Solid Mathematical Marbling

Shufang Lu, Xiaogang Jin, Aubrey Jaffer, Fei Gao, and Xiaoyang Mao

Abstract—Years of research have been devoted to computer-generated two-dimensional marbling. However, three-dimensional marbling has yet to be explored. In this paper, we present mathematical marbling of three-dimensional solids which supports a compact random-access vector representation. Our solid marbling textures are created by composing closed-form 3D pattern tool functions. The resulting representation is feature preserving and resolution-independent. When implemented on the GPU, our representation enables efficient color evaluation during the real-time solid marbling texture mapping. Our method consumes very little memory because only the mathematical functions and their corresponding parameters are stored. In addition, we develop an intuitive user interface and a genetic algorithm to facilitate the solid marbling texture authoring process. We demonstrate the effectiveness of our approach through various solid marbling textures and 3D objects carved from them.

Index Terms—Solid texture, texture generation, 3D marbling.

1 Introduction

Volumetric modeling is a topic of increasing interest in the computer graphics community. It enriches diverse graphics applications, such as scientific visualization, education, computer games, and training activities. Solid texture, as a special case of volumetric data, associates color or other attributes with each 3D point in a volume.

Marbling is a traditional art of aqueous surface design which produces patterns for decorations. In the real world, marbling textures are widely used for decoration from jewelry and scarves to greeting cards and tissue boxes. In computer graphics, there are several methods to simulate marbling art and create textures accordingly. One important application of these marbling textures is to map them onto the surface of 3D objects for rendering. However, all the current computer-generated marbling textures belong to 2D textures. Solid marbling texturing has several notable advantages over its 2D counterpart. The most important one is that the solid texturing method can well represent 3D objects that are carved out of some continuous 3D solid materials. The tissue box model in Figure 7 is a good example while 2D approaches may have difficulties in maintaining continuity across all faces of the box. Moreover, a same solid marbling texture can be easily reused for different 3D shapes because no surface parameterization is necessary. In addition, solid texturing supports cross sectioning and volume rendering of 3D shapes. Therefore, it is desirable to develop a solid marbling texture generation method for 3D texturing.

- S. Lu and G. Fei are with the College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, P. R. China. E-mail: {sflu, feig}@zjut.edu.cn.
- X. Jin is with the State Key Lab of CAD&CG, Zhejiang University, Hangzhou 310058, P. R. China. E-mail: jin@cad.zju.edu.cn.
- A. Jaffer is with Digilant, Boston, MA, USA. E-mail: agj@alum.mit.edu.
- X. Mao is with University of Yamanashi, Kofu, Yamanashi, Japan. E-mail: mao@yamanashi.ac.jp.

Nevertheless, constructing high-quality solid marbling textures with real-time rendering performance and limited video memory is a challenging task. First, conventional solid textures consume large amounts of memory for the 3D grid due to the third dimension, and this consumption becomes worse with the increasing resolution of the solid texture. Second, in many applications, one is required to render high-resolution results preserving sharp features and smooth color variations on surfaces. Third, the creation of solid marbling textures should be flexible and intuitive in real-time.

Real-world marbling patterns are created by first dropping color paints onto the surface of water and then stirring the surface with tools. Some existing digital marbling systems generate marbling textures by solving the 2D Navier-Stokes equations on the computer by regarding it as a 2D Computational Fluid Dynamics (CFD) problem [1]. To generate solid marbling textures, a straightforward solution would be to solve the Navier-Stokes equations directly in the volumetric space. However, achieving this goal is not currently feasible. The CFD method is costly both in terms of computation time and memory consumption. To meet the aforementioned requirements, we develop 3D marbling pattern tool functions inspired by the 2D mathematical marbling method [2]. Our approach attains the desired properties of compact storage, resolution independence, and efficient random access, and it is particularly suitable for real-time rendering. Color variations within a volume are compactly represented using mathematical functions which support efficient color evaluation during the rendering process. Sharp features of rendered objects remain crisp at any magnification due to the vector representation.

Our paper has the following contributions. (1). We present solid mathematical marbling by means of closed-form 3D mathematical pattern tool functions. To the best of our knowledge, this is the first attempt to simulate solid marbling. (2). We develop a novel volumetric representation for solid marbling textures which is compact, resolution-independent and supports random access for real time rendering. (3). We develop an intuitive user interface and an evolutionary design scheme to facilitate the authoring of solid marbling textures.

2 Related Work

In this section, we focus on techniques for marbling texture simulation and solid texture generation which are most relevant to our work.

