國立交通大學

多媒體工程研究所

碩 士 論 文

由 二 維 紋 理 形 成 具 有 向 量 場 控 制
的 實 體 紋 理 合 成

Solid Texture Synthesis with Vector Field Control
from a 2D Texture

研 究 生：林菊穗

指導教授：施仁忠 教授

張勤振 教授

中 華 民 國 九 十 八 年 六 月

由二維材質形成具有向量場控制的實體紋理合成
Solid Texture Synthesis with Vector Field Control from a 2D Texture

研 究 生：林菊穗　　Student：Chu-Sui Lin

指導教授：施仁忠 教授　　Advisor：Dr. Zen-Chung Shih

張勤振 教授　　Dr.Chin-Chen Chang

國 立 交 通 大 學

多 媒 體 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Multimedia Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年六月

由二維紋理形成具有向量場控制的實體紋理合成

Solid Texture Synthesis with Vector Field Control from a 2D Texture

研究生：林菊穗　　　　　指導教授：施仁忠　教授

張勤振　教授

國立交通大學多媒體工程研究所

摘　要

本論文輸入一張二維紋理來合成實體紋理(solid texture synthesis)。由於二維紋理在三維空間中無法得知足夠的資訊，故本論文利用預先計算好的三維候選點(3D-candidates)來做實體紋理合成，三維候選點是分別由三個方向(x, y, z)的 5×5 二維切面(2D slices)所構成，而且必須保持三個切面交叉軸(crossbar)的顏色一致性(color consistency)，接著利用金字塔合成方法(pyramid synthesis method)來完成實體紋理的合成。在合成過程中使用表面向量值(appearance vector)取代傳統只用色彩值(RGB color value)來做鄰近點比對，有了資料量豐富的表面向量值，我們就可以分別只比對 4 個點來建立 5×5 個點的三平面內的鄰近點(neighborhood)資料，並利用額外的向量場(vector field)來達到紋理控制(texture control)的目的。

Solid Texture Synthesis with Vector Field Control from a 2D Texture

Student：Chu-Sui Lin Advisor：Dr. Zen-Chung Shih

Dr. Chin-Chen Chang

Institute of Multimedia Engineering

National Chiao Tung University

ABASTRACT

In this thesis, we achieve solid texture synthesis from a 2D input texture. Because 2D input texture does not have enough information in 3D space, we introduce a method that using a set of pre-computed 3D-candidates, each being a triple of interleaved 5×5 2D slices from three coordinates. Moreover, our approach can keep the color consistency along the crossbar of the 3D-candidates. Then we can synthesize solid textures by applying pyramid synthesis method. Appearance vectors are used to replace RGB color values. With these information-rich vectors, 4 locations for each coordinate are used to obtain 5×5 slice neighborhoods. Moreover, we introduce our approach for controllable texture synthesis with vector fields.

Acknowledgements

First of all, I would like to thank my advisors, Dr. Zen-Chung Shih and Dr. Chin-Chen Chang, for their supervision and helps in this work. Then I want to thank all the members in Computer Graphics & Virtual Reality Lab for their comments and instructions. Especially two persons, Yu-Ting Tsai and Ya-Lin Su, I want to thank them for their suggestions and helps. Finally, special thanks for my dear family, and this achievement of this work dedicated to them.

Contents

摘　　要 I
ABASTRACT........ II
Acknowledgements III
Contents........ IV
List of Figures V
Chapter 1　Introduction 1
1.1　Motivation 1
1.2　Overview 2
1.3　Thesis Organization 3
Chapter 2　Related Works 5
2.1　Texture Synthesis with Control Mechanism 5
2.2　Solid Texture Synthesis 7
2.3　Texture Synthesis and Vector Field........ 9
Chapter 3　Solid Synthesis Process........ 11
3.1　Feature Vector Generation........ 11
3.2　Similarity Set Generation 13
3.3　3D-Candidate Generation 13
3.4　Pyramid Solid Texture Synthesis........ 15
3.4.1 Pyramid Upsampling 15
3.4.2 Jitter Method........ 16
3.4.3 Voxel Correction 17
Chapter 4　Anisometric Synthesis Process 22
4.1　3D Vector Field 22
4.2　Anisometric Solid Texture Synthesis 25
4.2.1 Pyramid Upsampling 25
4.2.2 Voxel Correction 26
Chapter 5　Implementation and Results........ 31
5.1　Vector Fields........ 32
5.2　Synthesis Results 36
Chapter 6　Conclusions and Future Works 72
Reference........ 74

List of Figures

Figure 1.1 System flowchart 4
Figure 3.1 Overview of texture data transformation 12
Figure 3.2 The process for feature vector generation: 12
Figure 3.3 The diagram of 3D-candidate: 14
Figure 3.4 Synthesis from one voxel to $m \times m \times m$ solid texture 15
Figure 3.5 Twelve neighbors for $N_{s_l}(v)$: 18
Figure 3.6 Three sub-neighbors for each neighbor of voxel v: 19
Figure 3.7 The process of forming triple candidates in direction x: 21
Figure 4.1 5×5×5 3D vector field with orthogonal axes 23
Figure 4.2 5×5×5 3D vector field with a circle pattern on XY plane 24
Figure 4.3 Twelve warped neighbors for $\tilde{N}_{s_l}(v)$: 27
Figure 4.4 Three warped sub-neighbors for warped neighbors of voxel v 29
Figure 4.5 The process of forming warped triple candidates in direction x: 30
Figure 5.1 5×5×5 3D vector field about circular & emissive control 32
Figure 5.2 5×5×5 3D vector field about slant control 33
Figure 5.3 5×5×5 3D vector field about zigzag control 34
Figure 5.4 5×5×5 3D vector field about 3D slant control 35
Figure 5.5 Input and result data for case_1 37
Figure 5.6 Anisometric results with circular control for case_1 38
Figure 5.7 Anisometric results with emissive control for case_1 39
Figure 5.8 Anisometric results with slant control for case_1 40
Figure 5.9 Anisometric results with zigzag control for case_1 41
Figure 5.10 Anisometric results with 3D slant control for case_1 42
Figure 5.11 Input and result data for case_2 44
Figure 5.12 Anisometric results with circular control for case_2 45
Figure 5.13 Anisometric results with emissive control for case_2 46
Figure 5.14 Anisometric results with slant control for case_2 47
Figure 5.15 Anisometric results with zigzag control for case_2 48
Figure 5.16 Anisometric results with 3D slant control for case_2 49
Figure 5.17 Input and result data for case_3 51
Figure 5.18 Anisometric results with circular control for case_3 52
Figure 5.19 Anisometric results with emissive control for case_3 53
Figure 5.20 Anisometric results with slant control for case_3 54

Figure 5.21 Anisometric results with zigzag control for case_3 55
Figure 5.22 Anisometric results with 3D slant control for case_3 56
Figure 5.23 Input and result data for case_4..58
Figure 5.24 Anisometric results with circular control for case_4..59
Figure 5.25 Anisometric results with emissive control for case_4......................................60
Figure 5.26 Anisometric results with slant control for case_4 61
Figure 5.27 Anisometric results with zigzag control for case_4 62
Figure 5.28 Anisometric results with 3D slant control for case_4 63
Figure 5.29 Input and result data for case_5..65
Figure 5.30 Anisometric results with circular control for case_5..66
Figure 5.31 Anisometric results with emissive control for case_5......................................67
Figure 5.32 Anisometric results with slant control for case_5 68
Figure 5.33 Anisometric results with zigzag control for case_5 69
Figure 5.34 Anisometric results with 3D slant control for case_5 70

Chapter 1

Introduction

1.1 Motivation

Nowadays, a lot of textures can be synthesized well in 2D. But there is still a lack of techniques in generating 3D textures. There are different kinds of techniques for 3D surface texturing like texture mapping [8, 23, 24], procedural texturing [5, 15] and image-based surface texturing [18, 20, 22]. 3D surface texturing has some undesired problems, so solid texture is introduced to solve this problems.

Texture mapping is the easiest method for 3D surface texturing, but the quality of the results is not good. It may be suffered from some well-know problems such as distortion, discontinuity, and unwanted seams. Procedural texturing can solve the problems of distortion and discontinuity. However, it still has some drawbacks. Procedural texturing models only limit types of textures, such as marble. Furthermore, users may take time to understand and control parameters. The results will be decided the designers. Image-based surface texturing can synthesize more textures, but it can’t handle large structural textures like bricks. When the curvature is too large, it still has distortion and discontinuous problems. Furthermore, image-based surface texturing is non-reusable such that textures generated for one surface cannot be used for other surfaces.

Solid textures can be used to solve the above problems. By Peachey [14] and Perlin [15], solid textures are blocks of colored points in 3D space to represent real-world materials. Using solid textures, users do not need to find a parameterization for the surface of the object to be textured. Furthermore, solid textures provide texture information inside the entire volume.

Recently, some techniques [3, 4, 11, 16] used three orthogonal slices for neighborhood matching. We take advantage of this idea and present a method for real 3D space texture synthesis from a 2D texture. Moreover, we use vector fields to control the solid texture synthesis. Unlike above methods, we replace 2D k-coherent candidates with 3D-candidates. When we select the best matching candidates, we consider all the three directions instead of each direction independently. We use information-rich appearance vectors for neighborhood matching. The whole process is automatic. The results illustrate that our approach can do well on a wide range of textures.

1.2 Overview

The flowchart of our system is shown in Figure 1.1. First, input a 2D texture and repeat the texture thrice as three directional exemplars. In the pre-process, it generates feature vectors, similarity sets, and 3D-candidates for each pixel. For feature vector generation, it captures 5×5 window information and uses principal component analysis (PCA) to decrease the dimensions. For similarity set generation, it finds the three pixels most similar to each pixel. For 3D-candidates generation, it computes a small candidate set from the other two exemplars for each pixel. In the synthesis process, we apply the pyramid synthesis method [12] to our system. Upsampling, jitter and correction are used at each synthesis level to obtain the results. For anisometric synthesis, vector fields are used to control the synthesis. We input a vector field as the anisometric field and compute the inverse anisometric field. These two fields are used in synthesis process to get the results.

The major contributions of this thesis are as follows: First, we present an approach for synthesizing solid textures from a 2D texture. Using appearance vectors, only 12 locations (4 locations for each direction) are needed to synthesize solid textures. Second, we present a coherent anisometric synthesis method for solid textures according to vector field control.

1.3 Thesis Organization

The rest of this thesis is organized as follows: In Chapter 2, we review related works about texture synthesis with control and solid texture synthesis. In Chapter 3, we present our approach for synthesizing solid textures from a 2D texture. Chapter 4 presents the proposed anisometric synthesis approach for solid textures based on vector field control. Chapter 5 shows the implementation and results. Finally, conclusions and future works are discussed in Chapter 6.

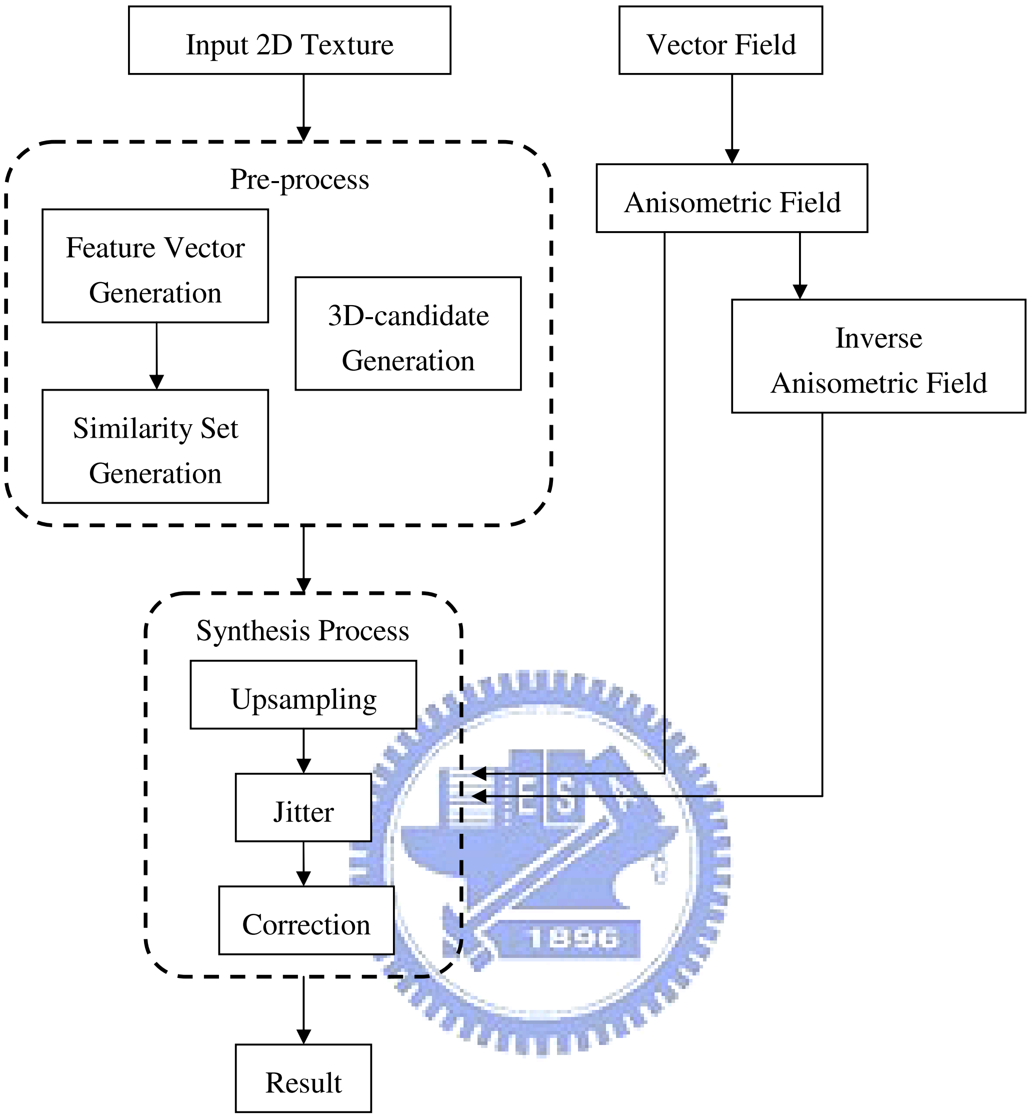


Figure 1.1 System flowchart

Chapter 2

Related Works

In this chapter, we review some recent and representative works.

2.1 Texture Synthesis with Control Mechanism

Ashikhmin [1] presented a texture synthesis algorithm for natural textures. He provided an interactive interface for users to control the texture synthesis process. He introduced an idea of “shifted-candidates” for neighborhood matching and found the best similar candidate. This approach uses a smaller neighborhood to obtain the quality characteristics of a larger neighborhood and maintains the coherence of the results. Users can use painting-style interface to indicate large-scale properties of the texture. Furrhermore, this method is fast and straightforward for the users. But it could not obtain good results if the user’s control does not contain significant amount of high frequency components.

Lefebvre and Hoppe [12] introduced a high-quality and parallel pyramid synthesis algorithm. Their method includes upsampling step to maintain patch coherence, jittering of exemplar coordinates to increase the randomness of the texture, and an order-independent correction step to keep the similarities between synthesized results and input textures. They obtained high-quality and efficient results thanks to the order-independent correction step. It corrects the pixel coordinate for more accurate neighborhood matching and the whole step can be divided into several subpasses to promote performance. However, it still has one drawback: when the features are too large to be captured by small neighborhoods, it could perform poorly. This drawback is also a well-known problem for other neighborhood-based per-pixel synthesis methods.

Lefebvre and Hoppe [13] proposed a structure for exemplar-based texture synthesis with anisometric control. They replaced traditional RGB values with appearance vectors for neighborhood matching. Their appearance space reduces runtime neighborhood vectors from 5x5 grids to only 4 locations, so the synthesis is more efficient, and the quality of results is good because of the information-rich appearance vectors. In addition, they combined their pyramid synthesis with appearance vectors to accelerate neighborhood matching and proposed novel methods for coherent anisometric synthesis which makes arbitrary affine deformation on textures. They presented a convenient way to control textures.

Kwatra et al. [9] introduced a method to control flows on 2D textures and presented an algorithm to perform texture control on 3D surfaces [10]. They proposed a vector advection technique with global texture synthesis to accomplish dynamically changing fluid surfaces. Users can define fluid velocity field to control the texture results on 3D surfaces. The neighborhood construction step in the process considers orientation coherent with the user-defined velocity field. This method not only makes the synthesized results similar to the input texture, but also keeps temporally coherent. However, it is difficult for users to define an orientation velocity field which is smooth everywhere.

2.2 Solid Texture Synthesis

Jagnow et al. [7] used traditional stereological methods to synthesize 3D solid textures from 2D images. They synthesized solid textures for spherical particles first and then extended the technique to apply to arbitrary-shaped particles. Their approach needs cross-section images to record the distribution of circle sizes on 2D slices, and then builds the relationship of 2D profile density and 3D particle density. Users could use the particle density to add one particle at a time to reconstruct the volume data, so it means the step is manual. This method uses many 2D profiles to construct 3D density for volume results, which are good for marble textures. However, their system is not automatic and only for particle textures.

Chiou and Yang [2] improved the above system to automatic process. They divided the synthesis into two parts: 2D analysis phase and 3D synthesis phase. First, they collected essential statistics to develop a probability model. Then they used this probability model to control the variation in particle size through the 3D synthesis procedure. However, this system inherits the above system so it is only for particle textures.

Qin et al. [16] introduced an image-based solid texturing in terms of basic gray-level aura matrices (BGLAMs) framework. They replaced traditional gray-level histograms with BGLAMs for neighborhood matching. They defined aura matrices from input exemplars and generated a solid texture from multiple view directions. In the volume result, they will only consider the pixels on the three orthogonal slices for neighborhood matching. The whole system is fully automatic without user interactions. Furthermore, they can generate reliable results for both stochastic and structural textures. However, it needs large storages for large matrices and the results are not good for color textures.

Kopf et al. [11] provided a solid texture synthesis method from 2D exemplars. They took advantage of 2D texture optimization techniques to do 3D solid texture synthesis and then preserved global statistical properties by histogram matching to achieve optimization. For each voxel, they only considered the neighborhood coherence in three orthogonal slices, and increased the similarity between the solid textures and the exemplar iteratively. Their approach could do well for wide range of textures**.** But they synthesized the texture with the information on the slices.

Takayama et al. [17] presented a method which fills a model with anisotropic textures. They had some volume textures and defined it how to map to 3D objects. Then they pasted solid texture exemplars repeatedly on the 3D object. Volumetric tensor over the mesh can be set by users, and the texture patches are located according to these fields. This method still has drawbacks. The patch seams became noticeable when using a texture with strong low-frequency components and the blurring artifacts appeared when using highly structured textures.

Dong et al. [4] introduced a new algorithm to restrict synthesis to a subset of the voxels, while enforcing spatial determinism. They reduced the dependency chain of neighborhood matching, so that each voxel only depended on a small number of other voxels. They synthesized a volume by using pre-computed 3D-candidates, each being a triple of interleaved 2D neighborhoods. These 3D-candidates are selected carefully to form consistent triples. Their approach could generate good results efficiently. However, if the three exemplars do not define a coherent 3D volume, the quality of the result will be poor.

2.3 Texture Synthesis and Vector Field

Many patterns are created by interactions between texture elements and surface geometry, so Turk [18] synthesized a texture directly on the surface of the model. An orientation field must be specified over the surface in order to preserve the directional nature of textures. They did this by allowing users to pick the directions at several locations and then interpolating vectors over the rest of the surface.

Ying et al. [22] presented a method that synthesized the texture directly on the surface, rather than synthesizing a texture image and mapping it to the surface. A 2D vector field was used to specify the correspondence between orientation on the surface and orientation in the domain of the example texture. A pair of orthogonal tangent vector fields is used for the purpose. They got one field first, and the second field is computed as the cross-product of the first field and the oriented surface normal.

Taponecco et al. [19] used Markov Random Field (MRF) texture synthesis method to implement 2D vector field visualization. In their approach, they defined a vector field by magnitude and direction. The magnitude could be easily computed using an appropriate norm of the values. Assigning the direction required a projection of the vector onto the image plane, and then the angle of the tangent in the vector field relative to the x-axis is gotten. The magnitude and direction were used to scale and rotate the image.

Wu et al. [21] proposed an approach to directly synthesize texture on arbitrary surface with texture sample. It had a tangential vector field which indicated the desired growing orientation of the texture on the surface. First, users specified tangential vectors at a few seed triangles, and then they interpolated vectors at the remaining triangles to build a tangential vector field. They recursively mapped triangles to texture space until the whole surface is mapped completely based on the vector field.

Chapter 3

Solid Synthesis Process

In this section, we present our approach for synthesizing solid textures from 2D textures. In Section 3.1, we describe feature vectors in appearance space and how we get the feature vectors. Then we use the similarity set to accelerate neighborhood matching in Section 3.2. In Section 3.3, we keep the color coherence of the three orthogonal slices by computing 3D-candidates. We introduce how to apply 2D pyramid texture synthesis to solid texture synthesis in Section 3.4. The upsampling process increases the texture sizes between different levels, that each voxel in parent level generates eight voxels in children level. The jitter step perturbs the textures to achieve deterministic randomness. The last step in pyramid solid synthesis is voexl correction, using neighborhood matching to make the results more similar to the exemplar.

