VOLUMETRIC MODELING OF NATURAL OBJECTS WITH COMPACT AND CONSISTENT REPRESENTATIONS

コンパクトかつ整合性のある表現形式による自然物のボリューメトリックなモデリング

by

Kenshi Takayama
高山 健志

A Doctor Thesis
博士論文

Submitted to
the Graduate School of the University of Tokyo
on December 15, 2011
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Information Science
and Technology in Computer Science

Thesis Supervisor: Takeo Igarashi　五十嵐 健夫
Professor of Computer Science

ABSTRACT

Modeling of 3D objects' internal information, or volumetric modeling, is useful for various computer graphics applications. For example, volumetric information would enable a rich set of natural and intuitive interactions with 3D models such as cutting and peeling. Volumetric information would also be important for applications such as rendering of heterogeneous translucent objects and simulation of heterogeneous deformable objects. Our goal in this thesis is to allow the user to interactively create volumetric models of natural objects such as vegetables, fruit, and organs that contain complex internal structures. In particular, we aim at providing the user with efficient and intuitive modeling user interfaces while achieving compact and consistent representations.

Volumetric modeling is fundamentally difficult mainly because of two reasons. First, we cannot perceive volumetric information directly; hence it is difficult for a user to specify volumetric information directly. Second, computational cost in volumetric modeling can often become prohibitively large. We propose two general principles to deal with these two difficulties. To deal with the first difficulty, we propose to let the user specify information such as color and orientation on some surfaces characteristic to the object's internal structures, and then let the system propagate such information over the volume (**Principle I**). To deal with the second difficulty, we propose to exploit the object's structural regularity such as repetitions of structures to achieve compact representations and efficient algorithms (**Principle II**). In order to comprehensively understand different types of structural regularities represented by volumetric fields, we first classify volumetric fields according to the scale of structures represented by them and their anisotropy. We further consider appropriate representations for each type of volumetric fields, and realize that the raster-based approach is suited for representing detailed structures while the vector-based approach is suited for representing global structures. Based on this analysis, we propose two methods for volumetric modeling, one in the raster-based approach and the other in the vector-based approach.

For the raster-based approach, we propose a method to represent volumetric objects with repetitive detailed structures using anisotropic solid textures. The basic idea is to repeatedly paste patches of an input anisotropic solid texture to a model's interior according to a user-specified volumetric orientation field, filling the model with overlapping patches of the solid texture. We propose efficient sketch-based user interfaces for modeling volumetric orientation fields. We provide intuitive modeling user interfaces and efficient synthesis algorithms tailored for different types of solid textures. Our representation is compact because we can reuse the input solid texture many times and only need to store the texture mapping information instead of the synthesized color at every 3D point.

For the vector-based approach, we propose a method to represent smooth volumetric color transitions among global structures using colored 3D surfaces. The basic idea is to represent a model as a set of colored 3D surfaces, whose volumetric color distribution is obtained

by diffusing the surface colors over the volume. When computing the color diffusion, it is computationally expensive to globally solve for the entire volumetric color distribution inside the model at once. Instead, we propose to solve for the diffused colors only locally at cross-sectional points when the user cuts the model. Our representation is compact because a volumetric color distribution can be represented using only a sparse set of colored 3D surfaces without actually storing the diffused color at every 3D point. For the creation of such colored 3D surfaces, we propose a simple sketch-based 3D modeling user interface that assumes rotational symmetry in the model's internal structures.

With our two methods, it is possible to interactively create a number of volumetric models of various natural objects with complex internal structures using compact and consistent representations. These successful results suggest the validity of our two principles for volumetric modeling which are utilized in both of our two methods. We believe our findings in this thesis will serve as a foundation for the future development of volumetric modeling techniques.

論文要旨

物体内部の情報のモデリング、すなわちボリューメトリックなモデリングは、コンピュータグラフィックスにおける様々なアプリケーションにおいて有用である。例えばボリューメトリックな情報によって、三次元モデルに対して切断や皮剥きといった自然で直感的なインタラクションを行うことが可能になる。またボリューメトリックな情報は、非均質な半透明物体のレンダリングや非均質な変形体のシミュレーション等のアプリケーションにとって重要である。本博士論文の目的は、野菜や果物、臓器など複雑な内部構造を含む自然物のボリューメトリックなモデリングを、ユーザがインタラクティブに行えるようにすることである。特に我々は、効率的で直感的なモデリングユーザインタフェースを提供しつつ、コンパクトかつ整合性のある表現形式を実現することを目指す。

ボリューメトリックなモデリングは、主に以下の二点において本質的に難しい。第一に、ボリューメトリックな情報を人間が直接見ることができないため、ユーザはボリューメトリックな情報を直接指定することができない。第二に、ボリューメトリックなモデリングにおいてはその計算コストが莫大になることが多い。これらの難しさに対処するために、本論文で我々は以下の二つの原則を提案する。一つ目の難しさに対処するために、物体内部構造の特徴を表すいくつかのサーフェス上でユーザに色や方向などの情報を指定させ、その情報をシステムが物体内部のボリューム上に伝播させる、という原則を提案する (**原則 1**)。二つ目の難しさに対処するために、物体内部構造における繰返しなどの規則性を利用することで、コンパクトな表現形式と高速なアルゴリズムを実現する、という原則を提案する (**原則 2**)。物体内部構造の様々な規則性について体系的に理解するために、まず我々はボリューメトリックな場を、それが表す構造のスケールと異方性に基づき分類する。さらに、ボリューメトリックな場の各タイプに適した表現形式のアプローチについて考察し、微細構造を表すにはラスタ的アプローチが、大局的構造を表すにはベクタ的アプローチが適していることに着目した。これらの分析を元に、本論文ではラスタ的アプローチとベクタ的アプローチのそれぞれについて、上記の原則に基づいたモデリング手法を一つずつ提案する。

ラスタ的アプローチとして、物体内部の繰返しのある微細構造を、異方性ソリッドテクスチャを用いて表現する手法を提案する。基本的な考え方は、入力として与えられた異方性ソリッドテクスチャのパッチを、ユーザが指定したボリューメトリックな方向場に沿って物体内部に繰返し貼り付けることで、物体内部を重なり合うテクスチャパッチで満たす、というものである。我々はボリューメトリックな方向場を効率的にモデリングするためのユーザインタフェースを提案する。またソリッドテクスチャの各タイプに適した直感的なモデリングユーザインタフェースと効率的な合成アルゴリズムを提供している。提案法では入力のソリッドテクスチャを繰返し再利用するため、計算結果として各三次元位置における色ではなくテクスチャマッピングの情報だけを記憶すれば良いので、コンパクトな表現形式を実現できる。

ベクタ的アプローチとして、物体内部の大局的構造の間で滑らかに変化するようなボリューメトリックな色分布を、色付き三次元サーフェスを用いて表現する手法を提案する。基本的な考え方は、モデルを色付き三次元サーフェスの集合として表現し、サーフェスの色をボリューム上で拡散させることでボリューメトリックな色分布を得る、というものである。色の拡散を計算する

際、モデル内のボリューメトリックな色分布全体を一度に求めようとすると、計算コストが高くなりすぎるという問題がある。そこで我々は色の拡散を、ユーザがモデルを切断した際の断面上の各点でのみ局所的に計算する方法を提案する。提案法では、各三次元位置での拡散された色を記憶することなく、疎な色付き三次元サーフェスの集合のみでボリューメトリックな色分布を表現できるので、コンパクトな表現形式を実現できる。このような色付き三次元サーフェスを作成する方法として、我々は内部構造の回転対称性を仮定したシンプルなスケッチベースの三次元モデリングユーザインタフェースを提案する。

提案した二手法を用いることで、複雑な内部構造を持つ様々な自然物のボリューメトリックなモデルを、コンパクトかつ整合性のある表現形式でインタラクティブに作成することができる。これらの結果は、我々が提案したボリューメトリックなモデリングのための二つの原則の有効性を示唆している。本研究で得られた知見は、ボリューメトリックなモデリングの技術の今後のさらなる発展の基礎となることを信じている。

Acknowledgments

First of all, my deepest gratitude goes to my supervisor Takeo Igarashi. He always encouraged me to explore what I am really interested in, and taught me the right kind of optimism and criticism essential for conducting a good research. In particular, I appreciate his thorough training of me for conveying ideas with clear writing and concise presentation. He also encouraged me to go outside the lab and experience different environments around the world; I was able to have quite a few opportunities to visit universities overseas. Starting my graduate study under his supervision is absolutely the best choice I have ever made in my life. I am also grateful to my thesis committee members: Katsushi Ikeuchi, Takehsi Naemura, Tomoyuki Nishita, Reiji Suda, and Shigeo Takahashi, for providing suggestions essential for improving this thesis.

I thank lab members I met through my five and half years: Daisuke Sakamoto, Kazutaka Kurihara, Makoto Okabe, Masatomo Kobayashi, Takashi Ijiri, Takeshi Nishida, Yuki Igarashi, Hideki Todo, HyoJong Shin, Hidehiko Abe, Kaisuke Nakajima, Hirotaka Ikoma, Toshio Nakamura, Puripant Ruchikachorn, Motoi Washida, Tatsuya Takeda, Nobuyuki Umetani, Okihide Teramoto, Rapee Suveeranont, Koichiro Honda, Akihito Sakurai, Shojiro Oku, Sosuke Okamura, Yuji Yasuda, Jun Kato, Andrei Ostanin, Xander Caldwell, Vladimir Alves, Akira Ohgawara, Seung-Tak Noh, Makoto Nakajima, Ding Chen, Yuji Tanada, Saidi Farid, Shuntaro Hirakawa, Yutaro Hiraoka, Ryo Kajiwara, Yusuke Iida, Takeo Asai, Yuki Koyama, Genki Furumi, Sho Yamauchi, and Fangzhou Wang, for spending good time with me. In particular, my first SIGGRAPH paper owes much to Makoto Okabe's inspiring words. I enjoyed collaborating with Takashi on his ProcDef project and two projects in the medical field. I also enjoyed collaborating with Nobuyuki on his project of integrating FEM simulations and interactive shape design interfaces [108]. I spent fun and exciting time right before my graduation collaborating with Yuki Koyama in submitting his bachelor thesis project to SIGGRAPH.

I thank international students who visited the lab: Jim Young, Alec Rivers, Floraine Berthouzoz, Ken Christen, Erik Andersen, Lasse Farnung Laursen, and Oliver Mattausch, for spending good time with me and teaching me English. In particular, I cannot thank Jim enough for his kindness; he invited me to his sweet home both in Japan and in Canada quite a few times. He also hosted my visit to University of Calgary and University of Manitoba. Alec is perhaps the craziest person I have ever met; he always made all the lab members burst into laughter with his humor. He also introduced Rob Wang to me who kindly offered me a place to sleep when I visited Massachusetts Institute of Technology. Erik kindly hosted my visit to University of Washington.

One of the most important research activity in my graduate study was a visit to New York University hosted by Olga Sorkine. My SIGGRAPH Asia paper would have not been possible without her strong encouragement. I also appreciate Andy Nealan's collaboration on the project. I thank people I met during the visit: Tino Weinkauf, Ashish Myles, Nico Pietroni, Yotam Gingold, Adrian Secord, Denis Kovacs, Matt Grimes, Murphy Stein, Qingnan Zhou, and Alec Jacobson, for spending good time with me. In particular, I thank Yotam for introducing the awesome Oishi Judo Club to me which made my stay so fun.

Thanks to Olga's broad connection with worldwide researchers, I was able to collaborate with Tamy Boubekeur, Ryan Schmidt, and Karan Singh, on an exciting project about geometry editing [102], which is unfortunately not included in this thesis. After this collaboration, Karan kindly let me visit his DGP group in University of Toronto, where I spent good time with Nick Kim, Bardia Sadri, Janis Libeks, Noah Lockwood, James McCrae, Peter O'Donovan, Cloud Shao, Michael Tao, Jonathan Deber, Chung-Lin Wen, Koji Yatani, Edy Garfinkiel, Hanieh Bastani, Khai Truong, Aaron Hertzmann, and Marc Alexa.

I also thank my roommates during these visits: Shigeo and Jun Sakai in New York and Morgonn Ewen in Toronto for being kind to me.

I was lucky to be able to collaborate with researchers in the medical field: Kazuo Nakazawa, Ryo Haraguchi, and Takashi Ashihara. They constantly provided me with many interesting problems in the medical field where computer graphics techniques can possibly be an effective solution. I also thank people in JST ERATO Igarashi Design Interface Project: Jun Mitani, Myung Geol Choi, Manfred Lau, and Rubaiat Habib, for having exciting discussions.

Lastly, I am deeply grateful to my parents Eiki and Ikue Takayama and my brother Ryo Takayama for their constant support and belief in me.

I have received financial support from various sources. During the first year of my master's program, I was supported by Mitou Youth program of Information-technology Promotion Agency, Japan. My three years of doctoral study have been funded by the fellowship of Japan Society for the Promotion of Science. My visit to NYU was supported by JSPS Excellent Young Researcher Overseas Visit Program. I was supported by Graduate School of Information Science and Technology, the University of Tokyo, for a few times when I visited various places overseas.

Contents

Chapter 1

Introduction

In 3D computer graphics, most objects are represented as surface models; the geometric shape of an object is represented as a manifold surface residing in the 3D space, and various properties such as color and reflectance are defined on the manifold as surface textures. There are numerous approaches to creating both surface geometries and surface textures, such as manual approaches of 3D sculpting and painting interfaces, procedural approaches of using rules and simulations, and measurement-based approaches using various sensors. While surface modeling still remains difficult, thanks to the significant advancements in the past decades, surface modeling is widely accepted by general users and is used to model a variety of things ranging from organic to man-made objects.

On the other hand, surface modeling is insufficient for several applications where information about the interior of an object is needed (Figure 1.1). One such application is interactive cutting of objects such as organ tissues and foodstuff in the context of virtual training systems, where an appropriate cross-sectional image needs to be shown immediately when the user cuts a model. Another example is photorealistic rendering of translucent heterogeneous objects where volumetric information plays an important role in producing highly realistic rendering results difficult with surface models. Simulating deformations of complex heterogeneous objects would also require volumetric information. Modeling of 3D objects' internal information, or volumetric modeling, is important for these applications. Volumetric modeling can be viewed as a task of constructing a volumetric field: a mapping from 3D position to some attribute values such as color, orientation, or density. The term "solid texture" is often used as a synonym for a volumetric field.

There are three major approaches to volumetric modeling. The first approach

Figure 1.1: Examples of applications in which volumetric information is useful: a virtual training system for cutting foodstuff [73] (left), a film *"Ratatouille"* [84] (middle), and simulating deformations of heterogeneous soft tissues [4] (right).

is to use simple mathematical expressions to procedurally define volumetric fields (Figure 1.2 left). The foremost advantage of this approach is its compactness; only mathematical expressions need to be stored to represent volumetric fields. On the other hand, this approach requires devising an appropriate mathematical expression for a particular modeling target which is a difficult task in general. The second approach is to synthesize cross-sectional images of a 3D model from 2D photographs (Figure 1.2 middle). This approach allows the user to quickly create a plausible volumetric model by simply specifying correspondences between 2D photographs and the 3D model's cross-sections. On the other hand, this approach often produces unnatural cross-sectional images for some cutting configurations, because the cross-sectional images are computed solely based on 2D photographs without considering 3D structures. The third approach, which we take in this thesis, is to explicitly model a volumetric field stored on a spatial data structure such as a voxel grid (Figure 1.2 right). This approach can be used to represent general objects, and is well suited for applications that require consistent 3D structures such as volume rendering.