2.1 Marbling Texture

In recent years, researchers have attempted to develop digital marbling design systems to create marbling textures on the computer. In the process of real-world marbling, patterns are created through the manipulation of floating pigment on the liquid surface. Therefore, most of the marbling simulation methods are based on the 2D computational fluid dynamic simulation by numerically solving the Navier-Stokes equations. This is a time-consuming task. To support real-time feedback, Jin et al. first implemented the marbling process on the GPU to achieve high efficiency [1]. Xu et al. [3] and Zhao et al. [4] further revised this method to reduce the blurring effects. In these methods, to solve the equations numerically, a grid-based Euler technique is employed by discretizing the computational domain to a grid of $M \times N$ cells, where M, N are the width and height of the simulation domain. The parameters (velocity, density, pressure, and others) are defined at the center of each cell. Several arrays are required to store the states for one time step. By analogy, to create solid marbling textures, one naive way is to simply extend these methods from 2D to 3D. However, numerically solving 3D Navier-Stokes equations for solid marbling texture generation is challenging. First, solving 3D Navier-Stokes equations requires large memory consumption by 3D grids with respect to solid texture resolution. Second, numerical calculation of 3D Navier-Stokes equations is computationally expensive and may have difficulties providing real-time feedback on the current commodity hardware. Third, the solid textures created in this way are resolution dependent and insufficient resolution often leads to blurry interpolated texture mapping results. In contrast to previous physically-based methods, in which the shape is tracked with a concentration density field in Eulerian grids, Ando and Tsuruno presented an interactive system for designing 2D marbling textures in vector format based on a fast CFD simulator and an explicit surface tracking method [5]. In their method, marbling paints are represented as enclosed contours which are moved by fluid flow to form the silhouette of marbling texture. Differing from these methods, Lu et al. develop mathematical marbling using closed-form mathematical expressions [2]. The system they developed supports five incompressible types of pattering tools which are frequently used in the traditional marbling process. The transform functions are reversible, and have explicit forward and backward forms. Therefore, we develop solid marbling by extending the 2D mathematical marbling tools to their 3D counterparts.

2.2 Solid Texture Generation

Solid textures can be used in various scenarios, including modeling natural elements like wood and stone, describing the interior or cross sections of volumetric objects, and representing the 3D objects with consistent textures both on the surface and in the interiors in rendering. Please refer to the recent survey [6] on solid texture synthesis techniques for detailed references in this field.

Procedural solid textures: Procedural approaches are the most common solid texture synthesis methods by introducing turbulence noise functions. A representative example is the famous Perlin noise introduced in 1985 [7]. Recently, Lagae et al. [8] presented a new procedural noise based on the sparse convolution and the Gabor kernel that offers accurate control over the intuitive parameters of the noise. These procedural solid texture algorithms do not save explicit copies, but generate texture colors over volumetric space by point-wise evaluations. Specifically, to evaluate the color of the point being textured, the 3D position of the point is used in the noise function on the fly. Our vector solid marbling texture representation adopts a similar point-wise color evaluation as procedural methods. It can be regarded as another kind of procedural approach for designing solid textures, but offers more flexible and intuitive control of solid texture generation and anti-aliased rendering of 3D surfaces.

Vector solid textures synthesis: Texture synthesis is a popular method for solid texture generation. This kind of method is usually based on 2D or 3D texture exemplars. However, most of these solid textures are in voxel level and resolution-dependent. The idea of compact vector representation for solid textures is introduced by Wang et al. [9]. It is resolution independent and feature preserving which is achieved by dividing a texture volume into multiple regions. Region boundaries corresponding to the original sharp features are represented by SDFs, and color variations within the regions are calculated by RBFs. Zhang et al. [10] adopt gradient solids to compactly represent solid textures following significant features guided by tensor fields derived from curves sketched by the user. Combining the procedural texture and exemplar-based synthesis seamlessly, Shu et al. [11] generate impressive vector-represented results for aggregate solid materials. However, most of the solid texture synthesis methods require synthesizing the solid texture in an entire solid block and are time consuming.

3 Solid Marbling Texture Representation

In our method, we use several 3D pattern tool functions derived from the 2D mathematical marbling method [2]

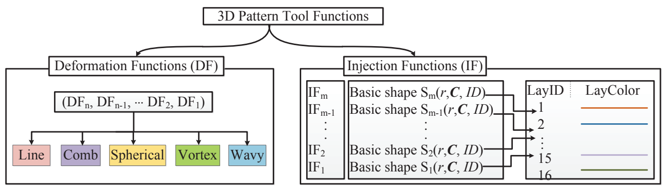
3D Pattern Tool Functions
Deformation Functions (DF)
(DF$_n$, DF$_{n-1}$, ··· DF$_2$, DF$_1$)
Line
Comb
Spherical
Vortex
Wavy
Injection Functions (IF)
IF$_m$
IF$_{m-1}$
IF$_2$
IF$_1$
Basic shape S$_m$(r,$\boldsymbol{C}$, ID)
Basic shape S$_{m-1}$(r,$\boldsymbol{C}$, ID)
Basic shape S$_2$(r,$\boldsymbol{C}$, ID)
Basic shape S$_1$(r,$\boldsymbol{C}$, ID)
LayID
LayColor
1
2
15
16

Fig. 1. The representation of solid marbling textures.

to create solid marbling textures in three-dimensional volumes. Our 3D pattern tool functions have the same incompressibility and reversibility properties as their 2D mathematical marbling tools. In 2D marbling, ink injections, lines, and circles pattern functions induce deformations in the plane. Analogically, for solid marbling, injections, lines, circles (cylinders, tori) and wavy pattern functions induce deformations in 3D space.