3.1 Feature Vector Generation

Solid texture synthesis using RGB color values for neighborhood matching needs larger neighborhood size and data. Appearance vectors have been proved that they are continuous and low-dimensional for neighborhood matching. Hence, we transform the texture data values in color space into feature vectors in appearance space. As shown in Fig. 3.1, we transform texture data T into appearance space texture data T'.

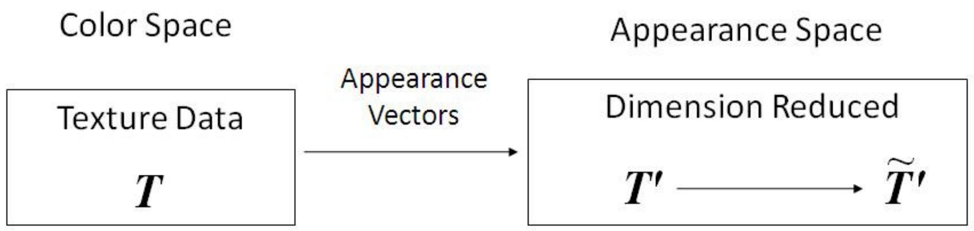


Figure 3.1 Overview of texture data transformation

According to Lefebvre and Hoppe [13], we take the RGB values in 5×5 windows (Fig. 3.2 (b)) to construct appearance vectors for each pixel of an input texture T (Fig. 3.2 (a)). The exemplar T' consists of the feature vectors at each pixel. There are 75 dimensions (25 for grids and 3 for RGB) for each pixel in T', and then we perform principal component analysis (PCA) to reduce the dimensions for a transformed exemplar $\tilde{T}'$ (Fig. 3.2 (c)).

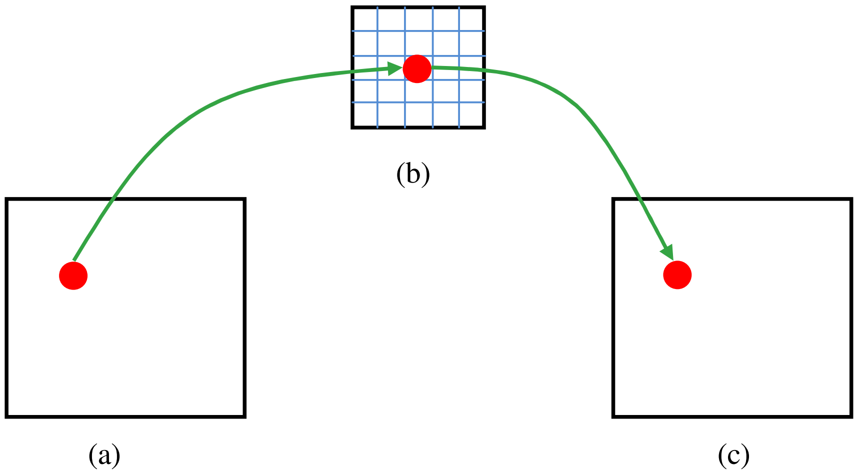


Figure 3.2 The process for feature vector generation:

(a) input texture data T (b) 5×5 windows structure for feature vectors

(c) transformed exemplar $\tilde{T}'$

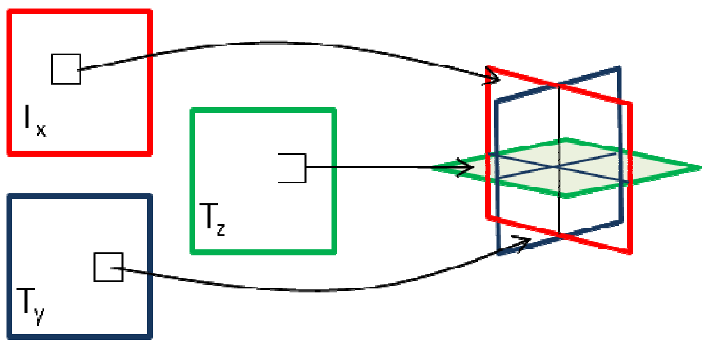
3.2 Similarity Set Generation

By the k-coherence search method [24], searching from the candidates can accelerate neighborhood matching because we do not have to search from each pixel in the exemplar for neighborhood matching. Thus, we construct a similarity set to record the candidates similar to each pixel.

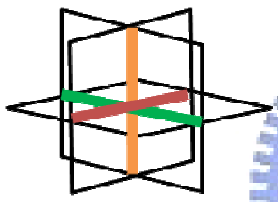
Based on the principle of coherence synthesis [1], searching candidates from the $n \times n$ neighbors of pixel p in the exemplar T can accelerate the synthesis process. Hence, we find the k most similar pixels from the $n \times n$ neighbors of pixel p in the transformed exemplar $\tilde{T}'$ to construct the similarity set $C^l_{1...k}(p)$ for pixel p, where k is a user-defined parameter, and l is the pyramid level, $C^l_1(p) = p$. Note that n is a user-defined parameter to control the window size for coherent synthesis. In the experiments, n is set as 7.

3.3 3D-Candidate Generation

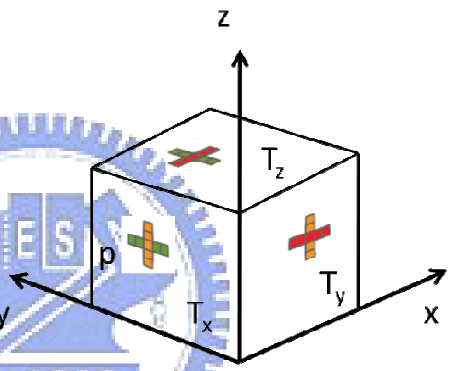
First, we repeat the input texture thrice as three directional exemplars T_x, T_y and T_z. By Dong et al. [4], we generate a small set of 3D-candidates for each pixel of the three exemplars to build the relation between the three exemplars. The 3D-candidate is by three 5x5 slices (2D neighborhoods) (Fig. 3.3 (a)). It is important that a suitable candidate should be consistent across the crossbar. The crossbar is the strip that is intersected by two slices (Fig. 3.3 (b)). Therefore, we seek to minimize the color disparity between the strips shared by interleaved slices.



(a)



(b)



(c)

Figure 3.3 The diagram of 3D-candidate:

(a) Three input exemplars T_x, T_y, T_z and a corresponding 3D-candidate.

(b) The crossbars are defined by the three slices.

(c) A consistent triple from each exemplar.

We define the triple to be the three coordinates which is pointed by the center pixel of the three slices. For each pixel of each exemplar, we form triples using the pixel and two neighborhoods from the other two exemplars. We select the triples which have the smaller crossbar error. The 3D-candidate set is composed of these triples. For instance, as shown in Fig. 3.3(c), assuming we want to compute a 3D-candidate set for pixel p in T_x, we first find in T_y the K pixel strips best matching the orange line from T_x, and in T_z the K pixel strips best matching the green line from T_x. We use the current pixel p as the third coordinate to produce all possible K^2 triples, and then order them based on the crossbar error which is the sum of color differences for the three of pixel strips. In the experiments, we keep the 12 best triples as 3D-candidates for each pixel and set a value K as 65.

3.4 Pyramid Solid Texture Synthesis

3.4.1 Pyramid Upsampling

The pyramid synthesis method [6] synthesizes textures from coarse level to fine level. There are $l+1$ levels in synthesis process, l=0~$\log_2 m$, where m is the size of the target texture. We apply this 2D pyramid synthesis method to 3D space.

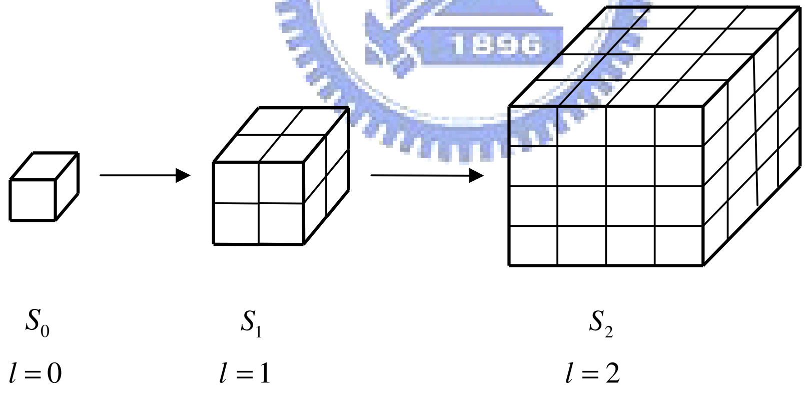


Figure 3.4 Synthesis from one voxel to $m \times m \times m$ solid texture

In our approach, we synthesize from one voxel to a m×m×m solid texture, from $S_0 \sim S_L$, where $L = \log_2 m$, as shown in Fig. 3.4. We synthesize a volume data S in which each voxel $S[v]$ stores three coordinate values, indicating $\tilde{T}_x'$, $\tilde{T}_y'$, $\tilde{T}_z'$

exemplar's pixel, respectively. First, we build a voxel and assign value (1,1), (1,1), (1,1) to it as triple coordinates. Then, we upsample the coordinates of parent voxels for next level, assigning each eight children the scaled parent coordinates plus child-dependent offset as

$$S_l[ijk]_x = S_{l-1}[\left\lceil \frac{i}{2} \right\rceil, \left\lceil \frac{j}{2} \right\rceil, \left\lceil \frac{k}{2} \right\rceil]_x + h_l(j \bmod 2, k \bmod 2),$$

$$S_l[ijk]_y = S_{l-1}[\left\lceil \frac{i}{2} \right\rceil, \left\lceil \frac{j}{2} \right\rceil, \left\lceil \frac{k}{2} \right\rceil]_y + h_l(i \bmod 2, k \bmod 2),$$

$$S_l[ijk]_z = S_{l-1}[\left\lceil \frac{i}{2} \right\rceil, \left\lceil \frac{j}{2} \right\rceil, \left\lceil \frac{k}{2} \right\rceil]_z + h_l(i \bmod 2, j \bmod 2),$$

$$h_l = 2^{(\log_2 m - l)}$$

where h_l denotes the regular output spacing of exemplar coordinates, and ijk means the location of voxel v. $S_l[ijk]_x$ is the new coordinate value at location ijk within the volume of level l for exemplar $\tilde{T}_x'$. $S_l[ijk]_y$ is the new coordinate value at location ijk within the volume of level l for exemplar $\tilde{T}_y'$. $S_l[ijk]_z$ is the new coordinate value at location ijk within the volume of level l for exemplar $\tilde{T}_z'$.

3.4.2 Jitter Method

After upsampling the coordinates, we have to jitter our texture to achieve deterministic randomness. We plus the upsampled coordinates at each level a jitter function value to perturb it. The jitter function $J_l(v)$ is produced by a hash function $H(v): \mathbb{Z}^2 \to [-1,+1]^2$ and a user-defined parameter r_l.

$$S_l[v] = S_l[v] + J_l(v)$$

$$J_l(v) = h_l H(v) r_l$$

3.4.3 Voxel Correction

In order to make the coordinates similar to those in the exemplars T_x, T_y, T_z, we take the jittered results to recreate neighbors. There is a feature value for each pixel after constructing feature vectors. For each voxel v, we collect the feature values of its neighbors to obtain the neighborhood vectors $N_{s_l}(v)_x$, $N_{s_l}(v)_y$, $N_{s_l}(v)_z$ for each direction, respectively. Then, we search the most similar pixel from the transformed exemplars $\tilde{T}_x'$, $\tilde{T}_y'$, $\tilde{T}_z'$ to make the result similar to the exemplars T_x, T_y, T_z.

In neighborhood matching, we take 4 diagonal locations for voxel v in each direction to obtain the neighborhood vectors $N_{s_l}(v)_x$, $N_{s_l}(v)_y$, $N_{s_l}(v)_z$:

$$N_{s_l}(v)_x = \left\{ \tilde{T}_x'[S[v+\Delta_x]] \,\middle|\, \Delta_x = \begin{pmatrix} 0 \\ \pm 1 \\ \pm 1 \end{pmatrix} \right\}$$

$$N_{s_l}(v)_y = \left\{ \tilde{T}_y'[S[v+\Delta_y]] \,\middle|\, \Delta_y = \begin{pmatrix} \pm 1 \\ 0 \\ \pm 1 \end{pmatrix} \right\}$$

$$N_{s_l}(v)_z = \left\{ \tilde{T}_z'[S[v+\Delta_z]] \,\middle|\, \Delta_z = \begin{pmatrix} \pm 1 \\ \pm 1 \\ 0 \end{pmatrix} \right\}$$

Fig. 3.5 shows that each direction has 4 diagonal locations for each voxel v.

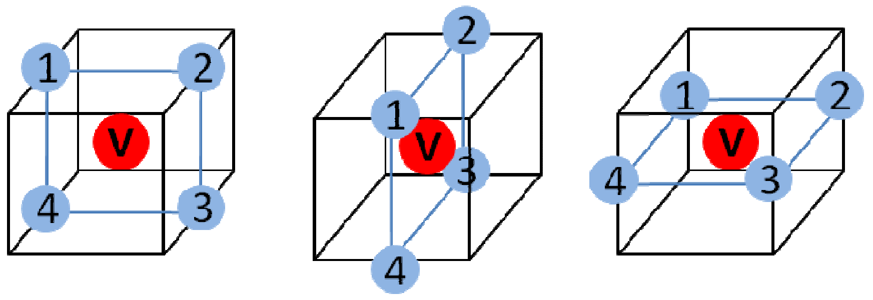


Figure 3.5 Twelve neighbors for $N_{s_l}(v)$:
(a) 4 neighbors for $N_{s_l}(v)_x$ (b) 4 neighbors for $N_{s_l}(v)_y$
(c) 4 neighbors for $N_{s_l}(v)_z$

Based on [13], for $N_{s_l}(v)_x$, we average the voxels nearby voxel $v+\Delta_x$ to improve convergence without increasing the size of the neighborhood vector $N_{s_l}(v)_x$. We average the appearance values from 3 synthesized voxels nearby $v+\Delta_x$ as the new feature value at voxel $v+\Delta_x$, and then use the new feature values at 4 diagonal voxel to construct neighborhood vector $N_{s_l}(v)_x$. Also, $N_{s_l}(v)_y$ and $N_{s_l}(v)_z$ can be done by the same process. $N_{s_l}(v;\Delta_x)$ means the averaged feature value at voxel $v+\Delta_x$. $N_{s_l}(v;\Delta_y)$ means the averaged feature value at voxel $v+\Delta_y$. $N_{s_l}(v;\Delta_z)$ means the averaged feature value at voxel $v+\Delta_z$.

$$\boldsymbol{N}_{s_l}(v;\Delta_x)=\frac{1}{3}\sum_{\Delta'=M\Delta_x, M\in\Psi_x}\tilde{T}_x{}'[S[v+\Delta_x+\Delta']-\Delta']$$

$$\Psi_x \in \{\begin{pmatrix}0&0&0\\0&0&0\\0&0&0\end{pmatrix},\begin{pmatrix}0&0&0\\0&1&0\\0&0&0\end{pmatrix},\begin{pmatrix}0&0&0\\0&0&0\\0&0&1\end{pmatrix}\}$$

$$\boldsymbol{N}_{s_l}(v;\Delta_y)=\frac{1}{3}\sum_{\Delta'=M\Delta_y, M\in\Psi_y}\tilde{T}_y{}'[S[v+\Delta_y+\Delta']-\Delta']$$

$$\Psi_y \in \{ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \}$$

$$N_{s_l}(v;\Delta_z) = \frac{1}{3}\sum\nolimits_{\Delta'=M\Delta_z, M\in\Psi_z} \tilde{T}_z{}'[S[v+\Delta_z+\Delta']-\Delta']$$

$$\Psi_z \in \{ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \}$$

Fig. 3.6 shows the locations of 3 synthesized voxels for each neighbor.

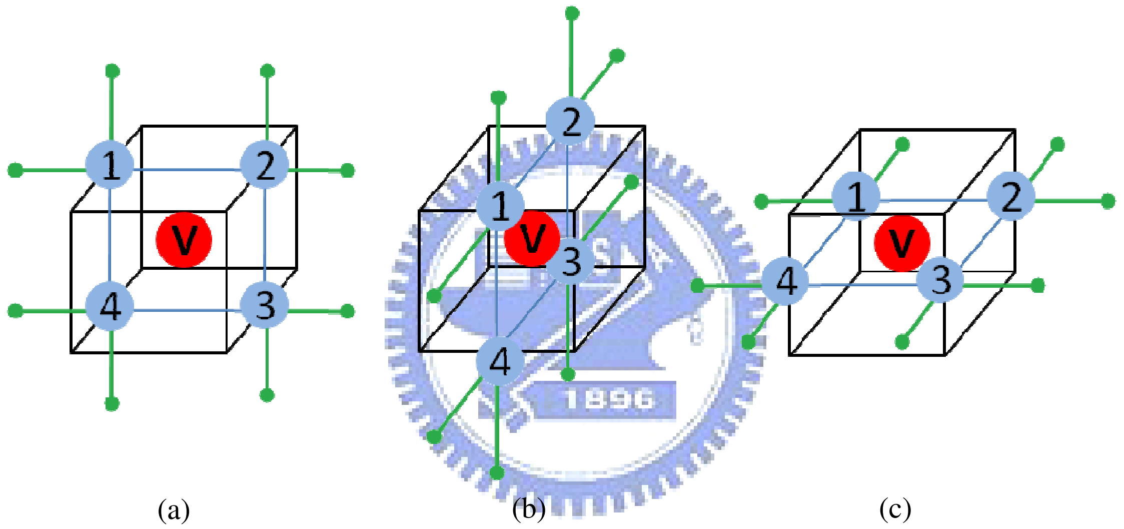


Figure 3.6 Three sub-neighbors for each neighbor of voxel v:
(a) in $N_{s_l}(v)_x$ (b) in $N_{s_l}(v)_y$ (c) in $N_{s_l}(v)_z$

We use the similarity sets and coherence synthesis method in the searching process, utilizing the 4 voxels for each direction nearby voxel v. Taking direction x for example, for the neighbor voxel i_x ($i_x = 1 \sim 4$), we can get the most similar 3 pixels (i_{x1}, i_{x2}, i_{x3}) in exemplar T_x for voxel i_x from the similarity set. And then we the offset between voxel i_x and voxel v from 3D space to 2D space to infer the candidates $(i_{x1v}, i_{x2v}, i_{x3v})$ in exemplar T_x for voxel v (Fig. 3.7(b)). In order to keep color consistence in three directions, we use the 3D-candidate set to infer the other two coordinates ($i_{(x\to y)v}$, $i_{(x\to z)v}$) in exemplars T_y and T_z (Fig. 3.7(c)). In addition, directions y, z are done by the same steps.

Now, for each v, we can get a set of triple candidates $TC_{1...k}$ which point towards pixels in exemplars T_x, T_y, T_z. We compute the neighborhood vectors $N_{s_l}(TC_{1...k})_x$, $N_{s_l}(TC_{1...k})_y$, $N_{s_l}(TC_{1...k})_z$ by the averaged feature values from the 4 nearby pixels. Finally, we sum up the total difference between $N_{s_l}(TC_{1...k})_x$, $N_{s_l}(TC_{1...k})_y$, $N_{s_l}(TC_{1...k})_z$ and $N_{s_l}(v)_x$, $N_{s_l}(v)_y$, $N_{s_l}(v)_z$, then replace the triple coordinate for voxel v with the best matching triple candidate.

