Volume Illustration Using Wang Cubes

AIDONG LU
University of North Carolina at Charlotte
and
DAVID S. EBERT, WEI QIAO, MARTIN KRAUS, and BENJAMIN MORA
Purdue University

To create a new, flexible system for volume illustration, we have explored the use of Wang Cubes, the 3D extension of 2D Wang Tiles. We use small sets of Wang Cubes to generate a large variety of nonperiodic illustrative 3D patterns and texture, which otherwise would be too large to use in real applications. We also develop a direct volume rendering framework with the generated patterns and textures. Our framework can be used to render volume datasets effectively and a variety of rendering styles can be achieved with less storage.

Specifically, we extend the nonperiodic tiling process of Wang Tiles to Wang Cubes and modify it for multipurpose tiling. We automatically generate isotropic Wang Cubes consisting of 3D patterns or textures to simulate various illustrative effects. Anisotropic Wang Cubes are generated to yield patterns by using the volume data, curvature, and gradient information. We also extend the definition of Wang Cubes into a set of different sized cubes to provide multiresolution volume rendering. Finally, we provide both coherent 3D geometry-based and texture-based rendering frameworks that can be integrated with arbitrary feature exploration methods.

Categories and Subject Descriptors: I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Interaction techniques*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Color, shading, and texture*

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Wang Tiles, Wang Cubes, volume illustration, volume rendering, nonphotorealistic rendering, scientific illustration

ACM Reference Format:

Lu, A., Ebert, D. S., Qiao, W., Kraus, M., and Mora, B. 2007. Volume illustration using Wang Cubes. ACM Trans. Graph. 26, 2, Article 11 (June 2007), 18 pages. DOI = 10.1145/1243980.1243985 http://doi.acm.org/10.1145/1243980.1243985

1. INTRODUCTION

Rendering and exploring features in scientific datasets is important to many research areas, including medicine, biology, and archaeology. Traditionally, direct volume rendering and isosurfacing techniques have been used for visualization and exploration of these datasets. However, these traditional techniques alone are not sufficient to highlight important features in datasets or deemphasize unimportant detail. These techniques also lack the flexibility to add additional information and detail into the resulting visualization for educational and explanatory uses. Therefore, inspired by the effectiveness of scientific and medical illustration, researchers have started to render their scientific datasets in an illustrative way by simulating certain scientific illustration techniques, including small primitives [Saito 1994], hatching [Nagy et al. 2002], and stippling [Lu et al. 2003]. Most recently, Owada et al. [2004] presented an interactive system for designing and browsing volumetric illustrations through 2D synthesized textures. Many of these illustrative techniques, however, require specific rendering approaches for each primitive and can use large amounts of storage for the primitives. We have, therefore, developed a system based on the 3D extension of Wang Tiles [Cohen et al. 2003], Wang Cubes, to provide enriched illustrative volume detail, compact storage, and interactive rendering of scientific datasets for more expressive rendering, education, and training applications.

1.1 Motivation

As with previous work in illustrative visualization, our work is inspired by the ability of illustrations to succinctly convey features, remove visual clutter and unimportant detail, and enrich the visual representation with textural detail. As shown in Figure 1, many illustrators apply patterns and textures to objects to add structural and textural detail for more informative communication. Since the

This work was supported by the National Science Foundation under Grant Nos. 0081581, 0121288, 0328984, 0633150, Department of Energy under Grant No. DE-FG02-06ER25733, and the U.S. Department of Homeland Security.

Authors' addresses: A. Lu, Computer Science Department, University of North Carolina at Charlotte, 9201 University City Blvd, Charlotte, NC 28223; email: aidong.lu@uncc.edu; D. S. Ebert, W. Qiao, M. Kraus, and B. Mora, Purdue University, School of Electrical and Computer Engineering, 465 Northwestern Ave, West Lafayette, IN 47907; email: {ebertd, wqiao, kraus, bmora}@purdue.edu.

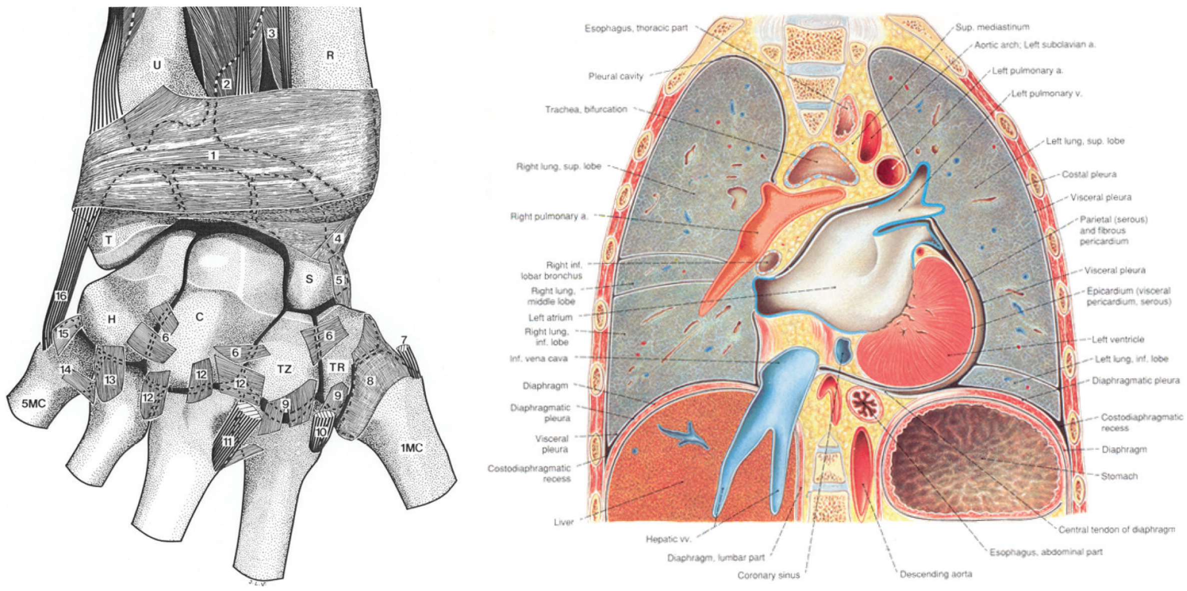

Fig. 1. Patterns and textures are commonly employed in scientific illustrations. The two example illustrations come from *Atlas of Human Limb Joints* [1990] and *Sobotta Atlas of Human Anatomy* [1990], respectively.

resolution of volumetric datasets is currently too low to allow for the capture of such textural details, it could be useful to have a computer graphics rendering method that allows for enriching acquired volumetric datasets with simulated detail. For example, Figure 1 (left) shows the human limb joints of the right wrist [Guyot 1990]. In this black and white image, bones (C, H, 1MC, 5MC, R, S, T, TR, TZ, U) are drawn with stipples, tendons (7, 10, 11, 16) with wide line patterns, and muscles and ligaments with thin line patterns. The right image shows the thoracic viscera of an adult human after removal of the anterior thoracic wall [Staubesand and Taylor 1990]. Almost all the structures, including the heart, lung, skin, vessels, diaphragm, etc., are all rendered with different colored textures.

In scientific illustrations, although the patterns and textures are usually chosen to simulate the shape and color of the real objects, they do not have to be exactly the same. Sample textures or even simple repeated point and line patterns are also commonly used in many illustrations [Hodges 1989]. By applying specific patterns or textures to objects, the human eye can easily pick out the objects within an image even when several objects or parts overlap. This fact demonstrates the great potential for using patterns and textures in volume rendering, which can show more internal structure than surface geometry alone. Scientific illustrators [Hodges 1989] also point out that almost every rendering technique or style has its advantages and disadvantages. For example, points are perfect to illustrate surfaces, while lines are good for silhouettes. Therefore, different results may be achieved when an object is rendered in various styles. The choice of rendering primitives plays an important role in reflecting the textures and properties of objects. In many applications, one single rendering technique is not sufficient. Instead, patterns composed of various primitives are widely used in scientific illustrations as shown in Figure 1. In nonphotorealistic rendering (NPR), the patterns are usually evenly, randomly, or nonperiodically distributed to achieve artistic effects. Compared to the artistic work of scientific illustration, there have been no competitive direct volume rendering techniques that are as rich in their choices of textures and rendering styles.

For a visualization system, the advantages of using primitive and pattern-based rendering must be weighed against the storage and running time requirements of an interactive visualization system. Many NPR techniques need to store the positions of all the primitives (e.g., points, lines) to render an object. When constraints from adjacent regions are added, a local search is needed for every primitive, such as when distributing points in a volume evenly or extending a long line segment on an object. Since the primitives are usually assigned per-voxel to achieve these effects, extra space and time are needed. These disadvantages can be overcome by using a set of patterns which saves time and space, gives the closest appearance to real objects, and provides the capability of various rendering styles.

1.2 Texture and Pattern for Shape Perception

Illustration techniques are very effective at providing appropriate visual cues to understand three-dimensional objects, their shape, and their spatial relationships. Research efforts in both computer graphics and visual perception have explored shape perception by textures or patterns. Healey et al. [2004] present a nonphotorealistic visualization method based on brush strokes, which uses the results from psychophysics that many properties of a texture (e.g., color, orientation, size, and contrast) are detected by the low-level visual system and can be used to encode information. It has been widely accepted that "shape perception can be enhanced by the addition of appropriate texture under generic viewing and shading conditions" [Kim et al. 2004]. Various aspects of shape perception from texture have also been studied [Interrante et al. 2002; Kim et al. 2003; Zaidi and Li 2002; Li and Zaidi 2000; Knill 2001; Todd et al. 1997; Rosenholtz and Malik 1997; Ware and Knight 1995; Wanger et al. 1992]. These research results show that general patterns and textures can affect human perception and understanding. Therefore, as in scientific illustrations, appropriate usage of patterns and textures could be more effective than traditional visualization methods that use a uniform color or colormaps to render an object.

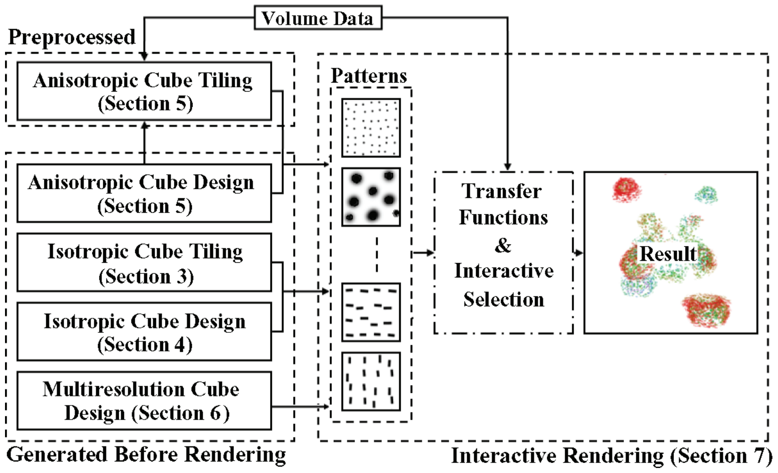
Volume Data
Preprocessed
Anisotropic Cube Tiling (Section 5)
Anisotropic Cube Design (Section 5)
Isotropic Cube Tiling (Section 3)
Isotropic Cube Design (Section 4)
Multiresolution Cube Design (Section 6)
Generated Before Rendering
Patterns
Transfer Functions & Interactive Selection
Result
Interactive Rendering (Section 7)

Fig. 2. The system diagram. The left column shows our techniques to generate various 3D patterns and textures which are then sent to the volume rendering framework. By using Wang Cubes, users can interactively adjust transfer functions and select objects of interest for rendering without redesigning the patterns or textures for any volume. The 3D patterns and textures can be used to achieve multiple illustrative styles and provide additional information compared to traditional renderings with color transfer functions.