Algorithm 1 Solid Marbling Texture Calculation for **P**

```
Input: The coordinates of current point $\mathbf{P}$;
       The set of deformation functions $(DF_n, ..., DF_1)$;
       The set of basic shapes $(S_m, ..., S_1)$;
       The set of injection functions $(IF_m, ..., IF_1)$;
       The set of layer colors $\mathbf{LayColor}[16]$;
       The background color $\mathbf{bc}$;
Output: The color of point $\mathbf{P}$;
 1: for $i = n \to 1$ do
 2:     $\mathbf{P} \leftarrow DF_i(\mathbf{P})$
 3: end for
 4:
 5: for $j = m \to 1$ do
 6:     if $||\mathbf{P} - \mathbf{C}_j|| \le r_j$ then
 7:         return $\mathbf{LayColor}[ID_j]$
 8:     else
 9:         $\mathbf{P} \leftarrow IF_j(\mathbf{P})$
10:     end if
11: end for
12: return bc
```

3D pattern tool functions are of two types: an injection function for basic pattern creation, and a set of deformation homeomorphisms (continuous bijections) to transform the basic pattern to complex marbling effects. The injection (basic pattern) contains a set of basic shapes having as parameters: radius r, center **C** and layer ID for color. The deformation functions include line, comb, spherical, vortex, and wavy pattern operations. The solid marbling textures are defined by the composition of these mathematical functions with parameters. As shown in Figure 1, a solid marbling texture representation consists of a set of deformation functions where DF_1 is the first deformation function and DF_n is the last deformation function of the sequence, and a set of basic shapes where S_1 is the first shape and S_m is the last shape injected, and the corresponding injection functions from IF_1 to IF_m. To compute the color of a point $\mathbf{P} \in R^3$, we first use **P** as input to obtain the new position of the point under the applied deformation functions. Then, we use the point to identify which basic shape the point is located in. If it is inside the current basic shape, its color is assigned by the color of the basic shape. Otherwise, the point will be further moved for the following basic shapes. The pseudo code for calculating the color for a single point is shown in Algorithm 1. The whole texture can be obtained by computing all the points on the GPU in parallel.

In the next section, we describe our 3D pattern tool functions for solid marbling textures in detail.

3.1 Injection Function

We focus on creating solid marbling textures from scratch rather than synthesize textures from exemplars. Therefore, at the start of the process, inclusions in volumetric space will be created by injection operations. We use a series of spheres as the basic shape in the basic pattern. The previously injected shapes are pushed aside by the subsequently injected ones and their shapes may change. Given a point **P** and an injected sphere with center **C** and radius r, if $||\mathbf{P} - \mathbf{C}|| < r$, point **P** is within the sphere and takes its color. Otherwise, point **P** is transformed to a new position radially from the center **C**:

$$IF(\mathbf{P}) = \mathbf{C} + (\mathbf{P} - \mathbf{C})\sqrt{1 - \frac{r^2}{||\mathbf{P} - \mathbf{C}||^2}}. \tag{1}$$

In addition, we can reuse the 2D injection function to generate cylinders parallel to the one of the XYZ axis. Take the cylinder parallel to the Z-axis as an example, its deformation function is given by

$$\begin{aligned} IF(\mathbf{P}) &= [C_x, C_y, P_z] + \\ &([P_x, P_y, 0] - [C_x, C_y, 0])\sqrt{1 - \frac{r^2}{||(P_x - C_x)^2 + (P_y - C_y)^2||}}. \end{aligned} \tag{2}$$

Others shapes can be injected in a similar way. Once a pattern composed of basic shapes is created, we use deformation tools to advect this basic pattern to create complex marbling effects. Our mathematical function tools are named line, comb, spherical, vortex, and wavy. The inputs for these functions are the coordinate vector **P** and the tool parameters; the output is coordinate vector **Q**.

3.2 Line Pattern Function

This function provides an interface for pattern deformation by running lines through the 3D space in any direction. It is governed by:

$$\mathbf{Q} = \mathbf{P} - \frac{\omega\lambda}{d + \lambda}\mathbf{M}, \tag{3}$$

where scalars ω and λ control the maximum shift and sharpness of the shift gradient, respectively. **M** is the unit vector in the direction of the line, **N** is a unit vector perpendicular to the line. $d = |(\mathbf{P} - \mathbf{A}) \cdot \mathbf{N}|$, where **A** is a point on the line, and $\cdot$ denotes the dot product.

According to the deformation function given in Eq. (3), the transformation caused by this operation is roughly inversely proportional to the distance from the point to the tine.

3.3 Comb Pattern Function

The deformation function representing a single line stroke can be composed with others to form a comb pattern function. The comb pattern function is useful for designing a pattern with evenly spaced multiple parallel lines. The mapping function of the comb tool can be represented by composing all the lines' deformation functions directly. However, there is a problem that the computation time will be substantially increased as the number of lines increases. To make the algorithm more efficient, we adopt a single distance calculation function to represent the displacement from the closest parallel lines as

$$d' = s/2 - |fmod(d, s) - s/2|, \tag{4}$$

where d is the distance from $\mathbf{P}$ to the arbitrary line, and s is the spacing between the parallel lines. As a result, the comb pattern function is the same with the line pattern function in Eq. (4) by replacing d with d'. It is computed only once regardless of the number of the parallel lines.

3.4 Spherical Pattern Function

The spherical function is responsible for generating a spherical tine-line pattern. Under this operation, as shown in Figure 2, the point $\mathbf{P}$ is mapped to $\mathbf{Q}$ by

$$\mathbf{Q} = \mathbf{C} + (\mathbf{P} - \mathbf{C})\mathbf{R}, \tag{5}$$

where $\mathbf{C}$ is the center of the spherical tine line, and $\mathbf{R}$ is the rotation matrix calculated by:

$$\mathbf{R} = \mathbf{R}_z(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_x(\gamma), \tag{6}$$

where α, β, γ are the yaw, pitch, and roll angles, respectively. $\mathbf{R}_x(\gamma)$, $\mathbf{R}_y(\beta)$, $\mathbf{R}_z(\alpha)$ are the matrices for rotation about the X, Y, and Z axis of a coordinate system, and they are governed by

$$\mathbf{R}_x(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(b\gamma) & -\sin(b\gamma) \\ 0 & \sin(b\gamma) & \cos(b\gamma) \end{bmatrix},$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos(b\beta) & 0 & \sin(b\beta) \\ 0 & 1 & 0 \\ -\sin(b\beta) & 0 & \cos(b\beta) \end{bmatrix},$$

$$\mathbf{R}_z(\alpha) = \begin{bmatrix} \cos(b\alpha) & -\sin(b\alpha) & 0 \\ \sin(b\alpha) & \cos(b\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The parameter b controls the direction of the spherical pattern as follows:

$$b = \begin{cases} 1, & \text{if direction is clockwise,} \\ -1, & \text{if direction is counterclockwise.} \end{cases} \tag{7}$$