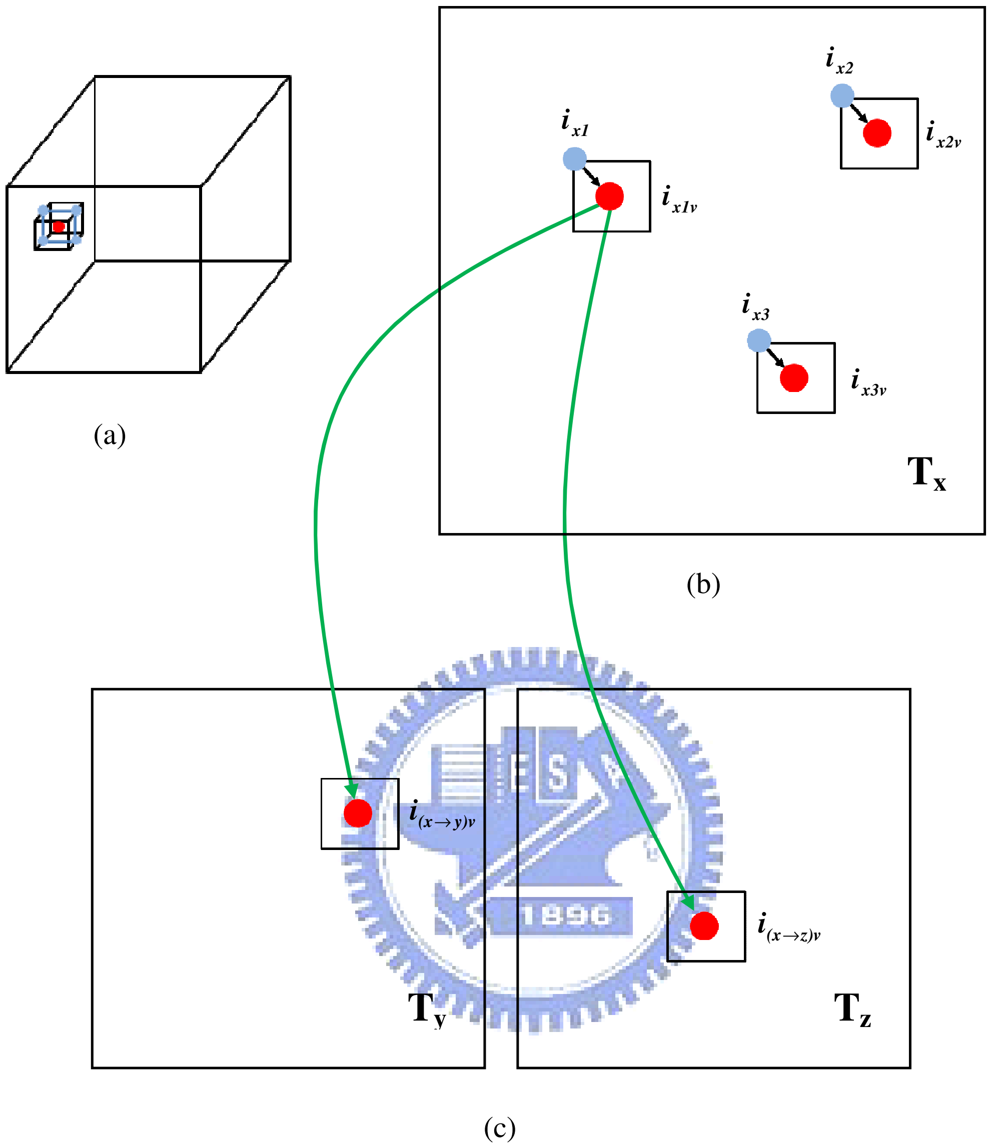


Figure 3.7 The process of forming triple candidates in direction x:

(a) A voxel v is corrected.

(b) The candidates in exemplar T_x.

(c) Find the other two candidates in exemplar T_y and T_z.

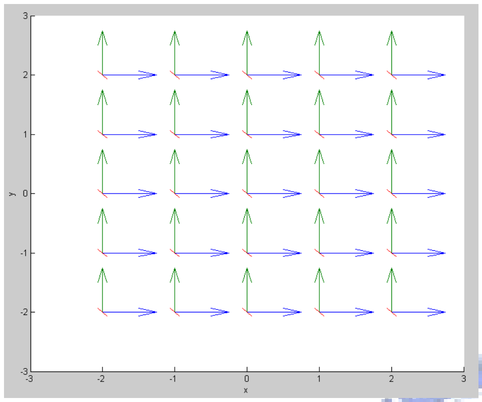
Chapter 4
Anisometric Synthesis Process

In this section, we present the proposed anisometric synthesis approach for solid textures with vector field control. In Section 4.1, we introduce 3D vector fields and how we generate anisometric fields and inverse anisometric fields with the 3D vector fields. In Section 4.2, we introduce the differences between solid synthesis and anisometric synthesis. These different steps are upsampling and correction. The jitter step is the same as it in the solid synthesis process.

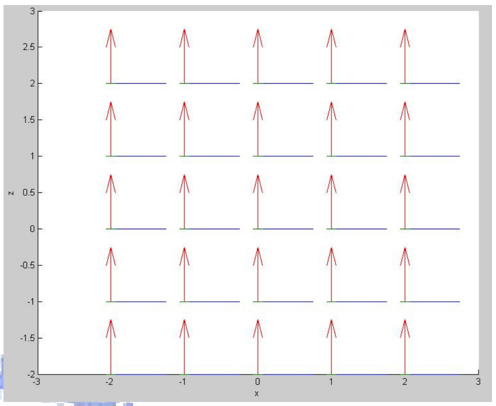
4.1 3D Vector Field

We need the user-defined 3D vector fields to implement anisometric solid texture synthesis. We use these 3D vector fields to control the result.

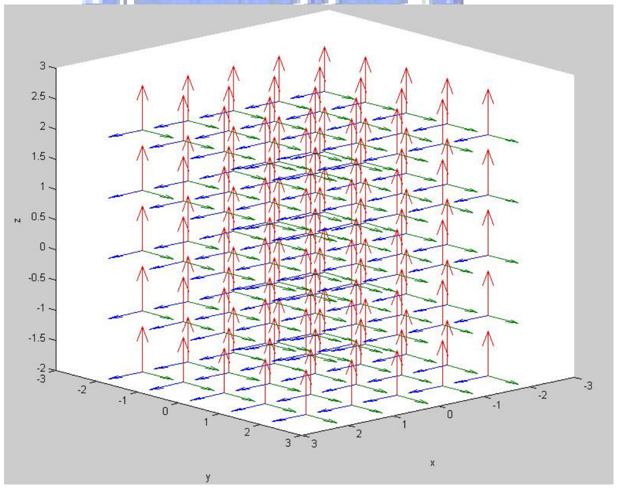
First, we design a 3D space which contains three orthogonal axes at every point, then we use mathematics formulas to control the three axes. Fig. 4.1 shows a 3D vector field with orthogonal axes at each point, and the space size is 5×5×5. We make the three axes various, and expect that the texture results would be changed with the fields. For example, we design a circular field, and there will be a circular pattern on the texture. Fig. 4.2 shows the 3D vector field with a circular pattern on XY plane. The vector field should be the same size as the texture result.



(a)



(b)

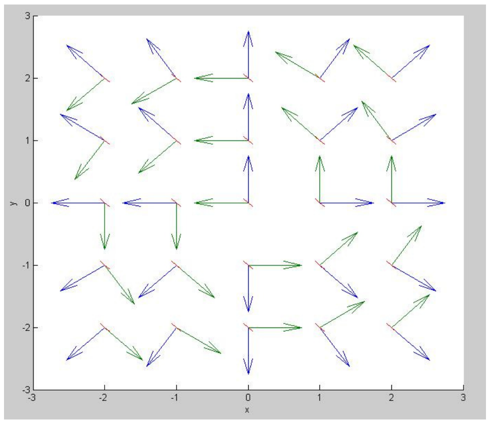


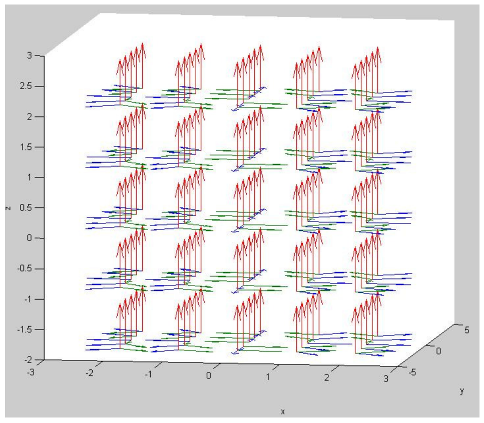
(c)

Figure 4.1 5×5×5 3D vector field with orthogonal axes

(a) XY plane (b) XZ plane

(c) three orthogonal axes at each point


(a)


(b)

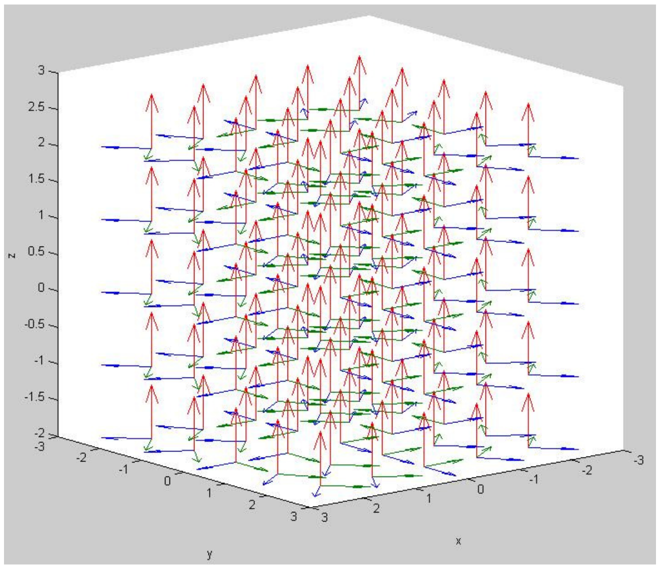

(c)

Figure 4.2 5×5×5 3D vector field with a circle pattern on XY plane

(a) XY plane (b) and (c) are three axes at each point

For each level, we have to make the anisometric field A and the inverse anisometric field A^{-1} based on the user-defined 3D vector field. The anisometric field A is created by downsampling the 3D vector field, and we obtain A_l for each level. Afterward we inverse the A_l to get inverse anisometric field A_l^{-1} for each level. In anisometrc synthesis process, the upsampling and correction steps will refer to the fields

A_l and A_l^{-1} at each level.

4.2 Anisometric Solid Texture Synthesis

4.2.1 Pyramid Upsampling

The goal for upsmapling step in anisometric synthesis is the same as it in isometric synthesis. It synthesizes from coarse level to fine level. We upsample the coordinate values of parent voxels for the next level.

The difference is that the child-dependent offset for upsmapling step is dependent on the anisometric field A. The anisometric field A is used to compute the distance for spacing.

$$S_l[v]_x = S_{l-1}[v-\Delta_x]_x + h_l\Delta_x \cdot A_l(v)$$
$$S_l[v]_y = S_{l-1}[v-\Delta_y]_y + h_l\Delta_y \cdot A_l(v)$$
$$S_l[v]_z = S_{l-1}[v-\Delta_z]_z + h_l\Delta_z \cdot A_l(v)$$
$$h_l = 2^{(\log_2 m - l)}$$

$$\Delta_x \in \{ \begin{pmatrix} 0 \\ -0.5 \\ -0.5 \end{pmatrix}, \begin{pmatrix} 0 \\ 0.5 \\ -0.5 \end{pmatrix}, \begin{pmatrix} 0 \\ -0.5 \\ 0.5 \end{pmatrix}, \begin{pmatrix} 0 \\ 0.5 \\ 0.5 \end{pmatrix}, \begin{pmatrix} 0 \\ -0.5 \\ -0.5 \end{pmatrix}, \begin{pmatrix} 0 \\ 0.5 \\ -0.5 \end{pmatrix}, \begin{pmatrix} 0 \\ -0.5 \\ 0.5 \end{pmatrix}, \begin{pmatrix} 0 \\ 0.5 \\ 0.5 \end{pmatrix} \}$$

$$\Delta_y \in \{ \begin{pmatrix} -0.5 \\ 0 \\ -0.5 \end{pmatrix}, \begin{pmatrix} 0.5 \\ 0 \\ -0.5 \end{pmatrix}, \begin{pmatrix} -0.5 \\ 0 \\ 0.5 \end{pmatrix}, \begin{pmatrix} 0.5 \\ 0 \\ 0.5 \end{pmatrix}, \begin{pmatrix} -0.5 \\ 0 \\ -0.5 \end{pmatrix}, \begin{pmatrix} 0.5 \\ 0 \\ -0.5 \end{pmatrix}, \begin{pmatrix} -0.5 \\ 0 \\ 0.5 \end{pmatrix}, \begin{pmatrix} 0.5 \\ 0 \\ 0.5 \end{pmatrix} \}$$

$$\Delta_z \in \{ \begin{pmatrix} -0.5 \\ -0.5 \\ 0 \end{pmatrix}, \begin{pmatrix} 0.5 \\ -0.5 \\ 0 \end{pmatrix}, \begin{pmatrix} -0.5 \\ 0.5 \\ 0 \end{pmatrix}, \begin{pmatrix} 0.5 \\ 0.5 \\ 0 \end{pmatrix}, \begin{pmatrix} -0.5 \\ -0.5 \\ 0 \end{pmatrix}, \begin{pmatrix} 0.5 \\ -0.5 \\ 0 \end{pmatrix}, \begin{pmatrix} -0.5 \\ 0.5 \\ 0 \end{pmatrix}, \begin{pmatrix} 0.5 \\ 0.5 \\ 0 \end{pmatrix} \}$$

where h_l is the regular output spacing of exemplar coordinates. Δ_x means the relative locations for 8 children in direction x, and Δ_y, Δ_z as well.

4.2.2 Voxel Correction

The goal for correction step is to make the coordinates similar to those in the exemplars T_x, T_y, T_z. For each voxel v, we collect the feature values of warped neighbors by the anisometric field A_l and the inverse anisometric field A_l^{-1} to obtain $N_{s_l}(v)_x$, $N_{s_l}(v)_y$, $N_{s_l}(v)_z$. Then we search the most similar voxel from the transformed exemplars $\tilde{T}_x^{'}$, $\tilde{T}_y^{'}$, $\tilde{T}_z^{'}$ to make the result similar to the exemplars T_x, T_y, T_z according to the 3D vector field.

Lefebvre and Hoppe [13] presented a method for anisometric synthesis which is able to reproduce arbitrary affine deformations, including shears and non-uniform scales. They only accessed immediate neighbors of pixel p to construct the neighborhood vector $N_{s_l}(p)$. They used the Jacobian field J and the inverse Jacobian field J^{-1} to infer which pixel neighbors to access, and the results will be transformed by the inverse Jacobian field J^{-1} at the current point. We apply this method to 3D space.

First, we have to know which 4 voxel neighbors in each direction to voxel v. We use the inverse anisometric field A_l^{-1} to infer the 4 warped neighbors for voxel v, and construct the three warped neighborhood vectors $\tilde{N}_{s_l}(v)_x$, $\tilde{N}_{s_l}(v)_y$, $\tilde{N}_{s_l}(v)_z$:

$$\tilde{N}_{s_l}(v)_x = \left\{ \tilde{T}_x^{'}[S[v+\tilde{\varphi}(\Delta_x)]] \;\middle|\; \Delta_x = \begin{pmatrix} 0 \\ \pm 1 \\ \pm 1 \end{pmatrix} \right\}$$

$$\varphi(\Delta_x) = A_l^{-1}(v) \cdot \Delta_x$$

$$\tilde{\varphi}(\Delta_x) = \frac{\varphi(\Delta_x)}{\left\| \varphi(\Delta_x) \right\|}$$

$$\tilde{N}_{s_l}(v)_y = \left\{ \tilde{T}_y^{'}[S[v+\tilde{\varphi}(\Delta_y)] \middle| \Delta_y = \begin{pmatrix} \pm 1 \\ 0 \\ \pm 1 \end{pmatrix} \right\}$$

$$\varphi(\Delta_y) = A_l^{-1}(v) \cdot \Delta_y$$

$$\tilde{\varphi}(\Delta_y) = \frac{\varphi(\Delta_y)}{\|\varphi(\Delta_y)\|}$$

$$\tilde{N}_{s_l}(v)_z = \left\{ \tilde{T}_z^{'}[S[v+\tilde{\varphi}(\Delta_z)] \middle| \Delta_z = \begin{pmatrix} \pm 1 \\ \pm 1 \\ 0 \end{pmatrix} \right\}$$

$$\varphi(\Delta_z) = A_l^{-1}(v) \cdot \Delta_z$$

$$\tilde{\varphi}(\Delta_z) = \frac{\varphi(\Delta_z)}{\|\varphi(\Delta_z)\|}$$

where $\tilde{\varphi}(\Delta)$ keeps its rotation but removes any scaling.

Fig. 4.3 shows the 4 warped neighbors for each voxel v for each direction. Their locations are changed from diagonal locations because of the inverse anisometric field A_l^{-1}.

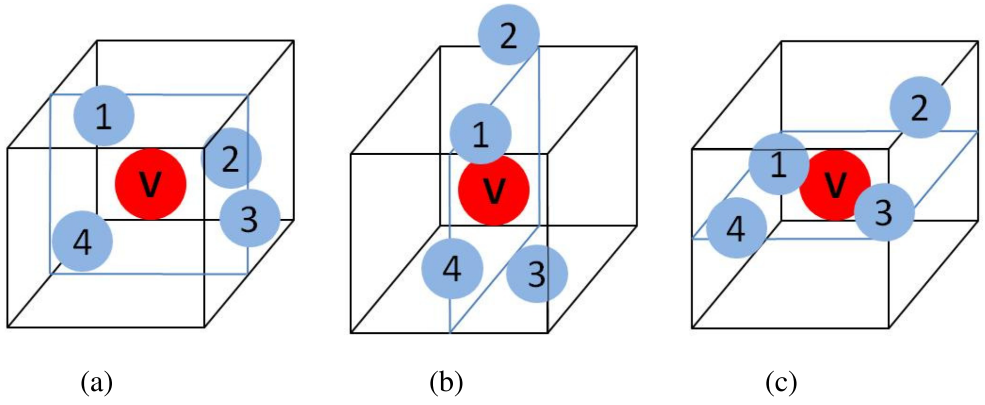


Figure 4.3 Twelve warped neighbors for $\tilde{N}_{s_l}(v)$:

(a) 4 neighbors for $\tilde{N}_{s_l}(v)_x$ (b) 4 neighbors for $\tilde{N}_{s_l}(v)_y$

(c) 4 neighbors for $\tilde{N}_{s_l}(v)_z$

Second, we have to find the 3 synthesized voxels nearby warped neoghborhod voxels of voxel v for each direction. Taking direction x for example, we use the inverse anisometric field A^{-1} to infer the 3 synthesized voxels for voxel $v+\tilde{\varphi}(\Delta_x)$, and compute the averaged feature value as the new feature value at $v+\tilde{\varphi}(\Delta_x)$. Also, directions y, z are done by the same process. Fig. 4.4 shows the locations of 3 warped synthesized voxels for each warped neighbor.

$$\tilde{N}_{sl}(v;\Delta_x) = \frac{1}{3}\sum_{\tilde{\varphi}'(\Delta_x)=\tilde{\varphi}(M\Delta_x), M\in\Psi_x} \tilde{T}_x{}'[S[v+\tilde{\varphi}(\Delta_x)+\tilde{\varphi}'(\Delta_x)]-M\Delta_x]$$

$$\Psi_x \in \{ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \}$$

$$\tilde{N}_{sl}(v;\Delta_y) = \frac{1}{3}\sum_{\tilde{\varphi}'(\Delta_y)=\tilde{\varphi}(M\Delta_y), M\in\Psi_y} \tilde{T}_y{}'[S[v+\tilde{\varphi}(\Delta_y)+\tilde{\varphi}'(\Delta_y)]-M\Delta_y]$$

$$\Psi_y \in \{ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \}$$

$$\tilde{N}_{sl}(v;\Delta_z) = \frac{1}{3}\sum_{\tilde{\varphi}'(\Delta_z)=\tilde{\varphi}(M\Delta_z), M\in\Psi_z} \tilde{T}_z{}'[S[v+\tilde{\varphi}(\Delta_z)+\tilde{\varphi}'(\Delta_z)]-M\Delta_z]$$

$$\Psi_z \in \{ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \}$$

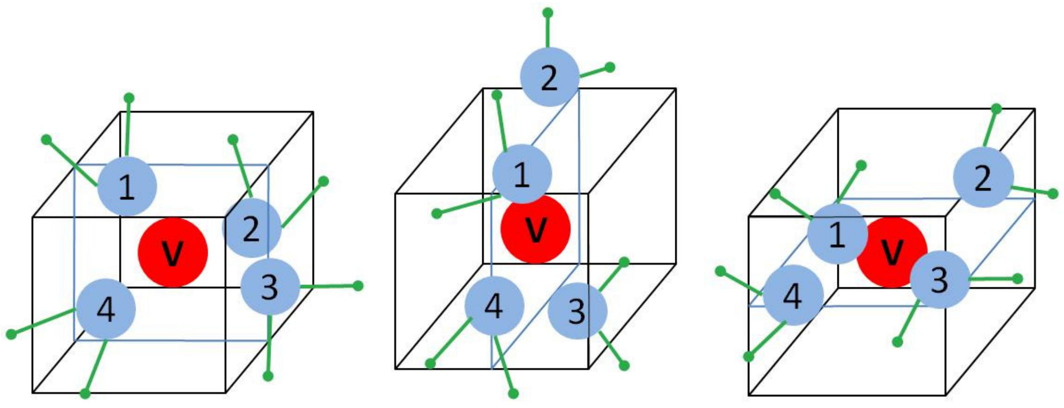
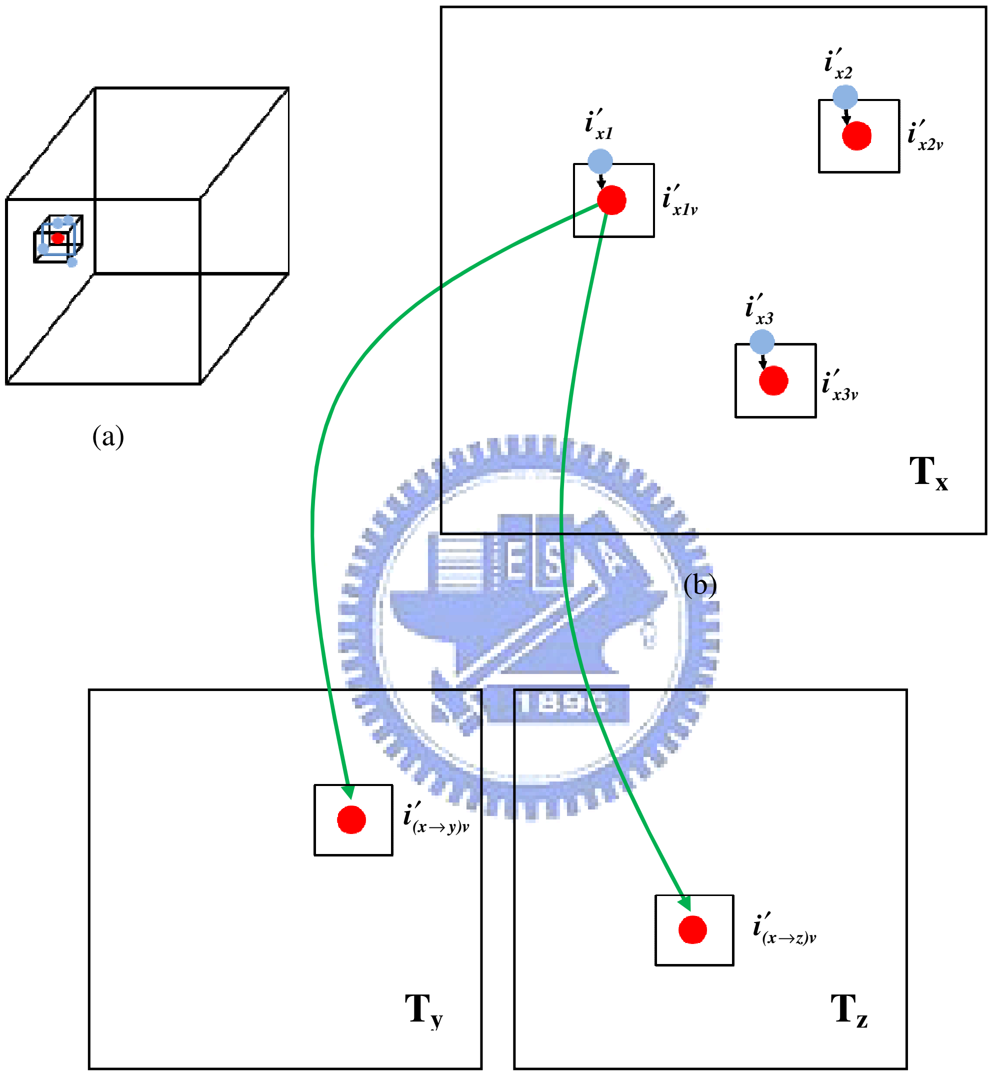