Our goal in this thesis is to allow the user to interactively create volumetric models of natural objects such as vegetables, fruit, organs, and geological structures that contain complex internal structures (Figure 1.3), using efficient modeling interfaces. At the same time, we aim at retaining consistent 3D structures of volumetric models by explicitly storing volumetric fields. Volumetric modeling in this approach is difficult mainly because of two reasons. First, it is difficult for the user to specify volumetric information directly, since volumetric information cannot be perceived directly. Second, computational cost can often become prohibitively large, since the entire volumetric information inside the model needs to be stored and processed explicitly. We propose two general principles for volumetric modeling to deal with these two difficulties. To

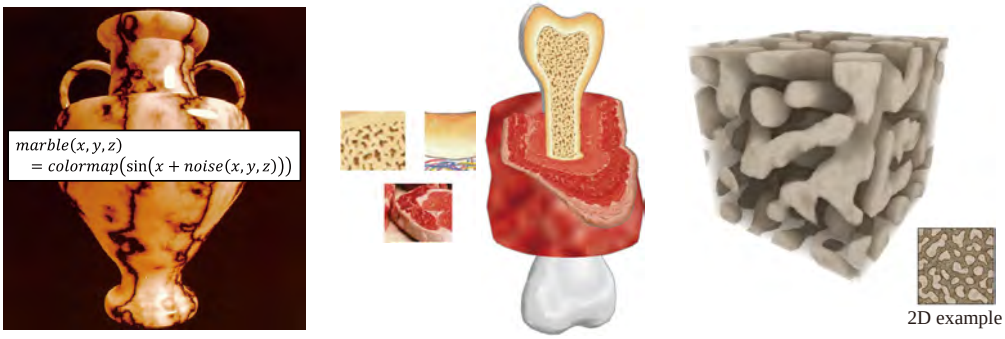

Figure 1.2: Three major approaches to volumetric modeling: procedural definition using mathematical expressions (left), synthesis of cross-sectional images from photographs (middle), and explicit modeling of volumetric fields (right).

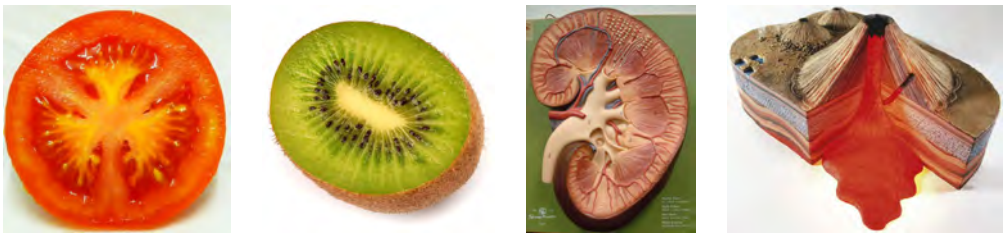

Figure 1.3: Examples of natural objects that we aim to model volumetrically in this thesis.

deal with the first difficulty, we propose to let the user specify attribute values (e.g., color and orientation) on surfaces characteristic to the object's internal structures, and then let the system propagate such values over the volume (**Principle I**). To deal with the second difficulty, we propose to exploit the object's structural regularity such as repetitions of structures to achieve compact representations and fast algorithms (**Principle II**). In order to comprehensively understand different types of structural regularities represented by volumetric fields, we first classify volumetric fields into several types based on the scale of structures represented by them and their anisotropy. We further consider appropriate representations for different types of volumetric fields, and realize that volumetric fields representing smaller and larger scale structures can be well modeled using raster-based and vector-based approaches, respectively. Based on this analysis, we propose two methods for volumetric modeling, one in the raster-based approach and the other in the vector-based approach, both of which make use of the two principles mentioned above.

For the raster-based approach, we propose a method to represent volumetric ob-

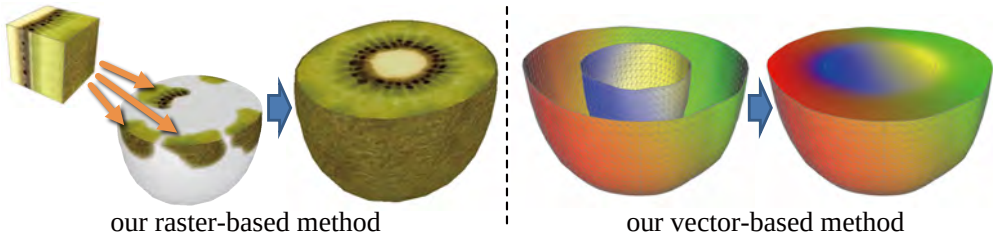

Figure 1.4: Our two methods for volumetric modeling, one in the raster-based approach (left) and the other in the vector-based approach (right).

jects with repetitive detailed structures using anisotropic solid textures. The basic idea is to repeatedly paste patches of an input anisotropic solid texture to a model's interior according to a user-specified volumetric orientation field, filling the model with overlapping patches of the solid texture (Figure 1.4 left). We propose efficient sketch-based user interfaces for modeling volumetric orientation fields. We provide an intuitive modeling user interface and an efficient synthesis algorithm tailored for each texture type in our classification of solid textures. Our representation is compact because we can reuse the input solid texture many times and only need to store the texture mapping information (i.e., 3D texture coordinates), instead of the synthesized color at every 3D point.

For the vector-based approach, we propose a method to represent smooth volumetric color transitions among global structures using colored 3D surfaces. The basic idea is to represent a model as a set of colored 3D surfaces, whose volumetric color distribution is obtained by diffusing the surface colors over the volume (Figure 1.4 right). When computing the color diffusion, it is computationally expensive to globally solve for the entire volumetric color distribution inside the model at once. Instead, we propose to solve for the diffused colors only locally at cross-sectional points when the user cuts the model. Our representation is compact because a volumetric color distribution can be represented using only a sparse set of colored 3D surfaces without actually storing the diffused color at every 3D point.

With our two methods, it is possible to interactively create a number of volumetric models of various natural objects with complex internal structures using compact and consistent representations. These successful results suggest the validity of our two principles for volumetric modeling which are utilized in both of our two methods. We believe our findings in this thesis will serve as a foundation for the future development of volumetric modeling techniques.

1.1 Thesis Overview

In Chapter 2, we first detail our two principles for volumetric modeling: propagating user-specified information on surfaces through the volume (Principle I), and exploiting structural regularities to achieve compact representations and fast algorithms (Principle II). We then present our classification of volumetric fields based on two aspects: variation level and anisotropy level. Finally, we describe raster-based and vector-based approaches to volumetric modeling, each of which is suited for modeling different type of volumetric fields.

In Chapter 3, we first review existing volumetric modeling methods in three major approaches: procedural definition using mathematical expressions, synthesis of cross-sectional images from photographs, and explicit modeling of volumetric information. The third approach is further divided into the raster-based and vector-based approaches, each of which is reviewed. Finally, we also briefly mention two other topics highly related to volumetric modeling: realistic rendering of translucent materials and volume visualization.

In Chapter 4, we present two interactive sketch-based interfaces for modeling volumetric orientation fields, which are later used in our raster-based method presented in Chapter 5 for synthesizing anisotropic solid textures. We also demonstrate two applications of these user interfaces other than synthesizing solid textures: electrophysiological simulation of heart ventricles and active deformation of unarticulated characters.

In Chapter 5, we present our raster-based method to represent volumetric objects with repetitive detailed structures using anisotropic solid textures. The basic idea is to repeatedly paste patches of an input anisotropic solid texture to a model's interior according to a user-specified volumetric orientation field created using our interfaces presented in Chapter 4. We provide an intuitive modeling user interface and an efficient synthesis algorithm tailored for each type of solid textures.

In Chapter 6, we present our vector-based method to represent smooth volumetric color transitions among global structures using colored 3D surfaces. The basic idea is to represent a model as a set of colored 3D surfaces, whose volumetric color distribution is obtained by diffusing the surface colors over the volume. We propose an efficient algorithm for computing the color diffusion only locally at cross-sectional points. For the creation of such colored 3D surfaces, we propose a simple sketch-based 3D modeling user interface that assumes rotational symmetry in the model's internal structures.

In Chapter 7, we reconfirm how our two principles for volumetric modeling are

utilized in our two methods, and verify the validity of the principles. We then discuss capabilities and limitations of our methods to make clear what kind of objects our methods can and cannot model. We also present possible ideas for combining the raster-based and vector-based approaches to enable the hybrid approach. Finally, we close this thesis by describing areas for future research.

1.2 Publications

Below is a list of publications that compose this thesis:

- Our user interface for modeling volumetric vector field described in Section 4.3 was presented as *A Sketch-Based Interface for Modeling Myocardial Fiber Orientation* [100] at Smart Graphics 2007 in Kyoto, Japan, in collaboration with Takeo Igarashi from the University of Tokyo and Ryo Haraguchi and Kazuo Nakazawa from the National Cardiovascular Center Research Institute.

- Our user interface for modeling volumetric frame field described in Section 4.4 was published as *A Sketch-Based Interface for Modeling Myocardial Fiber Orientation that Considers the Layered Structure of the Ventricles* [98] in The Journal of Physiological Sciences, in collaboration with Takashi Ashihara from Shiga University of Medical Science, Takashi Ijiri and Takeo Igarashi from the University of Tokyo, and Ryo Haraguchi and Kazuo Nakazawa from the National Cardiovascular Center Research Institute.

- The application of volumetric frame fields to active deformations of unarticulated characters described in Section 4.6 was presented as *ProcDef: Local-to-global Deformation for Skeleton-free Character Animation* [42] at Pacific Graphics 2009 in Jeju, Korea, in collaboration with Takashi Ijiri and Takeo Igarashi from the University of Tokyo and Hideo Yokota from RIKEN.

- Our raster-based method for volumetric modeling described in Chapter 5 was presented as *Lapped Solid Textures: Filling a Model with Anisotropic Textures* [101] at ACM SIGGRAPH 2008 in Los Angeles, USA, in collaboration with Makoto Okabe, Takashi Ijiri, and Takeo Igarashi from the University of Tokyo.

- Our vector-based method for volumetric modeling described in Chapter 6 was presented as *Volumetric Modeling with Diffusion Surfaces* [103] at ACM SIGGRAPH Asia 2010 in Seoul, Korea, in collaboration with Olga Sorkine from New

York University, Andrew Nealen from Rutgers University, and Takeo Igarashi from the University of Tokyo.

- Our algorithm for synthesizing layered solid textures described in Appendix A was presented as *Layered Solid Texture Synthesis from a Single 2D Exemplar* [99] at ACM SIGGRAPH 2009 Posters in New Orleans, USA, in collaboration with Takeo Igarashi from the University of Tokyo.

Chapter 2

Analysis of Volumetric Modeling

In this chapter, we describe our high level analysis of volumetric modeling that is essential to this thesis. We first identify two fundamental difficulties in volumetric modeling, and then propose two principles to deal with these difficulties. Next, in order to comprehensively understand different types of structures represented by volumetric fields, we classify volumetric fields into several types based on the scale of structures represented by them and their anisotropy. We further consider appropriate representations for different types of volumetric fields containing different scales of structures, and realize that raster-based and vector-based representations are suited for modeling smaller and larger scale structures, respectively. Based on this analysis, in Chapters 5 and 6 we propose our two methods for volumetric modeling, one in the raster-based approach and the other in the vector-based approach, both of which make use of our two principles. This analysis is also useful for relating other existing methods for volumetric modeling to ours, as detailed in Chapter 3.

2.1 Fundamental Difficulties in Volumetric Modeling

Volumetric modeling is fundamentally more difficult than surface modeling, because the data is distributed three-dimensionally in the case of volumetric modeling whereas it is distributed two-dimensionally in the case of surface modeling. This can be seen as one form of the curse of dimensionality in a sense, as new difficulties arise in the case of volumetric modeling which did not exist in the case of surface modeling. We identify two fundamental difficulties in volumetric modeling as follows.

The first difficulty is regarding the burden on the user when specifying volumetric information. Since volumetric information distributed three-dimensionally cannot be perceived directly by humans, it is difficult for the user to specify volumetric infor-

mation directly. One possible way would be to let the user specify sparse constraint values at arbitrary 3D locations which are then spatially interpolated. This approach would be, however, still difficult for the user because of the unnecessarily high degree of freedom in specifying constraint values at arbitrary 3D locations.

The second difficulty is regarding the computational cost when storing and processing a volumetric field. For 3D volumes the amount of data storage increases much more rapidly as the resolution increases than that for 2D images. For example, let us compare the amount of data needed to store a square-shaped image represented as a regular pixel grid and a cube-shaped volume represented as a regular voxel grid with the same number of elements along one coordinate axis. As the number of elements along the coordinate axis increases from 64 to 256 and 1024, the data size of the image increases from 12KB to 200KB and 3.1MB, respectively, whereas that of the volume increases from 79KB to 50MB and 3.2GB, respectively (when a 24bit RGB color is stored for each element). Note that in addition to storing such a large amount of data, it is often necessary to process it in various ways such as performing k-nearest neighbor search (when performing texture synthesis) and applying filter kernels (when solving partial differential equations). If we naively store and process individual voxels, the computational cost can easily become prohibitively large as the resolution increases.

2.2 Principles for Effective Volumetric Modeling

In order to deal with the fundamental difficulties in volumetric modeling described above, we propose two general principles for volumetric modeling as follows.

2.2.1 Principle I: Propagating User-Specified Information on the Surfaces Through the Volume

Our first principle to deal with the first difficulty of specifying volumetric information directly is to let the user specify desired values on surfaces that characterize an object's internal structures, which we refer to as *characterizing surfaces*, and then have the system propagate such user-specified values on the characterizing surfaces through the volume (**Principle I**). Examples of such characterizing surfaces include the object's external boundary surface, isosurfaces of the object's volumetric depth field if the object has a layered internal structure, and sharp features (i.e., surfaces across which values change suddenly) in the object's volumetric field. Because the dimensionality is reduced from three to two in this approach, it is easy for the user to specify desired values on the characterizing surfaces using ordinary interfaces such

as sketching and painting. Besides, this approach prevents the user from creating arbitrary and meaningless volumetric fields by limiting the degree of freedom appropriately using the characterizing surfaces. Lastly, the characterizing surfaces are also useful for modifying the resulting volumetric field to satisfy some application-specific requirements. For example, when modeling a volumetric vector field inside an object that has a layered internal structure, we can make the resulting volumetric vector field be always tangent to the depth layers by projecting the vector to the layer at every 3D location. It would be much more difficult for the user to achieve the same goal using the approach of specifying arbitrary vectors at arbitrary 3D locations.

2.2.2 Principle II: Exploiting Structural Regularities to Achieve Compact Representations and Fast Algorithms

Our second principle to deal with the difficulty of rapid increase in computational cost is to exploit the object's structural regularity to achieve compact representations and fast algorithms (**Principle II**). Examples of such structural regularities include repetitions of detailed structures, smooth transitions of the volumetric field with a few sharp features, and symmetries in the object's internal structures. By following this principle, we no longer need to store and process individual voxels, leading to compact representations and fast algorithms.

In order to comprehensively understand structural regularities represented by volumetric fields, in the next section we classify volumetric fields based on the scale of structures represented by them and their anisotropy.

2.3 Classification of Volumetric Fields

In this section, we propose a classification of volumetric fields in order to provide a comprehensive understanding of structural regularities represented by volumetric fields. Our classification is based on two aspects of volumetric fields: variation level and anisotropy level. (We refer to volumetric fields as solid textures synonymously here.) This classification forms a basis of our raster-based method presented in Chapter 5 when designing interfaces and algorithms for arranging different types of solid textures inside 3D models. In addition, this classification is also useful for depicting relationships among other volumetric modeling methods, including existing ones reviewed in Chapter 3 and our own presented in Chapter 6.