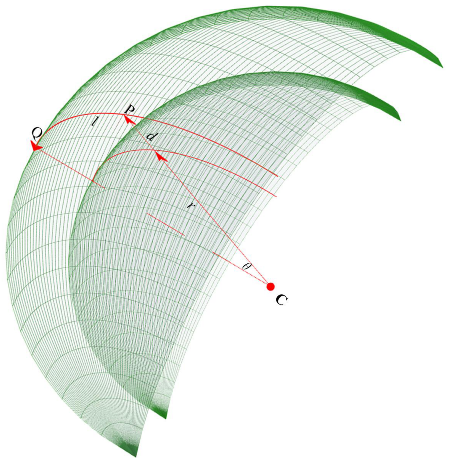

Fig. 2. Illustration of the spherical tine-line pattern function.

Any rotation matrix $\mathbf{R}$ in volume space can be obtained by these three using matrix multiplication.

The spherical motion is computed similarly to the line motion. For a given center point $\mathbf{C}$, the motion along the arc containing a point $\mathbf{P}$ is made inversely proportional to the minimum radial distance from $\mathbf{P}$ to the tine sphere. The scalars ω and λ in this operation play the same role as they are in the line pattern function. In this case, as shown in Figure 2, points are displaced along arcs around a center point $\mathbf{C}$, where its angle subtended at $\mathbf{C}$ is $\theta = l/(||\mathbf{P} - \mathbf{C}||)$, the length of the displacement arc is $l = \omega\lambda/(d + \lambda)$, and $d = ||\mathbf{P} - \mathbf{C}|| - r$. Similar to the injection function, the spherical function can also be generalized to a torus or cylinder in 3D space by replacing d with $||(P_x, P_y) - (C_x, C_y)|| - r$.

3.5 Vortex Pattern Function

The vortex pattern function is used for designing vortices patterns by using the same mapping function as the spherical pattern function (see Eq. (5)) except for the displacement term d. For this function, we set $d = ||\mathbf{P} - \mathbf{C}||$, where $\mathbf{C}$ is the center of the vortex. As a result, the moving distance of a point $\mathbf{P}$ in the vortex pattern function only depends on its position and center, while the spherical pattern function depends on the position, center, and radius.

3.6 Wavy Pattern Function

This function provides an interface for generating wavy patterns in 3D space. Given a rotation matrix $\mathbf{R}$ as described in Section 3.4, let $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ be the transformed x, y, z unit vectors:

$$\mathbf{u} = \mathbf{R} \cdot \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T,$$

$$\mathbf{v} = \mathbf{R} \cdot \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T,$$

$$\mathbf{w} = \mathbf{R} \cdot \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T.$$

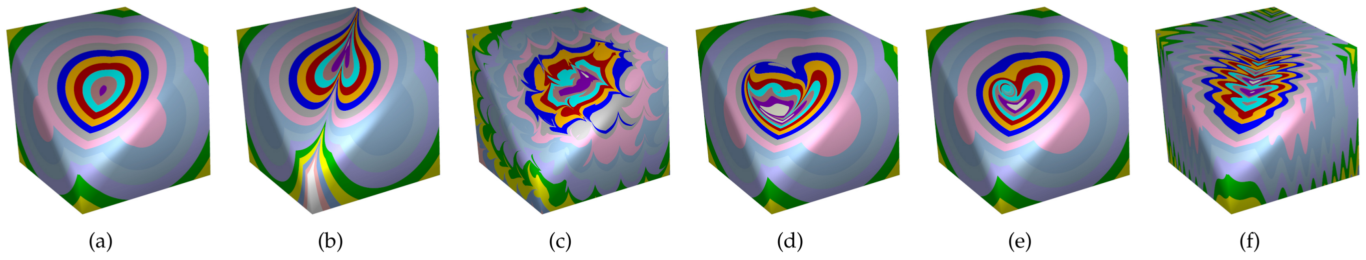

Fig. 3. Results with different pattern functions. Given the original input (a), different intricate textures emerge by applying the line pattern function (b), comb pattern function (c), spherical pattern function (d), vortex pattern function (e), and wavy pattern function (f), respectively.

The wavy pattern coordinate transformation is:

$$\mathbf{Q} = \mathbf{P} - f\left(\mathbf{P} \cdot \mathbf{u}, \mathbf{P} \cdot \mathbf{v}\right) \mathbf{w}, \tag{8}$$

where f is a sinusoidal displacement function. Various kinds of displacement functions f can be designed. We present two examples here, they are an egg-crate in Eq. (9) and the concentric ripples in Eq. (10), respectively.

$$f(x, y) = a_1 \sin(\omega_1 x + \varphi_1) + a_2 \sin(\omega_2 y + \varphi_2), \tag{9}$$

$$f(x, y) = a_1 \sin(\omega_1 (x^2 + y^2) + \varphi_1). \tag{10}$$

By interactively specifying the amplitudes a_1 and a_2, the wavelengths ω_1 and ω_2, and the phases φ_1 and φ_2, we obtain a variety of wavy paths.