1.3 Our New Approach

Based on these advantages of textures for efficient and illustrative visual perception and object detection, we have developed a new interactive volume illustration system using textures/patterns to effectively illustrate volumetric datasets. Using textures provides not only enriched illustrative visualizations but also enables multiple rendering styles within the visualization, providing the flexibility needed for illustrative visualization. The majority of NPR techniques are designed for one or two specific rendering styles, although several papers have achieved multiple styles by successfully using textures (e.g., image analogies [Hertzmann et al. 2001] and real-time hatching [Praun et al. 2001]).

To provide the advantages and flexibility of illustrative volume visualization while meeting the requirements of compact storage, little preprocessing, and interactive rendering, we use Wang Cubes to tile the patterns throughout space. We explore both isotropic and anisotropic cube design with small sets of Wang Cubes. The cubes can be filled with geometric primitives, patterns, and textures to generate a large number of 3D patterns. The design of the cubes and the cube tiling guarantee a consistent pattern over the whole volume and saves storage. All the cube contents and cube tilings are quick to generate, and special care is taken to ensure their temporal and spatial coherence. Our framework, shown in Figure 2, can be integrated with arbitrary transfer function and feature selection methods to select features in a volume and to assign the patterns and styles for the features during interactive rendering and exploration. We have developed two systems to effectively render object features: a geometry-based system that renders OpenGL geometry primitives and a texture-based system that is implemented in a fragment program. In this article, we show that Wang Cubes are a convenient tool to add various illustrative nonperiodic details to volume datasets with both compact storage and very little preprocessing.

We begin by summarizing previous work in Wang Tiles, NPR, and volume illustration. In Section 3, we extend the Wang Tile stochastic tiling algorithm [Cohen et al. 2003] to 3D cube tilings and modify it for multipurpose tilings. In Section 4, we discuss three kinds of isotropic pattern generation which have similar properties over the entire volume. By using information from a dataset (volume data, curvature direction, and tangent orientation) in Section 5, we automatically generate cubes with anisotropic patterns which utilize the features of a volume. In Section 6, we extend the definition of Wang Cubes into a set of different sized cubes to provide multiresolution renderings. The issues of direct volume rendering with Wang Cubes are discussed in Section 7. Finally, we discuss the advantages and disadvantages of using Wang Cubes in volume rendering.

2. RELATED WORK

Wang Tiles [Wang 1961, 1965] are square tiles used to generate a plane tiling. They are placed on a plane according to their face color rules to compose the desired pattern. Analogous to Wang Tiles, Culik and Kari [1995] introduced Wang Cubes with colored faces. Several researchers have used Wang Tiles to generate patterns in computer graphics. Stam [1997] employed 16 Wang Tiles to simulate water-like surfaces. Neyret and Cani [1999] used triangles with homogeneous textures to tile surfaces. Cohen et al. [2003] presented a stochastic method to design sets of Wang Tiles with different tile numbers and generated nonperiodic images automatically with sample textures. They also indicated that Wang Cubes could be used for three-dimensional applications in the future. More recently, Sibley et al. [2004] performed video synthesis and geometry placement by using Wang Cubes, and Lagae and Dutré [2005] proposed generating procedural object distribution functions using Wang Tiles. In this article, we propose using Wang Cubes to compose 3D textures for direct volume visualization.

In addition to Wang Tiles, other methods are used to generate patterns or textures for image synthesis and object rendering. Many methods try to obtain a Poisson disc distribution to randomly and uniformly place primitives for rendering. Hiller et al. [2003] use a user-controllable relaxation of a Voronoi tessellation to adjust the distribution of stipples and arbitrary shapes to render two-dimensional images, taking into account object boundaries. Interrante [1997] effectively uses patterns and curvature directions to illustrate surfaces within a volume dataset. Patterns and textures are also used to achieve better visual effects in volume rendering, such as glyphs in vector fields or flow visualizations [Wittenbrink et al. 1996]. Kirby et al. [1999] utilized concepts from paintings to

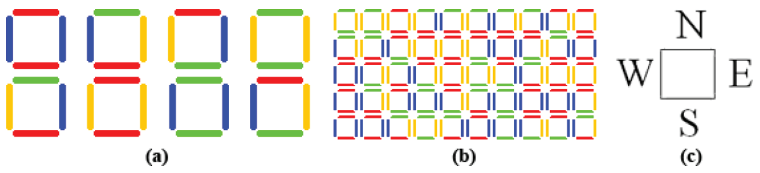

Fig. 3. (a) A set of 8 Wang Tiles designed with face colors (NSWE) and 2 colors for each edge. (b) A nonperiodic 5 × 10 tiling. (c) The 4 tile edge directions.

combine multiple data values in an image for 2D flows. As opposed to these studies, our patterns and textures are generated for general volumes without knowledge of the exact geometric mesh or shape.

Using NPR techniques in volume rendering has been shown to be effective in the visualization of three-dimensional (volume) data by highlighting portions of data, improving three-dimensional structure and shape cues, focusing viewer attention, and reducing visual clutter. Ebert and Rheingans [2000] combined nonphotorealistic rendering and volume rendering techniques to enhance important features and regions. Treavett et al. [2001] implemented artistic procedures in various stages of the volume-rendering pipeline. Lum and Ma [2002] implemented a combination of NPR methods at interactive frame rates for large datasets with a parallel hardware-accelerated rendering technique. Hadwiger et al. [2003] combined nonphotorealistic techniques to effectively render segmented datasets with high quality on consumer graphics hardware. The usage of silhouettes and contours has also been explored by several researchers [Csébfalvi et al. 2001; Kindlmann et al. 2003; Burns et al. 2005].

3. WANG CUBES

As shown in Figure 2, our framework first generates 3D patterns and textures in the preprocessing phase, then feeds these volumetric patterns and textures into the interactive rendering phase to illustrate a volume dataset. During the rendering phase, transfer functions or feature extraction methods are used to control the rendering properties of the selected 3D patterns and textures to form the shape and appearance of the volumetric objects (the details are discussed in Section 7). To save texture space and processing time, we use a mathematical tool called Wang Cubes to generate various 3D patterns and textures, which is the 3D extension of 2D Wang Tiles [Cohen et al. 2003]. A set of Wang Cubes can quickly fill any sized volume by following certain rules, making the rendering of arbitrary volume datasets more convenient. Two key components are generated for a set of Wang Cubes: a *cube tiling* and the *cube contents*. The cube tiling mainly determines the repetitive property of the 3D patterns or textures, while the cube contents determine the type (primitive, size, opacity, color, etc.) of the 3D patterns or textures. Different cube tilings and cube contents are combined to generate a large number of 3D patterns and textures, providing many choices for the rendering phase.

In the following, we briefly review Wang Tiles and then explain the general rules of Wang Cubes and the cube tilings. In later sections, the generation of isotropic cube contents and the generations of anisotropic cube tilings and cube contents are discussed.

3.1 Wang Tiles

Wang Tiles [Wang 1961, 1965] are square tiles used to generate a plane tiling. Each edge of the square tile is assigned a constant parameter (color) that is used to determine how they can be placed together to tile a plane; they are placed in the plane edge-to-edge only if the adjacent edges share the same color. Figure 3(a) shows an example Wang Tile set with 8 tiles. The edges along North(N)/South(S) and West(W)/East(E) each use exactly two colors. Figure 3(b) shows a 5 × 10 tiling created using this tile set. Wang Tiles are not allowed to be rotated; therefore, the edges in the same direction should share the same color set, while the colors in different directions are totally independent.

For practical usage in computer applications, Cohen et al. [2003] presented a stochastic tiling procedure to generate the tile patterns automatically. This stochastic tiling procedure guarantees that the generated tilings are nonperiodic, which is usually more visually pleasing than simple repeated patterns.

When designing 2D patterns and textures to fill each tile, the patterns and textures along the same colored edge should be the same to satisfy the tiling procedure. Therefore, new larger nonperiodic patterns or textures can be easily generated by putting the pattern or texture of each tile into the corresponding position of the tiling.

3.2 Wang Cube Tiling Procedure

Wang Cubes are a 3D extension of 2D Wang Tiles. As with Wang Tiles, Wang Cubes are cubes with colored faces in the sense that two cubes can be put together only if the adjacent faces have matching colors, as shown in Figure 4. Similarly, we denote the cube faces as North(N), South(S), West(W), East(E), Front(F) and Back(B). Similar to Wang Tiles, Wang Cubes are not supposed to be rotated. Therefore, two faces in the same direction (NS, WE, and FB) must share one set of colors to ensure that they can be put together, while the face colors on different directions are independent.

We extend the stochastic nonperiodic tiling process of 2D Wang Tiles [Cohen et al. 2003] to 3D Wang Cubes. Without loss of generality, we fill the volume from West-to-East, from North-to-South, and from Front-to-Back. Apart from the cubes on the boundary of the volume, each position has three constraints (each cube must have N, W, and F faces that match the S, E, and B faces already placed). To determine how to tile the space, we assume each face has n possible colors; therefore, the NWF faces create n^3 cases. As long as there is at least one cube in the cube set for each NWF case, a valid tiling of a volume exists. We can start from the NWF corner, use the 2D tiling process to tile the first slice in the volume by treating the NSWE face colors as the edge colors in Wang Tiles, and then tile the rest of the slices from Front-to-Back by adding the additional constraint that the front color of the cube should match the back color of the cube in the same position of the previous slice.

As long as there is one cube for each NWF case, the previous cube tiling process can be guaranteed to compose a valid tiling. The cube face colors at each direction need to be larger or equal to 2 to generate nonperiodic cube patterns. When there are at least 2 cubes for each NWF case, every step in the cube tiling process has at least 2 valid choices and a solution can be randomly chosen from them. Analogous to Wang Tiles, this process is similar to the random process of a coin flip; therefore, the cube tiling is guaranteed to be nonperiodic. Assuming m cubes are generated for each NWF case, $m \times n^3$ cubes are needed. Under our cube tiling process, the

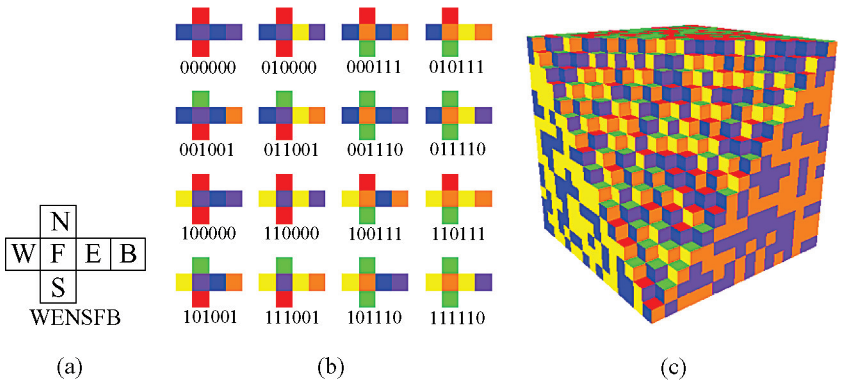

Fig. 4. (a) shows the face directions for flattened cube faces, and the sequence of cube face directions. (b) shows a set of 16 Wang Cubes with colored face where each face has 2 colors. The color numbers for the 6 faces are shown below each cube in the sequence shown in (a) (bottom). (c) shows a truncated volume with a nonperiodic tiling.

minimum number of cubes is 16, which requires 2 colors for each face and 2 cubes for each NWF case.

3.3 Reducing the Repetitive Appearance

After deciding the number of cube face colors and the number of cubes for each NWF case, we need to design the face colors for every cube. One common problem with the tiling generated by Wang Tiles and Wang Cubes is the repetitive appearance. Since the tiling is generated with a finite set of samples, the same composite pattern may repeat itself in different locations of the entire tiling, resulting in a local repetitive appearance. Even though the tiling is nonperiodic on the whole, the repetitive appearance may reduce the image quality. Kari [2000] has proven that the repetitive appearance is inevitable with any finite tile sets. Larger sets of cubes generally have a less repetitive appearance since there are more cubes/samples in the composed patterns. However, they require more storage space and memory swapping; therefore, they are not desirable for applications where storage is limited.