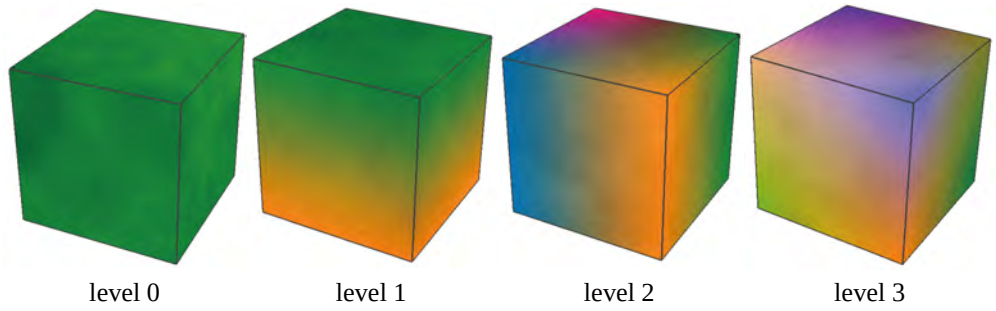

Figure 2.1: Variation level of volumetric fields.

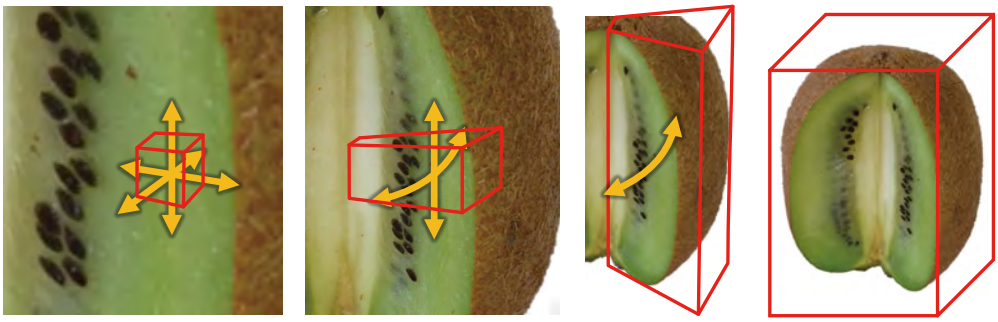

Figure 2.2: An example using a kiwi fruit showing how variation levels correspond to scales of structures. Red wireframes represent the texture volumes, while the orange arrows represent directions in which texture patterns repeat.

2.3.1 Variation level

The variation level describes how the texture pattern varies spatially (Figure 2.1). A texture with variation level 0 represents a homogeneous structure; the texture pattern repeats in all directions in 3D. A texture with variation level 1 represents a layered structure; there exists a depth direction in the texture along which the texture pattern varies, while the texture pattern repeats in the directions perpendicular to the depth direction. A texture with variation level 2 represents a sweeping structure; there exists a sweeping direction in which the texture pattern repeats, while the texture pattern varies arbitrarily in the directions perpendicular to the sweeping direction. A texture with variation level 3 represents an arbitrary structure; the texture pattern varies arbitrarily in all directions in 3D. The variation level corresponds to the scale of the structure represented by the texture; the higher the variation level, the larger the scale of the structure represented by the texture (Figure 2.2).

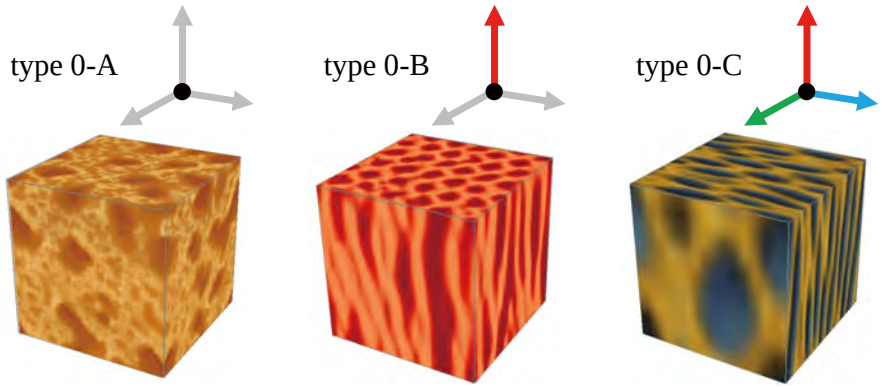

Figure 2.3: Three types of homogeneous (i.e., variation level 0) textures with different levels of anisotropy. The distinguishability of the coordinate axes of the space of repetition is depicted using colors.

2.3.2 Anisotropy level

For textures with variation levels 0 and 1, texture patterns repeat three- and two-dimensionally, respectively. We refer to such a space over which the texture pattern can repeat as *the space of repetition*. The anisotropy level, the other aspect of our classification, describes whether the coordinate axes of the space of repetition are distinguishable from each other.

For textures with variation level 0, there exist three types of textures depending on the anisotropy level: type 0-A which does not distinguish any of the three coordinate axes of the space of repetition, type 0-B which distinguishes one coordinate axis from the other two, and type 0-C which distinguishes all the three coordinate axes (Figure 2.3). A texture of type 0-A can be arranged in the volume without any orientation information, while a texture of types 0-B and 0-C need a volumetric vector field and a volumetric frame field (i.e., set of three vector fields that are orthogonal to each other everywhere, see Chapter 4), respectively, in order to be arranged in the volume.

For textures with variation level 1, there exist two types of textures depending on the anisotropy level: type 1-A which does not distinguish the two coordinate axes of the space of repetition, and type 1-B which does distinguish them (Figure 2.4). A texture with variation level 1 needs a volumetric depth field in order to be arranged in the volume. A texture of type 1-B additionally needs another volumetric vector field which is orthogonal to the depth direction everywhere, in order to be arranged in the volume.

The anisotropy level is undefined for textures with variation levels 2 and 3, since

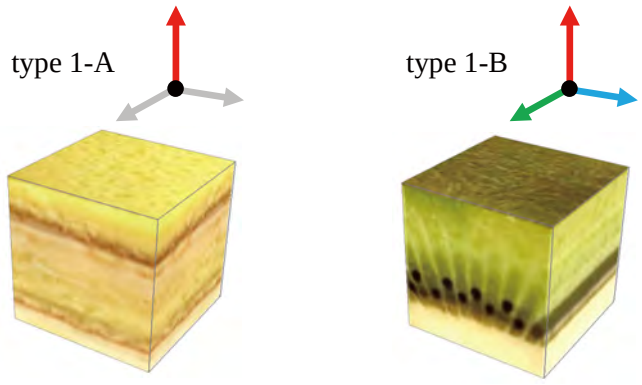

Figure 2.4: Two types of layered (i.e., variation level 1) textures with different levels of anisotropy.

structures in textures with variation level 2 repeat only one-dimensionally and structures in textures with variation level 3 does not repeat in any directions. In summary, our classification categorizes solid textures into seven types as 0-A, 0-B, 0-C, 1-A, 1-B, 2, and 3 (Figure 2.5). We consider appropriate representations for different types of solid textures in the next section.

2.4 Representations for Volumetric Fields

Different types of volumetric fields can be better modeled using different representations. In this section, we consider which representation is suited to model which type of volumetric fields. For this, we borrow ideas from the well-known raster-based and vector-based representations for 2D images. The raster-based representation stores an image as a regular pixel grid holding individual pixel values, while the vector-based representation stores an image as analytic information about the image content's geometric structures (e.g., region boundaries) along with smoothly varying colors. It is also well known that the raster-based representation is suited for encoding small-scale structures and noise-like details while the vector-based representation is suited for encoding large-scale structures and smooth color transitions with sharp color boundaries (Figure 2.6).

Analogous to the case of 2D images, we can also consider raster-based and vector-based approaches for representing 3D volumes. In our classification, volumetric fields with smaller variation levels represent smaller scale structures, and therefore they can be modeled well using raster-based approaches. Similarly, volumetric fields with larger variation levels represent larger scale structures, and therefore they can be modeled

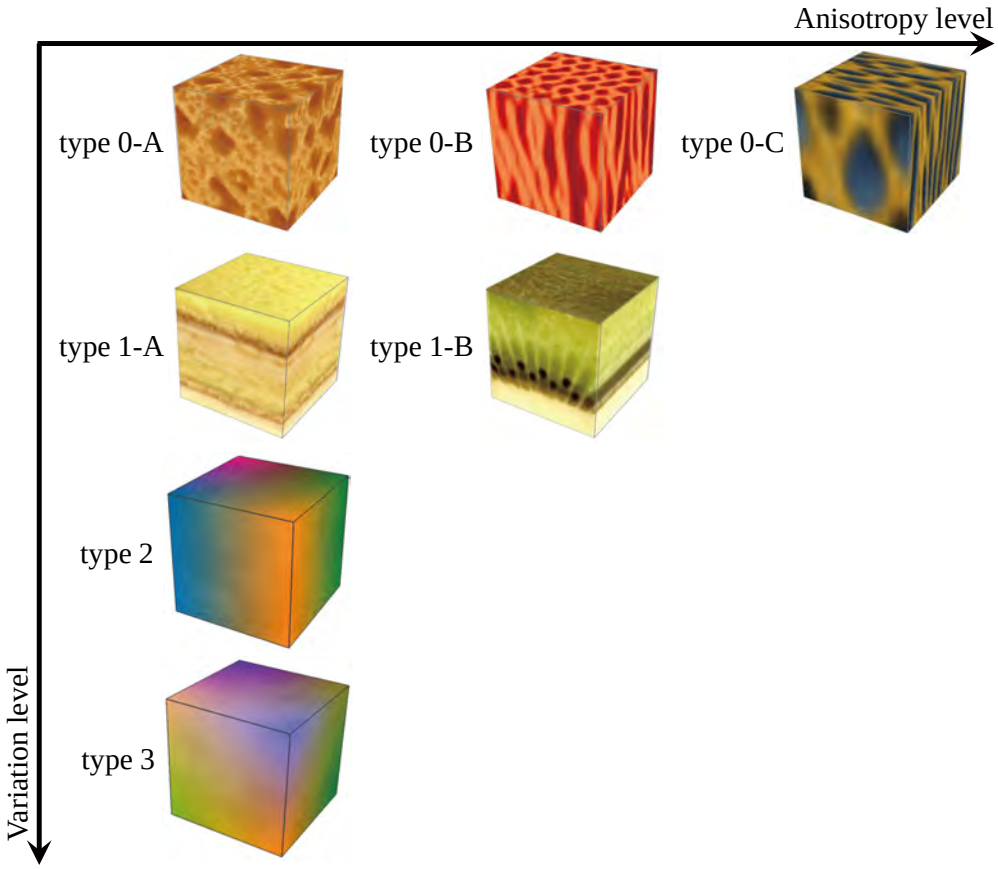

Figure 2.5: Seven types of solid textures in our classification.

Figure 2.6: The raster-based (left) and vector-based (right) representations for 2D images.

well using vector-based approaches. From this perspective, we can categorize existing volumetric modeling methods into raster-based and vector-based approaches, as we detail in Chapter 3. In this thesis we present two methods for volumetric modeling, one in the raster-based approach and the other in the vector-based approach.

Our raster-based method presented in Chapter 5 arranges textures with variation levels 0 and 1 inside 3D models with spatially-varying texture orientation and scale to produce textures of types 2 and 3. In Chapter 4, we present sketch-based interfaces for modeling volumetric orientation fields needed for arranging textures of types 0-B, 0-C, and 1-B, as well as a painting interface for modeling volumetric depth fields needed for arranging textures of types 1-A and 1-B. Our method is general enough to handle textures of type 2. Such textures, however, can only be arranged one dimensionally, limiting our method's advantage of compactness by reusing the same texture data. Besides, textures of type 2 would be more difficult to create than textures with variation levels 0 and 1 because such textures involve larger scale structures. We therefore chose not to support textures of type 2 in our method.

Our vector-based method presented in Chapter 6 can be regarded as a method for modeling textures of type 3, similar to other vector-based methods [15, 111, 110]. In addition, our 3D modeling user interface that assumes rotational symmetry can be regarded as an interface for creating textures of type 2 whose structures repeat along the circumferential direction around the axis of rotational symmetry. Our method is not well suited for creating textures with variation levels 0 and 1 that represent detailed structures, since numerous tiny surfaces required for representing such detailed structures would limit our method's advantage of compactness by representing a volumetric color distribution with a sparse set of colored 3D surfaces.

Chapter 3

Related Work

In this chapter, we review existing methods for volumetric modeling such that this thesis can be placed in an appropriate context. There are three approaches to volumetric modeling: procedural definition using mathematical expressions, synthesis of cross-sectional images from photographs, and explicit modeling of volumetric information. The third approach, which we take in this thesis, can be further divided into the raster-based approach and the vector-based approach. We review existing methods in each of these approaches and discuss their relations to this thesis. Lastly, we also mention other topics relevant to volumetric modeling: realistic rendering of translucent materials and volume visualization.

Pietroni et al. [82] presented a survey on solid texture synthesis which has some overlap with this chapter, while we categorize existing methods for volumetric modeling in our own manner.

3.1 Procedural Definition using Mathematical Expressions

In the early time of computer graphics, researchers first explored the approach of procedural definition using mathematical expressions for volumetric modeling. In 1985, Peachey [79] and Perlin [81] independently introduced the very first idea of volumetric modeling where a combination of a scalar function in 3D space and a color map defines a volumetric color distribution.

Specifically, their idea is to define a scalar function $f : \mathbb{R}^3 \mapsto \mathbb{R}$ in 3D space and a color map $colormap : \mathbb{R} \mapsto \mathcal{C}$ with $\mathcal{C}$ being a space of color vector (e.g., RGB vector), so that their combination $colormap \circ f : \mathbb{R}^3 \mapsto \mathcal{C}$ defines a volumetric color distribution. They named this approach "solid texturing", a common term now, and demonstrated that it is very well suited for achieving "carved-out" effects (i.e., as if an object is

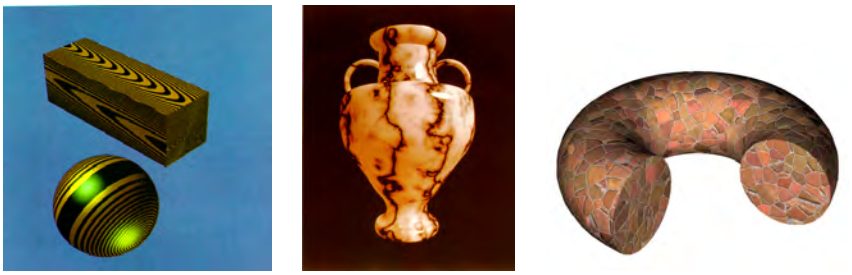

Figure 3.1: Examples of solid texturing demonstrated by Peachey (left), Perlin (middle), and Worley (right).

carved out of a certain material), as shown in Figure 3.1.

One significant invention by Perlin [81] is a noise function $noise : \mathbb{R}^3 \mapsto \mathbb{R}$ that outputs a random noise value at an arbitrary 3D position, which can be used effectively to simulate natural variations in the texture patterns. This noise function is designed to have a nice "band-limited" property (i.e., the magnitude of its frequency power spectrum is roughly limited within a certain range of radius), allowing us to sum up spatially scaled versions of the noise with different weights to define a noise that has a specific appearance desired by the artist. For example, a turbulence-like noise can be defined as

$$turbulence(x, y, z) = \sum_{i=0}^{N} \frac{noise(2^i x, 2^i y, 2^i z)}{2^i}.$$

Perlin demonstrated various impressive solid texturing examples by combining noise functions and some other basic math functions. For example, the solid texture used in the marble vase model in Figure 3.1 (middle) is defined as:

$$marble(x, y, z) = colormap(\sin(x + turbulence(x, y, z))).$$

Note that there have been continuous efforts on improving noise generation algorithms since Perlin's invention, with a notable recent development of Gabor noise by Lagae et al. in 2009 [56] which allows us to produce a wide variety of different texture patterns from just a few parameters. For more details about various noise generation algorithms, we refer the reader to a survey by Lagae et al [55].