Figure 3 illustrates different intricate textures generated by applying our five pattern functions to the same input shown in Figure 3(a). The combination use of these five functions can create complex visually pleasing solid marbling textures.

4 Creating Solid Marbling Textures

4.1 Creating from Scratch

Our system hides complex and abstract mathematical functions from users by providing intuitive and simple user interfaces. With these tool functions, users can drop or splash basic shapes onto the canvas using the injection functions to create a basic design. Then, by tuning the parameters of each composed function, users can create a wide variety of textures easily. All controls are exercised by dragging or clicking the mouse. Our system also provides users with flexible controls like undo and redo (history functions) to exploit their best designs interactively through trial and error. Even for a ready-made marbling textures, users can still instantly change the texture colors, which is obviously impossible for other solid texture synthesis methods. Furthermore, our system can save a user's design steps to a file for later reuse. Equipped with those user-friendly functions, our approach enables users to try out different tool functions without tedium and frustration.

4.2 Creating by Genetic Operations

In addition to direct design, our system provides an evolutionary design method especially well suited for non-experts to design solid marbling textures [12]. It has four genetic operations: random alteration, function recombination, swapping, and changing initial state. These operations create new textures by modifying either the basic pattern creation or deformation stages. Random alteration introduces variations to selected textures by modifying the parameters of the deformation functions. Function recombination is used to exchange deformation pattern functions of selected parents to create offspring. Swapping exchanges the initial pattern and deformation functions of two selected designs. "Changing the initial state" replaces the initial pattern with a new one. The workflow of the evolutionary design of solid marbling textures is:

- Step 1. Generation of an initial population: A population of 9 solid marbling textures is assembled either by random generation or by loading existing designs.
- Step 2. Selection: A user evaluates the aesthetic quality of the textures and picks the preferred designs.
- Step 3. Genetic algorithm: New solid marbling textures are created using one of the four genetic operations.
- Step 4. Repeat from Step 2 until the user is satisfied with a resulting solid marbling texture.

5 Rendering with Solid Marbling Textures

As stated in Section 3, the solid marbling texture is represented by mathematical functions and supports random access by any given 3D coordinate. Therefore, for a given 3D object, our method can render high-quality views by parallel computing the marbling texture on-the-fly. The texture rendered on the object may have aliasing artifacts because it is based on the pattern deformation. To reduce these artifacts, we have implemented antialiasing using the two-pass deferred shading technique shown in Figure 4:

- In the first pass of rendering, the positions and normals of 3D objects are rendered into the geometry buffer (G-buffer) as a series of textures.

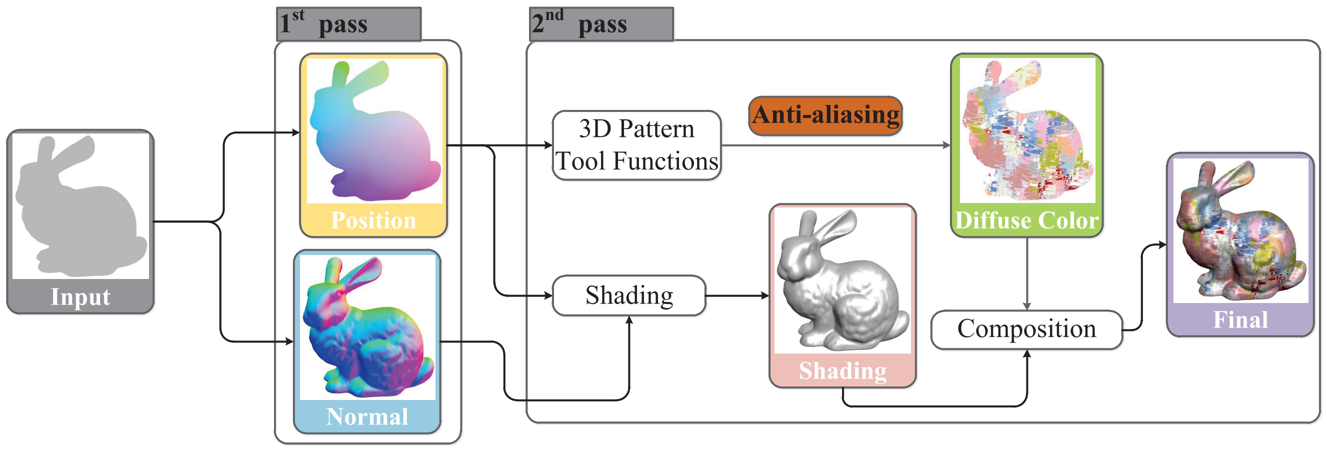

Fig. 4. Two-pass rendering process.

Dataset	#Deformation Functions	#Injection Functions	Storage (KB)	Rendering (FPS) No-antialiasing	3×3	5×5
Fig. 5(b)	10	110	4	1,556	225	84
Fig. 6(a)	16	126	5	945	143	54
Fig. 7(a)	15	27	3	1,225	312	127
Fig. 7(b)	0	83	2	1,840	429	171
Fig. 7(c)	4	150	4	873	156	59
Fig. 7(d)	49	185	11	368	57	21
Fig. 7(e)	5	29	2	2,440	530	209
Fig. 7(f)	0	500	10	272	34	13

TABLE 1
Rendering performance with different numbers of tool functions for the solid marbling textures. The performance is measured by rendering the textured 3D object in a 512×512 screen resolution on an NVIDIA GeForce GTX 660.