Figure 4.4 Three warped sub-neighbors for warped neighbors of voxel v

We utilize the 4 warped voxels for each direction nearby voxel v. Taking direction x for example, the warped neighbor voxel i'_x ($i'_x = 1 \sim 4$), we can get the most similar 3 voxels $(i'_{x1}, i'_{x2}, i'_{x3})$ in exemplar T_x for voxel i'_x from the similarity set. Then we use the warped relationship with the anisometric fields A between voxel i'_x and voxel v to infer the candidate voxels $(i'_{x1v}, i'_{x2v}, i'_{x3v})$ in exemplar T_x for voxel v, as shown in Fig. 4.5(b). We want to keep the color consistence in three directions, so we use the precomputed 3D-candidate set to infer the other two coordinates $(i'_{(x\to y)v}, i'_{(x\to z)v})$ in exemplar T_y and T_z (Fig. 4.5(c)). In addition, directions y, z are done by the same steps.

Now, for each v, we can get a set of triple candidates $TC'_{1...k}$ which point towards pixels in exemplars T_x, T_y, T_z. With the inverse anisometric field A_l^{-1}, we can compute the warped neighborhood vectors $\tilde{N}_{s_l}(TC'_{1...k})_x$, $\tilde{N}_{s_l}(TC'_{1...k})_y$ and $\tilde{N}_{s_l}(TC'_{1...k})_z$. Finally, we sum up the total difference between $\tilde{N}_{s_l}(TC'_{1...k})_x$,

$\tilde{N}_{s_l}(TC'_{1...k})_y$, $\tilde{N}_{s_l}(TC'_{1...k})_z$ and $\tilde{N}_{s_l}(v)_x$, $\tilde{N}_{s_l}(v)_y$, $\tilde{N}_{s_l}(v)_z$, and then replace the triple coordinate for voxel v with the best matching triple candidate.



(c)

Figure 4.5 The process of forming warped triple candidates in direction x:

(a) A voxel v is corrected.

(b) The candidates in exemplar T_x

(b) Find the other two candidates in exemplars T_y, T_z

Chapter 5

Implementation and Results

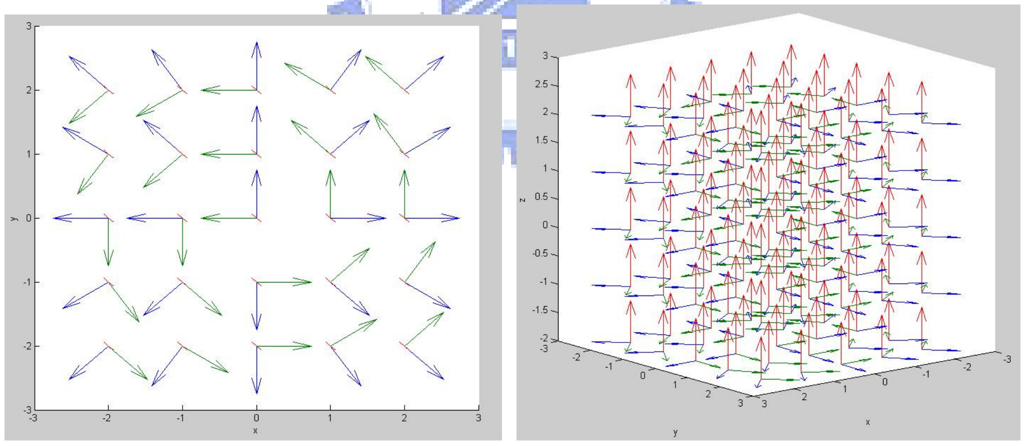
We implement our system on a PC with 3.00GHz and 3.00GHz Core2 Extreme CPU and 8.0GB of system memory. We use MATLAB to implement our method. We always use 128×128 input texture, and we can synthesize to any target size which we want. It needs about 2.5 seconds to construct a transformed exemplar from feature vectors. It takes various times to construct a similarity set based on characteristic of the input texture. However, the time does not exceed 11 minutes. And we spend about 3.5 hours constructing a 3D-candidate set. The transformed exemplar from feature vectors and the similarity set can be reused for synthesis process. It means that once the feature vectors and similarity sets are constructed, we can use them for other syntheses with different target sizes for results and with different vector fields.

For a 128×128×128 result data, it needs about 11~12 hours for synthesis process. For a 256x256x256 result data, it needs about 5 days for synthesis process. We will our results with 128×128 input texture data and 128×128×128 result data in this chapter. The detail computation times for different textures are shown in Table 5.1. In Section we show some vector fields which we use in anisometric synthesis, and we show synthesis results in Section 5.2.

5.1 Vector Fields

We use different vector field controls: circular pattern on XY plane, emissive pattern on XY plane, slant pattern on XY plane, zigzag pattern on XY plane, and slant control on 3D space.

The vector field about circular and emissive control is in Fig. 5.1. In circular control, we use the green arrows as the primary vectors. On the contrary, we use the blue arrows as the primary vectors in emissive control.

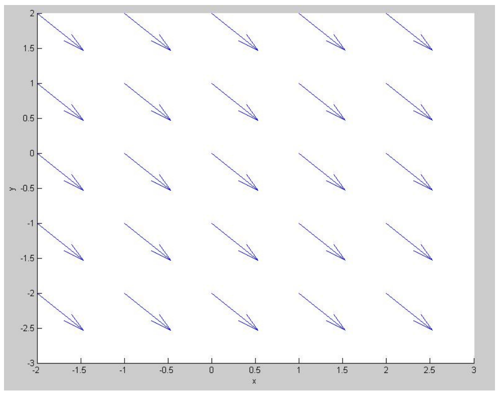


(a) (b)

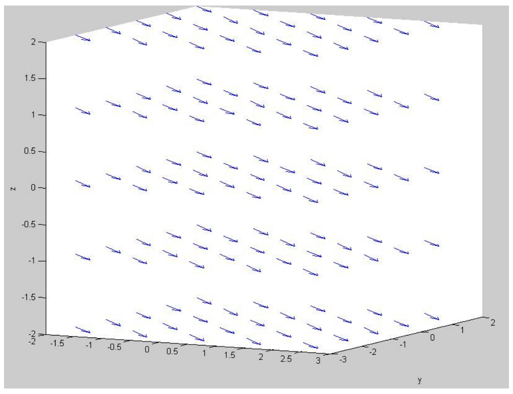
Figure 5.1 5×5×5 3D vector field about circular & emissive control

(a) XY plane (b) three orthogonal axes at every point

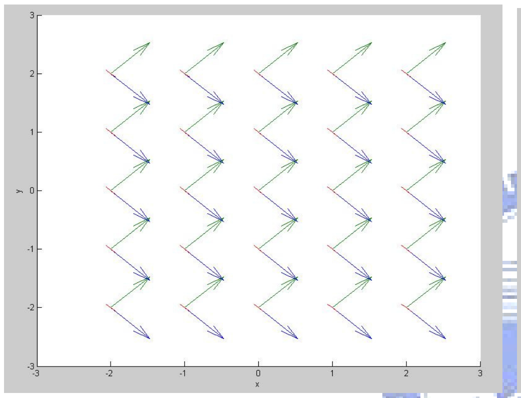
The vector field about slant control is in Fig. 5.2.



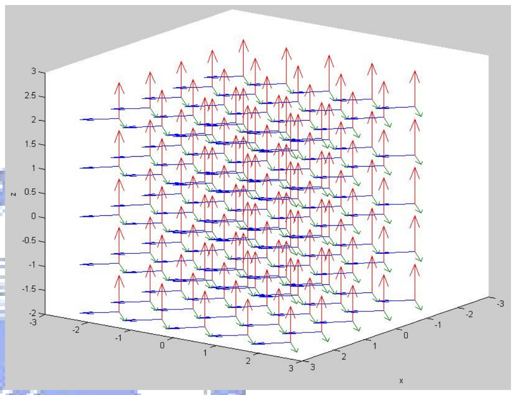
(a)



(b)



(c)



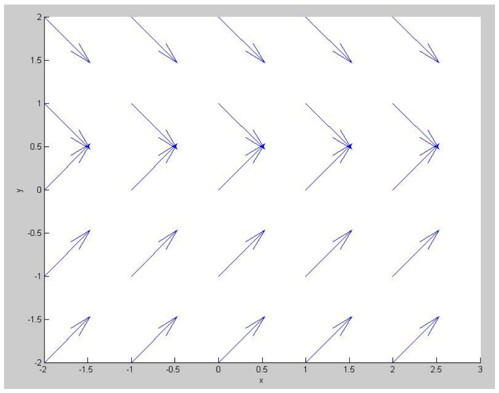
(d)

Figure 5.2 5×5×5 3D vector field about slant control

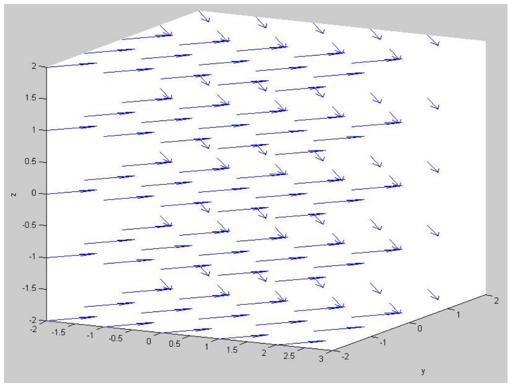
(a) one axes on XY plane (b) one axes at every point

(c) three axes on XY plane (d) three orthogonal axes at every point

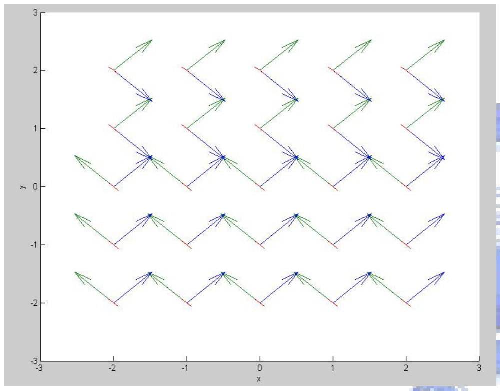
The vector field about zigzag control is in Fig. 5.3, and we make it changed by two different directions for slant.



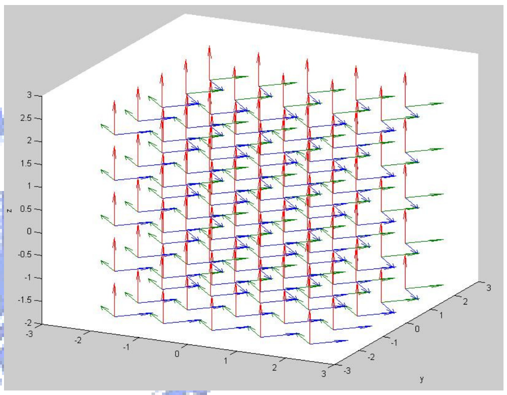
(a)



(b)



(c)



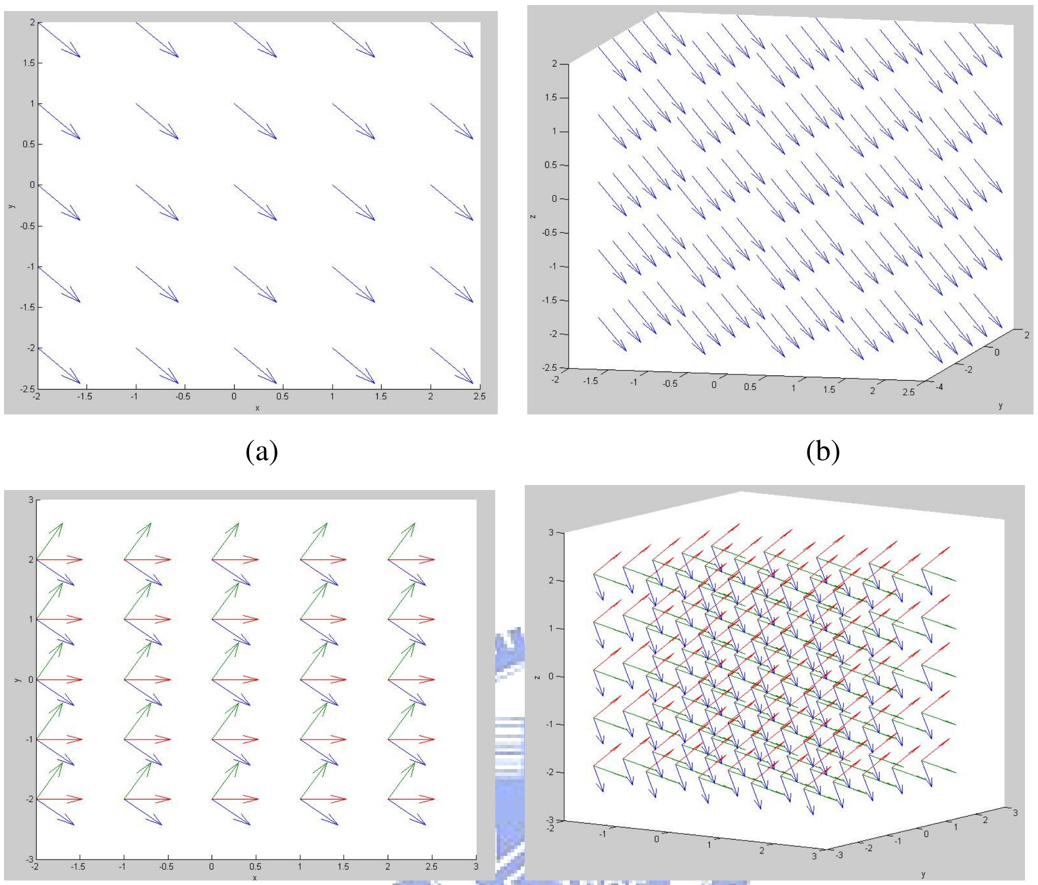
(d)

Figure 5.3 5×5×5 3D vector field about zigzag control

(a) one axes on XY plane (b) one axes at every point

(c) three axes on XY plane (d) three orthogonal axes at every point

The vector field about 3D slant control is in Fig. 5.4.



(a) (b)

(c) (d)

Figure 5.4 5×5×5 3D vector field about 3D slant control

(a) one axes on XY plane (b) one axes at every point

(c) three axes on XY plane (b) three orthogonal axes at every point

5.2 Synthesis Results

The input data in Fig. 5.5(a) (case_1) is a particle-like texture. It contains few kinds of color, and it is very different between particles and background. The particles in case_1 are the same kind. As long as there are few complete particle patterns in the input data, we can synthesize good result, as shown in Fig. 5.5(b)~(e).

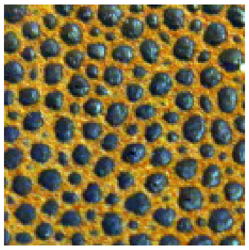
We add the circular vector field control to the synthesis, as shown in Fig. 5.6(a)~(d). Because this texture is not structural enough, the effect by vector field is not obvious. But we still can see the arc-like patterns.

We add the emissive vector field control to the synthesis, as shown in Fig. 5.7(a)~(d). The effect by vector field is also obscure. Only little emissive-like patterns are appeared in the cross section.

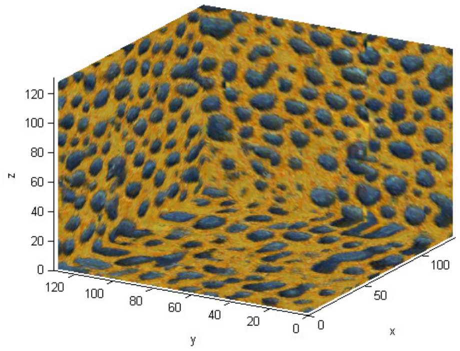
We add the slant vector field control to the synthesis, as shown in Fig. 5.8(a)~(d). On XY plane, the patterns have slanted directional consistency.

We add the zigzag vector field control to the synthesis, as shown in Fig. 5.9(a)~(d). In this case, the effect by zigzag vector field control is obscure, just like no vector field.

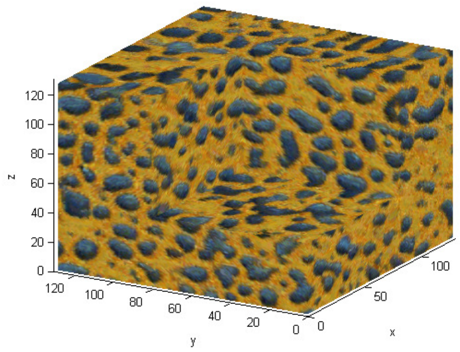
We add the 3D slant vector field control to the synthesis, as shown in Fig. 5.10(a)~(d). On XY plane, the slant patterns are obvious. On the other two planes, the effect is less than the XY plane, but the spots are squelched.