There are a few different ways to define procedural solid textures other than combining noise and math functions, including cellular texturing method by Worley [116]. The reader may refer to a book by Ebert et al. [25] which covers various methods for procedural texture design.

3.1.1 Discussion

The foremost advantage of this approach is its compactness; only mathematical expressions need to be stored to represent the volume. In addition, once an appropriate mathematical expression is found for a particular modeling target, often that mathematical expression is easy to implement and requires low computational cost, making this approach suited for practical applications such as games and films.

On the other hand, this approach requires devising an appropriate mathematical expression for a particular modeling target which is a difficult task in general. Besides, it would be challenging, if not impossible, to represent general volumetric objects only using combinations of math and other functions.

3.2 Synthesis of Cross-Sectional Images from Photographs

The second approach to volumetric modeling is synthesis of plausible cross-sectional images based on user-specified correspondences between the 3D model and 2D cross-sectional photographs. This approach seems to have received relatively less attention compared to other approaches, and only two methods have been proposed in this approach to our knowledge.

3.2.1 2D Texture Synthesis on Cross-Sections

Owada et al. [76] proposed to mimic the illusion of a volumetric model without actually constructing large volumetric data by synthesizing a plausible 2D texture image on a cross-section of a 3D model when the user cuts the model. Figure 3.2 shows an overview of their approach. Starting with a 2D source texture and a 3D surface model, the user specifies *control maps* for both the source texture and a cross-section of the 3D model, indicating the relationship between the source texture and the 3D model. Note that information stored in the control map and the way to specify it vary depending on the type of texture (e.g., when creating layered textures, the control map stores a scalar depth field which is specified by painting). Given these control maps, the system synthesizes a plausible 2D texture on the cross-section using modified versions of standard example-based 2D texture synthesis algorithms [114, 106]. Note that with respect to our classification of solid textures, their method conceptually considered textures of types 0-A, 0-B, and 1-A, though not being based on using solid textures.

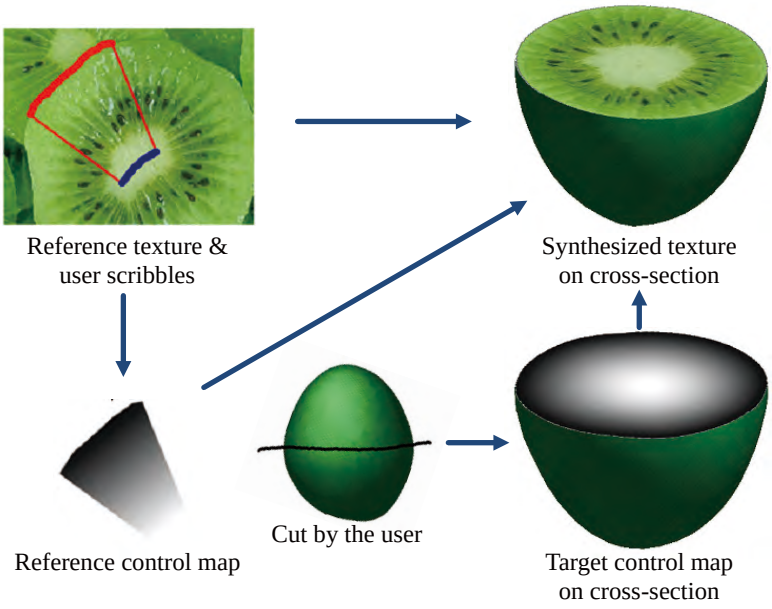

Figure 3.2: Overview of the volumetric illustration system by Owada et al, [76].

3.2.2 Morphing of Cross-Sectional Photographs

Pietroni et al. [83] proposed to perform morphing between a set of cross-sectional photographs of an object and paste the morphed image onto the cross-section of a 3D model. The input to their system consists of a set of cross-sectional photographs and a 3D surface mesh representing the overall shape of the object. The only required user interaction is to arrange the set of cross-sectional photographs in the 3D space appropriately (Figure 3.3 left). Then the system can compute a color for an arbitrary 3D point by projecting it onto the user-specified cross-sectional image planes and blending the colors of the projected image pixels with different weights according to the distances from the planes (Figure 3.3 middle left).

3.2.3 Discussion

This approach has mainly two advantages. First, it allows the user to quickly create a plausible volumetric model by simply specifying correspondences between 2D photographs and the 3D model's cross-sections. Second, the representations in this approach are both compact, because only the set of 2D photographs need to be stored to represent a volumetric model.

The downside of this approach is that unnatural cross-sectional images often result

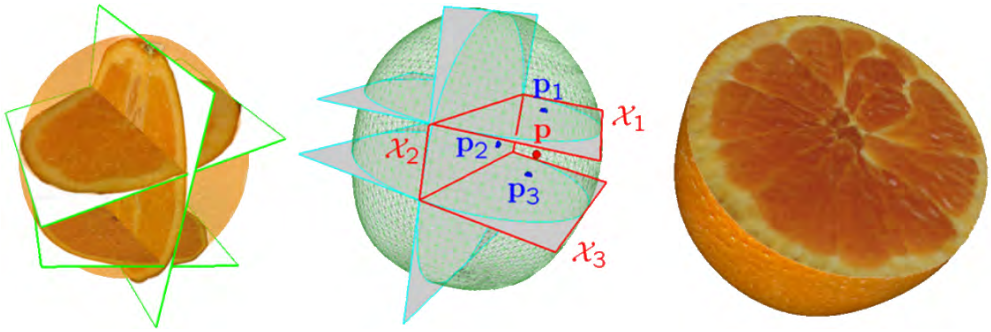

Figure 3.3: The approach of Pietroni et al. [83] of morphing cross-sectional photographs.

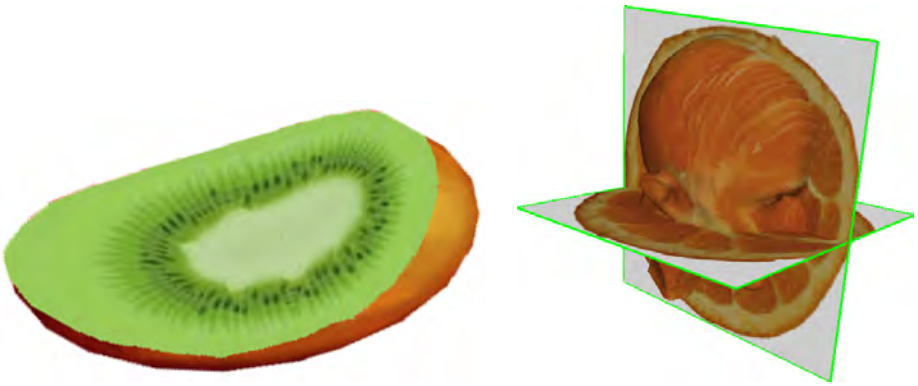

Figure 3.4: Unnatural cross-sectional images produced by Owada et al.'s method (left) and Pietroni et al.'s method (right).

for some cutting configurations (Figure 3.4), because the cross-sectional images are computed solely based on 2D photographs without considering 3D structures. For the similar reason, this approach is also not suited for applications such as volume rendering that require consistent volumetric information.

3.3 Explicit Modeling of Volumetric Field

The third approach, which we take in this thesis, is to explicitly store a volumetric field on a spatial data structure such as a voxel grid. This approach can be used to represent general objects, and is well suited for applications that require consistent 3D structures such as volume rendering. As mentioned earlier, two approaches exist for representing volumetric fields: raster-based approach and vector-based approach. The raster-based approach represents a field by storing individual field values on a

regular lattice, while the vector-based approach represents a field by storing analytic information about the field's geometric structures (e.g., region boundaries) along with smoothly varying field values. The raster-based approach is suited for representing detailed structures, while the vector-based approach is suited for representing global structures. In this thesis we present two methods for volumetric modeling, one in the raster-based approach and the other in the vector-based approach. Here we review existing methods in each of the raster-based and vector-based approaches, and discuss the differences of our methods from these existing methods.

3.3.1 Raster-Based Approach

The raster-based approach mainly consists of methods for synthesizing a bitmap solid texture (i.e., 3D volume stored on a voxel grid) from 2D images. The goal is to take a few 2D exemplar images and synthesize a solid texture such that its cross-sections in certain orientations appear similar to the input 2D exemplar images everywhere. Parametric methods were explored in the early period, while nonparametric methods were later explored to overcome the limitations of parametric methods. The reader may refer to a survey by Wei et al. [113] that extensively covers various techniques of example-based texture synthesis for both 2D and 3D. We also mention an approach of manually painting on solid textures using colored particles.

Parametric Solid Texture Synthesis

In the parametric approach, a certain underlying computational model that can generate a certain class of solid textures is assumed, and the goal is to estimate the parameters for such a model based on an input exemplar image so that the output solid texture visually matches with the exemplar image. Statistical techniques are often used for this parameter estimation.

Pyramid Decomposition and Histogram Matching: Heeger and Bergen [37] presented a method for parametric texture synthesis using two statistical techniques: pyramid decomposition and histogram matching, which are applicable to both 2D images and 3D volumes.

In pyramid decomposition, the original image I_0 is first downsampled with a factor of $\frac{1}{2}$ as

$$I_1 = downsample(I_0),$$

which is the low-pass component of I_0. The high-pass component of I_0 is obtained by subtracting the low-pass component upsampled with a factor of 2 as

$$H_0 = I_0 - upsample(I_1).$$

This decomposition is repeated recursively, resulting in a sequence of H_0, H_1, $\cdots$, H_N and a leftover low-pass component I_{N+1} at the coarsest level. Note that the original image I_0 can be trivially reconstructed by reversing the decomposition procedure.

Histogram matching, the other key technique of their method, transforms the colors in the synthesis image so that its color histogram matches that of the exemplar image. To do this, for each color channel, histograms are constructed for both of the exemplar and the synthesis images as $histogram_E$ and $histogram_S$, respectively (i.e., $histogram_E(c)$ counts the number of pixels in the exemplar image whose values equal to c). Next, the cumulative distribution function (CDF) for the exemplar image is computed as

$$cdf_E(c) = \sum_{c' \leq c} histogram_E(c')$$

and is then normalized such that $cdf_E(c_{max}) = 1$, where c_{max} is the possible maximum color value (e.g., 255 when using 8 bit integer for each channel). The CDF for the synthesis image cdf_S is computed similarly. Then, the histogram matching operation replaces the color of each pixel in the synthesis image using the color mapping function $cdf_E^{-1} \circ cdf_S$.

With these two techniques at hand, Heeger and Bergen's texture synthesis method starts by generating a 3D volume of white noise, then applying histogram matching to the pair of the exemplar image and the synthesis volume. Then, the pyramid decomposition is performed on both the exemplar image and the synthesis volume, followed by histogram matching applied to the pairs of corresponding components in the exemplar pyramid and the synthesis pyramid. The output solid texture is finally reconstructed from the synthesis pyramid. Figure 3.5 (left) shows one of their results.

Spectral Analysis: Ghazanfarpour and Dischler [34] proposed another parametric texture synthesis method based on spectral analysis of the input exemplar image. The input exemplar image $I(x, y)$ is first converted into the power spectrum $S(f_x, f_y)$ in the frequency domain using Fourier transform. Next, a new randomized image $I'(x, y)$ is obtained as a sum of modulated cosine waves corresponding to frequencies whose amplitudes in $S(f_x, f_y)$ are larger than a threshold. Finally, an output solid texture

Figure 3.5: Results of parametric solid texture synthesis techniques demonstrated by Heeger and Bergen (left) [37], Ghazanfarpour and Dischler (middle) [34], and Jagnow et al. (right) [43].

$V(x, y, z)$ is obtained as

$$V(x, y, z) = I'(x + p(x, y, z), y + p(x, y, z)),$$

where $p(x, y, z) = whitenoise(x, y, z) \otimes h(x, y)$ is a random perturbation function defined as a convolution of a white noise with a 2D filter kernel $h(x, y)$ which is determined somehow based on $S(f_x, f_y)$. Figure 3.5 (middle) shows one of their results.

The same authors later extended the above method in several ways, such as spectral analysis on multiple views [35] and combining spectral analysis and histogram matching [20]. We refer the reader to their survey [19] for more details.

Stereological Techniques: Jagnow et al. [43] proposed stereological techniques for solid textures, aiming specifically at generating volumetric materials consisting of aggregate particles and binding media, such as concrete, asphalt, and sponge. Assuming that the 3D particle shapes are known but their relative sizes (i.e., uniform scaling) are unknown, their goal is to estimate the unknown distribution of particle sizes in the 3D volume based on an input 2D cross-sectional image. This estimation is achieved based on theories in stereology, a field that studies the relationship between 3D structures and their 2D cross-sections.

In their algorithm, the output 3D volume and the input 2D image are both characterized by histograms of the sizes of particles contained within a unit space, denoted by $N_V(i)$ and $N_A(i)$, respectively, with i referring to an index of the histogram bin. Here, N_A is easily obtained by measuring the areas of particle cross-sections in the 2D image, and N_V is the unknown parameter of the underlying texture model that their algorithm seeks for.

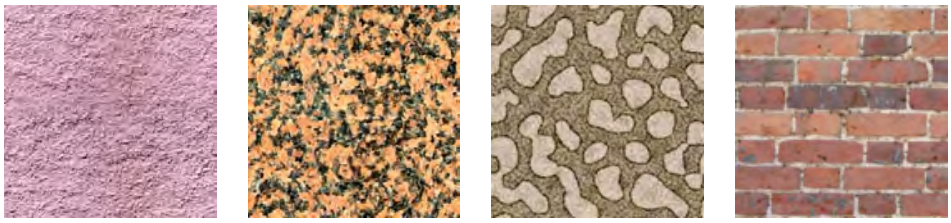

Figure 3.6: Examples of stochastic textures (left, middle left) and structured textures (middle right, right).

According to stereology, we can relate N_V to N_A as

$$N_A(i) = \sum_j k_{ij} N_V(j).$$

Intuitively, k_{ij} corresponds to the probability that a 3D particle with the j-th size in the 3D volume appears as a 2D particle profile with the i-th size in the 2D image. This matrix $K = (k_{ij})$ is specific to the individual particle shape which is known, and it is estimated using a Monte-Carlo approach where the 3D particle is cut at random positions and orientations and its cross-sectional area is measured for numerous times. Once K is obtained, N_V can be computed as $N_V = K^{-1} N_A$.

In the final synthesis step, the system iteratively generates each individual 3D particle by first choosing its size according to N_V, and then randomly choosing its position and orientation. Collisions among particles are resolved using simulated annealing. Colors and texture details inside the particles and the binding medium are generated using Heeger and Bergen's technique [37]. Figure 3.5 (right) shows one of their results.

Nonparametric Solid Texture Synthesis

An inherent limitation of parametric methods is that they are successful only for a limited range of materials. In particular, approaches that rely on general image statistics [37, 34] are only successful with *stochastic* textures (i.e., textures with small-scale noise-like details) but not with *structured* textures (i.e., textures with macro-scale features). Examples of these two classes of textures are shown in Figure 3.6.

To deal with more general types of solid textures including structured textures, researchers started to explore nonparametric methods later. In a nonparametric approach, the system does not assume any underlying global texture models, and synthesizes textures based on the information about local texture neighborhoods.