- In the second pass, we render screen-aligned quadrilaterals by retrieving the per-pixel attributes stored in the G-buffer in the first pass.

The position of each pixel is utilized to compute the diffuse color using the applied 3D pattern tool functions of the solid marbling texture. Specifically, given a 3D point $\mathbf{P}$, the rendering process first backward traverses the deformation functions to obtain the new coordinate of the point, and then uses the new coordinate to backward traverse the injection functions to obtain the diffuse color as the pseudo code shown in Algorithm 1. The position and normal of each pixel are together used for calculating the shading. The final composition is obtained by combining the diffuse color and shading effects. During this process, super-sampling is employed as an antialiasing scheme to obtain high-quality texture mapping effects. Figure 5 shows the improved rendering result with antialiasing (Figure 5(b)) in contrast to the one without antialiasing (Figure 5(a)). Figures 5(c-d) are the results of zooming in on Figures 5(a-b), respectively.

6 Results and Discussions

We have implemented our rendering algorithm on the GPU to obtain high efficiency at interactive frame rates. Our computer is equipped with a 3.2GHz Intel Core i5-3470 CPU and an NVIDIA GeForce GTX 660 graphics card. The implementation of our rendering algorithm uses HLSL pixel shaders. Similar to the deferred shading system, the rendering time of solid marbling texture mapping is proportional to the number of tool functions and screen resolution, while independent of the viewpoint and model complexity. For the 3D objects in this paper, the implementation achieves real-time performance at a screen resolution of 512 x 512 (in pixel), even with 5×5 supersampling antialiasing.

Table 1 lists the performance of 3D pattern functions (including deformation functions and injection functions), the memory consumption, and rendering performance for different solid marbling textures applied to 3D objects. At fixed resolution, the rendering cost and memory storage is linearly dependent on the number of 3D pattern tool functions. The rendering performance decreases with an increasing number of tool functions and supersampling grids.

Compared to other vector solid texture synthesis methods, the vector representation proposed by Wang et al. [9] consumes 17%-26% of the storage of a bitmap version at the same grid resolution for a solid texture with three 8-bit color channels. The method by Shu et al. [11] has made the storage no more than 1MB

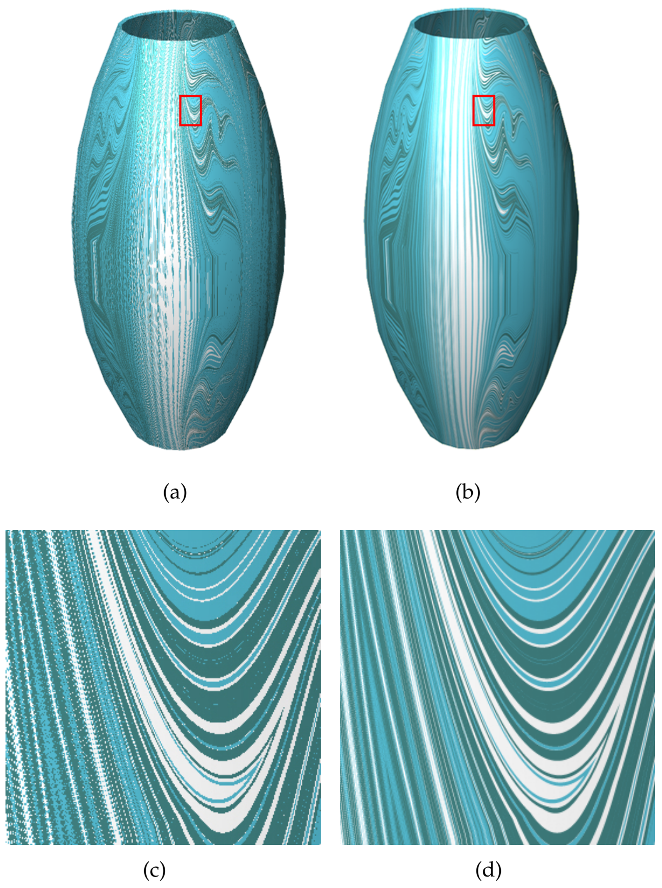
Fig. 5. Comparison between non anti-aliasing (a) and anti-aliasing (b) solid marbling results applied on a vase. (c-d) are the results of zooming in on the vase in (a-b), respectively.

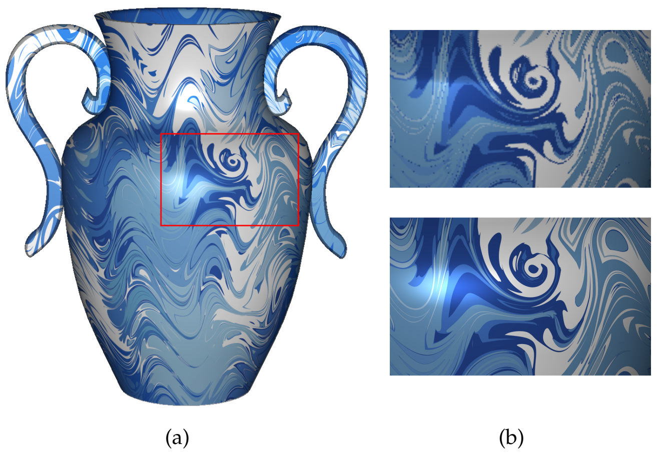
Fig. 6. Comparison between a vector and a bitmap solid marbling texture. (a) A vase rendered with a solid marbling texture. (b) The upper inset shows a close-up view of the bitmap texture on the vase, while the lower inset is the same view of the vector texture on the vase. The bitmap texture generated from rasterizing the vector version has the resolution of $512 \times 512 \times 512$.

memory space even without compression. As shown in Table 1, the storage consumption of our method is much smaller than that of current vector solid texture techniques, occupying only a few kilobytes.