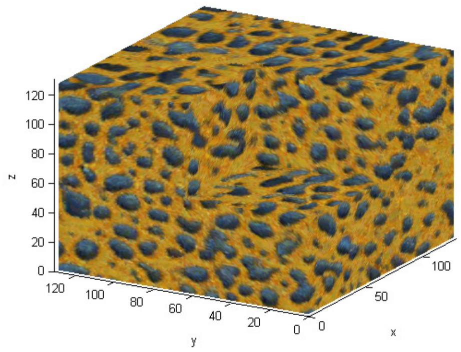
(a)



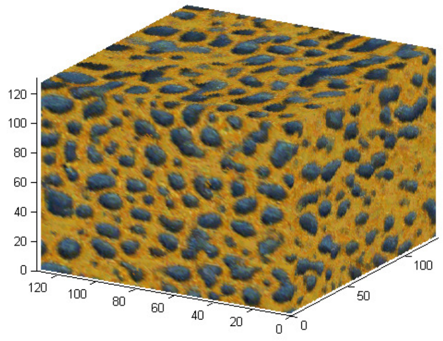
(b)



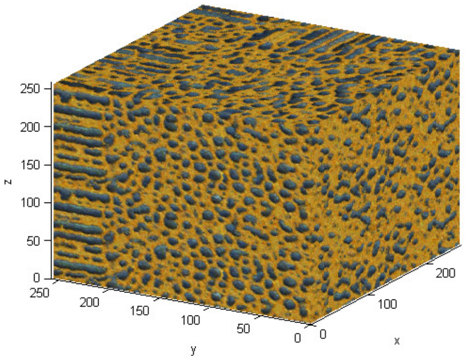
(c)



(d)



(e)



(f)

Figure 5.5 (a) Input and result data for case_1

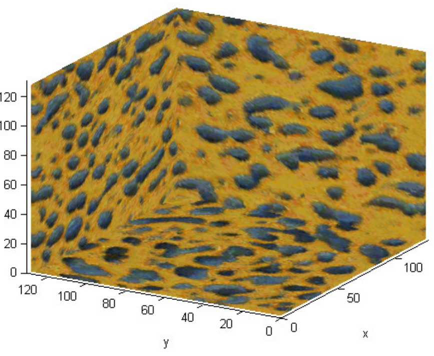
(b) cross section at X=126, Y=126, and Z=126 for result data

(c) cross section at X=80, Y=80, and Z=80 for result data

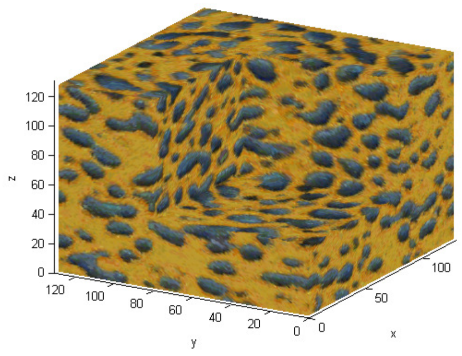
(d) cross section at X=64, Y=64, and Z=64 for result data

(e) 128×128×128 result volume data for case_1

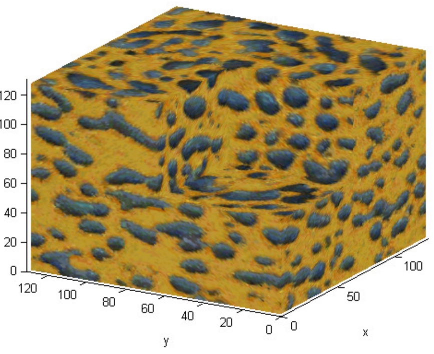
(f) 256×256×256 result volume data for case_1



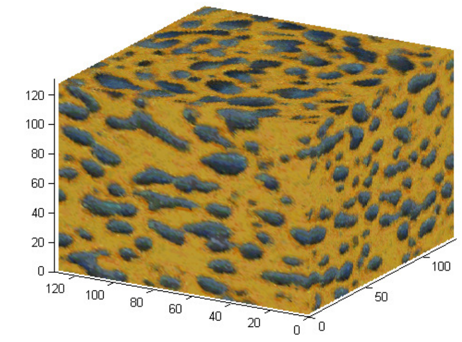
(a)



(b)



(c)



(d)

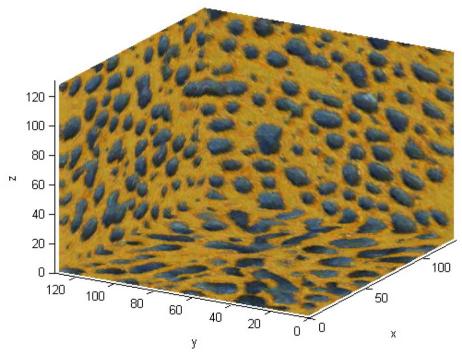
Figure 5.6 Anisometric results with circular control for case_1

(a) cross section at X=126, Y=126, and Z=126 for result data

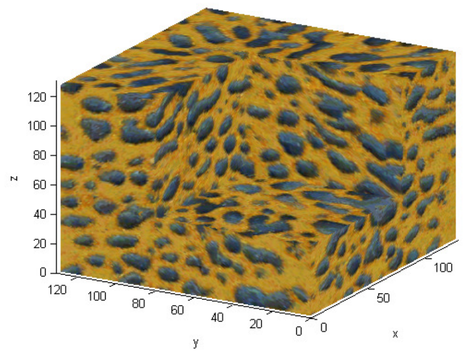
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

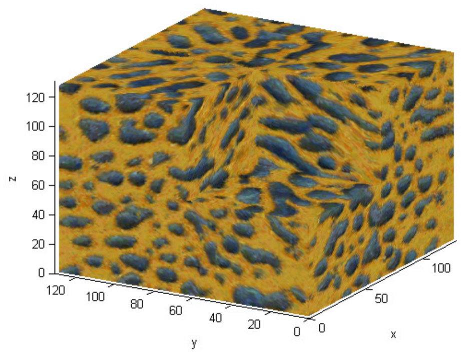
(d) anisometric result with circular control for case_1



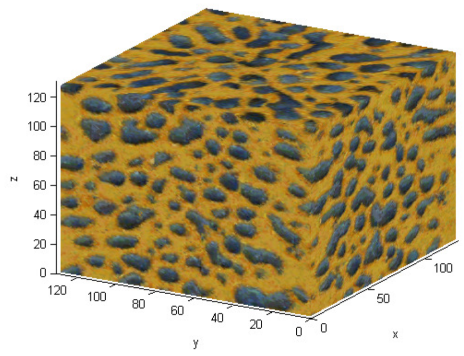
(a)



(b)



(c)



(d)

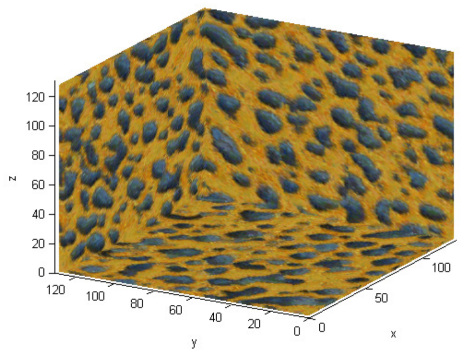
Figure 5.7 Anisometric results with emissive control for case_1

(a) cross section at X=126, Y=126, and Z=126 for result data

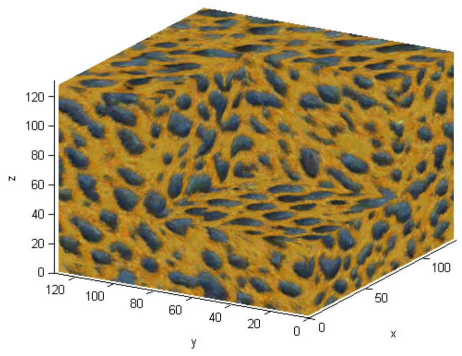
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

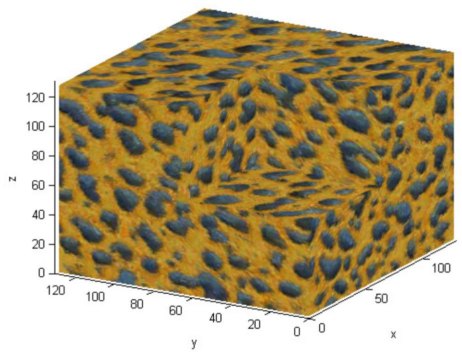
(d) anisometric result with emissive control for case_1



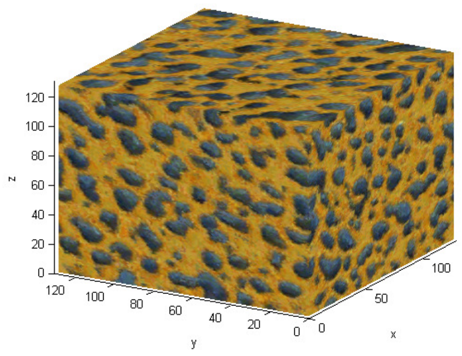
(a)



(b)



(c)



(d)

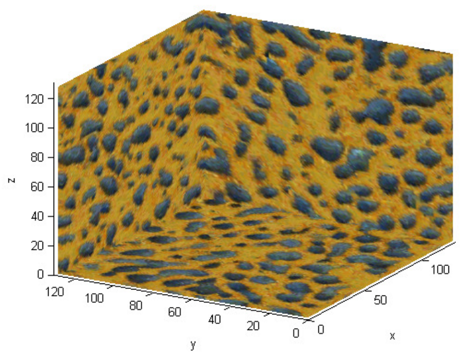
Figure 5.8 Anisometric results with slant control for case_1

(a) cross section at X=126, Y=126, and Z=126 for result data

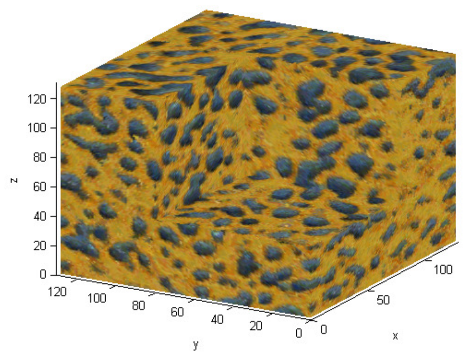
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

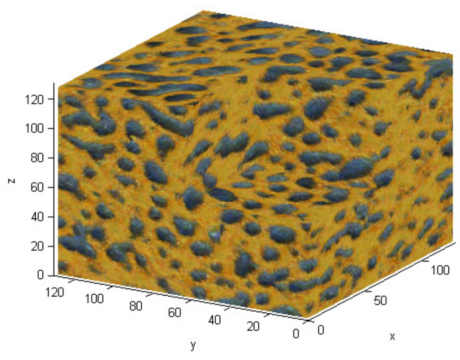
(d) anisometric result with slant control for case_1



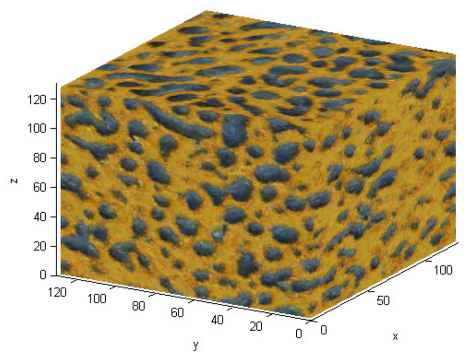
(a)



(b)



(c)



(d)

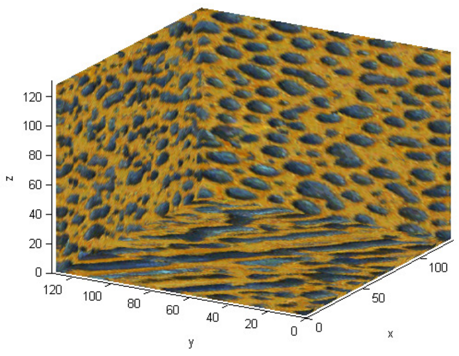
Figure 5.9 Anisometric results with zigzag control for case_1

(a) cross section at X=126, Y=126, and Z=126 for result data

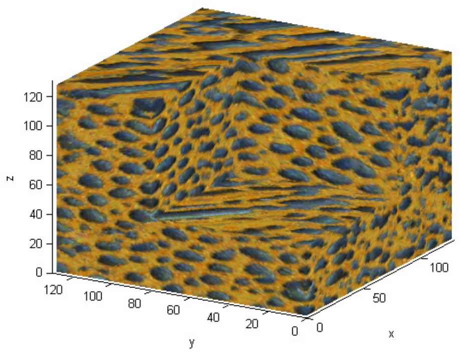
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

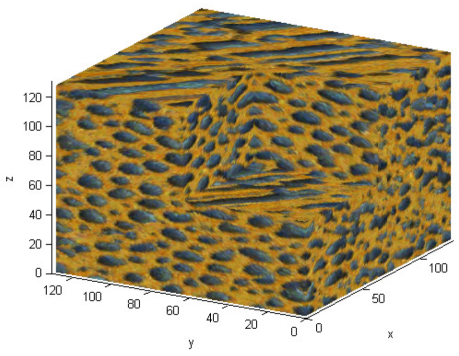
(d) anisometric result with zigzag control for case_1



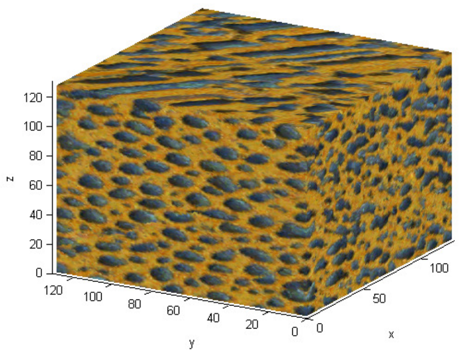
(a)



(b)



(c)



(d)

Figure 5.10 Anisometric results with 3D slant control for case_1

(a) cross section at X=126, Y=126, and Z=126 for result data

(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

(d) anisometric result with 3D slant control for case_1

The input data in Fig. 5.11(a) (case_2) is stochastic and marble-like texture. It only contains two kinds of colors, and it is vivid. It has rich information so that only needing small amount of data to represent the whole texture. It means that we can synthesize larger results with this kind of textures. Fig. 5.11(b)~(e) show the result.

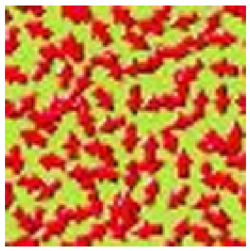
We add the circular vector field control to the synthesis, as shown in Fig. 5.12(a)~(d). As we can see, there are circular patterns on XY plane.

We add the emissive vector field control to the synthesis, as shown in Fig. 5.13(a)~(d). On XY plane, we can see some emissive-patterns.

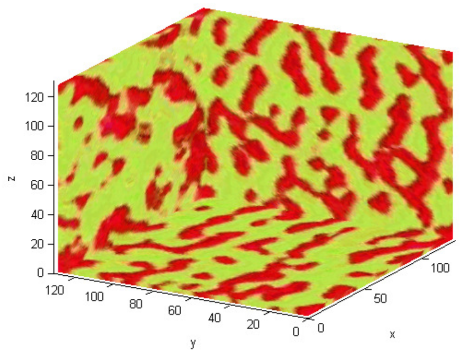
We add the slant vector field control to the synthesis, as shown in Fig. 5.14(a)~(d). On XY plane, the patterns have slanted directional consistency.

We add the zigzag vector field control to the synthesis, as shown in Fig. 5.15(a)~(d). In this case, the effect by the zigzag vector field is obscure. The result looks like random distribution.

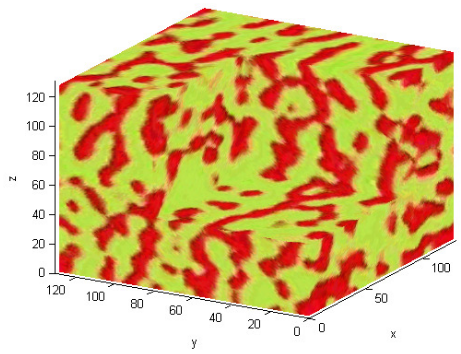
We add the 3D slant vector field control to the synthesis, as shown in Fig. 5.16(a)~(d). On XY plane, there are blocks of slanted patterns. On the other two planes, the patterns are squelched.



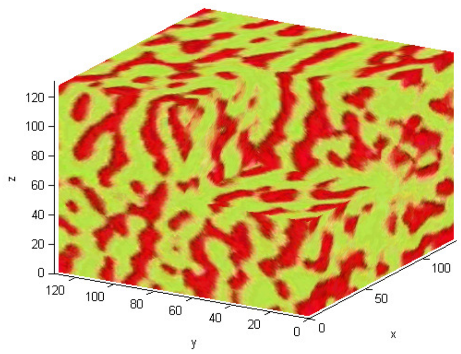
(a)



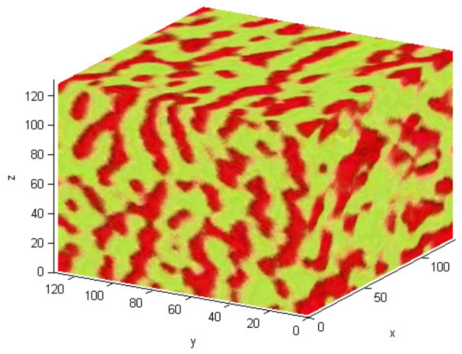
(b)



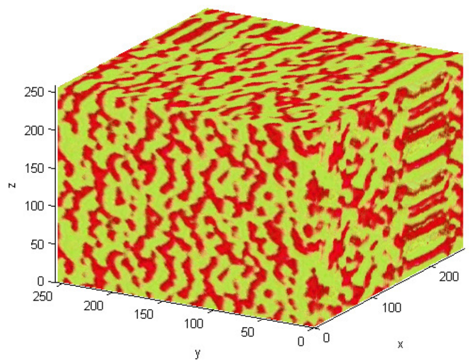
(c)



(d)



(e)



(f)

Figure 5.11 (a) Input and result data for case_2

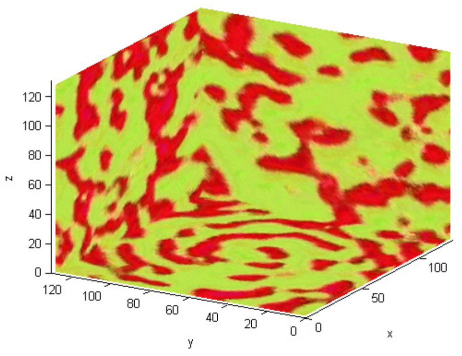
(b) cross section at X=126, Y=126, and Z=126 for result data

(c) cross section at X=80, Y=80, and Z=80 for result data

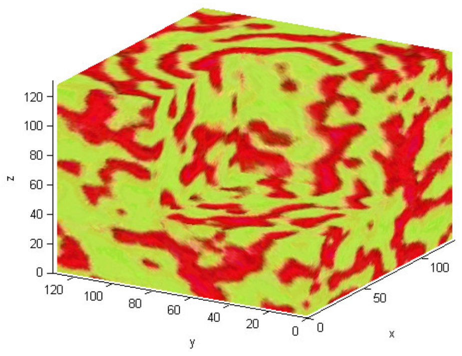
(d) cross section at X=64, Y=64, and Z=64 for result data

(e) 128x128x128 result volume data for case_2

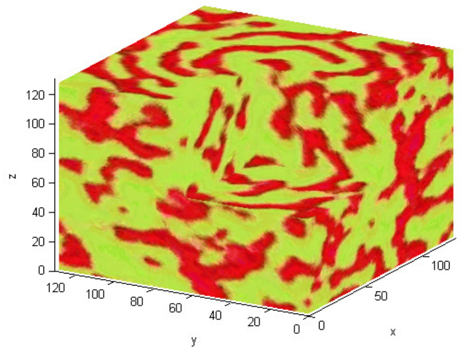
(f) 256x256x256 result volume data for case_2



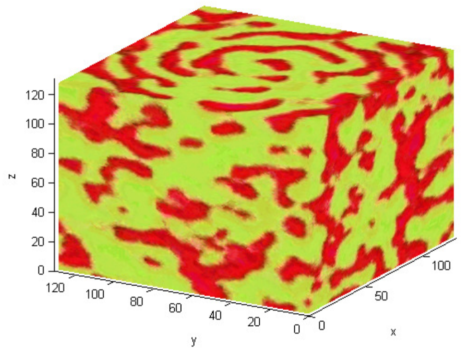
(a)



(b)



(c)



(d)

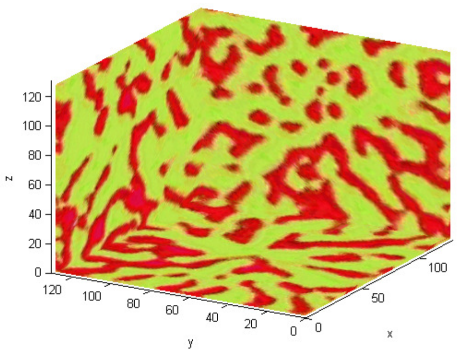
Figure 5.12 Anisometric results with circular control for case_2

(a) cross section at X=126, Y=126, and Z=126 for result data

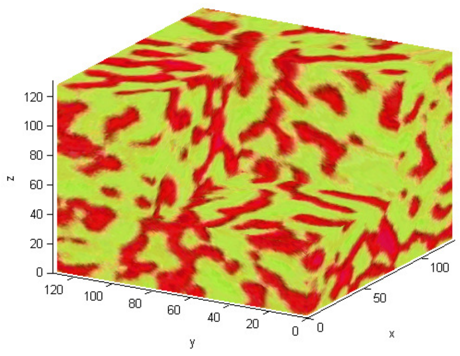
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

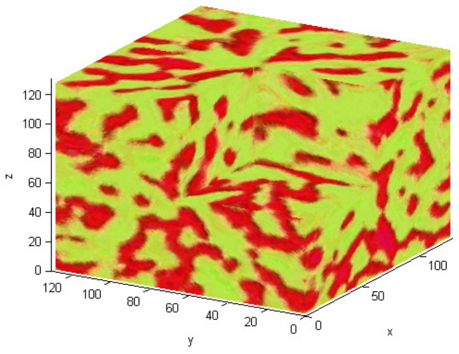
(d) anisometric result with circular control for case_2



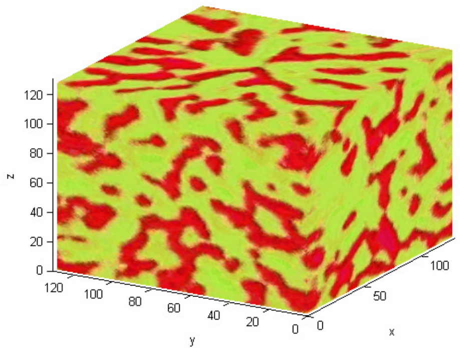
(a)



(b)



(c)



(d)

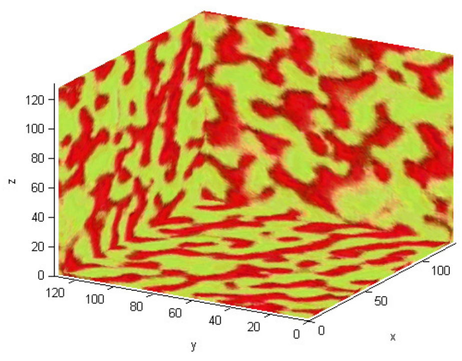
Figure 5.13 Anisometric results with emissive control for case_2