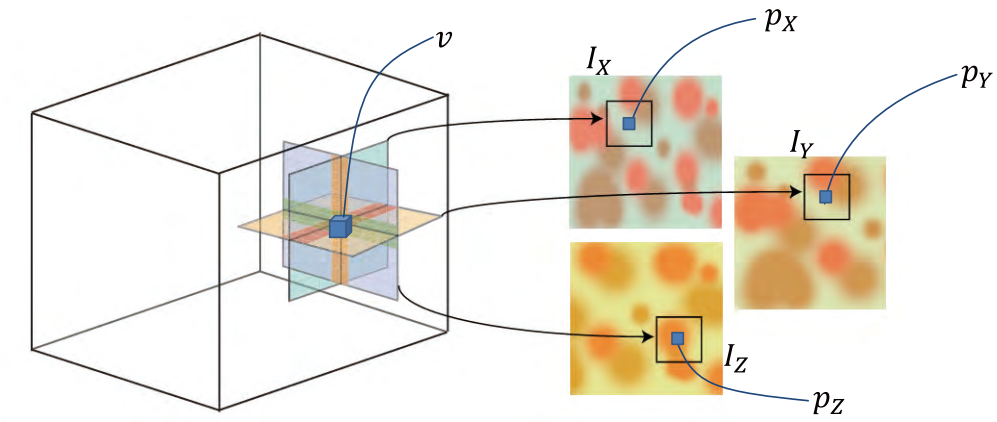

Figure 3.7: Notations used in nonparametric solid texture synthesis using local neighborhood matching [112, 53].

Local Neighborhood Matching: In 2003, Wei [112] proposed for the first time a nonparametric solid texture synthesis technique based on local neighborhood matching. The basic idea is to formulate the solid texture synthesis as an energy minimization problem. Given as input a set of 2D texture exemplars I_X, I_Y, and I_Z which specify the desired appearance of the cross-sections of the output volume perpendicular to the X-, Y-, and Z-axes, respectively, the goal is to find a solid texture V that minimizes the following energy:

$$E(V) = \sum_{v \in V} \sum_{D \in \{X,Y,Z\}} \min_{p_D \in I_D} \|N_D(v, V) - N(p_D, I_D)\|^2$$

where $v \in V$ is a voxel of V, $p_D \in I_D$ is a pixel of I_D, $N_D(v, V)$ is a neighborhood vector formed by concatenating all the voxel values contained in an n by n square slice of V perpendicular to the D-axis and centered at v (with n being a user-exposed parameter), and $N(p_D, I_D)$ is a neighborhood vector formed similarly but from I_D. Figure 3.7 illustrates these quantities visually. Note that this energy definition involves searching of the pixel p_D with the nearest neighborhood for each voxel v.

To minimize this energy, Wei proposed an iterative procedure consisting of two interleaved phases: search and update. In the search phase, the above search is simply carried out based on the current synthesis result, which is the most computationally expensive part of the entire procedure. In the update phase, the energy is decreased by replacing the color of each voxel v with the average color of p_X, p_Y, and p_Z. Although the above idea is simple and seemingly promising, unfortunately, Wei could only obtain some interesting but not satisfactory results (Figure 3.8 left).

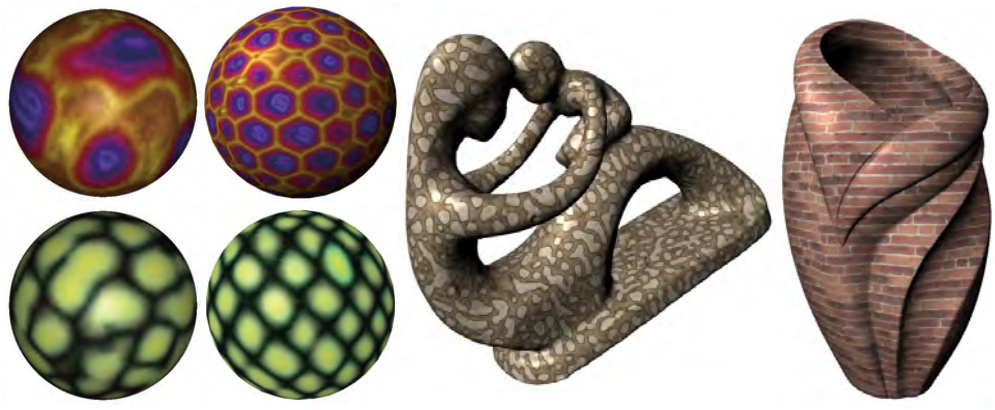

Figure 3.8: Comparison of solid texture synthesis algorithms by Wei (left) and Kopf et al. (middle left), and two examples of applying Kopf et al.'s resulting solid textures to 3D models (middle right, right).

In 2007, Kopf et al. [53] reformulated the above idea using the texture optimization framework [54] where the update phase is modified as weighted averaging of all overlapping neighborhood pixel colors, and introduced a scheme of adjusting these weights such that the color histogram of the synthesis volume becomes similar to that of the exemplars, which is inspired by Heeger and Bergen's technique [37]. These modifications made it possible to generate high quality solid textures of various types (Figure 3.8).

While producing stunning results, Kopf et al.'s technique takes a long time to compute and also requires a huge amount of storage especially when synthesizing high resolution solid textures. To address these problems, Dong et al. [21] proposed a significantly faster solid texture synthesis technique by reducing the search space and employing the GPU's parallelism, which is inspired by a previous fast 2D texture synthesis technique [59]. One notable advantage of Dong et al.'s technique is that it enables on-the-fly evaluation, an ability to query a synthesized color value at an arbitrary 3D position without synthesizing the entire color volume. This means that we only need to maintain compact information needed for the synthesis, and we do not need to store the entire color volume to texture a given 3D geometry. Note that this advantage of data compactness has previously been exclusively attributed to the procedural definitions of volumes as in Section 3.1.

Aura Matrices: Slightly before Kopf et al.'s work, Qin and Yang [87] presented another different technique for solid texture synthesis using Basic Gray Level Aura

Matrices (BGLAMs). Their technique assumes that the color channels of the input image can be decorrelated, and it works separately on each grayscale image as follows.

BGLAMs of a 2D image $I(x, y)$ are a set of matrices $\left\{A^{(\mathbf{d})} = \left(a_{ij}^{(\mathbf{d})}\right)\right\}$ where $\mathbf{d} \in \{-n, \cdots, -1, 0, 1, \cdots, n\}^2 \setminus \{(0, 0)\}$ is a neighborhood displacement vector (with n being a user-exposed parameter), and each $a_{ij}^{(\mathbf{d})}$ is defined as the number of pixel pairs $\mathbf{p}$ and $\mathbf{q} = \mathbf{p} + \mathbf{d}$ in the image that satisfies $I(\mathbf{p}) = i$ and $I(\mathbf{q}) = j$. The matrix $A^{(\mathbf{d})}$ is thus m by m when the range of the grayscale value in the image is discretized into m levels, which is why it is necessary to work with each color channel separately.

An important property of BGLAMs is that two sets of BGLAMs computed from two images are identical if and only if the two images are identical. Using BGLAMs as a way to measure the similarity between two images, the algorithm synthesizes solid textures in an iterative fashion, where the grayscale value of each voxel is updated such that the BGLAMs computed on the volume slice get closer to the BGLAMs of the input 2D cross-sectional images.

The algorithm often produces color bleeding artifacts because the assumption that the color channels can be decorrelated does not always hold. In addition, it often fails to deal with structured textures where Kopf et al.'s method performs much better.

Manual Painting Interface for Volumes

Owada et al. [74] presented a sketch-based interface for directly painting colors onto the voxel grid. Their goal is to develop an easy-to-use interface for creating volumetric contents from scratch. The basic idea is to create 3D structures and textures only by 2D sketching on cross-sections of the model. Given a surface mesh representing the overall shape of the model as input, the modeling process consists of two steps: decomposition of the interior regions and distributing particles within each region.

To decompose regions, the user draws a 2D stroke on the cross-section to specify the shape of the region boundary. As 2D stroke information is generally not enough for determining 3D shape, some additional information or assumptions are needed to obtain 3D shape from a 2D stroke. To deal with this issue, they performed a user study on the human perception about the relationship between 2D profile curves on cross-sections and their corresponding 3D shapes, and developed a set of heuristics for determining 3D shapes from 2D strokes based on the study result (Figure 3.9).

The next step of distributing particles is achieved in an example-based manner. The user first manually puts a few particles on the cross-section as examples. The system then infers information about how these example particles are arranged (i.e.,

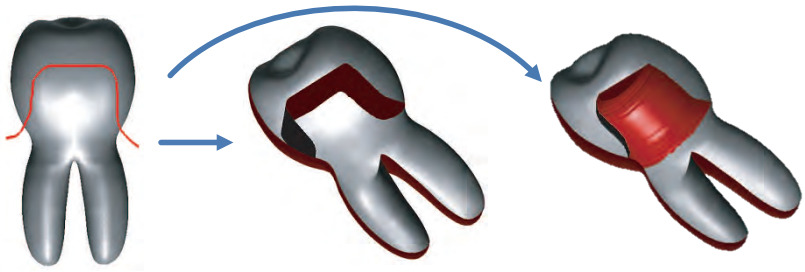

Figure 3.9: Two of the heuristics for generating a 3D surface from a 2D sketch (left) on a cross-section used in Owada et al.'s volume painting interface [74]: sweep-based (middle) and extrusion-based (right).

Figure 3.10: Examples of distributing particles according to example particle distributions given by the user.

density, orientation, and distance from the boundaries), and synthesizes new particles in the 3D space accordingly (Figure 3.10). Figure 3.11 shows some of the volumetric models created using their volume painting interface.

Discussion

While some of the aforementioned solid texture synthesis methods can synthesize homogeneous anisotropic solid textures corresponding to texture types of 0-A, 0-B, and 0-C in our classification of solid textures, orientation and scale of the texture are assumed to be uniform over the volume. In addition, since these methods synthe-

Figure 3.11: Some of the results in Owada et al.'s volume painting approach [74].

size individual voxels over the entire volume (except for Dong et al.'s method [21]), computational cost becomes prohibitively large when synthesizing a high resolution solid texture. In contrast, our raster-based method presented in Chapter 5 can handle spatially varying texture orientation and scale. Besides, our method requires low computational cost even when synthesizing a high resolution solid texture, since our method is can reuse the same texture data over and over again.

Owada et al.'s manual painting interface for volumes [74] is limited in expressiveness, as the system only supports constant colored particles and region fills. We believe that smooth volumetric color transitions are important when creating interesting volumetric data manually, and our vector-based method presented in Chapter 6 is better suited for from-scratch creation of volumetric data.

3.3.2 Vector-Based Approach

It is only recently that the vector-based approach to volumetric modeling started to receive much attention, and so far there exist three methods in this approach to our knowledge. They are all based on dividing the object's interior into separate regions and specifying volumetric color distribution inside each region in several ways such as using predefined solid textures, procedural descriptions, or radial basis functions. One method can define only layered regions by using distances from input 3D surfaces, while the other two methods can define more general shapes of regions by using signed distance functions (SDFs) that divide 3D spaces into two by their signs. Note that these vector-based methods can model global and arbitrary structures, and thus can be regarded as modeling textures of type 3 in our classification of solid textures.

Layer-Based Procedural Authoring

Cutler et al. presented the first method of the vector-based approach to volumetric modeling in 2002 [15] (although not described as such at that time). Their basic idea is to assume that the regions are arranged in a layered manner, and to define such layers based on some existing 3D surfaces. Specifically, the user can define layers with desired thicknesses around some existing 3D surfaces and assign a material to each layer. The user can procedurally define both the layer shapes and their materials using a scripting language. Listing 3.1 and Figure 3.12 show a simple example script code and its resulting model, respectively. The user can define more complicated layer shapes by using advanced features in the language such as combining different shapes and perturbing layer thicknesses.

```
STRIPED_CANDY = volume {
  distance_field = surface_mesh {
    file = candy.obj }
  layers = {
    interior_layer {
      material = CHOCOLATE
      thickness = fill }
    exterior_layer {
      material = WHITE_CHOCOLATE
      thickness = 0.10 }
    exterior_layer {
      material = STRIPED_CHOCOLATE
      thickness = 0.05 } } }
```

Listing 3.1: An example script code describing a simple chocolate candy model shown in Figure 3.12. Definitions for materials are omitted.

Figure 3.12: A volumetric chocolate candy model resulting from the script code in Listing 3.1.

Since their main goal is to use the resulting volumetric models with physics simulations, they also propose an algorithm for tessellating the regions into tetrahedral meshes and techniques to integrate simulation operators into their scripting language.

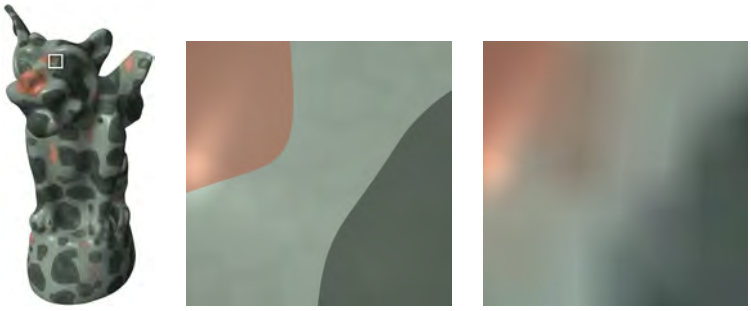

Figure 3.13: One of Wang et al.'s results of volume vectorization [111]. When extremely zoomed in, the vector solid texture representation still retains the sharpness of region boundaries (middle), whereas the bitmap solid texture representation exhibits severe blurring artifacts (right).

Volume Vectorization

Raster-based solid texture synthesis methods described in Section 3.3.1 generate bitmap solid textures as their final output. One fundamental problem with bitmap solid textures is that distinct visual features (e.g., profile of pebbles, boundaries between brick and mortar, etc.) cannot remain sharp when zoomed in, resulting in blurry artifacts. Simply increasing the texture resolution to alleviate this problem would obviously require a prohibitively large amount of memory.

Volume vectorization [111] proposed by Wang et al. specifically tackles this problem; it converts an input bitmap solid texture into a vector solid texture that retains sharp features when zoomed in while requiring compact storage. The basic idea is to represent the feature boundaries as the isosurfaces of an SDF, and approximate the color variation within each closed regions using radial basis functions whose parameters are determined by nonlinear optimization. Figure 3.13 shows one of their vectorization results.

Because their representation handles information about geometry and color separately, it is also possible to do some interesting editing operations that are difficult for bitmap solid textures. For example, they successfully generate self-similar fractal-like boundaries by evaluating the SDF recursively (Figure 3.14).

Multiscale Modeling

An obvious limitation of Cutler et al.'s approach [15] is that it assumes all the regions are arranged in a layered manner, which is not always the case for general objects.

Figure 3.14: Examples of vector solid textures with fractal-like boundaries achieved by evaluating SDFs recursively.

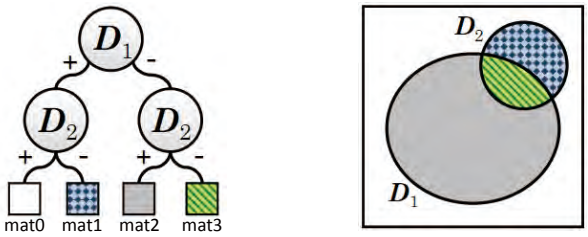

Figure 3.15: A visualization of SDF tree data structure specified in Listing 3.2 (left) and its graphical interpretation in 2D (right). Both of two disc-shaped SDFs `D1` and `D2` take positive (negative) values outside (inside) the disk.

Wang et al. [110] recently presented a new approach to authoring volumetric models that removes this restriction and enables the complexity that has been almost impossible previously. They address two problems specifically: (1) how to define separate regions arranged in arbitrary and complicated ways (not necessarily layered) in a concise manner, and (2) how to represent multi-scale structures in the objects (e.g., human skin consisting of hair, fat cells, and cell nucleus) compactly and efficiently.