Although we cannot fundamentally avoid this problem, we can reduce the repetitive appearance by properly designing the colors for the cubes. To ensure that the set of the cubes can produce nonperiodic patterns with our tiling process, we need to assign the NWF colors of the cubes in a way that has at least two cubes for each NWF case. For a set of 16 cubes, there are exactly two cubes for each NWF case. Under this restriction, we choose the SEB colors to meet two criteria to ensure an even distribution of the cubes in a volume. First, the SEB colors should cover all the color cases. Second, for every color on each of the NWF faces, the NS/WE/FB pairs in the cube set should have both the same and opposite colors. If the latter cannot be satisfied, we favor the cubes whose opposite faces have different colors because they are less likely to produce repetitive patterns.

We also reduce the repetitive appearance by adding varying probabilities for each cube. Initially, each cube has an equal chance of being selected as long as its face colors match the existing cube patterns. During the tiling procedure, we use a small count table to gather the local occurrence for every cube. The probability for selecting each cube changes according to the surrounding cube combinations: it is decreased if the cube appears more than the average frequency (total cube number divided by the observing window size) and increased if it appears less than the average frequency. We normally use the tiling procedure with varying probabilities for smaller volumes and without varying probabilities for larger volumes since, if the volume is large enough, every cube in the set will have an equal occurrence. But locally, as shown for 8^3 volumes, the average standard deviation of the cube set from the tiling with varying probabilities is 1/2 of the deviation from the tiling without varying probabilities. Therefore, the cube patterns are more evenly distributed.

Another application of varying the probability for the Wang Cubes is by generating cube patterns with preferences. With the local probability control, we can favor some cubes in the set more than others and, at the same time, guarantee the nonperiodic property of the cube pattern. We can also arrange the cube patterns according to a predefined probability field to smoothly vary the generated pattern. Figure 5(b)-(d) show the results of varying probabilities. Figure 5(b) and (c) show two $32 \times 32 \times 1$ results for controlling average cube occurrences. Figure 5(d) shows the result for generating cube patterns for a $32 \times 64 \times 1$ predefined field pattern.

4. ISOTROPIC PATTERN DESIGN

After tiling the Wang Cubes in a volume, the 3D space is filled with empty cube frames that are nonperiodically distributed. We still need to generate the cube contents to fill in the tiling so that we can compose a 3D pattern or texture to use in the final volume rendering. An essential point of using Wang Cubes is that the cube face colors are only used to generate the cube tiling, while the cube contents can be filled with arbitrary patterns or textures. The cube tiling procedure ensures the nonperiodic property, while the cube content design also plays a key role in the quality of the final composed 3D pattern.

Since multiple cube pattern designs can provide flexibility to render a volume with different styles, we propose two types of cube pattern generation techniques: isotropic design and anisotropic design. In this section, we discuss isotropic patterns, which have the same properties over the entire volume, such as a point volume with equal densities. In the next section, we use anisotropic patterns to match certain data features, and we generate them with additional constraints during the tiling process. With various pattern choices, the users have the flexibility to choose the patterns for different purposes. Simulating scientific illustrations, we usually choose patterns that can be visually distinguished from the surroundings or possess the closest appearance to a specific real object.

For isotropic cube design, our goal is to create a large number of illustrative patterns and textures with as small a set of cubes as possible. Therefore, we only need a small storage overhead to

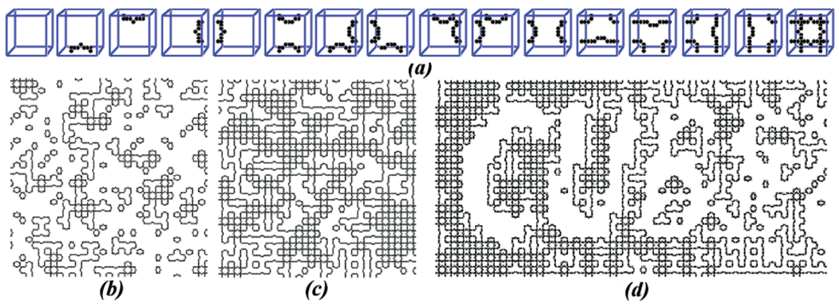

Fig. 5. (a) A set of Wang Cubes generated interactively from our system. (b) A uniform distribution where all the cubes in (a) have an equal chance. (c) A distribution favoring darker cubes. The darkness of a cube is calculated as the sum of the 6 face color values, which are 1 if the face has a point on it, 0 otherwise. (d) A tiling generated by a predefined field (a CUBE pattern) with the restriction changing from strict to loose from left-to-right. Therefore the right side has more random patterns according to the predefined field than the left side.

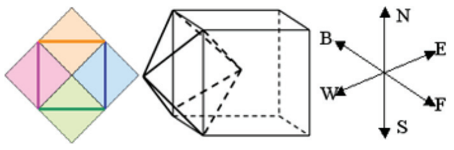

Fig. 6. The cube synthesis procedure. (Left) Wang Tile generation process: each tile is synthesized with four sample diamonds. (Middle) The octahedron for the W cube face. The cube generation process is a 3D extension of the 2D tile synthesis. (Right) The cube face directions.

provide many pattern choices for the final volume rendering. In Section 3.2, we showed that 16 is the minimum number for a cube set to generate nonperiodic cube tilings under the stochastic tiling procedure. We use a 16 cube set to generate all of our isotropic patterns and textures. Depending on what is inside the cubes, our methods for designing the cube contents can be divided into two types namely, texture cubes and primitive cubes.

4.1 Texture Cubes

After the texture cubes are designed, they are used to compose new large 3D textures using the tiling process. We generate texture cubes with both an interactive drawing program and an automatic texture synthesis approach.

For texture generation in our system, the user can interactively design cubes by manually drawing the cube contents using our drawing program with a brush model such as a 2D or 3D sample. Figure 5(a) shows an interactive design example. The patterns are drawn along a vertical plane within the cubes.

The cubes can also be automatically generated from 3D sample textures, extending the 2D tile method [Cohen et al. 2003] into 3D cubes. For 2D Wang Tiles, Cohen et al. [2003] select four sample diamonds that correspond to the edge colors of the tile as shown in Figure 6 (left). Then, each tile is constructed by finding cutting paths to combine these four sample diamonds. We extend the 2D tile generation to 3D cubes by using an octahedron to correspond to a face color and synthesize a cube with 6 octahedra. Figure 6 (middle) shows the octahedron for the west cube face. When the N/S cube faces have *n* colors, *n* octahedra are selected from the sample textures, each corresponding to a face color. Similarly, *n* octahedra are selected for each of the W/E and F/B directions. Next, we put these subvolumes to every cube according to the face colors of the cube. Finally, a corner-to-corner synthesis process is proceeded to quilt the adjacent octahedra together by finding the cutting plane between them. Figure 17(a) shows an example of 16 cubes (each has 8^3 voxels) which are synthesized from a filtered noise volume [Perlin and Hoffert 1989].

4.2 Primitive Cubes

Since a large number of primitives are used in scientific illustrations and NPR, we design cubes filled with geometric primitives. The primitives in illustration examples are usually distributed in a pleasant way, and NPR methods simulate this effect by distributing the primitives uniformly in the rendering. Therefore, we design an automatic method to fill the cubes uniformly with geometric primitives. Before we discuss the primitive cube design, we address three common problems for nonphotorealistic volume rendering that put some requirements on the pattern design.

—To achieve different shading and density effects at varying resolutions, we need to design the cube patterns at multiple levels.
—To provide temporal coherence during rendering, the higher-level representations should include all the primitives of the lower levels. For example, point drawing [Deussen et al. 2002; Lu et al. 2003] always draws the prefix of the point lists, while hatching [Praun et al. 2001] uses line patterns at several levels. Using the same idea, our geometry-based rendering draws the prefix of the geometry lists per-voxel and our texture-based rendering draws the cubes at several levels.
—To improve the visual quality of the image and simulate a Poisson distribution, an even distribution of primitives is needed. Cohen et al. [2003] use Lloyd's method to optimize pre-generated point positions among Wang Tiles, and Praun et al. [2001] hierarchically select the best-fitting line from a pool of candidates. Our method is derived from the combination of these two methods.

The tiling process of Wang Cubes ensures that the cube patterns are nonperiodic, therefore, we concentrate on the connectivity of the cubes and the even and random distribution of the primitives at multiple levels. We next discuss cube design for points, lines, and general primitives, respectively.

4.2.1 *Points.* We use an iterative procedure to generate the point (stipple) patterns. For point patterns, we recursively divide the volume of a cube into 8 candidate subregions based on a predefined depth. We also maintain an octree structure to store the number of points contained in the corresponding candidate region. Points are iteratively added to each cube by the following process, illustrated in Figure 7. For each point primitive to be added, we hierarchically

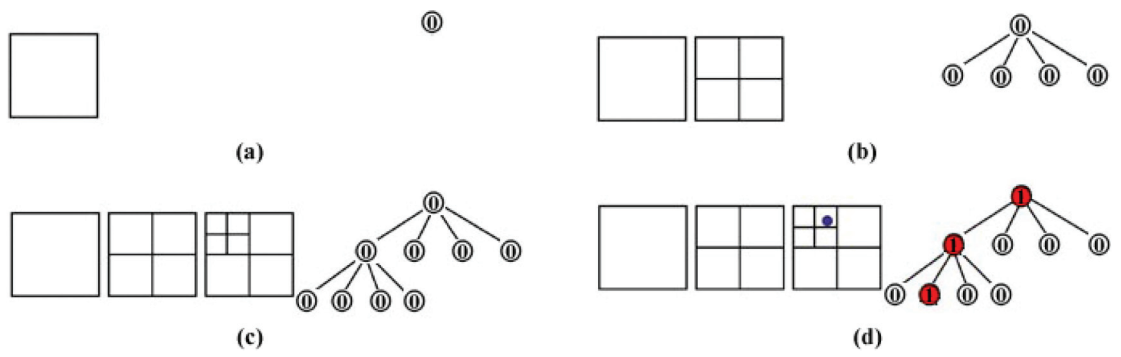

Fig. 7. The process of point generation. (a) An empty cube. (b) The first subregion is selected. (c) The second subregion is selected. (d) A point is randomly generated within the selected region and the octree is updated.

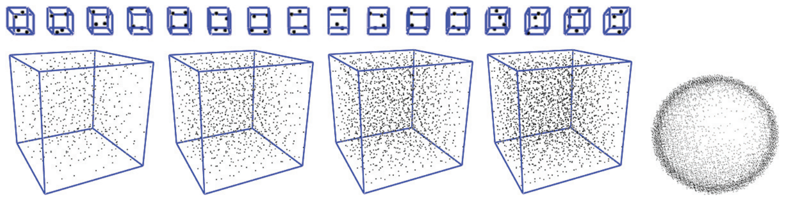

Fig. 8. 16 cubes with points and four 8^3 volumes with densities of 1 to 4. The volumes achieve roughly uniform point distributions. In the right image, the density of points per-voxel is calculated to show the lighting effect on a sphere. The rendering process will be discussed in Section 7.

calculate the probability of the nodes in the octree from top-to-bottom, select a best leaf (discussed later), randomly generate a point inside the corresponding region of the leaf, and update the octree. The probability is calculated by the weighted sum of the following two factors.

The first factor is the point density, which is the number of points contained in a region. Let d_1 be the density of the current region and d_2 be the density sum of the 26 adjacent regions. To generate a uniform point distribution, we need to consider all the possible combinations from the cube tilings according to the face colors; therefore, adjacent regions from other cubes and their occurrence frequencies are also considered.