(a) cross section at X=126, Y=126, and Z=126 for result data

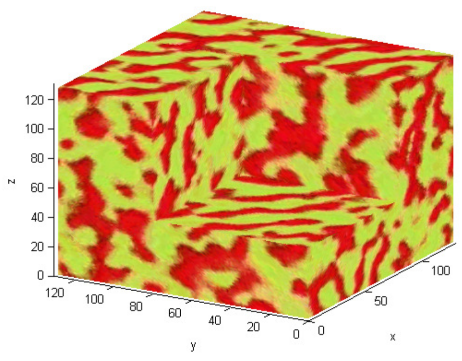
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

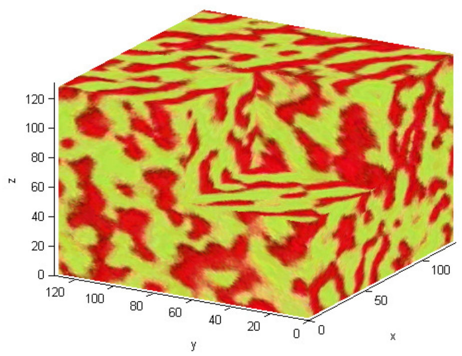
(d) anisometric result with emissive control for case_2



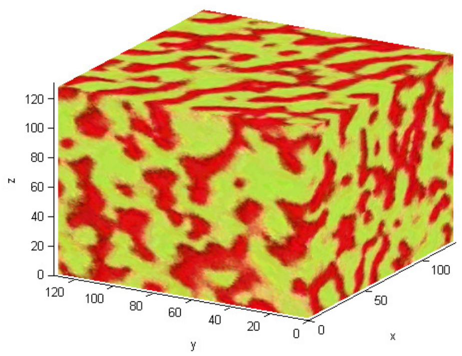
(a)



(b)



(c)



(d)

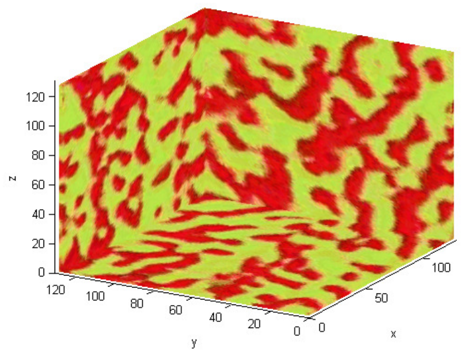
Figure 5.14 Anisometric results with slant control for case_2

(a) cross section at X=126, Y=126, and Z=126 for result data

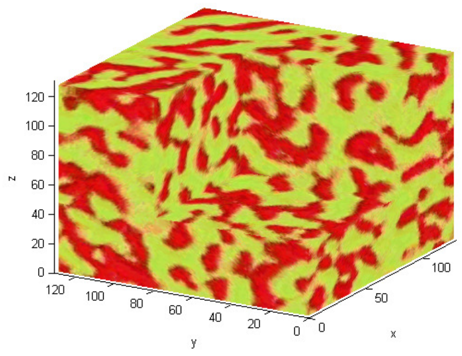
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

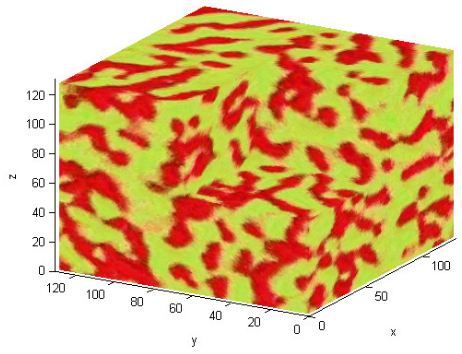
(d) anisometric result with slant control for case_2



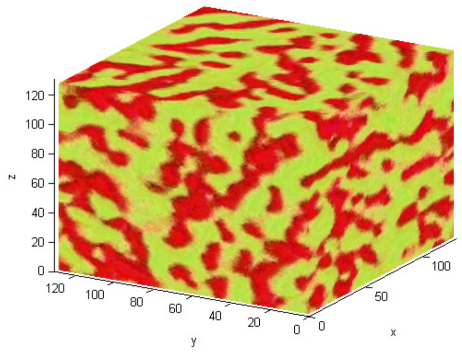
(a)



(b)



(c)



(d)

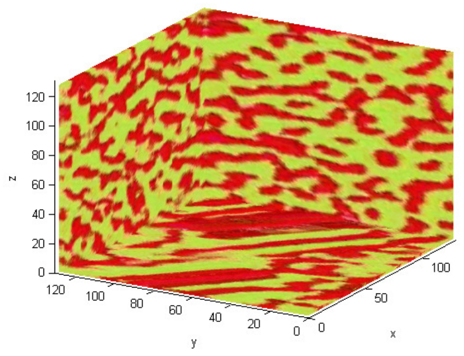
Figure 5.15 Anisometric results with zigzag control for case_2

(a) cross section at X=126, Y=126, and Z=126 for result data

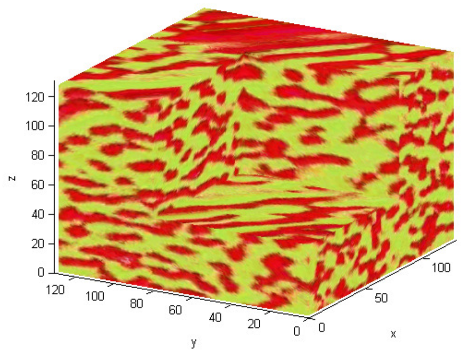
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

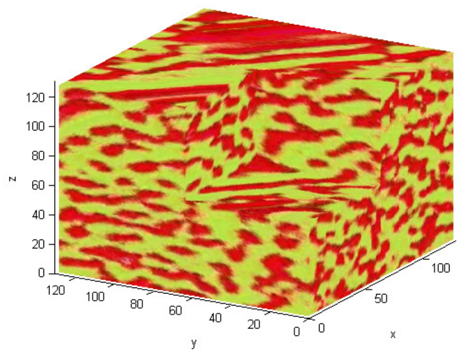
(d) anisometric result with zigzag control for case_2



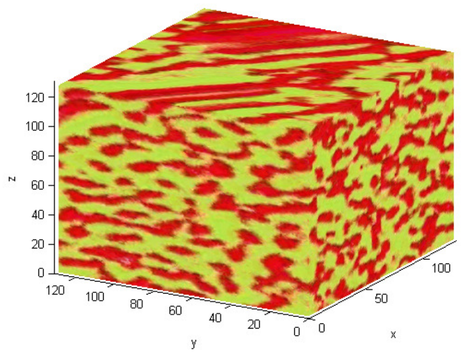
(a)



(b)



(c)



(d)

Figure 5.16 Anisometric results with 3D slant control for case_2

(a) cross section at X=126, Y=126, and Z=126 for result data

(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

(d) anisometric result with 3D slant control for case_2

The input data in Fig. 5.17(a) (case_3) is a kind of structural textures. The patterns in the input data are small and compact, so the texture is information-rich. Only small size for input data could provide enough patterns for synthesis. It can be synthesized with a few input data and get good results. The result is shown in Fig. 5.17(b)~(e).

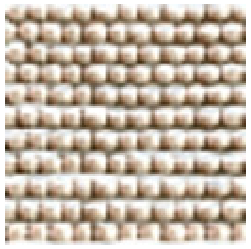
We add the circular vector field control to the synthesis, as shown in Fig. 5.18(a)~(d). As we can see, the result is good. The cross sections also display the character of the texture.

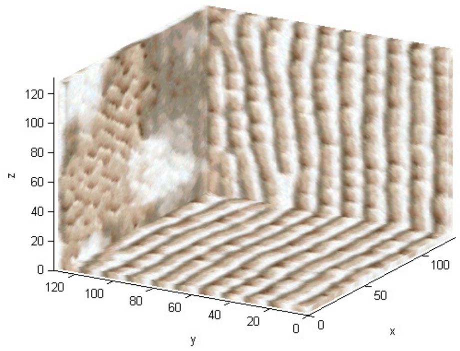
We add the emissive vector field control to the synthesis, as shown in Fig. 5.19(a)~(d). On XY plane, the result is good that it has emissive patterns completely. But on the other two planes, there are some discontinuous lines.

We add the slant vector field control to the synthesis, as shown in Fig. 5.20(a)~(d). We can see that the result is pretty good in all the three planes. It not only exhibits the character of the texture but also has continuous lines.

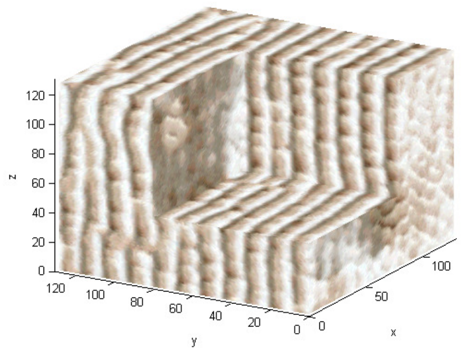
We add the zigzag vector field control to the synthesis, as shown in Fig. 5.21(a)~(d). It has good zigzag patterns on XY plane, but there is a little discontinuity on the other two planes.

We add the 3D slant vector field control to the synthesis, as shown in Fig. 5.22(a)~(d). The result is good. In all the three planes, it has slanted and continuous patterns completely.

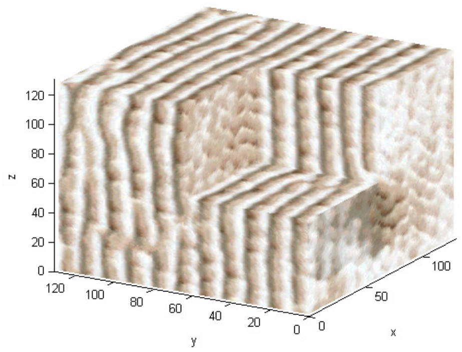

(a)



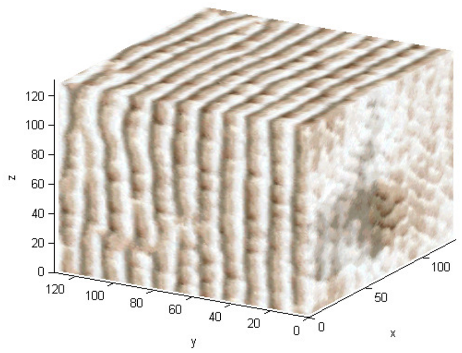
(b)



(c)



(d)



(e)

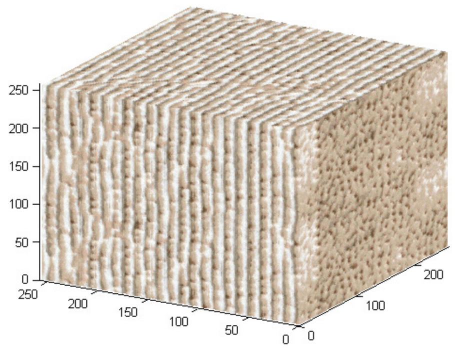

(f)

Figure 5.17 (a) Input and result data for case_3

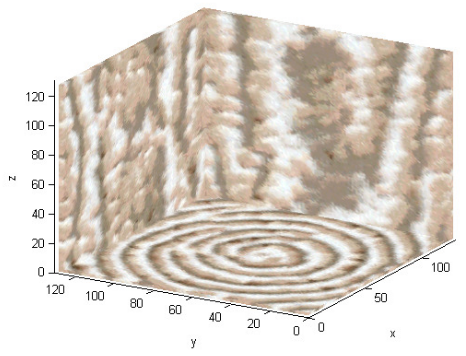
(b) cross section at X=126, Y=126, and Z=126 for result data

(c) cross section at X=80, Y=80, and Z=80 for result data

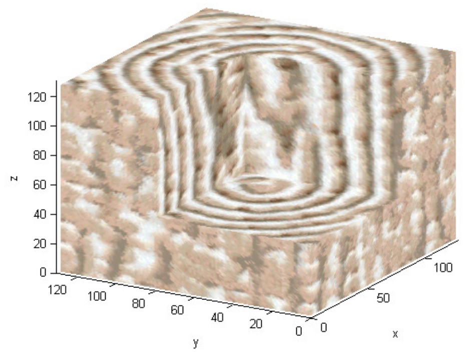
(d) cross section at X=64, Y=64, and Z=64 for result data

(e) 128×128×128 result volume data for case_3

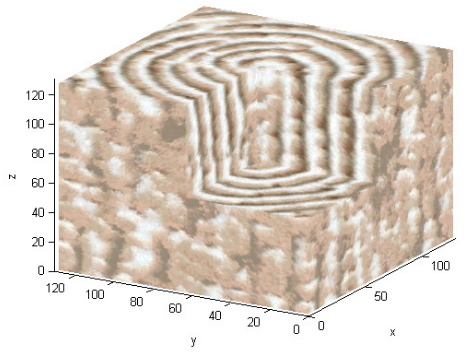
(f) 256×256×256 result volume data for case_3



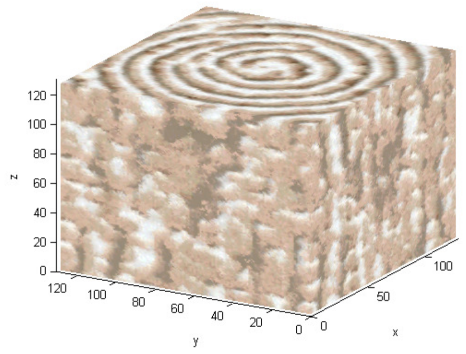
(a)



(b)



(c)



(d)

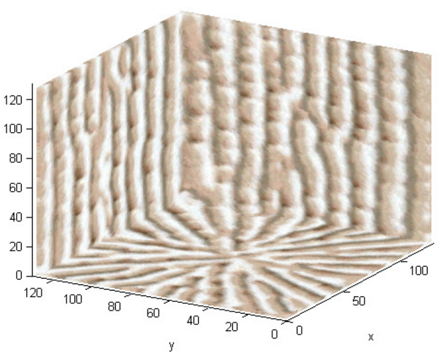
Figure 5.18 Anisometric results with circular control for case_3

(a) cross section at X=126, Y=126, and Z=126 for result data

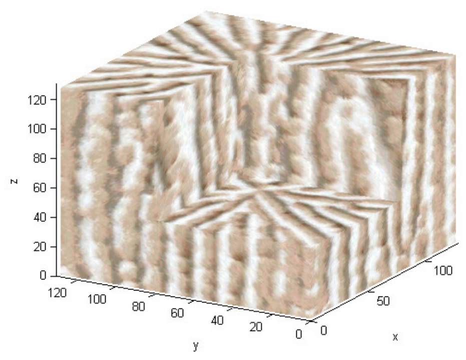
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

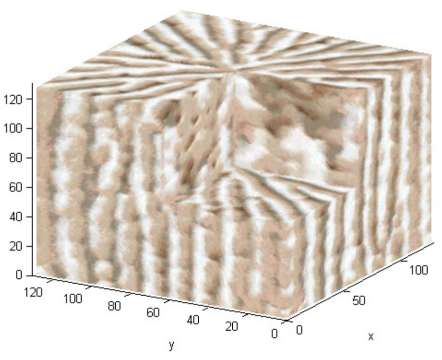
(d) anisometric result with circular control for case_3



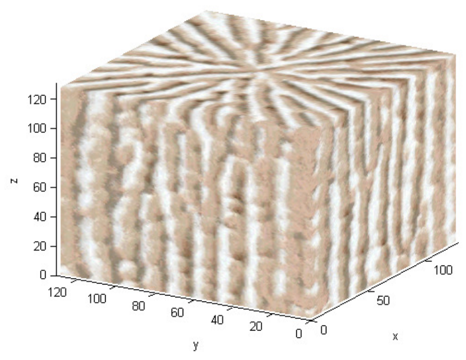
(a)



(b)



(c)



(d)

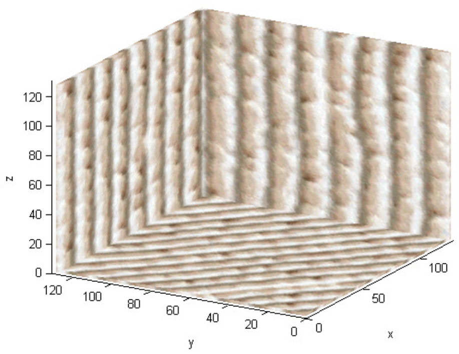
Figure 5.19 Anisometric results with emissive control for case_3

(a) cross section at X=126, Y=126, and Z=126 for result data

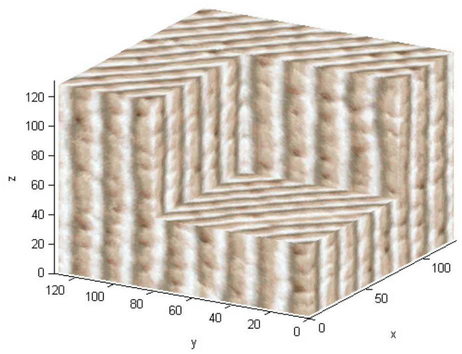
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

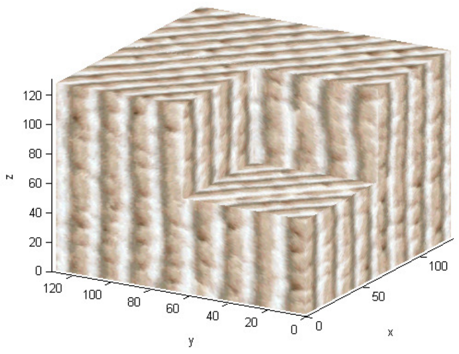
(d) anisometric result with emissive control for case_3



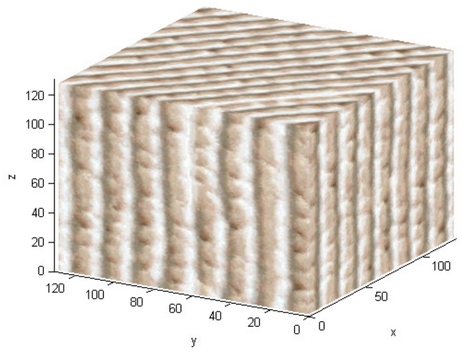
(a)



(b)



(c)



(d)

Figure 5.20 Anisometric results with slant control for case_3

(a) cross section at X=126, Y=126, and Z=126 for result data

(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

(d) anisometric result with slant control for case_3

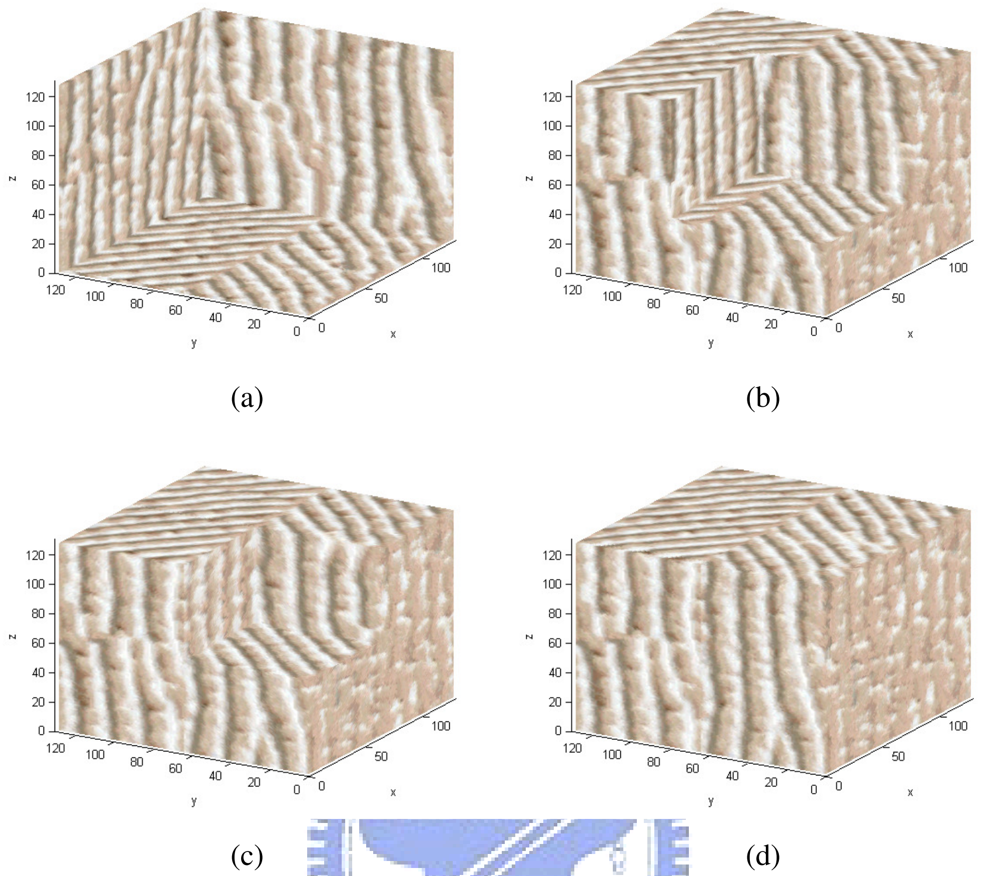


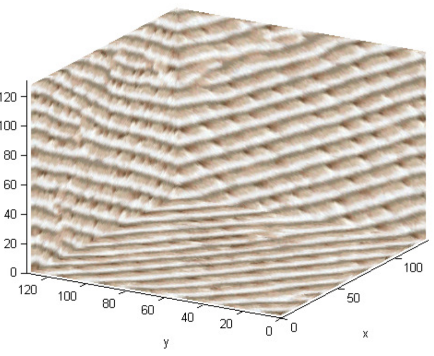
Figure 5.21 Anisometric results with zigzag control for case_3

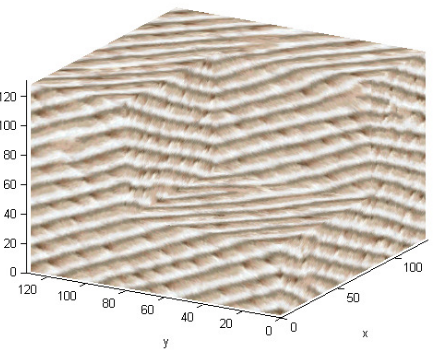
(a) cross section at X=126, Y=126, and Z=126 for result data

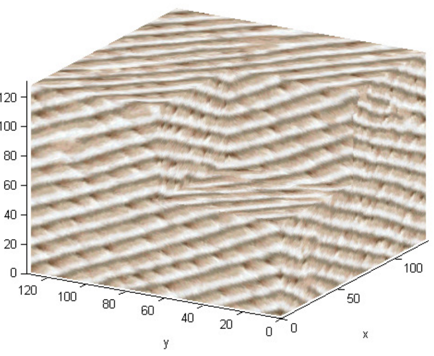
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

(d) anisometric result with zigzag control for case_3


(a)


(b)


(c)

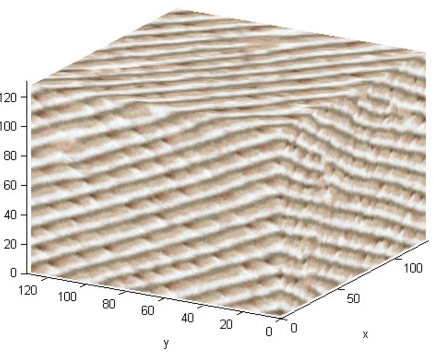

(d)

Figure 5.22 Anisometric results with 3D slant control for case_3

(a) cross section at X=126, Y=126, and Z=126 for result data

(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

(d) anisometric result with 3D slant control for case_3

The input data in Fig. 5.23(a) (case_4) is between structural and stochastic. It has different length and width patterns, but these patterns have the same slanted directions. Because of the slanted direction, it displays different effects comparing with case_3 adding vector field control. The result is shown in Fig. 5.23(b)~(e).