To deal with the first problem, they propose a data structure called SDF tree which can be considered as a generalization of binary space partitioning tree. In an SDF tree, each intermediate node is associated with a certain spatial region (e.g., the root node is associated with the entire 3D space) and an SDF which divides the region into two disjoint sub-regions based on its sign. Thus, each intermediate node has two child nodes, corresponding to the positive and negative regions of the SDF. Each child node can be set as an intermediate node further dividing its space, or as a leaf node by assigning a specific material type that fills the space. A unique advantage of SDF tree is that it can define multiple disjoint regions simultaneously and concisely. Listing 3.2 shows a simple example code of their XML-based scripting language defining an SDF tree, and Figure 3.15 shows its corresponding graphical interpretation in 2D.

```
<OBJECT name="example">
  <SDF name="D1" file="D1.sdf">
    <POSITIVE>
      <SDF name="D2" file="D2.sdf">
        <POSITIVE region="mat0"/>
        <NEGATIVE region="mat1"/>
      </SDF>
    </POSITIVE>
    <NEGATIVE>
      <SDF name="D2" file="D2.sdf">
        <POSITIVE region="mat2"/>
        <NEGATIVE region="mat3"/>
      </SDF>
    </NEGATIVE>
  </SDF>
</OBJECT>
```

Listing 3.2: An example script code describing an SDF tree. SDFs are stored in a coarse raster image separately as `*.sdf` files.

To represent multi-scale structures, they propose to link multiple SDF trees representing components in different scales. Specifically, an instance of an SDF tree representing a smaller scale component, with some affine transformations, can be embedded into a leaf node of an SDF tree representing a larger scale component. This allows us to represent multi-scale components very compactly because the same data can be reused many times. This object instancing can be specified either manually using a simple GUI or procedurally using algorithms such as Poisson disk sampling with some target density maps. Figure 3.16 shows one of their results. Such complex volumetric models containing components in multiple scales would have required a huge amount of storage to model previously.

Discussion

While all of the above methods partition the object's interior into separate regions, we believe that not all objects can be clearly partitioned into separate regions, which is especially the case for objects with sharp and soft color transitions such as kiwi fruit,

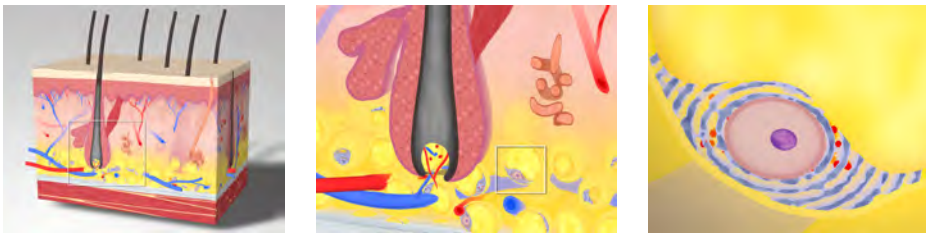

Figure 3.16: A multiscale volumetric model of human skin created using Wang et al.'s approach [110].

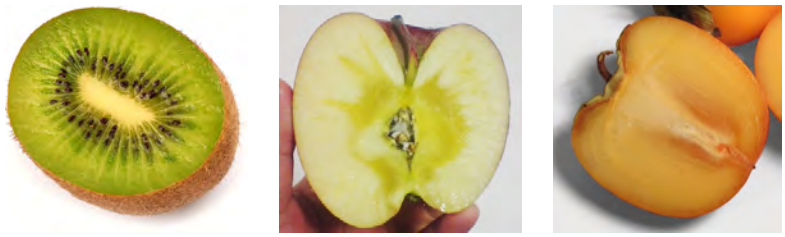

Figure 3.17: Examples of objects that cannot necessarily be partitioned into separate regions.

apple, and persimmon (Figure 3.17). In contrast, our vector-based method presented in Chapter 6 does not require partitioning the object's interior into separate regions, and allows open-ended boundaries to be placed at arbitrary locations inside the object. In addition, our method allows the user to specify the volumetric color distribution inside a model by painting colors on 3D surfaces, which we believe is more intuitive than placing RBF kernels in 3D space as in Wang et al.'s method [110].

3.4 Other Topics Related to Volumetric Modeling

In this section, we briefly mention other topics that are closely related to volumetric modeling but with different goals and contexts.

3.4.1 Realistic Rendering of Translucent Materials

Most of the works mentioned above use simple methods for rendering (e.g., Lambertian reflectance, volume ray casting, etc.) while focusing more on how to define the volumetric color distributions. On the other hand, there have been numerous works in the long history of computer graphics that focus on volumetric light interactions within participating media to achieve photorealistic rendering of natural materials and phenomena, such as cloud [50], fur [49, 80], stones [24], meso-structures [11, 104], het-

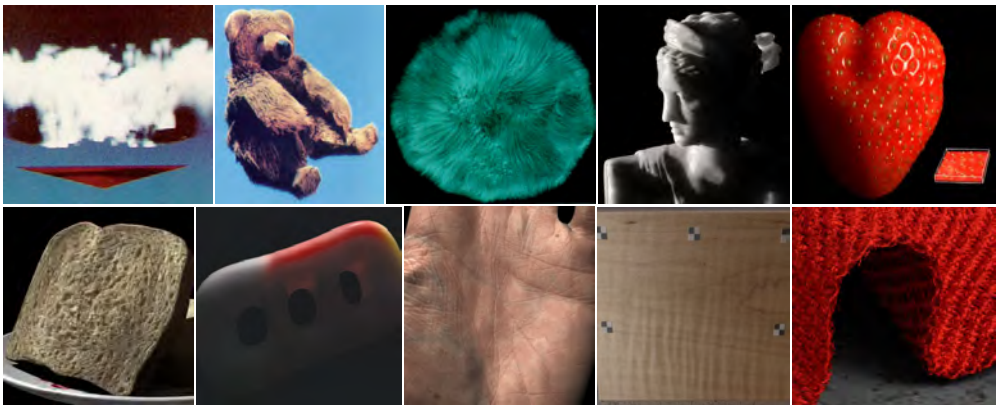

Figure 3.18: Some results of realistic rendering of translucent materials: (from top left to bottom right) Kajiya's early work on raytracing volume densities [50], fur rendering by Kajiya and Kay [49], hypertexture [80], weathered stones [24], shell texture functions [11], quasi-homogeneous materials [104], heterogeneous translucent materials [109], skin [23], anisotropic materials [44], and cloths [121].

erogeneous translucent materials [109], skin [45, 22, 23], anisotropic materials [44], and cloths [121], among many others (Figure 3.18).

Along with equations and algorithms to solve them, these works also develop methods for generating the actual volumetric dataset to render, including procedural definition, use of fluid simulation, measurements using photographs or CT images, and manual creation by artists. This part has a certain overlap with the field of volumetric modeling we have discussed so far; those volumetric modeling methods can be useful for the purpose of realistic rendering of participating media. Although these two fields have been separate possibly because of the difference in their goals, we expect that they will come close to each other in the near future.

3.4.2 Volume Visualization

There are numerous works that focus on visualizing CT or MRI volume data in the context of scientific visualization in various fields such as medicine, biology, and archaeology. They mainly focus on the purposes of visual analysis, explanation, and education, and typical approaches include illustrative rendering [8, 63] and interactive browsing using deformation [65, 13], among many others (Figure 3.19).

The main difference from the volumetric modeling techniques we have discussed in the previous sections is that they start with a given volume dataset. Also, they

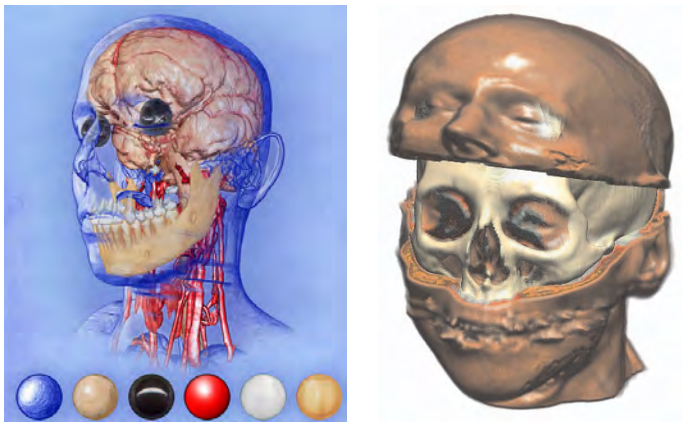

Figure 3.19: Examples of typical volume visualization approaches: illustrative rendering [8] (left) and use of deformation [13] (right).

mostly aim at non-photorealistic, illustrative, and explanatory depictions rather than realistic or plausible appearances.

Note that it would be useful to apply such visualization techniques to the results of the volumetric modeling techniques for further enhancing the browsing experiences.

Chapter 4

User Interfaces for Modeling Volumetric Orientation Fields

Manually creating a detailed volumetric color distribution from scratch is hard. Instead, we start the modeling process by first creating volumetric orientation fields that describe overall distributions of orientation of anisotropic structures of objects. This is because many natural objects, such as tree stumps, carrots, and kiwi fruits (Figure 4.1), have internal structures that can be seen as some common anisotropic structures repeating over and over throughout the object interior while changing its orientation.

Definition of "3D orientation" can differ depending on the application needs; it can either be a simple 3D vector, a frame (i.e., a set of three orthogonal 3D vectors), a symmetric tensor (i.e., 3×3 matrix) whose eigenvectors represent three orthogonal non-orientable directions, or a N-rotational symmetry (N-RoSy) which further generalizes the tensor form to represent a set of non-orthogonal non-orientable directions. In this chapter, we present two methods for modeling volumetric orientation fields, one for

Figure 4.1: Examples of natural volumetric objects whose internal structures can be seen as the repetition of some common anisotropic patterns along an overall orientation field.

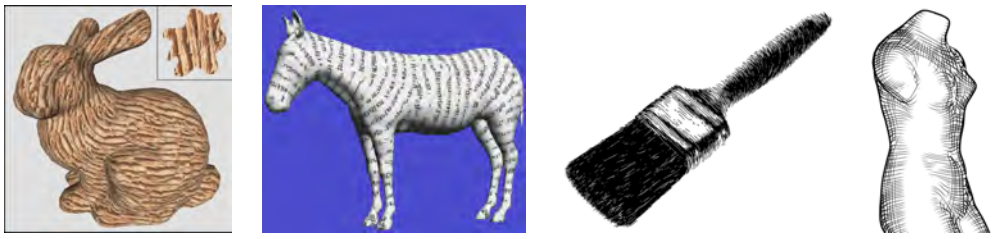

Figure 4.2: Early works that demonstrated the importance of creating orientation fields. (From left to right) Anisotropic texture synthesis on surfaces by Praun et al. [85] and Turk [106], pen-and-ink illustration of 2D images by Salisbury et al. [92] and of 3D smooth surfaces by Hertzmann and Zorin [38].

vector fields and the other for frame fields.

In general, specifying a 3D orientation in 3D space is perceptually much harder than specifying a 2D orientation in 2D space, and therefore modeling volumetric orientation fields is much more difficult than modeling orientation fields on a 2D plane or on a 3D manifold surface. Designing an appropriate user interface is thus crucial for enabling efficient modeling of volumetric orientation fields. In accordance with our **Principle I** for volumetric modeling (i.e., "*propagating user-specified information on the surfaces through the volume*"), we decompose the modeling process into simpler steps by exploiting some domain knowledge which reduces the excessive freedom in specifying 3D orientation.

Methods described in this chapter are used in the next chapter to create detailed volumetric color distributions by synthesizing solid textures along the created volumetric orientation fields. While our primary focus is on creating volumetric color distribution of objects, volumetric orientation fields can also be used for variety of other applications. Here we demonstrate two applications that utilize volumetric orientation fields created using our methods: electrophysiological simulation of heart ventricles and active deformation of unarticulated objects.

4.1 Related Work

The importance of creating orientation fields was first recognized by applications that use such orientation fields, such as synthesis of anisotropic textures on surfaces [85, 106] and pen-and-ink illustration of images [92] and smooth surfaces [38] (Figure 4.2).

In these early works, orientation fields were created using relatively naive approaches, such as simple Gaussian radial basis functions interpolation [85], low-pass

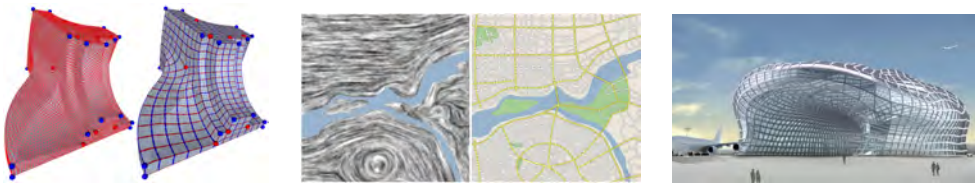

Figure 4.3: New applications emerging from recent orientation field modeling methods: (from left to right) mixed-integer quadrangulation [6], procedural city design [10], and planar quad mesh design useful for architectural glass structures [61].

filtering [106], manual painting of orientations [92], and smoothing of surface principal curvature tensors [38]. Since then, many methods for modeling orientation field have been proposed to improve these early methods. New representations more general than simple vector field [118, 31] have been proposed, such as symmetric tensor field [118], N-RoSy field [77, 89, 88, 14], and conjugate direction field [61]. Typical advantages of these new methods include more control on the field topology [118, 89, 88], the intrinsic rotation-invariant property [31], and a simpler formulation for interpolation [14]. Several new applications emerge from these new methods for modeling orientation fields, such as quadrangulation [6], procedural city design [10], and planar quad mesh design [61] (Figure 4.3). Note, however, that the above techniques all deal with orientation fields on a 2D plane or on a 3D manifold surfaces, instead of volumetric orientation fields.

Modeling of volumetric orientation field is a much less explored topic. The need for modeling volumetric orientation field was recognized for the first time when Owada et al. proposed the volumetric illustration system [76]. In their system, the user creates a volumetric vector field by placing several arrows on a cross-section of the 3D model, which are interpolated in 3D space using radial basis functions. The created volumetric vector field is then used to synthesize oriented 2D textures on cross-sections of the 3D model (Figure 4.4).

Concurrent to our work, Fu et al. [33] proposed a sketch-based hairstyle design system that uses a volumetric vector field to guide the growth of hair strands from the scalp. They propose a method for modeling volumetric vector fields that is specialized for hairstyle design. Specifically, their interface consists of three different modes of sketching curves: stream curve mode, dividing curve mode, and ponytail mode, each of which defines a certain set of constraints on the volumetric vector field (Figure 4.5 a and b). The system interpolates such constraints using Laplacian smoothing over a voxel grid to obtain a smooth volumetric vector field. They also propose simple schemes

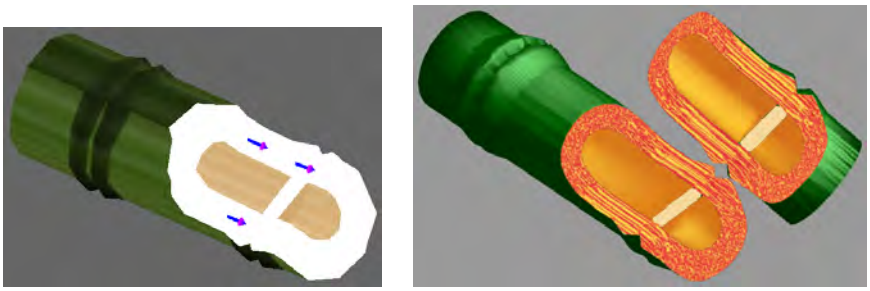

Figure 4.4: The volumetric illustration system by Owada et al. [76], in which a volumetric vector field is constructed from several user-specified 3D arrows (left) and 2D oriented textures are synthesized accordingly (right).