The second factor is based on several 3D restrictions. Different from 2D pattern generations, the appearance of a 3D pattern is the projection on the image plane, and thus it may look quite different from different viewing directions. Therefore, we need the points in the cubes to overlap with each other as little as possible to achieve the overall best effect for all the viewing directions. Our approach uses simple geometric relations: no three points should be in the same line, and no four points should be in the same plane. From the center of the selected region, we calculate the two following values for the final candidate probability:

(1) point-line distance d_3 (the average from the center to any line by any other two points in the near region)
(2) point-plane distance d_4 (the average from the center to any plane by any other three points in the near region).

The probability of each region is then calculated as $p = -\sum_{i=1}^{4} w_i d_i$, $w_{1,2} \geq 0$, $w_{3,4} \leq 0$, where the weights, w_i, can be interactively adjusted. For instance, weights of $100, 2, -10, -10$ are used in Figure 8 and 3 is used for the octree depth. A region with the maximum probability is considered a best region. If multiple best regions exist, we randomly choose one. If a varying point size is used, the point size should be proportional to the probability value. Larger points are allowed when the local point density is relatively low.

We generate one point at a time for each cube and repeat the process until the desired number of points is reached. Generally, we use cubes with 4–20 points, and the whole process takes place in several minutes. Compared to point distribution techniques [Cohen et al. 2003], we do not need to redistribute the points after the generation. Compared to line distribution techniques [Praun et al. 2001], we do not need to generate a candidate pool. Finally, during rendering, we draw the cubes at different levels (corresponding to varying point numbers), and we always draw from the beginning of the point lists. Therefore, our method yields an approximately uniform distribution as shown in the cubes in Figure 8.

4.2.2 *Lines.* Patterns composed of lines, strokes, and cross-hatchings are often used in scientific illustrations. To generate line patterns, we use the following four user-specified parameters: primary line direction, direction range, maximum line length, and length range. Different combinations of these four parameters allow us to generate various line patterns. The parameters are very intuitive and thereby easy to adjust. We will discuss the line generation for two cases: lines within one cube and lines spanning multiple cubes.

For lines within one cube, we first select a best point for a line to pass through, using the same method as for points. Then we randomly generate a line direction by the two direction parameters (primary line direction and direction range) and search for the best length of the line in both directions. We favor longer lines over shorter lines by using normalized probabilities along the line segment, $\sum_{i=1}^{l} p_i / l$, in choosing the best line length l [Praun et al. 2001], where p_i is the probability for each sample point along the line segment.

The lines spanning multiple cubes are generated by manually assigned length, corresponding to the number of cubes to cross.

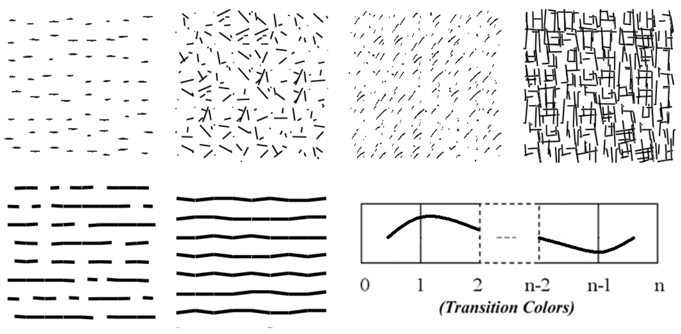

Fig. 9. The top four images show line patterns with different settings (density and direction). The bottom images show three kinds of lines: various length, infinite lines, and lines with assigned length n.

Instead of selecting the best point inside the cube, we choose the best point on the colored faces since we need to guarantee the connections of the lines when they are put together by their face colors in 3D. The best point on the faces is also selected according to the current primitive densities using the two factors from the previous section for any colored face. A line segment is then added for all the cubes with this colored face. According to the assigned length, we divide the line generation into four cases.

—*Length Two.* A face-inside line segment is added to the cube. One end point is the face point, the other is inside the cube.

—*Various Lengths.* We connect as many face points as possible. If a single point remains, we generate a face-inside line segment.

—*Infinite Length.* We generate an equal number of points for the corresponding faces and connect them into face-face lines.

—*Assigned Length n* ($n > 2$). All the lines cross n cubes. Following, we discuss two methods for this case.

For lines of length n ($n > 2$), we can add transition colors for all the related faces to transfer the connecting points as shown in Figure 9 (bottom right). The number of additional cubes depends on the line direction. If the line direction is near, but not parallel to the X, Y, or Z-axis, many transition points are needed and, therefore, many cubes. Another method is to divide a big cube into smaller cubes. This method might need more cubes than the first method, but it does not need to calculate transition colors.

We can generate cross-hatching using a similar method by choosing the center position and the line length along each line direction. One example pattern generated by our system is shown in Figure 9 (top right).

4.2.3 *General Primitives.* Since the structure of many geometric primitives can be expressed as points (center) and lines (center with direction), we use the center point or center line to determine their position. We randomly choose other properties of the primitives, such as rotation, to make the results look more random and more visually pleasing. One remaining problem is that general primitives have thickness which may cause them to exceed the boundary of the cubes. If we render using geometry, we draw the primitives at their center location; if we use 3D texture-based rendering, we need to treat the oversized primitives as face primitives and add them to all the cubes that have the matching colored face. Examples of illustrative renderings with various primitives are shown in Figure 14 and Figure 16.

Since the cube contents and cube tiling are totally independent, we can use one set of cubes to generate multiple sets of cube contents. All of the new composed patterns and textures can share the same cube tiling during rendering, since the cube tiling is only controlled by the cube face colors. Therefore, we can interactively select patterns or textures during the rendering by loading different cube contents.

5. ANISOTROPIC PATTERN DESIGN

Isotropic patterns are designed without the knowledge of any data values from a dataset. Different from isotropic patterns, this section discusses the design of anisotropic patterns, which utilize different data values, such as voxel values or gradient direction. Our basic idea is to use a user-specified data field as an additional restriction to compose the 3D pattern for rendering. The patterns composed for each dataset will be different even when a same set of cubes is used. Therefore, suitable anisotropic patterns may reflect the features of an object.

Anisotropic patterns have been designed for both 2D images [Cohen et al. 2003] and 3D models [Gorla et al. 2003]. Compared to these previous approaches, the main difference with our approach is that we try to generate anisotropic patterns with a set of cube samples instead of generating the patterns for specific voxels in the volume. Therefore, our approach is generally faster and we need to control the trade-off between the storage issue and final effects. We explore three methods to design anisotropic patterns by using volume data value, curvature direction, and tangent orientation, respectively. These methods can be used for other data values with similar types.

5.1 Volume Data: The General Corner Problem

Instead of coloring the faces of Wang Cubes, we can also color the corners. Coloring the cube corners was called the corner problem when originally introduced for Wang Tiles by Cohen et al. [2003]. The face colors and corner colors can be used to generate a tiling either separately or jointly.

For the corner problem of Wang Cubes, we assume each corner has n possible colors, and we can use transfer functions to map the volume data values into n colors. The tiling process for the corner problem of Wang Cubes is almost the same as the process for original Wang Cubes except that corner colors will be used to select the qualified cubes instead of cube face colors. Since all the corners from the N, W, and F faces need to be matched, a total of 7 corners are required, and there are n^7 combinations. Since we associate the volume data values with the corner colors, all the 8 corners need to be matched with the corresponding volume data value. Therefore, we need to consider all n^8 combinations.

Different corner colors may be used to represent different primitive densities or different kinds of primitives. To generate the patterns for the cubes, we add a density field for each voxel of every cube according to the corner colors. This density field g_i is calculated by the trilinear interpolation of the designed densities from each corner of a cube. It is organized in the same octree structure as the cube and used to modify the first factor in Section 4.1: $w_i' = g_i w_i$, $i = 1, 2$. A new primitive will be generated inside a region with the maximum probability. For example, we design the point densities (line lengths) on a scale from 0 to 2, where 0 corresponds to no point (line) and 2 corresponds to maximum point density (line length). Figure 10(a) shows 4 cube designs. The left two-cubes have different point densities with the green color indicating density scale 2 and the red color indicating density scale 1. The two right cubes are examples of different primitives. For point densities, the green color indicates scale 2 and the red color indicates scale 1. For line lengths, the green color indicates scale 0 and the red color indicates

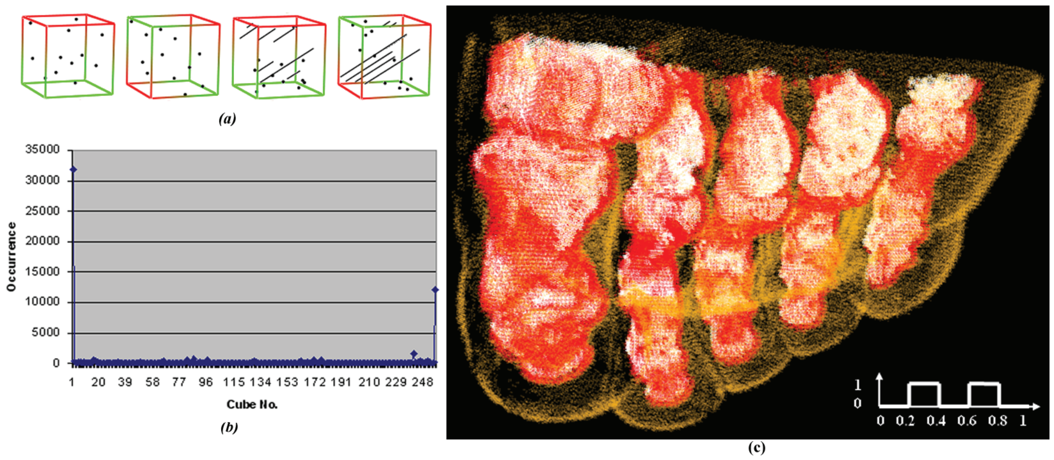

Fig. 10. (a) Primitive distributions by corner colors. The left two show points with different densities, the right two show hybrid distributions of points and lines. (b) The 256 cases distribution of (c) shows most occurrences are at case 0 and case 255. (c) Volume rendering of a foot dataset by point patterns with 2 corner colors. The yellow and white regions are cases 0 and 255, respectively (each case is rendered with 16 isotropic cubes), the red color represents the cases 1-254 (each case is rendered with 1 anisotropic cube). The function at the right-bottom corner is designed to color normalized volume data. The point densities for the two corner colors are: color 0 has 4 points and color 1 has 8 points. Although each of the cases 1-254 contains only 1 cube, the point patterns are random enough to simulate a manmade drawing.

scale 2. The user can freely design a transfer function to divide the whole data range into n colors. For example, one color may represent normal data ranges, and the others represent emphasized data ranges.

However, many cubes are required. Two colors yield 256 cases, and with more than two colors, the number of cases becomes insupportable. If we consider the face colors and corner colors at the same time, there will be at least 8×256 cases. As with coloring faces, we need to have 2 cubes for each case to assure that the patterns are nonperiodic. By statistically examining the cube occurrence for each case, the most frequent cases are 0 (all the corners are 0) and 255 (all the corners are 1), as shown in Figure 10(b). Since other cases occur based on object properties and only occur at some isosurfaces, they do not significantly generate periodic patterns. However, if we only color corners, cases 0 and 255 will generate a large number of repeated patterns. Therefore, for geometry-based rendering, we use 16 isotropic cubes for each of the two special cases and 1 cube for each of the rest. The corner cases are calculated on-the-fly, and we can reuse the isotropic cube tiling for the two special cases. An example with 2 corner colors is shown in Figure 10(c).

5.1.1 *Similar Corner Problem with Marching Cubes.* The Marching Cubes algorithm renders polygons to approximate isosurfaces according to the volume data at the corners of each cell [Lorensen and Cline 1987]. All the corners are divided into either inside or outside the isosurface according to the isovalue of the surface. From this corner classification, each cell is classified into a case of the surface intersecting the voxel with predesigned polygon intersection placements.

