Graph.*, 9(4):580–586, 2003.

[13] Johannes Kopf, Chi-Wing Fu, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski, and Tien-Tsin Wong. Solid texture synthesis from 2D exemplars. *ACM Trans. Graph.*, 26(3):Article 2, 2007.

[14] Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. Texture optimization for example-based synthesis. *ACM Trans. Graph.*, 24(3):795–802, 2005.

[15] Yu-Kun Lai, Shi-Min Hu, and Ralph R. Martin. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM Trans. Graph.*, 28(3):Article 85, 2009.

[16] Sylvain Lefebvre and Hugues Hoppe. Parallel controllable texture synthesis. *ACM Trans. Graph.*, 24(3):777–786, 2005.

[17] Sylvain Lefebvre and Hugues Hoppe. Appearance-space texture synthesis. *ACM Trans. Graph.*, 25:541–548, 2006.

[18] F. Lekien and J. Marsden. Tricubic interpolation in three dimensions. *J. Numerical Methods Engin.*, 63:455–471, 2005.

[19] Yong Li, Tao Ju, and Shi-Min Hu. Instant propagation of sparse edits on images and videos. *Computer Graphics Forum*, 29(7):2049–2054, 2010.

[20] Jianye Lu, A. S. Georghiades, A. Glaser, H. Wu, L.-Y. Wei, B. Guo, J. Dorsey, and H. Rushmeier. Context-aware textures. *ACM Trans. Graph.*, 26(1):Article 3, 2007.

[21] C. Ma, L.-Y. Wei, B. Guo, and K. Zhou. Motion field texture synthesis. *ACM Trans. Graph.*, 28(5):Article 110, 2009.

[22] Paul Ning and Lambertus Hesselink. Fast volume rendering of compressed data. In *Proc. IEEE Visualization*, pages 11–18, 1993.

[23] A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin. Diffusion curves: a vector representaiton for smooth-shaded images. *ACM Trans. Graph.*, 27(3):Article 92, 2008.

[24] Bin Pan, Fan Zhong, Shuai Wang, Wei Chen, and Qunsheng Peng. Salient structural elements based texture synthesis. *Science China Information Sciences*, 54(6):1199–1206, 2011.

[25] Darko Pavić and Leif Kobbelt. Two-colored pixels. *Computer Graphics Forum*, 29(2):743–752, 2010.

[26] D. R. Peachey. Solid texturing of complex surfaces. In *Proc. ACM SIGGRAPH*, pages 279–286, 1985.

[27] K. Perlin. An image synthesizer. In *Proc. ACM SIGGRAPH*, pages 287–296, 1985.

[28] N. Pietroni, P. Cignoni, M. A. Otaduy, and R. Scopigno. Solid-texture synthesis: a survey. *IEEE Computer Graphics and Applications*, 30(4):74–89, 2010.

[29] J. Sun, L. Liang, F. Wen, and H.-Y. Shum. Image vectorization using optimized gradient meshes. *ACM Trans. Graph.*, 26(3):Article 11, 2007.

[30] Kenshi Takayama, Makoto Okabe, Takashi Ijiri, and Takeo Igarashi. Lapped solid textures: filling a model with anisotropic textures. *ACM Trans. Graph.*, 27(3):Article 53, 2008.

[31] Kenshi Takayama, Olga Sorkine, Andrew Nealen, and Takeo Igarashi. Volumetric modeling with diffusion surfaces. *ACM Trans. Graph.*, 29(6):Article 180, 2010.

[32] J. Tumblin and P. Choudhury. Bixels: Picture samples with sharp embedded boundaries. In *Proc. Eurographics Symposium on Rendering*, pages 186–196, 2004.

[33] Lvdi Wang, Kun Zhou, Yizhou Yu, and Baining Guo. Vector solid textures. In *Proc. ACM SIGGRAPH*, page Article 86, 2010.

[34] L.-Y. Wei. Texture synthesis from multiple sources. In *SIGGRAPH 2003 Sketch*, 2003.

[35] L.-Y. Wei, S. Lefebvre, V. Kwatra, and G. Turk. State of the art in example-based texture synthesis. In *Eurographics State-of-Art Report*, 2009.

[36] Y. Wexler, E. Shechtman, and M. Irani. Space-time completion of video. *IEEE Trans. PAMI*, 29(3):463–476, 2007.

[37] T. Xia, B. Liao, and Y. Yu. Patch-based image vectorization with automatic curvlinear feature alignment. *ACM Trans. Graph.*, 28(5):Article 115, 2009.

[38] Tian Xia, Qing Wu, Chun Chen, and Yizhou Yu. Lazy texture selection based on active learning. *The Visual Computer*, 26(3):157–169, 2010.

[39] Kun Xu, Yong Li, Tao Ju, Shi-Min Hu, and Tian-Qiang Liu. Efficient affinity-based edit propagation using k-d tree. *ACM Trans. Graph.*, 28(5):118:1–118:6, 2009.

[40] Kun Xu, Jiaping Wang, Xin Tong, Shi-Min Hu, and Baining Guo. Edit propagation on bidirectional texture functions. *Comput. Graph. Forum*, 28(7):1871–1877, 2009.

[41] Boon-Lock Yeo and Bede Liu. Volume rendering of dct-based compressed 3d scalar data. *IEEE Trans. Vis. Comp. Graph.*, 1(1):29–43, 1995.

[42] Guo-Xin Zhang, Song-Pei Du, Yu-Kun Lai, Tianyun Ni, and Shi-Min Hu. Sketch guided solid texturing. *Graph. Mod.*, 73(3):59–73, 2011.

[43] Guo-Xin Zhang, Yu-Kun Lai, and Shi-Min Hu. Efficient solid texture synthesis using gradient solids. *Lecture Notes in Computer Science*, 7633:67–74, 2012.