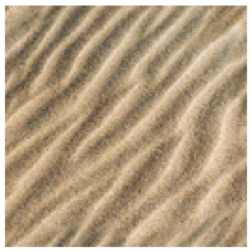
We add the circular vector field control to the synthesis, as shown in Fig. 5.24(a)~(d). As we can see, the result has vortex-like patterns on XY plane because of the input texture's slanted direction.

We add the emissive vector field control to the synthesis, as shown in Fig. 5.25(a)~(d). It has the similar result to circular vector field because of the same factor.

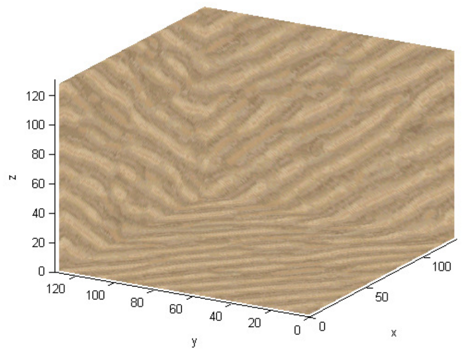
We add the slant vector field control to the synthesis, as shown in Fig. 5.26(a)~(d). We can see that the patterns on XY plane change their directions from slant to horizontal.

We add the zigzag vector field control to the synthesis, as shown in Fig. 5.27(a)~(d). On XY plane, the patterns separate to two parts. Each part has different effect by the zigzag vector field. The left part has horizontal direction, and the right part has slanted direction.

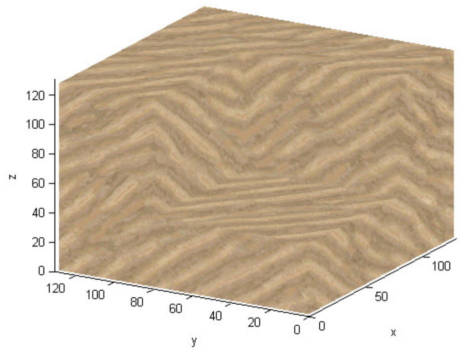
We add the 3D slant vector field control to the synthesis, as shown in Fig. 5.28(a)~(d). The results has horizontal direction on XY and YZ planes. But on XZ plane, it has point-like patterns.



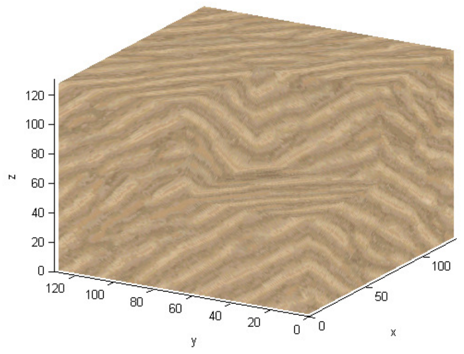
(a)



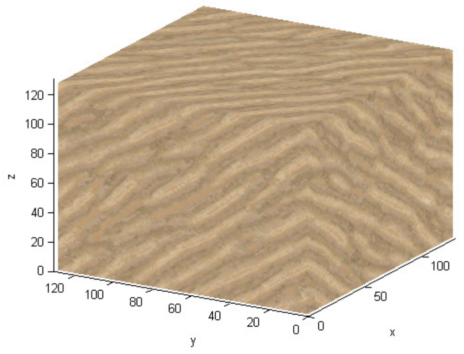
(b)



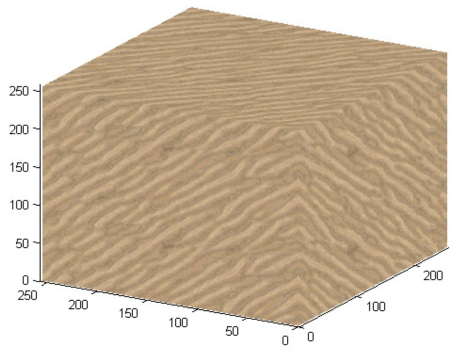
(c)



(d)



(e)



(f)

Figure 5.23 (a) Input and result data for case_4

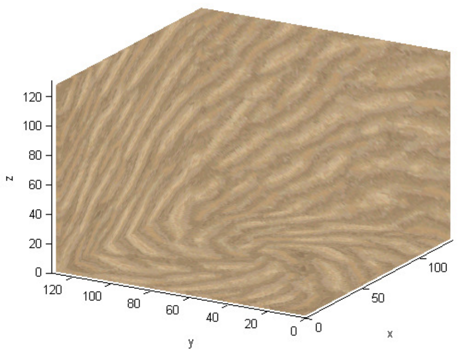
(b) cross section at X=126, Y=126, and Z=126 for result data

(c) cross section at X=80, Y=80, and Z=80 for result data

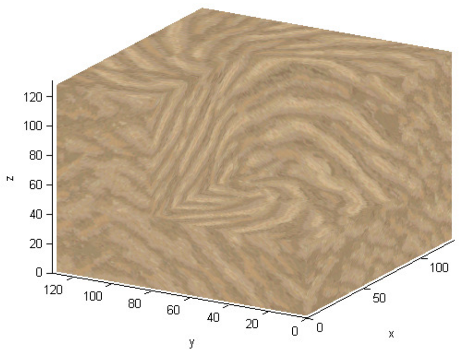
(d) cross section at X=64, Y=64, and Z=64 for result data

(e) 128x128x128 result volume data for case_4

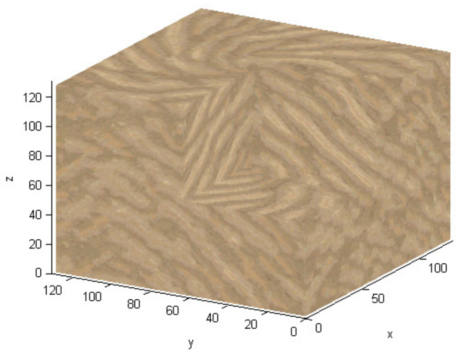
(f) 256x256x256 result volume data for case_4



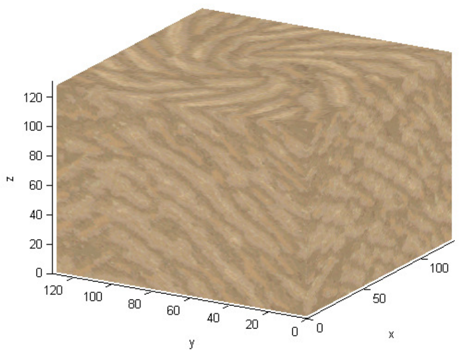
(a)



(b)



(c)



(d)

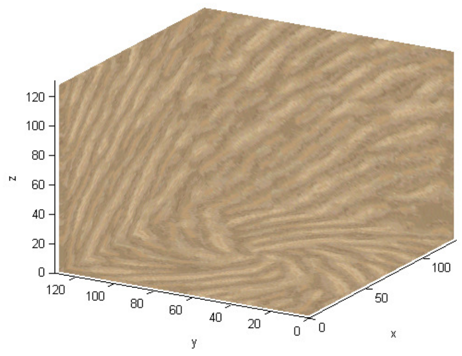
Figure 5.24 Anisometric results with circular control for case_4

(a) cross section at X=126, Y=126, and Z=126 for result data

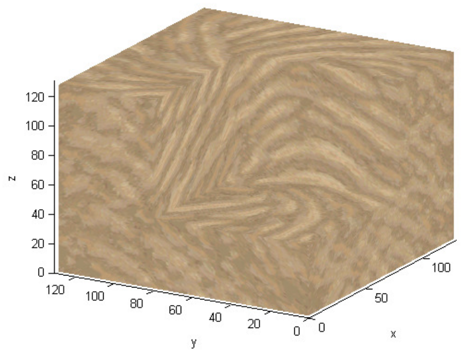
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

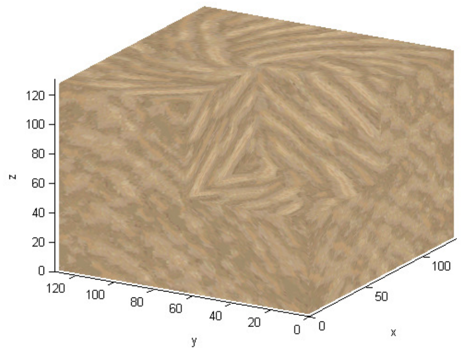
(d) anisometric result with circular control for case_4



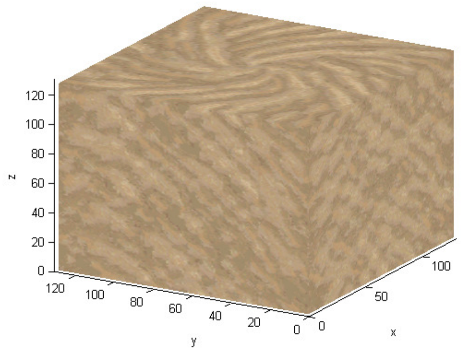
(a)



(b)



(c)



(d)

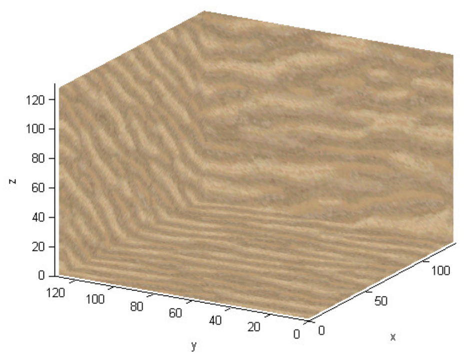
Figure 5.25 Anisometric results with emissive control for case_4

(a) cross section at X=126, Y=126, and Z=126 for result data

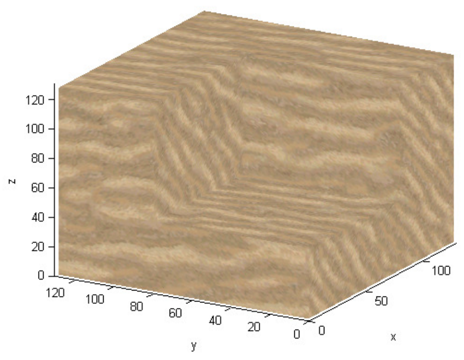
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

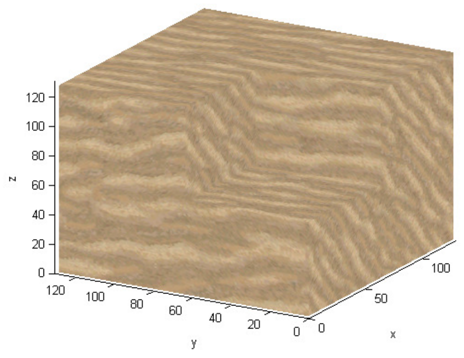
(d) anisometric result with emissive control for case_4



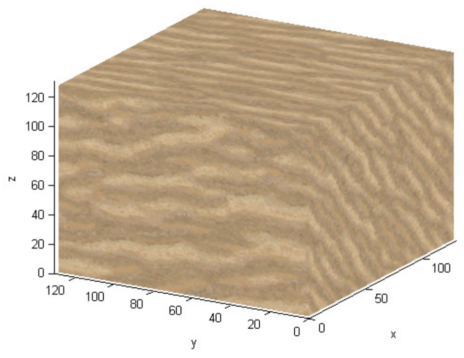
(a)



(b)



(c)



(d)

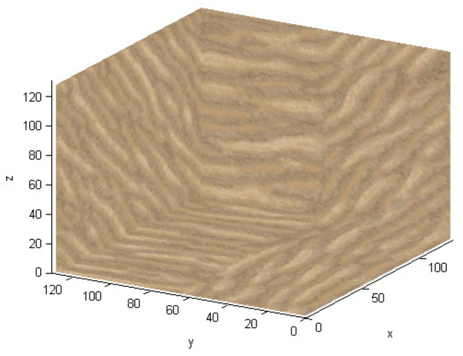
Figure 5.26 Anisometric results with slant control for case_4

(a) cross section at X=126, Y=126, and Z=126 for result data

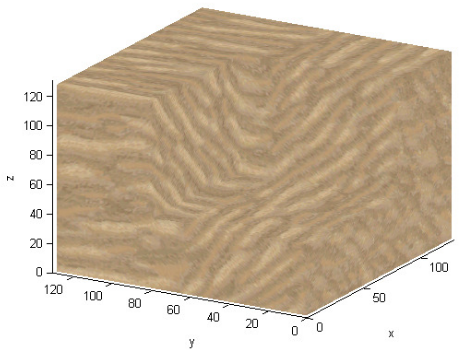
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

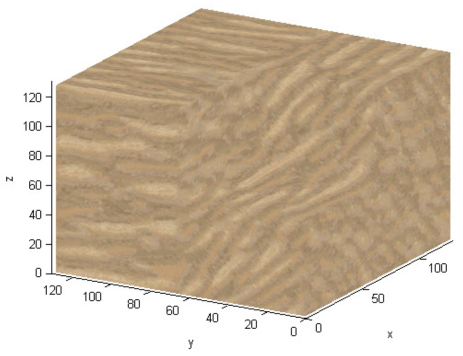
(d) anisometric result with slant control for case_4



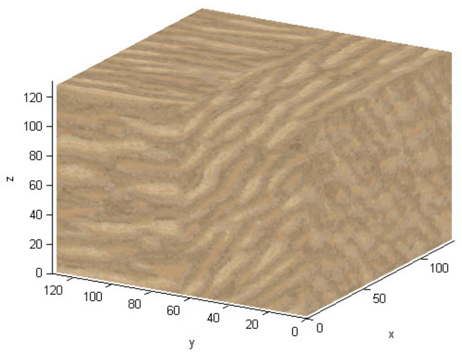
(a)



(b)



(c)



(d)

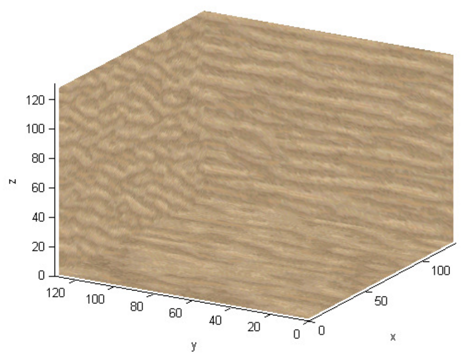
Figure 5.27 Anisometric results with zigzag control for case_4

(a) cross section at X=126, Y=126, and Z=126 for result data

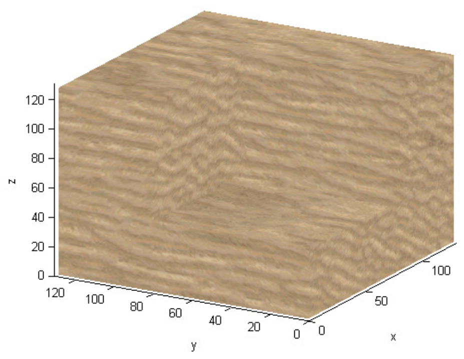
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

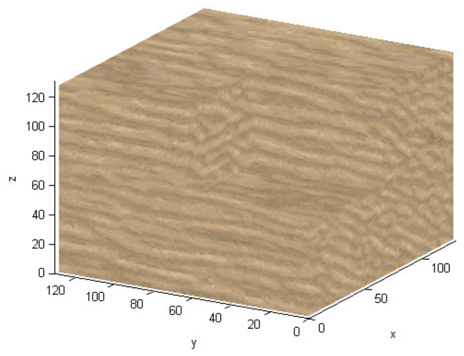
(d) anisometric result with zigzag control for case_4



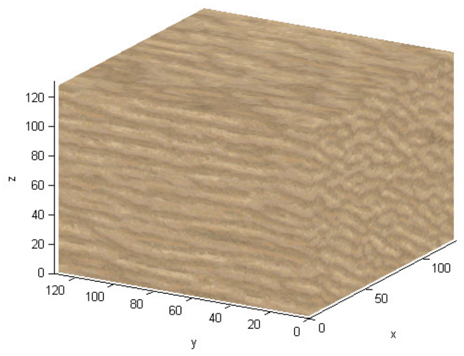
(a)



(b)



(c)



(d)

Figure 5.28 Anisometric results with 3D slant control for case_4

(a) cross section at X=126, Y=126, and Z=126 for result data

(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

(d) anisometric result with 3D slant control for case_4

The input data in Fig. 5.29(a) (case_5) is a structural texture. It has simple wooden patterns. Because of its sparse patterns, it has different effects for different vector fields, and we discuss later. The result is shown in Fig. 5.29(b)~(e).

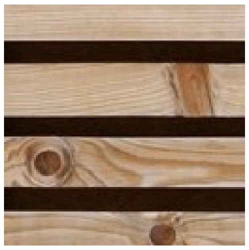
We add the circular vector field control to the synthesis, as shown in Fig. 5.30(a)~(d). The result is bad, just little arc-like patterns on XY plane. The other places in the volume are blurring, and the color of the result is unlike the input texture's color.

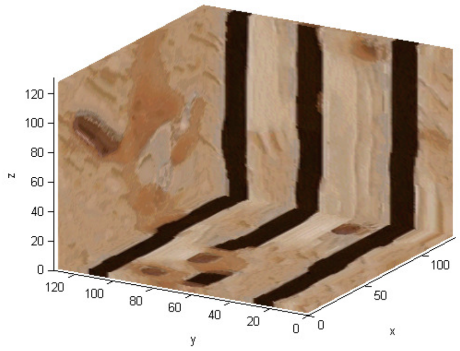
We add the emissive vector field control to the synthesis, as shown in Fig. 5.31(a)~(d). As we can see, there are emissive patterns on XY plane. The color of this result is also odd.

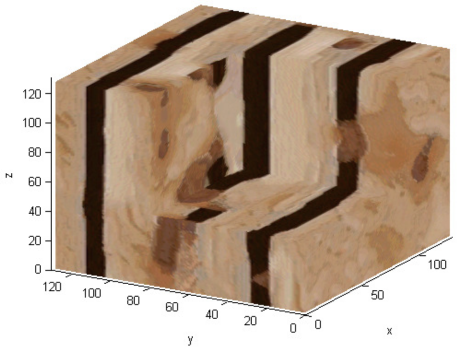
We add the slant vector field control to the synthesis, as shown in Fig. 5.32(a)~(d). The result is pretty good that it has slanted patterns on XY plane completely. Moreover, the characteristic of the wood is displayed well in the result.

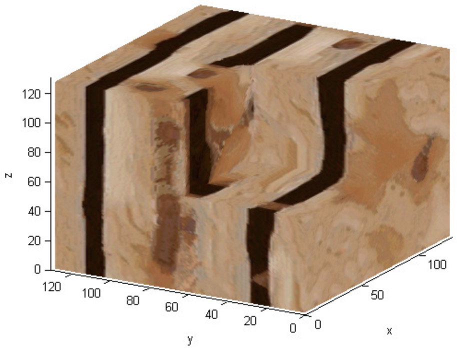
We add the zigzag vector field control to the synthesis, as shown in Fig. 5.33(a)~(d). On XY plane, we can see the zigzag patterns. The result is sparse because the input texture is also sparse.