With Wang Cubes, if we draw only the cubes corresponding to cases 1 to 254, the result is similar to an isosurface but generated by 3D patterns. In the sense of coloring corners, the Marching Cubes algorithm divides voxels into two groups, while the Wang Cubes algorithm divides the volume with transfer functions. Moreover, the Marching Cubes algorithm renders polygons, while Wang Cubes can render various patterns and textures in addition to polygons. These correlations demonstrate that there are similarities between the Marching Cubes algorithm and the corner problem of Wang Cubes.

As with isotropic patterns, we can use Wang Cubes to nonperiodically texture the isosurface with at least 2 texture samples. An example is shown in Figure 11. We use fixed vertex positions instead of interpolated values for speed in rendering, but this does not produce any obvious artifacts and image quality can improved be by adding more cubes. The representation fidelity and nonperiodical textures can both be achieved by a mixed rendering approach, which uses the interpolated vertex positions from Marching Cubes and maps the textures from Wang Cubes to each corresponding polygon.

5.2 Curvature Direction with Wang Cubes

Curvature direction plays an important role in conveying the shape of objects [Gorla et al. 2003]. We calculate the curvature direction by using the eigenvector method from Kindlmann et al. [2003]. An intuitive way to generate Wang Cubes from curvature information is to use quantized directions to color corners, but this requires too many cubes (n^8). Instead, we use connecting points on cube faces to generate line patterns.

Assume n connecting points are selected from 6 cube faces. We allow all the combinations for the connecting points: every point can be connected or disconnected. Therefore, there are 2^n cases. For each case, we randomly design m cubes to connect the connecting points; therefore, $m \times 2^n$ cubes are generated. The tiling process is the same as the process of cube tiling generation in Section 3. Generally for each position, we choose the cube which matches the existing connecting points on the N, W, and F faces and contains the closest line direction to the curvature direction from the processed voxel. Figure 12 uses the 6 center points of cube faces as the connecting points and has 2 cubes for each case, totaling 128 cubes representing 8 different directions. The tiling process takes about 5 seconds for a 128^3 volume.

Using the same idea, we can also generate patterns which align to other directions, such as gradient directions and vector fields.

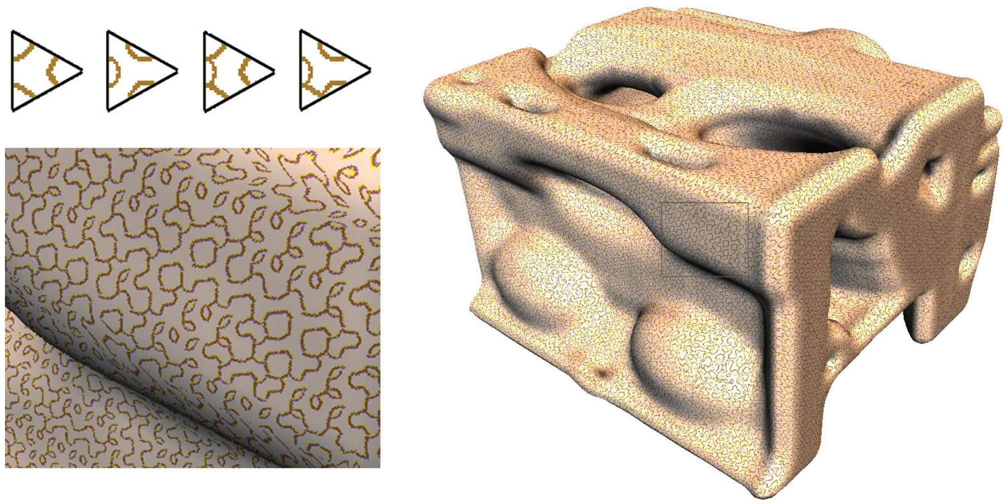
Fig. 11. An isosurface of a 64^3 engine block dataset rendered by polygons. 256×4 cubes are generated from the left 4 triangle textures to nonperiodically tile the surface. An enlarged region is shown in the left.

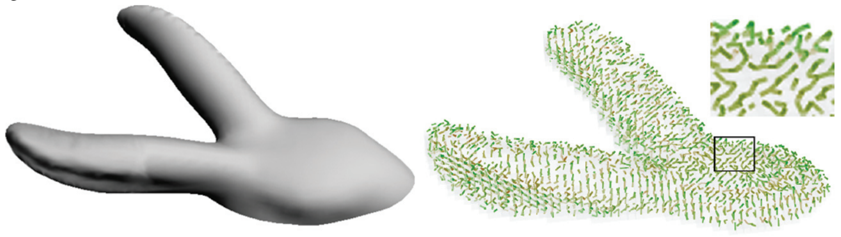
Fig. 12. The geometry of the ears of a bunny dataset (left) and the connected line patterns (right).

The original direction information is approximately embedded in the designed line patterns.

5.3 Tangent Orientation with Wang Cubes

Tangent orientation is commonly used in volume rendering. We can also generate cubes based on preprocessed tangent orientations to emphasize the gradient information. First, the 3D normalized vector space is quantized into n gradient directions. For each cube, we align the flat primitives orthogonally to the assigned quantized direction and randomly rotate them in the other free directions. If the cubes contain 3D textures, we need a total of $16n$ cubes. If we render geometry, we use only n cubes to indicate the placement of the flat primitive and use the point or line positions from the isotropic patterns as the skeleton of the primitives. In Figure 14 (d), we use small, different-shaped polygons aligned with the tangent plane to simulate a Pointillist drawing.

6. MULTIRESOLUTION CUBE DESIGN

Wang Cubes are cubes of the same-size, and all the previous sections discuss same-sized cubes. However, in volume rendering, we need to render datasets and different features at varying image resolutions. The introduction of multiresolution cubes brings three advantages to Wang Cubes. First, we can emphasize certain portions in a dataset and direct the user's attention to these more detailed features as illustrators do. Second, processing time is saved when we render more distant objects or surrounding features at a coarser scale. Third, since most scientific data has a fixed resolution, there will not be enough detail when a portion of the object is excessively enlarged. We can use multiresolution cubes to provide a method to continually add the necessary nonperiodic details during zooming.

We use eight small cubes in an octree to represent a large cube. By *represent*, we mean that both the textures and six face colors are matched from the small cubes to the large cube. If there existed a set of cubes which represents themselves, we could implement infinite resolutions by continuously representing the cubes. Unfortunately, by counting face colors, we know that it is impossible for a set of cubes with general 3D textures to represent itself.

However, if we only consider the colors of the cube faces, it is possible to find a set of cubes which can face-represent, themselves. By *face-represent*, we mean only the six face colors are matched from the small cubes to the large cube. For each cube in the set, we consider it as a 2^3 volume with face colors to see whether it can be filled. If every cube can be filled with the cubes in the set, then the set of cubes is *self-face-representing*, as shown in Figure 13(a). Usually there exist multiple mappings to self-face-represent a cube. For each cube, we choose one mapping which contains the largest number of different cubes. Therefore, for all the scales in the rendering, we reuse one set of face colors and one set of mappings. The cube tiling is generated at the coarsest scale, and we use the mapping to find the cube index for finer scales. The cube tiling may appear less random because only one mapping for each cube is used, but additional cubes can be added to alleviate this problem. We will now discuss two methods to generate the contents of the self-face-representing cubes.

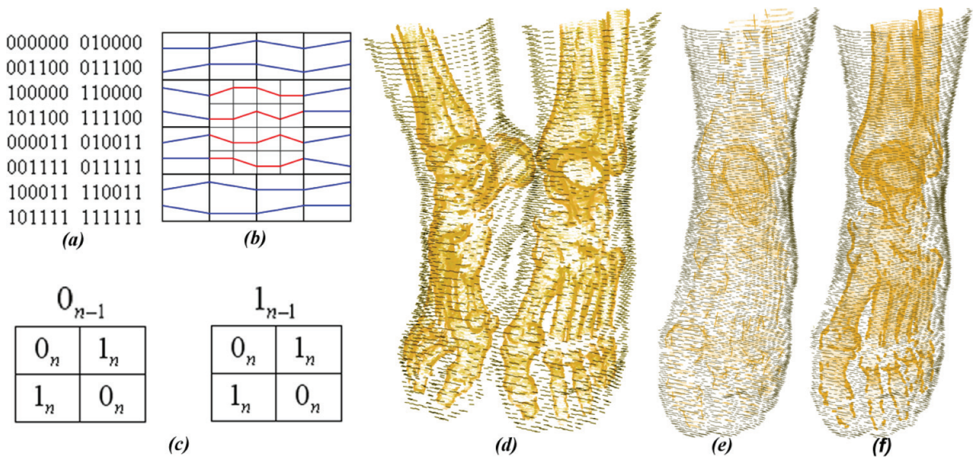

Fig. 13. (a) A sample set of self-face-representing cubes with face colors WENSFB. (b) A sample pattern on different scales. The finer scale (red) includes more details than the coarser scale (blue). (c) The mappings of the cube face colors from a coarser scale to a finer scale we use for (a), where a coarser face color is represented by 4 finer face colors. (d) The bones of the foot are rendered with the finer scale, while the skin uses the coarser scale. (e) The bones are on the coarser scale, while the skin is on the finer scale. (f) Both are on the finer scale.

One method is to generate one set of cubes and blend the cubes on the adjacent scales during rendering. In this way, infinite nonperiodic resolution can be achieved by continually blending the cubes of the current scale and the next finer scale. Another method is to use one set of cubes for each scale. First, we design the textures for the cubes of the finest scale. Then, we use the mapping to copy the textures of the smaller cubes into the larger cubes. Therefore, the connections of cubes are guaranteed across multiple scales, and the cubes can smoothly transfer from one scale to another without any manipulation. Generally for every isotropic pattern, we use two scales and 16 cubes for each scale. Figure 13(b) shows a sample pattern on two adjacent scales.

The volume data on the coarser scale is 1/8 of that on the finer scale; therefore, the running time significantly improves. The foot dataset in Figure 13(d) renders about 2 times faster than (f) when both features are rendered at the finer scale. It also keeps the exact shape of the bones and shows the position of the skin.

7. VOLUME RENDERING WITH WANG CUBES

With our cube tiling generation and cube design methods, we can generate various 3D nonperiodic patterns and textures for volume rendering. The flexibilities of Wang Cubes make these patterns and textures very convenient for rendering volume datasets. They occupy much less storage than traditional rendering methods, and they are very quick to compose any sized volumes once the sample cubes are generated. We will first generally discuss the rendering issues involved with Wang Cubes. Then, we will concentrate on our geometry-based system and our texture-based system. At the end, we will specifically discuss the storage and running time of this new volume rendering method.

To ensure the temporal coherence during rendering, we store the cube contents and the cube tilings for the volume so that, for every pattern, each voxel (or cell) in the volume has its fixed corresponding cube location. The shape of an object is achieved through rendering each position at different levels. Specifically, we change the number of primitives to draw for geometry-based rendering and the opacities for texture-based rendering. The rendering levels can be controlled for each voxel with arbitrary feature extraction and transfer functions, two common approaches in volume rendering. The voxel-level calculation is generally very accurate in preserving sharp data features. Since Wang Cubes provide a large number of patterns and textures, users can choose suitable patterns or textures according to the visualization purposes. The settings (primitive, density, size, and opacity) of these patterns can also be interactively assigned for each feature to achieve various styles. In Figure 14, 15, and 16, independent patterns and settings are used to simulate different styles. Specifically, the shapes and sizes of the primitives are designed to simulate the form of an artistic brush, and the opacities and color schemes are designed to simulate the properties of the brush. We use the feature extraction methods and color calculation from Lu et al. [2003], and we choose the patterns which are visually distinguishable from surroundings or which have the closest appearance to the real objects. Our framework can be used to effectively explore the features in a volume dataset and render them in various styles.