Most of solid texture synthesis methods generate bitmap solid textures [6]. Similar to traditional 2D bitmap textures, high resolution bitmap solid textures consume too much of GPU memory while low resolution causes blurry results when mapping onto 3D objects. Fortunately, our vector solid marbling textures solve these problems because of their resolution independent representation. Very sharp curves and details even at high zoom levels on 3D objects can be preserved while reducing the texture memory consumption. A comparison between vector solid marbling texture and bitmap solid marbling textures is shown in Figure 6. The bitmap solid marbling texture is derived by rasterizing the same vector solid marbling texture at the resolution of $512 \times 512 \times 512$.

The previous mathematical marbling method also supported rendering surface details on 3D objects [2]. However, there are significant differences between these techniques. The method of Lu et al. [2] applies a 2D marbling texture onto a 3D object. In comparison, the newly developed method not only renders vector textures, but also completely eliminates texture distortions and complex plane parameterizations.

Figure 7 demonstrates some representative results of our vector solid marbling textures, effortlessly applied onto a variety of 3D objects without planar parametrization. The upper inset shows the mapping results on different 3D objects, and the details are presented at the bottom (see Figure 7(a-f)).

Limitations. Compared to CFD-based approaches, our mathematical marbling suffers from a lack of "dynamic realism" in the design process. Although our approach can achieve real time performance, the employed supersampling scheme is time consuming. Since aliasing often occurs at high-frequency regions in the solid marbling textures, it is desirable to develop a better antialiasing method.

7 Conclusions

In this paper we have introduced a novel method for generating solid marbling textures by using 3D pattern tool functions. We have successfully extended the 2D mathematical marbling method to three dimensions. The experiments show that the method produces pleasing visual effects. The representation of our solid marbling texture is compact, supports random access, and is resolution independent. Our framework can render high-quality solid textures preserving sharp features in real time on the GPU. To facilitate the authoring of solid marbling textures, we have introduced an intuitive user interface and a genetic algorithm. Textures can be easily generated both by direct design and guided evolution.

Our approach provides a new means to design specific solid textures.

In addition to creating solid marbling textures from scratch and guided evolution, converting a given 2D marbling texture exemplar to a solid marbling texture is an interesting direction for further research. Furthermore, we are excited about the possibilities of using pattern tool functions for 3D modeling and editing in the future.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments to improve the manuscript. Shufang Lu was supported by the National Natural Science Foundation of China (Grant no. 61402410) and Zhejiang Provincial Natural Science Foundation of China (Grant no. LQ14F020004). Xiaogang Jin was supported by the National Natural Science Foundation of China (Grant no. 61472351).

References

[1] X. Jin, S. Chen, and X. Mao, "Computer-generated marbling textures: a gpu-based design system," *IEEE Computer Graphics and Applications*, vol. 27, no. 2, pp. 78–84, 2007.
[2] S. Lu, A. Jaffer, X. Jin, H. Zhao, and X. Mao, "Mathematical marbling," *IEEE computer graphics and applications*, vol. 32, no. 6, pp. 26–35, 2012.
[3] J. Xu, X. Mao, and X. Jin, "Nondissipative marbling," *IEEE Computer Graphics and Applications*, vol. 28, no. 2, pp. 35–43, 2008.
[4] H. Zhao, X. Jin, S. Lu, X. Mao, and J. Shen, "Atelierm++: a fast and accurate marbling system," *Multimedia Tools and Applications*, vol. 44, no. 2, pp. 187–203, 2009.
[5] R. Ando and R. Tsuruno, "Vector graphics depicting marbling flow," *Computers & Graphics*, vol. 35, no. 1, pp. 148–159, 2011.
[6] N. Pietroni, P. Cignoni, M. Otaduy, R. Scopigno *et al.*, "Solid-texture synthesis: a survey," *IEEE Computer Graphics and Applications*, vol. 30, no. 4, pp. 74–89, 2010.
[7] K. Perlin, "An image synthesizer," *ACM Siggraph Computer Graphics*, vol. 19, no. 3, pp. 287–296, 1985.
[8] A. Lagae and G. Drettakis, "Filtering solid gabor noise," in *ACM Transactions on Graphics (TOG)*, vol. 30, no. 4. ACM, 2011, p. 51.
[9] L. Wang, K. Zhou, Y. Yu, and B. Guo, "Vector solid textures," in *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4. ACM, 2010, p. 86.
[10] G.-X. Zhang, Y.-K. Lai, and S.-M. Hu, "Efficient synthesis of gradient solid textures," *Graphical Models*, vol. 75, no. 3, pp. 104–117, 2013.
[11] Y. Shu, Y. Qian, H. Sun, and Y. Chen, "Efficient texture synthesis of aggregate solid material," *The Visual Computer*, vol. 30, no. 6-8, pp. 877–887, 2014.
[12] S. Lu, P. Mok, and X. Jin, "From design methodology to evolutionary design: An interactive creation of marble-like textile patterns," *Engineering Applications of Artificial Intelligence*, vol. 32, pp. 124–135, 2014.