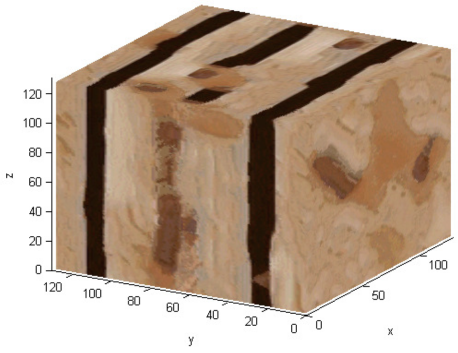
We add the 3D slant vector field control to the synthesis, as shown in Fig. 5.34(a)~(d). The result is not bad. It has slanted patterns on all the three planes.


(a)


(b)


(c)


(d)


(e)

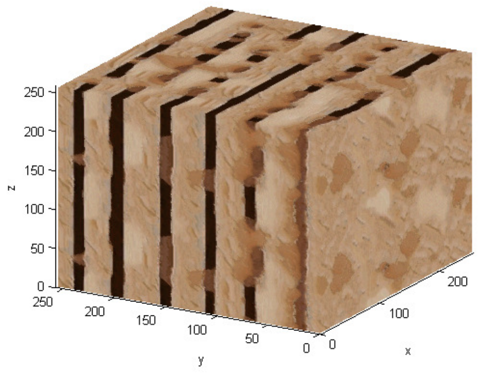

(f)

Figure 5.29 (a) Input and result data for case_5

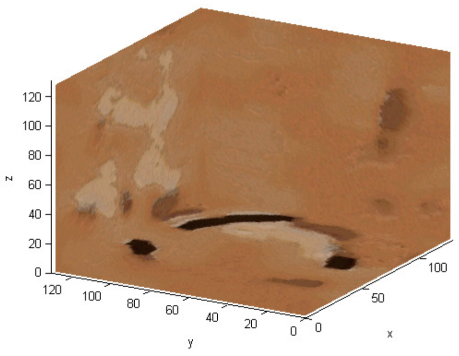
(b) cross section at X=126, Y=126, and Z=126 for result data

(c) cross section at X=80, Y=80, and Z=80 for result data

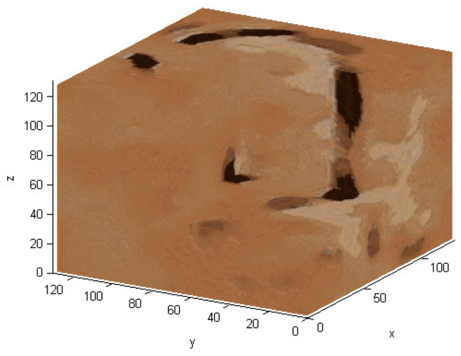
(d) cross section at X=64, Y=64, and Z=64 for result data

(e) 128x128x128 result volume data for case_5

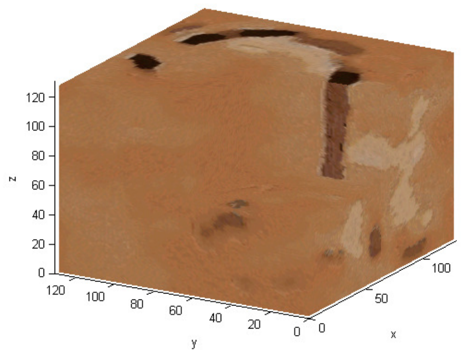
(f) 256x256x256 result volume data for case_5



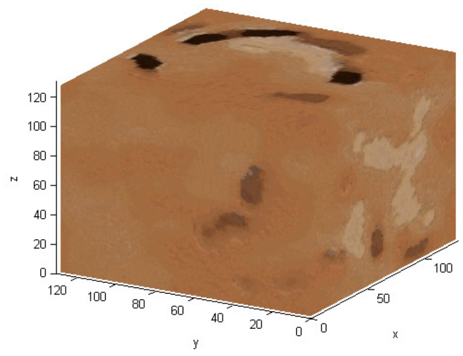
(a)



(b)



(c)



(d)

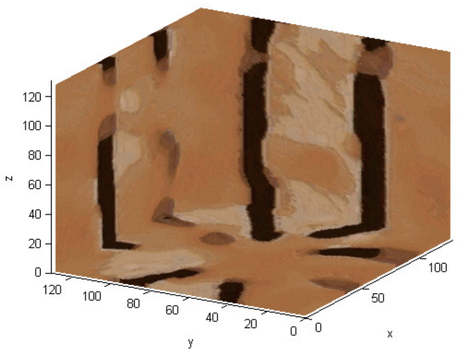
Figure 5.30 Anisometric results with circular control for case_5

(a) cross section at X=126, Y=126, and Z=126 for result data

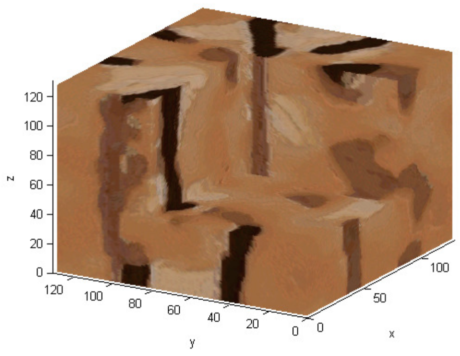
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

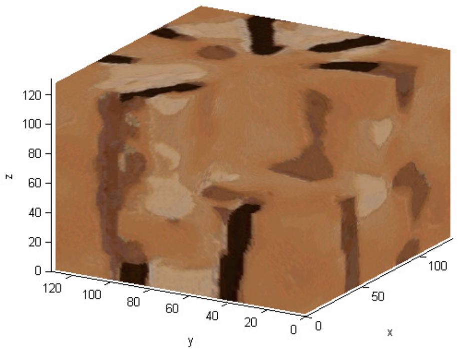
(d) anisometric result with circular control for case_5



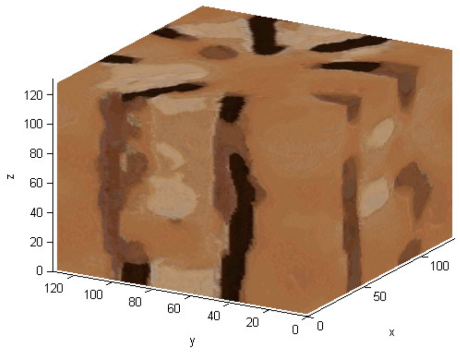
(a)



(b)



(c)



(d)

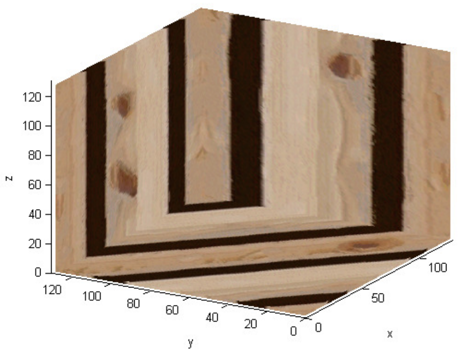
Figure 5.31 Anisometric results with emissive control for case_5

(a) cross section at X=126, Y=126, and Z=126 for result data

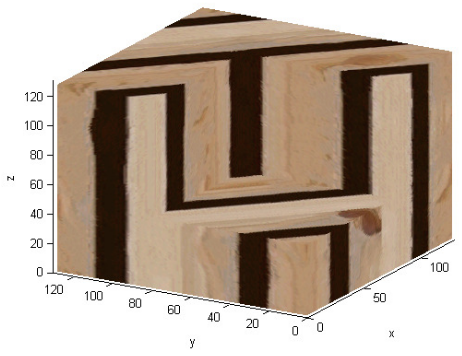
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

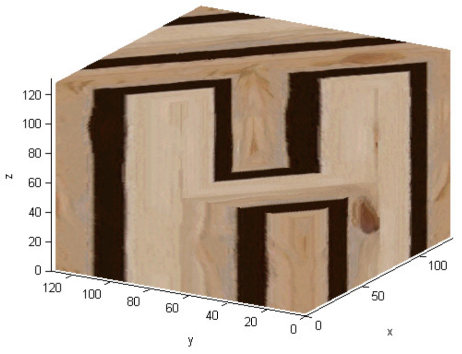
(d) anisometric result with emissive control for case_5



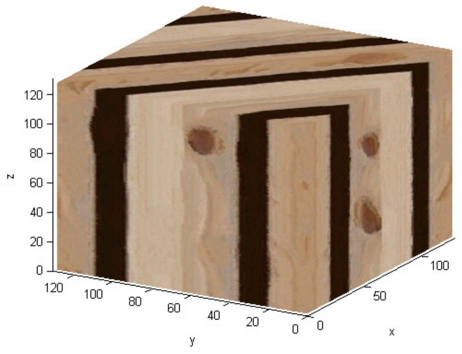
(a)



(b)



(c)



(d)

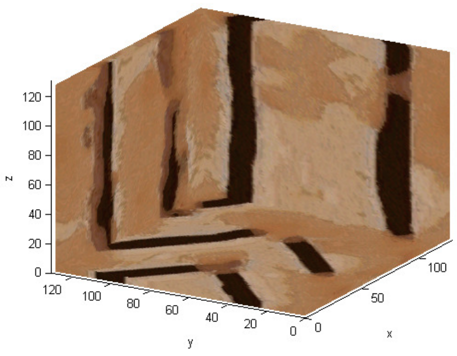
Figure 5.32 Anisometric results with slant control for case_5

(a) cross section at X=126, Y=126, and Z=126 for result data

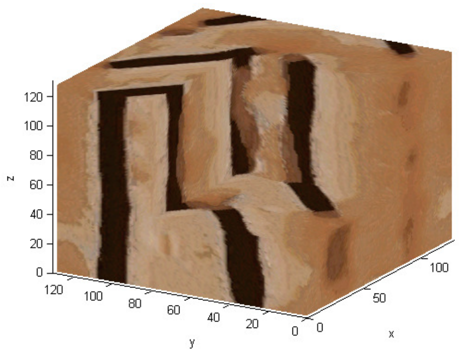
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

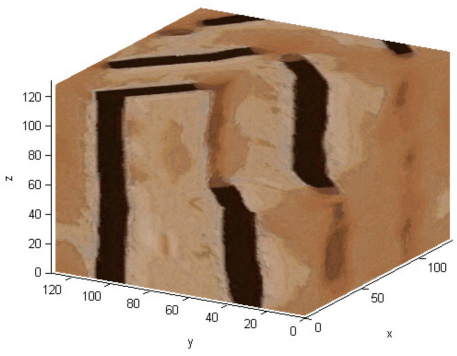
(d) anisometric result with slant control for case_5



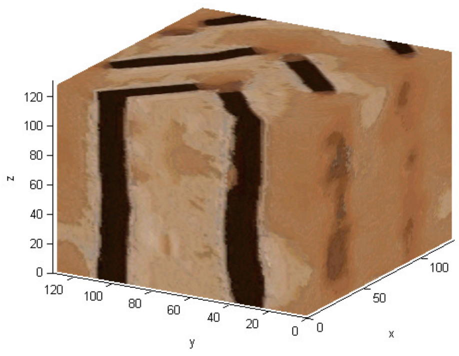
(a)



(b)



(c)



(d)

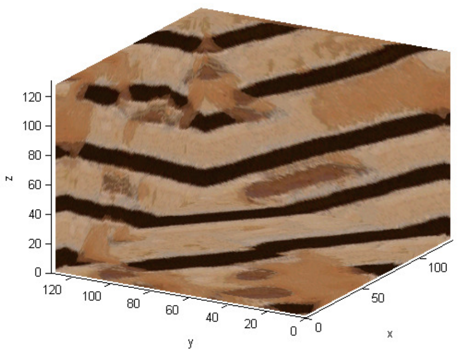
Figure 5.33 Anisometric results with zigzag control for case_5

(a) cross section at X=126, Y=126, and Z=126 for result data

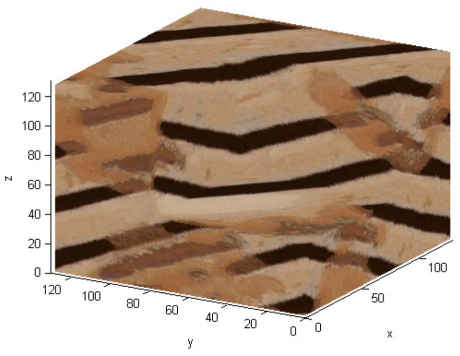
(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

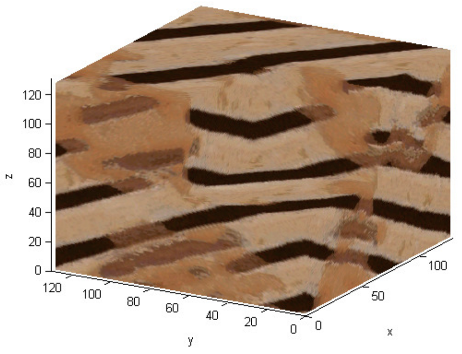
(d) anisometric result with zigzag control for case_5



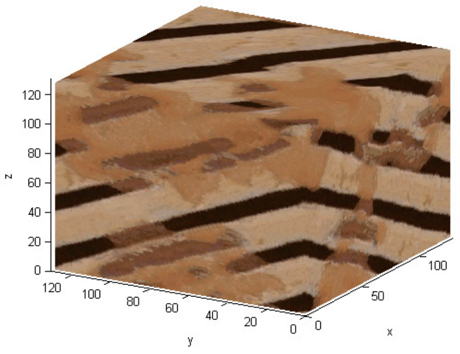
(a)



(b)



(c)



(d)

Figure 5.34 Anisometric results with 3D slant control for case_5

(a) cross section at X=126, Y=126, and Z=126 for result data

(b) cross section at X=80, Y=80, and Z=80 for result data

(c) cross section at X=64, Y=64, and Z=64 for result data

(d) anisometric result with 3D slant control for case_5

Table 5.1 Computation time without vector field control for different textures

	Feature Vector Construction	Similarity Set Construction	3D Candidate Construction	Synthesis Process (128x128x128)	Synthesis Process (256x256x256)
Case_1	2.496 seconds	10 minutes 19seconds	3 hours 32 minutes	11 hours 48 minutes	96 hours 38 minutes
Case_2	2.527 seconds	6 minutes 15 seconds	3 hours 42 seconds	12 hours 11 minutes	97 hours 14 minutes
Case_3	2.511 seconds	3 minutes 56 seconds	3 hours 26 seconds	11 hours 57 minutes	96 hours 52 minutes
Case_4	2.512 seconds	5 minutes 14 seconds	3 hours 29 minutes	12 hours 19 minutes	97 hours 27 minutes
Case_5	2.511 seconds	4 minutes 39 seconds	3 hours 30 minutes	12 hours 3 minutes	97 hours 2 minutes

By using the information-rich appearance vectors, we just use 12 locations (4 locations for each direction) in neighborhood matching. The method by Dong et al. [4] has to compare the corrected voxel’s direct neighborhoods, so it needs 27 (3x3x3) locations. The same points are following: we also separate the system to two processes which are pre-process and synthesis process. And we compute the same 3D-candidate set in pre-process to accelerate the synthesis process.

Chapter 6

Conclusions and Future Works

We have presented an exemplar-based system for solid texture synthesis with anisometric control from a 2D texture. In the preprocessing, we construct the feature vectors, similarity sets, and 3D-candidate sets from a 2D input texture. We replace traditional RGB values with the feature vectors to construct neighbor vectors for more accurate neighborhood matching. The similarity set which records 3 candidates for each pixel promotes the performance of neighborhood matching in 2D plane. The 3D-candidate set keeps color consistence in three directions and also helps more effective neighborhood matching in 3D space. In the synthesis process, we use the pyramid synthesis method to synthesize textures from coarse level to fine level, from one voxel to $m \times m \times m$ resulting data. For each direction, we can only use 4 locations of each voxel for neighborhood matching in synthesis process. In the anisometric synthesis process, we design vector fields and make the result textures changed by the vector fields.

In the future, we may control the anisometric textures with flow fields that make the results changed with time. Kwatra et. al. [9] presented a method for 3D surface texture synthesis with flow field. We may apply their method to synthesize anisometric textures changing with time in the 3D space. Besides, we may reduce the cost time: trying another algorithm to make the system more effective.

Reference

[1] Ashikhmin, M., “Synthesizing Natural Textures”, *ACM SIGGRAPH Symposium on Interactive 3D graphics,* pp. 217-226, 2001.

[2] Chiou, J. W., and Yang, C. K., “Automatic 3D Solid Texture Synthesis from a 2D Image”, *Master’s Thesis*, Department of Information Management National Taiwan University of Science and Technology, 2007

[3] Dischler, J. M., Ghazanfarpour, D., and Freydier, R., “Anisotropic Solid Texture Synthesis Using Orthogonal 2D Views”, *EUROGRAPHICS 1998*, vol. 17, no. 3, pp. 87-95, 1998

[4] Dong, Y., Lefebvre, S., Tong, X., and Drettakis, G., “Lazy Solid Texture Synthesis”, *Eurographics Symposium on Rendering 2008,* vol. 27, no.4

[5] Ebert, D.S., Musgrave, F.K., Peachey, K.P., Perlin, K. and Worley, S., *Texturing & Modeling: A Procedural Approach*, third ed. Academic Press, 2002.

[6] Heeger, D. J., and Bergen, J. R., “Pyramid-Based Texture Analysis Synthesis”, *ACM SIGGRAPH 1995*, vol. 14, no. 3, pp. 229-238, 1995

[7] Jagnow, D., Dorsey, J., and Rushmeier, H., “Stereological Techniques for Solid Textures”, *ACM SIGGRAPH 2004*, vol. 23, no. 3, pp. 329-335, 2004

[8] Kraevoy, V., Sheffer, A., and Gotsman, C., “Matchmaker: Constructing Constrained Texture Maps”, *ACM SIGGRAPH 2003*, vol. 22, no. 3, pp. 326-333, 2003

[9] Kwatra, V., Essa, I., Bobick, A., and Kwatra, N., “Texture Optimization for

Exampled-based Synthesis", *ACM SIGGRAPH 2005*, vol. 24, no. 3, 2005

[10] Kwatra, V., Adalsteinsson, D., Kim, T., Kwatra, N., Carlson, M., AND Lin, M., "Texturing Fluids", *IEEE Transactions on Visualization and Computer Graphics 2007*, vol.13, no. 5, pp.939 – 952, 2007

[11] Kopf, J., Fu, C. W., Cohen-Or, D., Deussen, O., Lischinski, D., and Wong, T. T., "Solid Texture Synthesis from 2D Exemplars", *ACM SIGGRAPH 2007*, vol. 26, no. 3, 2007

[12] Lefebvre, S., and Hoppe, H., "Parallel Controllable Texture Synthesis", *ACM SIGGRAPH 2005* , vol. 24, no. 3, pp. 777-786, 2005

[13] Lefebvre, S., and Hoppe, H., "Appearance-space Texture Synthesis", *ACM SIGGRAPH 2006*, vol. 25, no. 3, 2006

[14] Peachy, D. R., "Solid Texturing of Complex Surfaces", *ACM SIGGPRACH 1985*, vol. 19, no. 3, pp. 279-286, 1985

[15] Perlin, K., "An Image Synthesizer", *ACM SIGGPRACH 1985*, vol. 19, no. 3, pp. 287-296, 1985

[16] Qin, X., and Yang, Y. H., "Aura 3D Textures", *IEEE Transactions on Visualization and Computer Graphics 2007*, vol. 13, no. 2, pp.379-389, 2007

[17] Takayama, K., Okade, M., Ijirl, T., and Igarashi, T., "Lapped Solid Textures: Filling a Model with Anisotropic Textures", *ACM SIGGRAPH 2008*, vol. 27, no. 3, 2008

[18] Turk, G., "Texture Synthesis on Surfaces", *ACM SIGGRAPH 2001*, vol. 20, no. 3, pp. 347-354, 2001.

[19] **Taponecco**, F., Alexa, M., "*Vector Field Visualization using Markov Random Field Texture Synthesis*", Proceedings of Eurographics / IEEE TCVG Symposium on Visualization, 195–202, 2003.

[20] Wei, L.Y., and Levoy, M., “Texture Synthesis Over Arbitrary Manifold Surfaces”, *ACM SIGGRAPH 2001*, vol. 20, no. 3, pp. 355-360, 2001

[21] Wu, J. W., Mei, C. H., and Shi, J. Y., “Method of Direct Texture Synthesis on Arbitrary Surfaces”, *Journal of Computer Science and Technology,* Sept. 2004, Vo1.19, No.5, pp.643-649.

[22] Ying, L., Hertzmann, A., Biermann, H., and Zorin, D., “Texture and Shape Synthesis on Surfaces”, *Eurographics Workshop on Rendering 2001*, vol. 12, pp. 301-312, 2001.

[23] Zelinka, S., and Garland, M., “Interactive Texture Synthesis on Surfaces Using Jump Maps”, *Eurographics Workshop on Rendering 2003,* vol. 14, pp. 90-96, 2003

[24] Zhou, K., Wang, X., Tong, Y., Desbrun, M., Guo, B., and Shum, H. Y., “Synthesis of Bidirectional Texture Functions on Arbitrary Surfaces”, *ACM SIGGRAPH 2002*, vol. 21, no. 3, pp. 665-672, 2002.