The user is provided with a toolbox of patterns and is given a tool to design new patterns by our proposed design methods. These 3D patterns are composed in real-time from the cube contents and cube tilings. The cube contents only need to be computed once, and generally it takes from seconds to a few minutes according to the pattern type and can be used for all size volumes. One isotropic cube tiling and selected anisotropic cube tiling will be generated for each volume before rendering which is within a dozen seconds. To switch patterns and textures during the rendering, users just need to select a pattern type from a pop-up menu. Our system will automatically switch to the corresponding rendering by changing the cube contents and cube tiling. Therefore, the main user effort throughout the rendering is for common visualization purposes such as adjusting transfer functions and rotating the volume.

We use the same color set of cubes for all the isotropic tilings because the face color design of the cubes is independent of the components inside the cube. Since we have 16 cubes for each isotropic pattern, 4 bits are needed for each cube tiling. The size of each cube depends on the primitives inside and their densities. We will discuss this issue for geometry-based rendering and texture-based rendering, respectively.

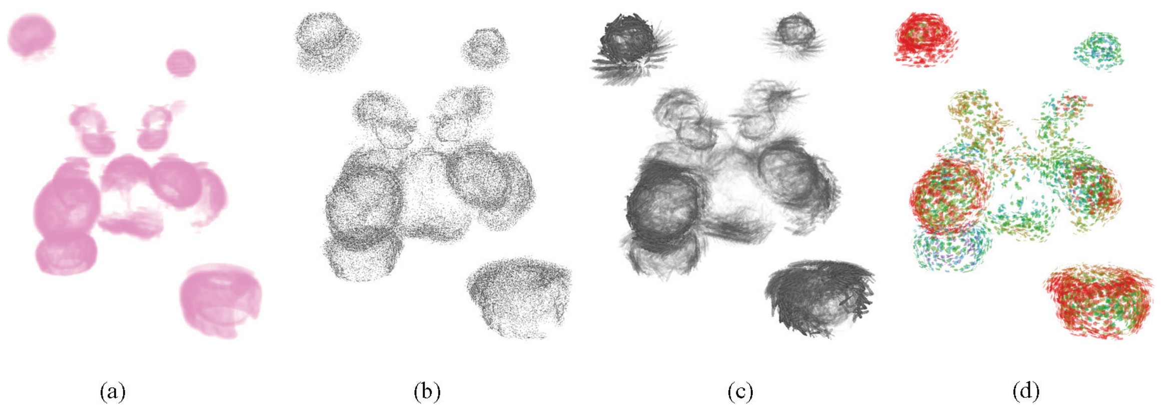

Fig. 14. An iron protein dataset rendered with different patterns and settings. (a) is rendered with 2D transfer functions and used for comparison with our illustrative results. The other three images simulate the following styles, respectively: (b) stipple for illustrating data shape, (c) stroke for highlighting gradient field, and (d) particles for emphasizing tangent orientation.

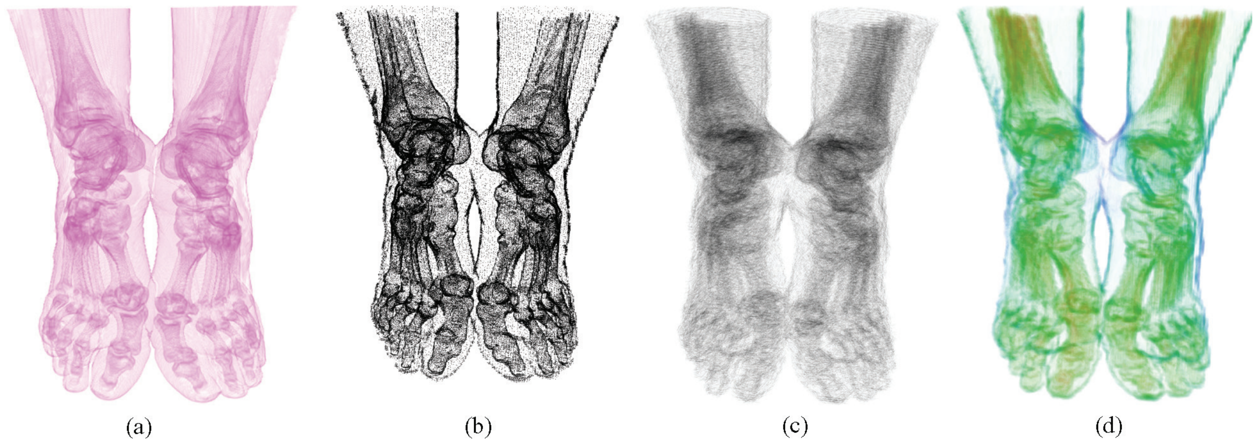

Fig. 15. A foot dataset rendered with different patterns and settings. Similar to the results in Fig. 14, these rendering styles can be used to render all the volume datasets. (a) is rendered with 2D transfer functions and used for comparison with the illustrative results. The other images simulate the styles of (b) stipple, (c) stroke, and (d) watercolor.

7.1 Geometry-Based Rendering

Geometric cubes usually require little storage space. For example, a cube with eight-3D float points costs 96 bytes (8(*points*) × 3(x, y, z *axis*) × 4(*float type*)). Therefore, we can create various cube patterns within a limited storage requirement. The variety of the cube patterns provides the possibility to simulate different rendering styles. When there are multiple objects in the volume, the user can choose suitable cube patterns, such as the closest appearance to a real object, to more effectively render each object. In our geometry-based system, we use 16 cubes for each isotropic cube set at each level, and we use several sets of cubes: 2 for points (one sparse, one dense), 3 for lines (along three orthogonal directions), and 1 for cross-hatching. For anisotropic cubes, we have 254 extra cubes for the corner pattern, 128 for directions, and 40 for tangent planes, for a total of no more than 1000 cubes. All the user-selected isotropic and anisotropic cubes and the isotropic cube tilings are generated and stored before rendering. For the anisotropic cube tilings, while the directions and tangent orientations need to be generated for each dataset, the cube tiling for the corner problem is calculated on-the-fly.

The rendering process is similar for all the geometric cubes. We traverse the volume and calculate the rendering level for each voxel. For the voxels with positive rendering levels, we use the cube tiling and the calculated value to decide which cube at which level to draw. To avoid pattern "popping," we draw the integer part of the value in full color and the fractional part semitransparent [Lu et al. 2003].

7.2 Texture-based Rendering

Our 3D texture-based system is developed on a physically-based multifield weather data visualization system [Riley et al. 2003]. One 3D texture is generated for each object in the volume, and they are rendered with a hardware-accelerated program that slices through the volume. The weather data contains several data components, such as cloud, rain, ice, and graupel. Similar to the minimum cube number in the geometry-based system, a set of 16 isotropic cubes is generated for each of these fields using the automatic method discussed in Section 4. All the cube textures are combined into one texture unit. The cube indices are divided by 16 to reduce them to the texel value range of [0,1), and the cube tiling is stored in a free component in a texture unit with the volume data. During the rendering,

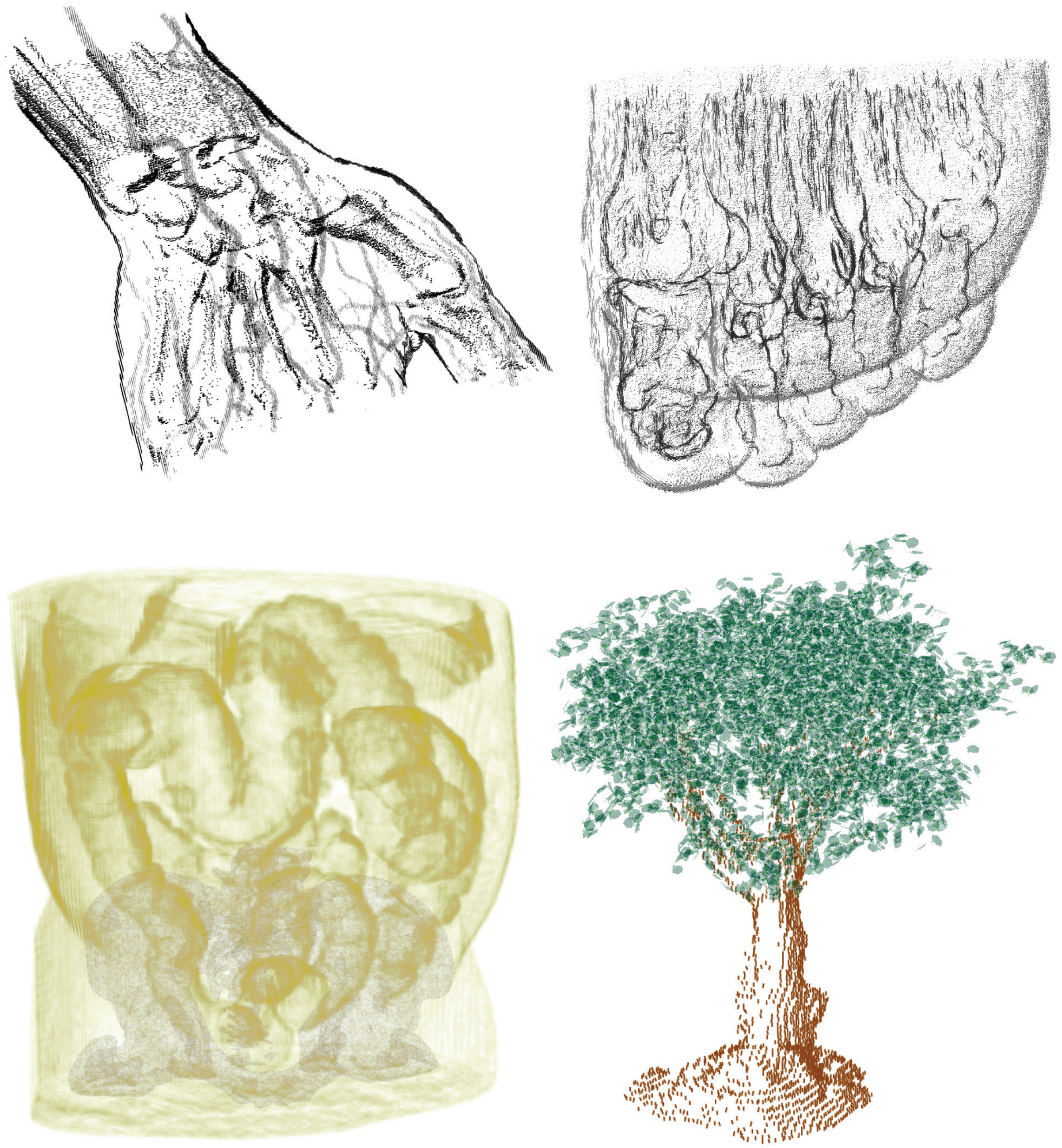

Fig. 16. Illustrations of a segmented hand dataset, a human foot from a posterior view, a colonoscopy dataset, and a bonsai tree dataset. Different patterns and settings (size, density, opacity, and color) are used in the rendering to distinguish each object in the volume.