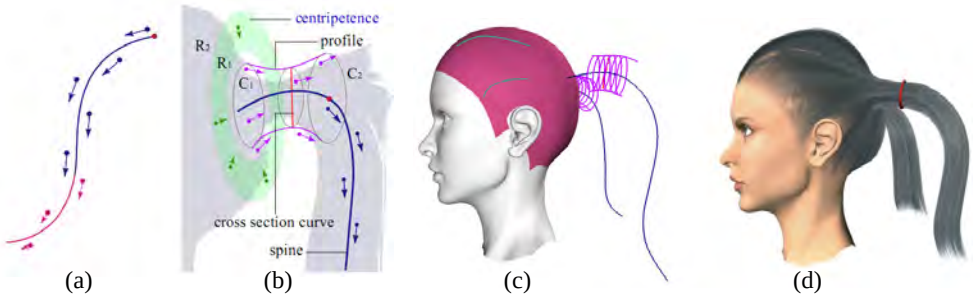

Figure 4.5: Sketch-based hairstyle design system by Fu et al. [33]. Constraints on the volumetric vector field defined by a stream curve (a) and a ponytail curve (b). 3D curves sketched by an artist (c) and the resulting hairstyle (d).

to circumvent the difficulty of making 3D curves by 2D sketching. For example, a curve point sketched on the scalp sticks to the scalp, a curve point crossing the scalp silhouette is assigned the same screen-space depth as the previous curve point, and the curve geometry can be modified by over-sketching on a supporting surface defined by sweeping the curve in a certain direction. One of their hairstyle design results is shown in Figure 4.5 d.

Later than our work, Zhang et al. [119] proposed a method for synthesizing anisotropic solid textures that follow user-defined volumetric frame fields. They develop a straightforward interface for modeling volumetric frame field; the user draws curves on a given 3D surface or on an arbitrary 3D plane, and each curve point defines a constraint frame formed by the curve tangent, the surface (or the plane) normal, and their cross-product (Figure 4.6 left). Internally, a frame is represented as a quaternion, and smooth volumetric frame fields are obtained by performing Laplacian smoothing

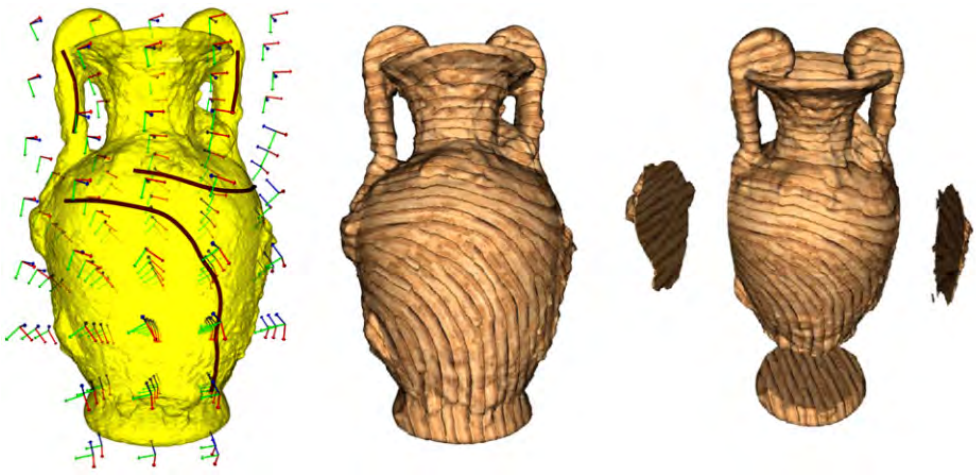

Figure 4.6: Sketch-guided solid texturing developed by Zhang et al. [119]: user-input 3D curves and an interpolated volumetric frame field (left), and the synthesis result (middle and right).

for the constrained quaternions. For the synthesis of solid textures, they use a modified version of Dong et al.'s algorithm [21]. Figure 4.6 middle and right show one of their results.

Recently, Huang et al. [39] propose a unique method for modeling volumetric orientation field. The primal application of their method is the generation of a hexahedral mesh that fills the interior volume of a given 3D surface while aligning as much as possible to the surface normals at its exterior boundaries. For such an application, a smooth 3D cross-frame field defined over a volume is useful for guiding the hexahedral meshing, much like a smooth 2D cross-frame field defined over a surface is useful for guiding the surface quadrangulation. A 3D cross-frame is different from a 3D frame (i.e., a set of three orthogonal vectors) in that it is non-orientable and the ordering of the three axes does not matter; i.e., if a 3D cross-frame is rotated about one of its axes by the multiple of $\frac{\pi}{2}$, it is considered equivalent to the original. Huang et al. propose a convenient way of representing such a 3D cross-frame using spherical harmonics, and successfully generate smooth 3D cross-frame fields that align well with the input boundary surface, subsequently generating well-shaped hexahedral meshes that fill the interior volumes (Figure 4.7). Note, however, that their main focus is on how to represent and interpolate 3D cross-frames, instead of intuitive user interfaces for manual control of orientations.

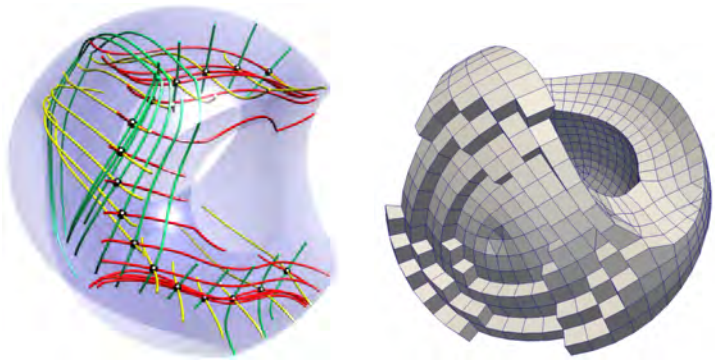

Figure 4.7: A streamline-based visualization of a volumetric 3D cross-frame field generated by Huang et al.'s algorithm [39] (left), and a volumetric hexahedral mesh sebsequently generated from the field (right).

4.2 Common Machinery: Laplacian Smoothing

Before describing our methods in detail, in this section we briefly review the basics of Laplacian smoothing, a common interpolation technique used for both of our methods for modeling volumetric orientation field described later. Laplacian smoothing and relevant gradient-domain processing techniques became popular after Yu et al. [117] and Sorkine et al. [96] proposed mesh deformation techniques based on Laplacian operator. A thesis by Nealen [68] gives a good introduction and overview of the field.

The fundamental advantage of Laplacian smoothing over other spatial interpolation techniques such as radial basis functions (RBF) [107] is that it can take the domain topology into account for interpolation. For example, given two points residing on a very concave surface that are nearby in Euclidean distance but actually far away in geodesic distance, simple spatial interpolation techniques such as RBF treat them as being close to each other, while Laplacian smoothing treats them as being distant.

Laplacian smoothing is an operation which can be defined on any kind of graph structure that has a set of vertices and connectivity information, such as triangular meshes, tetrahedral meshes, pixel grids, and voxel grids. Given the graph consisting of n vertices, the goal is to find a smoothly varying scalar field $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ over the graph that satisfies additionally specified linear constraints on $\boldsymbol{x}$ as much as possible.

To measure the smoothness, the Laplacian δ_i for each vertex i is defined as

$$\delta_i = x_i - \sum_{j \in N_i} w_{i,j} x_j,$$

where N_i is a set of vertices adjacent to the vertex i, and $w_{i,j}$ are weights that satisfy the partition of unity property (i.e., $\sum_{j \in N_i} w_{i,j} = 1$). The weights can be defined differently depending on the application requirements. In the simplest case, $w_{i,j}$ is set as $\frac{1}{|N_i|}$ which is often called the uniform Laplacian. For an irregularly sampled triangular mesh, the cotangent Laplacian [18] which accounts for the angles of the mesh triangles is a popular choice for achieving good approximations. In our method for modeling frame fields described later, we use a our specific definition for these weights. The entire smoothness energy over the graph can be expressed as the magnitude of the Laplacian vector $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_n)^T$, which can be written using the matrix notation as

$$E_s(\boldsymbol{x}) = \|\boldsymbol{\delta}\|^2 = \|L\boldsymbol{x}\|^2$$

where $L = (l_{i,j})$ is an $n \times n$ matrix (called Laplacian matrix) with

$$l_{i,j} = \begin{cases} 1 & (i = j) \\ -w_{i,j} & (j \in N_i) \\ 0 & (otherwise) \end{cases}.$$

Given m linear constraints whose k-th constraint is expressed as

$$\sum_{j=1}^{n} c_{k,j} x_j = b_k \quad (k = 1, \ldots, m),$$

the residual can be expressed as $C\boldsymbol{x} - \boldsymbol{b}$ where $C = (c_{k,j})$ and $\boldsymbol{b} = (b_1, \ldots, b_m)^T$. The quadratic energy for the linear constraints is thus given as

$$E_c(\boldsymbol{x}) = \|\Lambda(C\boldsymbol{x} - \boldsymbol{b})\|^2$$

where a diagonal matrix $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$ adjusts weights for individual constraints.

Our goal is to find

$$\arg\min_{\boldsymbol{x}} E_s(\boldsymbol{x}) + E_c(\boldsymbol{x})$$

which is a least squares problem whose solution has the following closed form:

$$\boldsymbol{x} = (L^T L + C^T \Lambda^2 C)^{-1} C^T \Lambda^2 \boldsymbol{b}.$$

Most of the time, the matrices L and C are both sparse, allowing us to use any existing sparse matrix solvers such as UMFPACK [16] which we always use. Note that it is often the case that the Laplacian matrix, once constructed from the original graph, remains the same regardless of the user interaction, thus it can be reused conveniently.

4.3 Modeling of Volumetric Vector Field

In this section, we present a method for modeling a volumetric vector field that fills the interior of a given boundary surface. In accordance with our **Principle I**, we make the modeling process simple and efficient by assuming that the volumetric vector field at the boundary surface is always parallel to the surface tangent. This assumption holds true for various examples such as a bunch of muscle fibers and a fluid flow contained within a bounded region. This assumption makes it possible to decompose the modeling process into two simpler steps: first sketching on the boundary surface to create a smooth tangent vector field which determines the overall flow of the volumetric vector field, and then sketching on arbitrary cross-sections of the volume to make detailed adjustments to the current volumetric vector field.

4.3.1 User Interface

The user first loads a closed manifold triangular mesh as a boundary surface. After the system completes some preprocessing which takes a few seconds, the user can start sketching on the surface to create a smooth tangent vector field. As the user draws a new stroke on the surface, the surface tangent vector field is updated instantly such that it locally aligns with the user-drawn strokes (Figure 4.8 left). The user can draw arbitrary number of strokes on the surface. After creating a satisfactory tangent vector field, the user presses a button to let the system compute the volumetric vector field based on the tangent vector field, which takes a few seconds. The user can cut the model at an arbitrary position by drawing a crossing stroke (Figure 4.8 middle) and browse the volumetric vector field sampled on the cross-section. To make further detailed adjustments, the user can draw arbitrary number of strokes on cross-sections. The user presses a button to let the system update the volumetric vector field according to the user-drawn strokes, which takes a few seconds (Figure 4.8 right). Note that the user can modify the tangent vector field at any time.

4.3.2 Algorithm

Figure 4.9 shows an overview of our algorithm for computing a volumetric vector field. A surface vector field is first constructed based on the user-drawn strokes on the surface (Figure 4.9a). Then, the surface vector field is transformed into a set of constraints on the boundary voxels (Figure 4.9b). Finally, the volumetric vector field is constructed based on these constraints and additional internal strokes drawn by the user (Figure 4.9c).

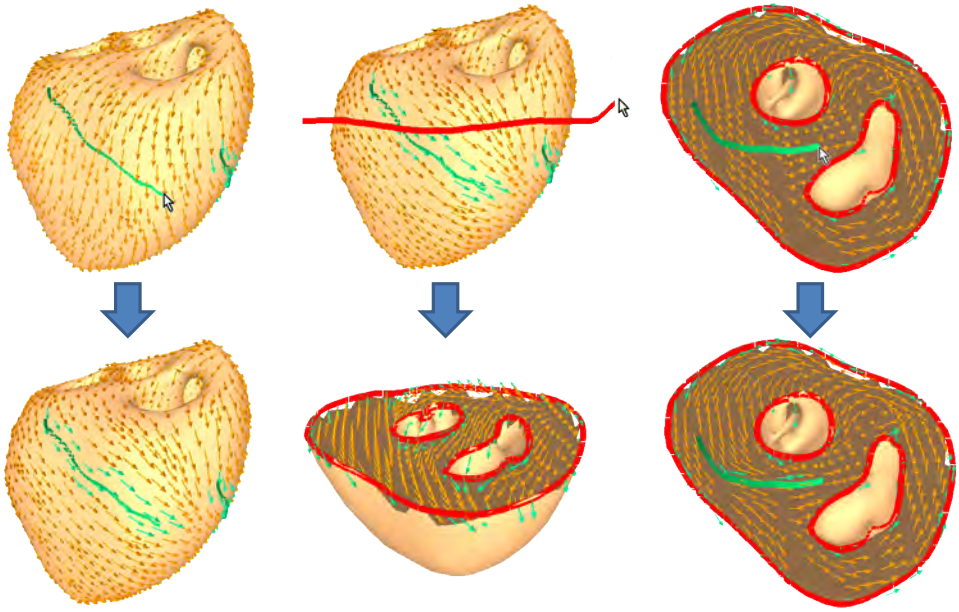

Figure 4.8: User interface for modeling volumetric vector fields. The user draws strokes on the surface to create the surface vector field (left), then cuts the model to see the volumetric vector field sampled on the cross-section (middle), and further draws strokes on the cross-section to modify the volumetric vector field at the interior (right).

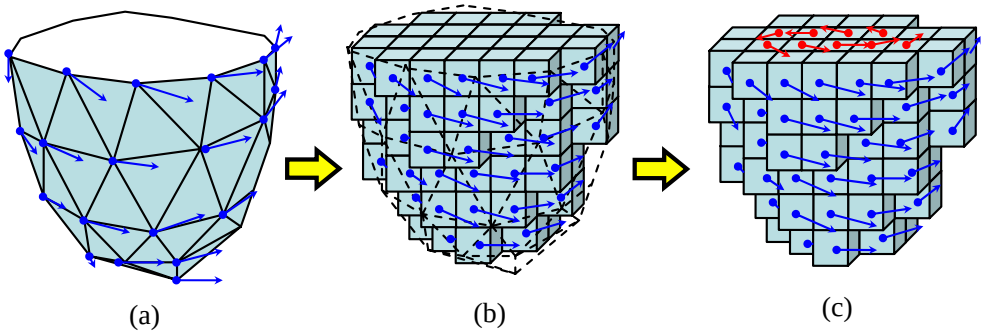

Figure 4.9: An overview of our two-step algorithm for computing a volumetric vector field. (a) The surface vector field is first constructed. (b) The surface vector field is transformed into a set of constraints for the boundary voxels. (c) The volumetric vector field is finally constructed based on these constraints.

The user first loads a triangular mesh as a boundary surface. The system then extracts a set of voxels inside the surface using a standard scanline-based voxelization technique. The surface vector field and the volumetric vector field are defined for the surface mesh and the voxel grid, respectively. The system next constructs two Laplacian matrices, one for the surface mesh and the other for the voxel grid, as a preprocessing. We use cotangent weights for the Laplacian matrix of the surface mesh, while we use uniform weights for the Laplacian matrix of the voxel grid.