Shufang Lu is an assistant professor at the College of Computer Science and Technology at Zhejiang University of Technology. Her research interests include marbling simulation, non-photorealistic rendering and image processing. She received her PhD in computer science from Zhejiang University. Contact her at sflu@zjut.edu.cn.

Xiaogang Jin is a professor at the State Key Laboratory of CAD & CG at Zhejiang University. His research interests include implicit surface computing, cloth animation, crowd and group animation, texture synthesis, and digital geometry processing. Jin received his PhD in applied mathematics from Zhejiang University. Contact him at jin@cad.zju.edu.cn.

Aubrey Jaffer is a mathematician and data scientist at Digilant in Boston MA. His research interests include convection, radiative transfer, numerical analysis, algebraic geometry, symbolic algebra, and space filling curves. Aubrey has a BS in Mathematics from the Massachusetts Institute of Technology. Contact him at agj@alum.mit.edu.

Fei Gao is a professor at the College of Computer Science and Technology at Zhejiang University of Technology. He received his PhD degree in mechanical engineering from Zhejiang University in 2004. His research interests include image processing, computer vision, and computer-aided design. Contact him at feig@zjut.edu.cn.

Xiaoyang Mao is a professor at the University of Yamanashi in Japan. Her research interests include non-photorealistic rendering, texture synthesis, perception and affect based rendering and visualization. Xiaoyang Mao recieved her BS in computer science from Fudan University in China and her MS and PhD in computer science from the University of Tokyo. Contact her at mao@yamanashi.ac.jp.

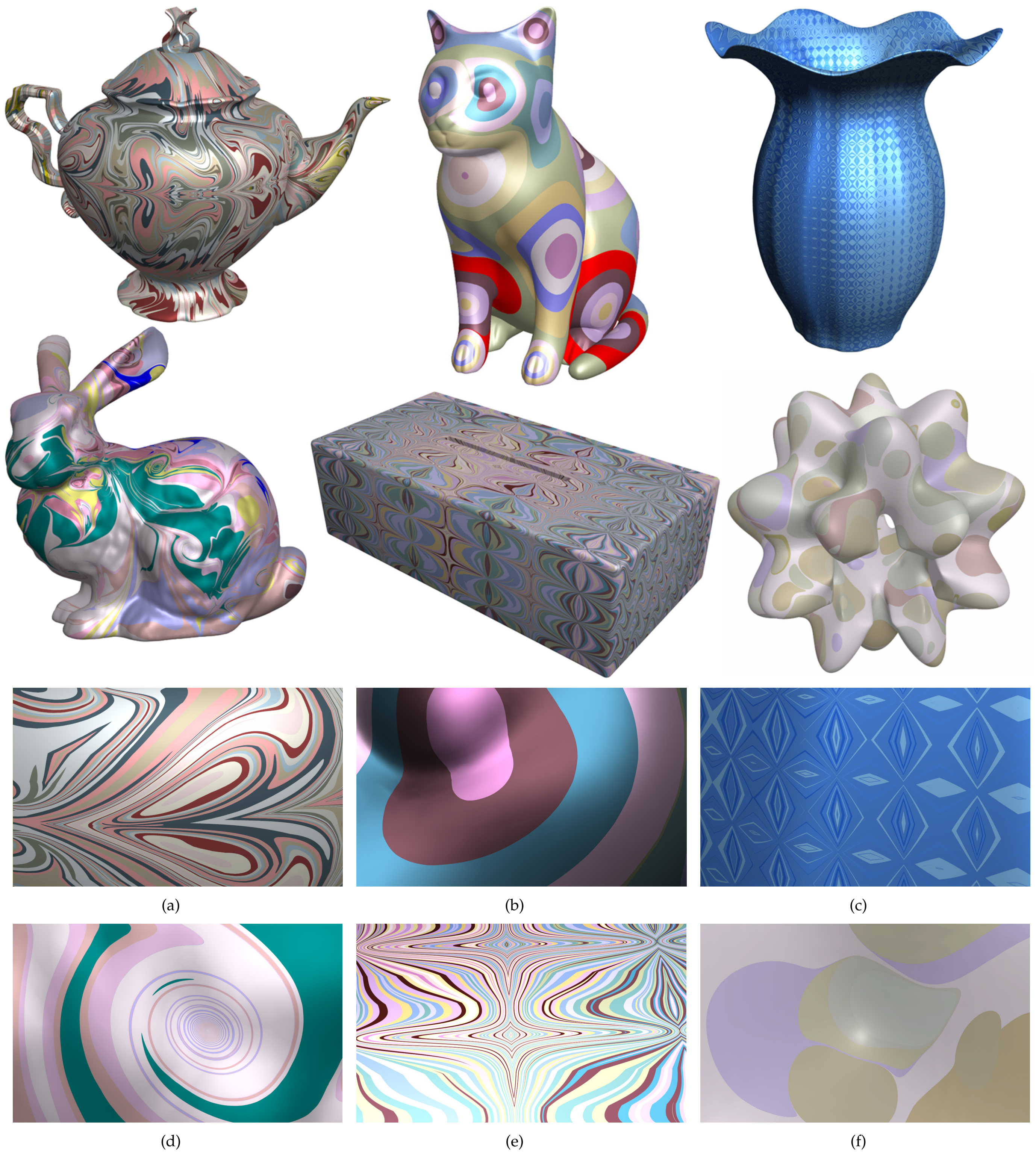

Fig. 7. Results of our vector solid marbling textures mapped onto different 3D objects. (a-f) are the details of zooming in on each object.