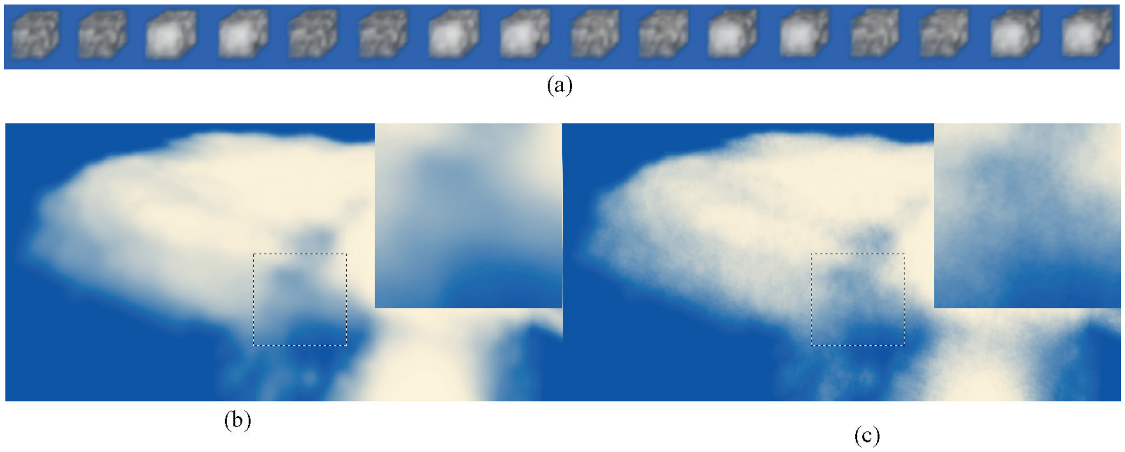

Fig. 17. (a) 16 cubes generated from a noise volume. (b) Cloud rendered without Wang Cubes. Low-resolution volume model results in a very smooth cloud. (c) Cloud rendered with Wang Cubes showing more detail. Enlarged regions are shown in the top-right corner.

the closest texel determines into which cube the current fragment maps, given a set of coordinates. We translate texture coordinates to coincide with an actual voxel for the uninterpolated cube index. We then translate the global texture coordinates into the local coordinates of the chosen Wang Cube and sample the cube patterns. The patterns are used to modulate opacity per data component, and all fields are blended according to their overall contribution. As shown in Figure 17, the cloud with Wang Cubes contains richer details than the cloud without, and voxel artifacts near the volume boundary are reduced.

To emulate example illustration styles, we develop an approach to generate 3D textures from 2D illustration examples and volume samples [Lu and Ebert 2005], as shown in Figure 18. We use the illustrative 3D textures to render volume datasets directly with segmentation information and/or transfer function selections. Similar to the usage of Wang Cubes in other renderings, we can provide additional information, enhancing the rendering of the original data and saving storage. Because of the ability to emulate example illustrations, this method requires much less user interaction in the rendering process.

7.3 Storage Requirement

We now specifically discuss the storage requirements for using Wang Cubes and also compare Wang Cubes with other rendering approaches. To use Wang Cubes in volume rendering, we first need to store all the cube contents. We also store a cube tiling that has the same size as the volume so that the composed volume patterns and textures are fixed to keep the temporal coherence of the rendering. Let's assume that the cube set has N_C cubes, each cube takes M_C bytes, each cube tiling takes M_T bytes, and a volume's size is N_T. The size of each cube depends on the primitive type and densities. To generate a nonperiodic pattern or texture without Wang Cubes, we need to store a cube content for every voxel in the volume, which takes $M_C \times N_T$ bytes. Therefore, the ratio of the storage with Wang Cubes and the storage without Wang Cubes is shown in Equation (1).

$$p = \frac{M_{with}}{M_{without}} = \frac{M_C \times N_C + M_T \times N_T}{M_C \times N_T}. \quad (1)$$

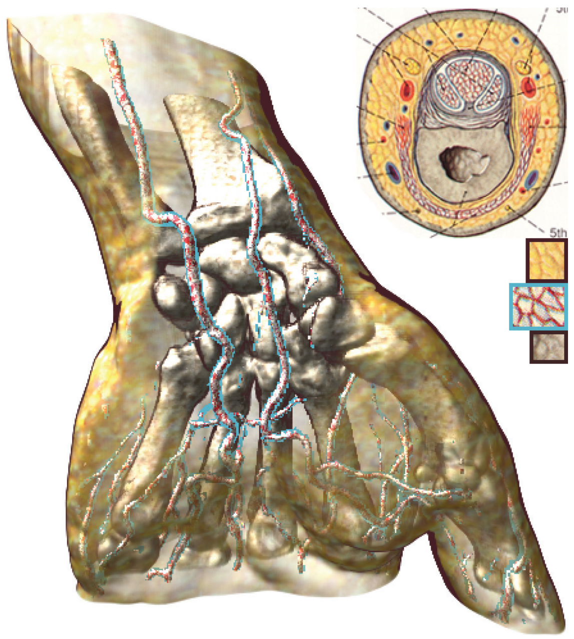

Fig. 18. A direct volume rendering of a segmented hand dataset [Lu and Ebert 2005]. It is rendered with recolored textures using their respective example illustrations. The three small images are cropped from the corresponding example regions and show the textures of fat, vessel, and bone, respectively. The silhouette colors are also selected from the examples and shown as the box colors around the samples.

When we use a set of 16 isotropic cubes, it only requires 4 bits for each voxel to store the cube tiling. We will use a 128^3 volume as an example. For geometry-based rendering, if we use 8 geometry points (3D floats) for each cube, $M_C = 96\,bytes$ and $p = 1/191.7$. For texture-based rendering, if we use a 8^3 volume for each textured cube, $M_C = 2048\,bytes$ and $p = 1/3855.1$.

When there are multiple objects in the volume, we usually use different cube contents to distinguish them from each other. Since the same cube set is used to generate all the cube contents, the

Table I. The Additional Storage Requirements Beside the Volume Data and Gradients. (The data is collected based on 8 geometry points, 8^3 textured cubes, and a 16 cube set. The data for the geometry-based Wang cubes, texture-based Wang Cubes, and texture-based rendering without wang cubes are collected as Equation (1) and (2).)

Dataset Name	Dimension	Original Stippling (bytes)	Geometry-based Wang Cubes (bytes)	Texture-based Wang Cubes (bytes)	Texture-based Rendering without Wang Cubes (bytes)
iron	64^3	3,164,928	132,608	163,840	536,870,912
abdomen	128^3	201,326,592	1,050,112	1,081,344	4,294,967,296
feet	128^3	20,698,560	1,050,112	1,081,344	4,294,967,296
hand	$256^2 \times 128$	805,306,368	4,195,840	4,227,072	17,179,869,184

content sets can share the same cube tiling. Assuming there are N_O objects in the volume, the ratio of the storage with Wang Cubes to the storage without Wang Cubes is shown in Equation (2). Therefore, the saving with Wang Cubes is even greater.

$$p' = \frac{M_{with}}{M_{without}} = \frac{N_O \times M_C \times N_C + M_T \times N_T}{N_O \times M_C \times N_T}. \qquad (2)$$

In real applications, the storage requirements are different for geometry-based rendering and texture-based rendering. In our geometry-based rendering, the storage of the cube contents is usually negligible compared to the storage of the cube tiling so the volume size is the major factor. In our original stipple-rendering [Lu et al. 2003], we save some storage by eliminating voxels that won't be used in the rendering based on transfer function settings. The average storage improvement is around 20 times in the original stipple-rendering system. In our current illustration system using Wang Cubes, $p = 1/192$ for the example of 8 geometry points. In the texture-based rendering, the cube tiling is stored in the same texture with the scalar value so that it does not occupy additional space. The size of each textured cube becomes the major factor. For 8^3 cubes, $p = 1/4096$. Table I provides the actual storage comparison for several datasets. From these statistics, it is clear that Wang Cubes can provide nonperiodic patterns and textures with much less texture space. This comparison is made for the nonperiodic case. Although an arbitrary 3D pattern or texture can provide more flexibility, the generation time and storage requirement make them impractical present for real applications at without using a technique such as Wang Cubes.

7.4 Running Time

The geometry-based Wang Cube rendering has different running times according to the geometry type and size, such that points are faster than the other primitives. Since the rendering with Wang Cubes only requires an additional indexing operation, our original stipple rendering and the stipple rendering with geometry Wang Cubes have roughly the same running times. The texture-based rendering has the same running time for same-sized volumes. Since we need additional calculations for each fragment to locate the cube content and the cube tiling texels, the rendering with Wang Cubes is slower than the rendering without Wang Cubes. Table II provides a detailed running time comparison that was collected on a NVidia GeForce 6800 Ultra graphics card. Figure 17 is rendered with the texture-based system, and it is slower than the renderings in Figure 15 because of the additional memory accesses and texel position calculations.

The major factor that affects the running time is the number of objects that users select. When multiple objects are specified from transfer functions, our implementation traverses all the selected patterns for every object at each voxel for geometry-based rendering and at each fragment for texture-based rendering. Therefore, the running time is proportional to the object number. For geometry-based rendering, a second factor is the proportion of actual rendered voxels to the volume size. The higher proportion takes longer running time. This factor does not affect our texture-based rendering since all the fragments are calculated in the same way.

Table II. The Recorded Running Time for the 64^3 Iron Protein Volume and 128^3 Feet Volume

Iron Image	Rendering	Running Time (frames per second)
Figure 14 (b)	points	66.0
Figure 14 (c)	textured lines	61.5
Figure 14 (d)	patches	35.2

Feet Image	Rendering	Running Time (frames per second)
Figure 16 (b)	points	6.6
Figure 16 (c)	textured lines	6.6
Figure 16 (d)	patches	6.3

The additional operations for handling cube-based textures are the accesses of corresponding cube tiling and cube contents and several float type calculations. Since these operations are independent of the size of the cube set, the running time does not change much as long as all the storage can be fitted in the memory.

7.5 Discussions

The main effort of this article is to generate various 3D patterns and textures and utilize them in volume visualization. Most current volume rendering approaches use single colors or colormaps to visualize data with transparent or opaque regions except texture-based flow visualization. It is the storage and generation problems of general 3D textures that have limited their usages in volume visualization. As we have shown in the storage section, several general 3D textures take 4GB memory for a 128^3 volume, which is currently too expensive for an interactive rendering algorithm. Procedure textures do not require much storage space, however, they need the abstraction of mathematical equations and thereby may not be available for every texture type. The generation of 3D textures is another important factor which limits the usage of general textures in volume rendering. Since the generation process is usually time-consuming, it is not practical if the textures need to be repeatedly generated for different sized volumes. We show in this article that Wang Cubes is a convenient tool for applying various 3D patterns and textures in volume visualization that can overcome both storage and generation problems.

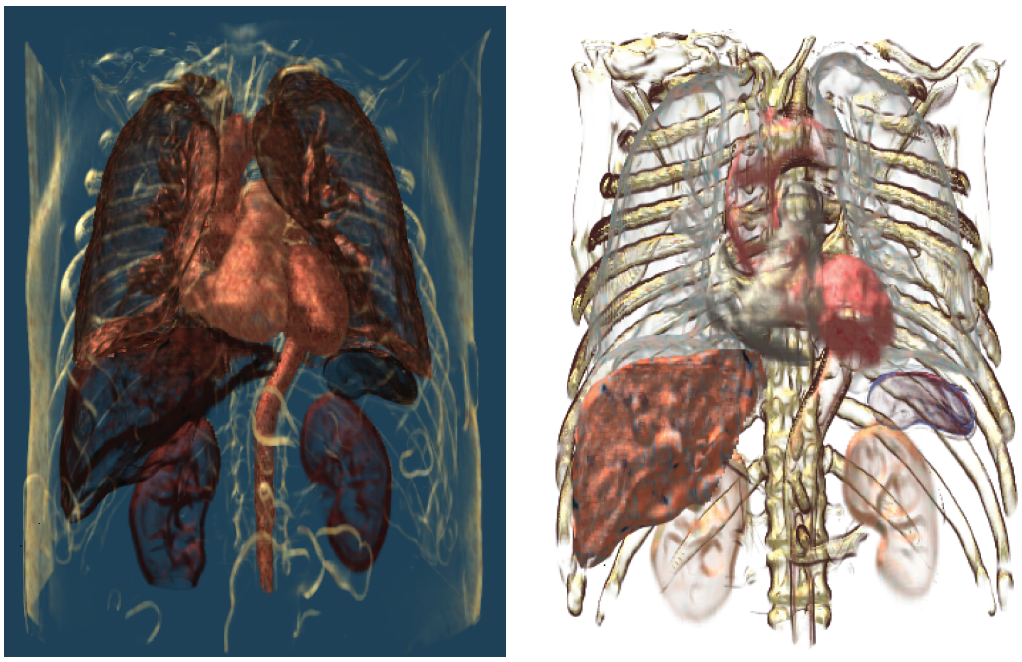

Fig. 19. An abdomen dataset rendered with different 3D textures that are composed by Wang Cubes [Lu and Ebert 2005].

The main difference between our proposed approach and conventional volume rendering approaches is the use of 3D patterns and textures. Since we use the original volume data at voxel levels for geometry-based rendering and at fragment levels for texture-based rendering, this approach achieves the same accuracy as other general volume rendering approaches. Our main motivation of the Wang Cube-based rendering is to enrich the visual representation capability of volume renderings by simulating the usages of textures in scientific and artistic illustrations (Figure 19). With various patterns and textures generated from Wang Cubes, users are allowed to choose suitable ones according to their major focus. For example, the textures synthesized from high-resolution data can be used to enrich the visualization of similar type volumes. More illustrative patterns can be used to emphasize certain object geometry features. Or different textures can be combined as an additional approach to differentiate one object from the others.

While the patterns and textures can be used to achieve better visual representations, we also realize that inappropriate usages may do harm to the visualization compared to the conventional volume rendering approaches. First, some textures may suggest unwanted patterns and create confusion. Second, overuse of textures in one rendering may be difficult for the user to visualize in each single object. Third, if the spatial frequency of the texture is lower than the real data, the accuracy of the object representation may be degraded. Therefore, the patterns and textures should be carefully selected to avoid these potential issues.

8. CONCLUSION AND FUTURE WORK

We have developed a 3D pattern and texture generation method using Wang Cubes that provides an interactive volume illustration system. Our cube tiling method produces uniform cube distribution and can be used for multipurpose tiling. To generate different style patterns, we automatically design three types of cubes: isotropic cubes, anisotropic cubes, and multiresolution cubes. The design of the pattern is per-voxel, therefore, it is accurate in conveying the shape of objects. Our method for automatically distributing geometry primitives in isotropic cubes is simple and flexible and can also distribute hybrid objects or primitives with different densities in the corner problem of Wang Cubes. We have developed both a geometry-based system and a 3D texture-based system to show that Wang Cubes provide various styled nonperiodic details for volume rendering. Our system is an interactive feature exploration tool that effectively renders the features of a volume with different patterns, textures, styles, and resolutions.

Wang Cubes have several advantages for volume illustration. First, Wang Cubes use little storage to provide various nonperiodic patterns and textures, which are more visually pleasing than the repetition of a single texture. Second, each cube is quick to generate and once the cubes are pregenerated, the filling of the textures for a whole volume is very quick, and the size of the volume can be arbitrarily large. Third, Wang Cubes can be extended to multiresolution cubes to provide continuous and necessary details to scientific datasets. With these 3D patterns and textures, rendering styles and volume transfer functions can be changed interactively by using our framework.

We believe that volume illustration is an effective way to distinguish different features in scientific datasets. Wang Cubes, with their great capability to generate various textures at a small cost, provides a large variety of choices for the rendering. Various textures can effectively convey the properties of an object and distinguish it from surrounding objects. Our method is a useful supplement to traditional volume rendering and has advantages for educational and artistic purposes. Initial feedback from a medical illustrator is positive and encouraging.

Our future work includes further exploration of Wang Cubes as a general 3D texture-based method. We would also like to combine our rendering with methods such as light fields to improve the performance. Although Wang Cubes are not supposed to be rotated, it would be useful to apply the case-counting method of Marching

Cubes to decrease the cube numbers. Also, we would like to explore effective approaches to further evaluate the usage of different patterns and textures.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their helpful suggestions in revising this manuscript.

REFERENCES

BURNS, M., KLAWE, J., RUSINKIEWICZ, S., FINKELSTEIN, A., AND DECARLO, D. 2005. Line drawings from volume data. In *Proceedings of ACM SIGGRAPH*. 512–518.

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. In *Proceedings of ACM SIGGRAPH*. 287–294.

CSÉBFALVI, B., MROZ, L., HAUSER, H., KÖNIG, A., AND GRÖLLER, M. E. 2001. Fast visualization of object contours by nonphotorealistic volume rendering. *Comput. Graph. For. 20*, 3, 452–460.

CULIK II, K. AND KARI, J. 1995. An aperiodic set of wang cubes. *J. Univer. Comput. Sci. 1*, 10, 675–686.

DEUSSEN, O., COLDITZ, C., STAMMINGER, M., AND DRETTAKIS, G. 2002. Interactive visualization of complex plant ecosystems. In *Proceedings of IEEE Visualization*. 219–226.

EBERT, D. AND RHEINGANS, P. 2000. Volume illustration: Non-photorealistic rendering of volume models. In *Proceedings of IEEE Visualization*. 195–202.

GORLA, G., INTERRANTE, V., AND SAPIRO, G. 2003. Texture synthesis for 3d shape representation. *IEEE Trans. Visuali. Comput. Graph.*, 512–524.

GUYOT, J. AND VANNSON, J. L. 1990. *Atlas of Human Limb Joints*. Springer-Verlag.

HADWIGER, M., BERGER, C., AND HAUSER, H. 2003. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *Proceedings of IEEE Visualization*. 301–308.

HEALEY, C. G., ENNS, J. T., TATEOSIAN, L. G., AND REMPLE, M. 2004. Perceptually-based brush strokes for nonphotorealistic visualization. *ACM Trans. Graph. 23*, 1, 64–96.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *Proceedings of ACM SIGGRAPH*. 327–340.

HILLER, S., HELLWIG, H., AND DEUSSEN, O. 2003. Beyond stippling-methods for distributing objects on the plane. In *Proceedings of Eurographics*. 515–522.

HODGES, E. 1989. *The Guild Handbook of Scientific Illustration*. John Wiley and Sons.

INTERRANTE, V., KIM, S., AND HAGH-SHENAS, H. 2002. Conveying 3d shape with texture: recent advances and experimental findings. In *Human Vision and Electronic Imaging VII, SPIE 4662*. 197–206.

INTERRANTE, V. L. 1997. Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution. In *Proceedings of ACM SIGGRAPH*. 109–116.

KARI, J. 2000. Recent results on aperiodic wang tilings. In *Proceedings of Pattern Formation in Biology, Vision and Dynamics*. 83–96.

KIM, S., HAGH-SHENAS, H., AND INTERRANTE, V. 2003. Showing shape with texture: two directions seem better than one. In *Human Vision and Electronic Imaging VIII, SPIE 5007*. 332–339.

KIM, S., HAGH-SHENAS, H., AND INTERRANTE., V. 2004. Conveying shape with texture: experimental investigations of texture's effects on shape categorization judgments. *IEEE Trans. Visuali. Comput. Graph.*, 471–483.

KINDLMANN, G., WHITAKER, R., TASDIZEN, T., AND MÖLLER, T. 2003. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *Proceedings of IEEE Visualization*. 513–520.

KIRBY, R. M., MARMANIS, H., AND LAIDLAW, D. H. 1999. Visualizing multivalued data from 2d incompressible flows using concepts from painting. In *Proceedings of IEEE Visualization*. 333–340.

KNILL, D. 2001. Contour into texture: information content of surface contours and texture flow. *J. Optical Soci. Amer. A*, 1, 12–35.

LAGAE, A. AND DUTRÉ, P. 2005. A procedural object distribution function. *ACM Trans. Graph.*, 1442–1461.

LI, A. AND ZAIDI, Q. 2000. Perception of three-dimensional shape from texture is based on patterns of oriented energy. *Vision Resea. 40*, 2, 217–242.

LORENSEN, W. E. AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of ACM SIGGRAPH*. Vol. 21. 163–169.

LU, A. AND EBERT, D. S. 2005. Example-based volume illustrations. In *Proceedings of IEEE Visualization*. 655–662.

LU, A., MORRIS, C., TAYLOR, J., EBERT, D., RHEINGANS, P.AND HANSEN, C., AND HARTNER, M. 2003. Illustrative interactive stipple rendering. *IEEE Tracs. Visualiz. Comput. Graph. 9*, 2, 127–139.

LUM, E. AND MA, K.-L. 2002. Hardware-accelerated parallel non-photoralistic volume rendering. In *Proceedings of Non-Photorealistic Animation and Rendering*. 67–74.

NAGY, Z., SCHNEIDER, J., AND WESTERMANN, R. 2002. Interactive volume illustration. In *Proceedings of Vision, Modeling, and Visualization*. 497–504.

NEYRET, F. AND CANI, M.-P. 1999. Pattern-based texturing revisited. In *Proceedings of ACM SIGGRAPH*. 235–242.

OWADA, S., NIELSEN, F., OKABE, M., AND IGARASHI, T. 2004. Volume illustration: Designing 3d models with internal textures. In *Proceedings of ACM SIGGRAPH*. 322–328.

PERLIN, K. AND HOFFERT, E. 1989. Hypertexture. In *Proceedings of ACM SIGGRAPH*. 253–262.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. In *Proceedings of ACM SIGGRAPH*. 579–584.

RILEY, K., EBERT, D., HANSEN, C., AND LEVIT, J. 2003. Visually accurate multi-field weather visualization. In *Proceedings of IEEE Visualization*. 279–286.

ROSENHOLTZ, R. AND MALIK, J. 1997. Surface orientation from texture: Isotropy or homogeneity (or both)? *Vision Resear. 37*, 16, 2283–2293.

SAITO, T. 1994. Real-time previewing for volume visualization. In *Proceedings of Symposium on Volume Visualization*. 99–106.

SIBLEY, P. G., MONTGOMERY, P., AND MARAI, G. E. 2004. Wang cubes for video synthesis and geometry placement. In *SIGGRAPH Poster*.

STAM, J. 1997. Aperiodic texture mapping. Tech. Rep. R046, European Research Consortium for Informatics and Mathematics (ERCIM).

STAUBESAND, J. AND TAYLOR, A. N. 1990. *Sobotta Atlas of Human Anatomy. Volume 1: Head, Neck, Upper Limbs, Skin. Volume 2: Thorax, abdomen, Pelvis, Lower Limbs*. Urban and Schwarzenberg Baltimore-Munich.

TODD, J., NORMAN, F., KOENDERINK, J., AND KAPPERS, A. 1997. Effects of texture, illumination, and surface reflectance on stereoscopic shape perception. *Perception 26*, 807–822.

TREAVETT, S. M. F., CHEN, M., SATHERLEY, R., AND JONES, M. W. 2001. Volumes of expression: Artistic modelling and rendering of volume datasets. In *Computer Graphics International*. 99–106.

WANG, H. 1961. Proving theorems by pattern recognition ii. *Bell Syst. Tech. J. 40*, 1–42.

WANG, H. 1965. Games, logic, and computers. *Scient. Amer.* 98–106.

WANGER, L. R., FERWERDA, J. A., AND GREENBERG, D. P. 1992. Perceiving spatial relationships in computer-generated images. *IEEE Comput. Graph. Appli. 12*, 3, 44–58.

WARE, C. AND KNIGHT, W. 1995. Using visual texture for information display. *ACM Trans. Graph. 14*, 1, 3–20.

WITTENBRINK, C., PANG, A., AND LODHA, S. 1996. Glyphs for visualizing uncertainty in vector fields. *IEEE Trans. Visualiz. Comput. Graph. 2*, 266–279.

ZAIDI, Q. AND LI, A. 2002. Limitations on shape information provided by texture cues. *Vision Resear. 42*, 815–835.

Received September 2005; revised August 2006; accepted December 2006