The surface vector field is computed based on the user-drawn strokes on the surface. Every small segment of each stroke assigns the unit vector of its direction to its closest mesh vertex as a constraint (Figure 4.10 left). From these constraints, the smooth surface vector field is obtained by performing Laplacian smoothing on the surface mesh.

The volumetric vector field is computed based on the surface vector field and the additional strokes drawn on cross-sections. The surface vector field is transformed into constraints for the volumetric vector field as follows. For each boundary voxel (i.e., the most exterior voxel), the system knows information about the relationship between the voxel and the surface mesh (i.e., the closest triangle and corresponding barycentric coordinates), which is determined when voxelizing the surface mesh. Thus a linear interpolation of the surface vector field using that information is assigned to the boundary voxel as a constraint (Figure 4.10 middle). Strokes on cross-sections are transformed into constraints much like in the case of surface vector field; every small segment of each stroke assigns the unit vector of its direction to its closest voxel as a constraint (Figure 4.10 right). From these constraints, the smooth volumetric vector field is obtained by performing Laplacian smoothing on the voxel grid.

Note that each of the x, y, and z components of 3D vector is processed separately, and later combined. When computing the smooth surface vector field using Laplacian smoothing, it is not guaranteed that the resulting vector field is always tangent to the surface. Thus, the system projects each of the resulting vectors along its surface normal direction. Also, since we aim at modeling orientation fields, we chose not to care about the magnitude (or norm) of vectors. Therefore, we always perform normalization of vectors after each calculation as a post-process. Normalization of a vector is undefined if it is exactly a zero vector; we simply assign a random value when this rare case happens.

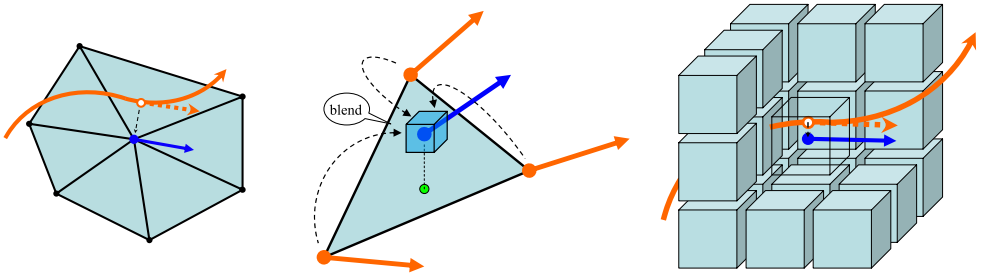

Figure 4.10: Constraints for the Laplacian smoothing. (Left) A stroke segment on the surface assigns a unit vector parallel to its direction to its closest surface mesh vertex. (Middle) A boundary voxel is assigned a linearly interpolated vector from its nearest surface mesh triangle. (Right) A stroke segment inside the volume assigns a unit vector parallel to its direction to its closest voxel.

4.4 Modeling of Volumetric Frame Field

In this section, we describe a method for modeling volumetric frame fields inside an object. In accordance with our **Principle I**, we make the modeling process efficient by assuming that one of the frame direction (which we call "primary direction") correspond to the gradient direction of a scalar "depth" field over the volume, and the other two directions of the frame (which we call secondary and tertiary directions) form a smooth 2D tangent frame field for each "depth layer" surface (i.e., iso-surface of the depth field). This assumption holds true particularly for myocardial fiber orientation of heart ventricles as reported by Nielsen [70], but also for other natural objects such as kiwi fruits, carrots, and tree stumps which have some notion of depths (i.e., innermost and outermost parts) and an additional "up" direction perpendicular to the depth direction (Figure 4.11). This assumption allows us to decompose the entire process of modeling volumetric frame field into two simpler steps: creating a scalar depth field over the volume, and specifying the secondary direction on each depth layer. This decomposition ensures that the secondary direction is always perpendicular to the primary (i.e., depth) direction, forming a valid frame.

4.4.1 User Interface

The user first loads a tetrahedral mesh in which the volumetric frame field is going to be modeled. Such a tetrahedral mesh can be constructed from a closed triangular mesh using common tools such as TetGen [94].

The user then starts the first step of creating a scalar depth field inside the model.

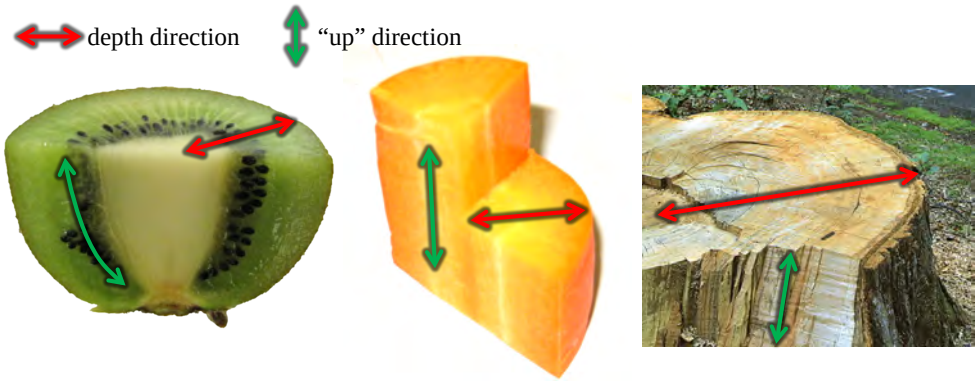

Figure 4.11: Examples of real-world objects which have the notion of depth and "up" directions.

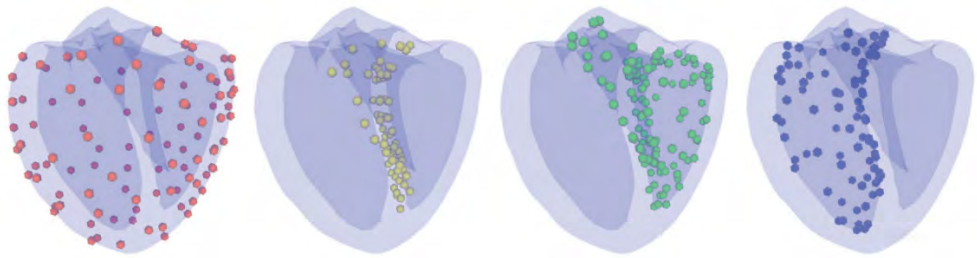

Figure 4.12: Points with constraint depth values placed by the user for a heart ventricle model.

The system provides a palette of predefined depth values (which are 0, 0.25, 0.5, 0.75, and 1 in our current implementation) from which the user chooses as a constraint depth value. The user places points with constraint depth values at several locations (Figure 4.12), and presses an "update" button. Then the system computes a smooth depth field interpolating those user constraints (Figure 4.13). Here the depth values are visualized using colors (i.e., the color becomes red, yellow, green, cyan, and blue as the depth changes from 0 to 1). The depth layers are constructed at the same time, and the user can browse different layers by rolling the mouse wheel. The user continues adding or removing colored points until obtaining satisfactory layer shapes.

After the depth field is created, the user moves on to the next step of specifying the secondary directions on the depth layers by sketching strokes (Figure 4.14a). When the user presses an "update" button, the system computes a smooth volumetric frame field. The secondary directions can be visualized using streamlines (Figure 4.14b).

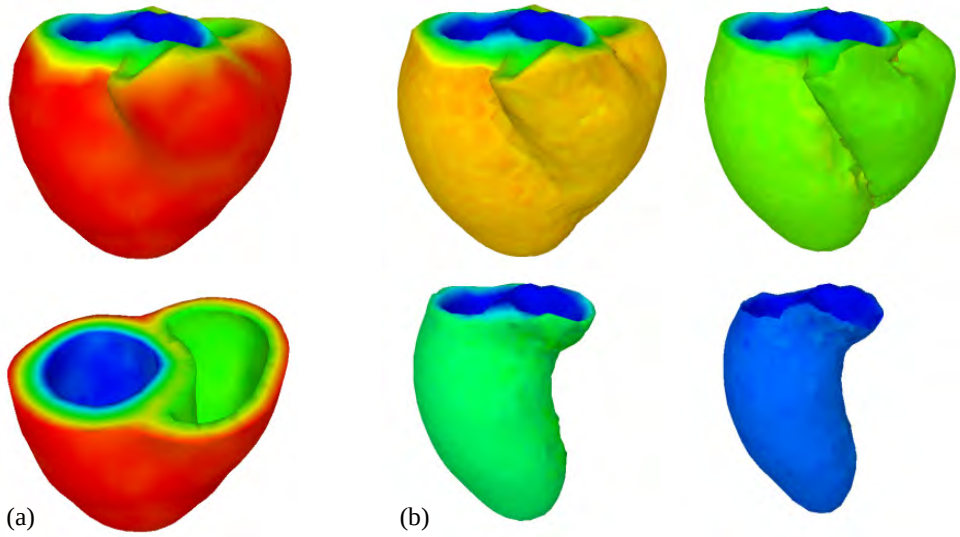

Figure 4.13: A smooth depth field interpolating the user constraints, visualized on a cross-section (a) or on layers (b).

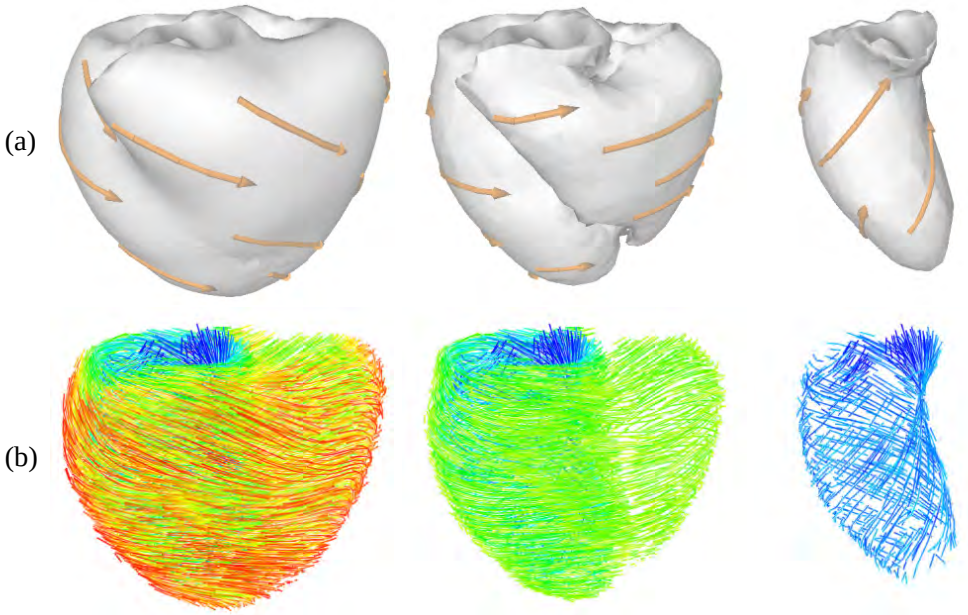

Figure 4.14: Secondary directions of the frame field. The user draws strokes on each depth layer (a), and the interpolated frame field is visualized using streamlines (b).

4.4.2 Algorithm

For computing smooth depth fields, we use RBF interpolation [107] instead of Laplacian smoothing, since we want to compute a smooth scalar field and its gradient. The depth layers are trivially extracted using the marching tetrahedra algorithm [105].

For computing the secondary directions, we use Laplacian smoothing on the tetrahedral mesh. In order to enforce more smoothness among vertices with similar depths, we alter the definition for the weights in the Laplacian operator as follows. For two adjacent vertices i and j, the weight for the Laplacian operator is defined as

$$w_{i,j} = \exp(-\lambda_R r_{i,j}^2 - \lambda_D d_{i,j}^2)$$

where $r_{i,j}$ and $d_{i,j}$ are the Euclidean distance and the depth difference between the two vertices, respectively, with λ_R and λ_D weighting the corresponding factors. We set λ_D relatively higher with respect to λ_R such that the continuity of the field on the same depth layer is enforced appropriately. (Note that $w_{i,j}$ is further normalized to satisfy the partition of unity property.) Similarly, a constraint value c specified at a 3D position $\boldsymbol{p}$ is incorporated into the linear constraint form as follows. Assuming $\boldsymbol{p}$ is within a tetrahedron T whose four vertices are denoted as i_1, i_2, i_3, and i_4, the linear constraint is expressed as

$$w_1 x_{i_1} + w_2 x_{i_2} + w_3 x_{i_3} + w_4 x_{i_4} = c$$

with $w_k = \exp(-\lambda_R r_k^2 - \lambda_D d_k^2)$ and r_k and d_k being the Euclidean distance and the depth difference between $\boldsymbol{p}$ and the vertex i_k, respectively. (Note that w_k is further normalized such that $\sum_{k=1}^{4} w_k = 1$.) Similar to the modeling of surface vector field described in the previous section, we process each of the x, y, and z components of 3D vector (i.e., secondary direction) separately. The Laplacian smoothing operation does not guarantee that the resulting secondary direction is always perpendicular to the primary direction. Therefore, the system removes components parallel to the primary direction from the secondary direction as a post-process. This way, we ensure that the secondary direction is always perpendicular to the primary direction. The tertiary direction is simply obtained as a cross-product of the primary and the secondary directions.

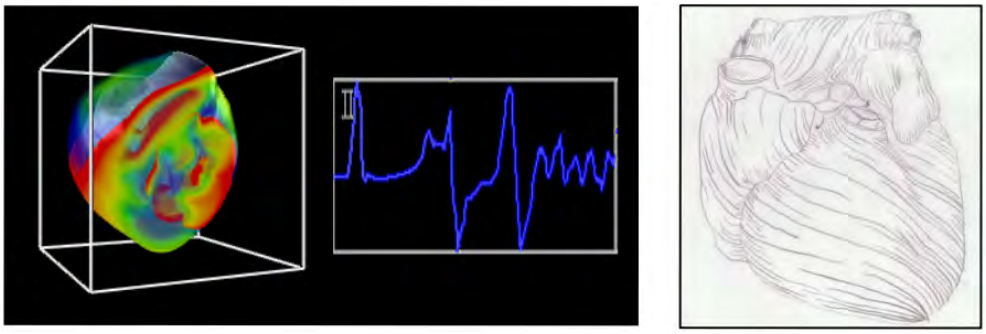

Figure 4.15: Electrophysiological simulation with a virtual heart model (left), and an illustration of myocardium of heart ventricles (right).

4.5 Application 1: Electrophysiological Simulation of Heart Ventricles

4.5.1 Background and Motivation

Many people suffer from abnormal heart rhythm, and effective treatment is much desired. One approach to understanding the mechanism of this disease is electrophysiological simulation of the heart [67] (Figure 4.15 left). Various parameters are required for this simulation, and orientation of myocardial fiber (i.e., muscle fiber of heart as shown in Figure 4.15 right) is one of the key elements that determines the behavior of signal propagation [1, 2], because the speed of excitation propagation is about three times faster in the longitudinal direction of myofibers than in the transverse direction.

In order to study the direct influence of the myocardial fiber orientation on the simulation result, it is necessary for physicians to manually create various models of fiber orientation based on their expert knowledge. Approaches employed previously include specifying 3D vectors on discrete 2D slices manually, or generating the volumetric vector field by hard cording, both of which are difficult and time-consuming.

In this section, we utilize our interfaces for modeling volumetric orientation fields to help this process. We asked Dr. Takashi Ashihara, a cardiologist at Shiga University of Medical Science, to create myocardial fiber orientations using both of our interfaces for modeling vector fields and frame fields described in Sections 4.3 and 4.4, respectively. We applied the resulting myocardial orientation fields to a simplified electrophysiological simulator [36]. Since the simulator takes as input a volumetric vector field representing the myocardial fiber orientations, for the case of using our interface for modeling frame fields, we treat the field of the secondary direction as the

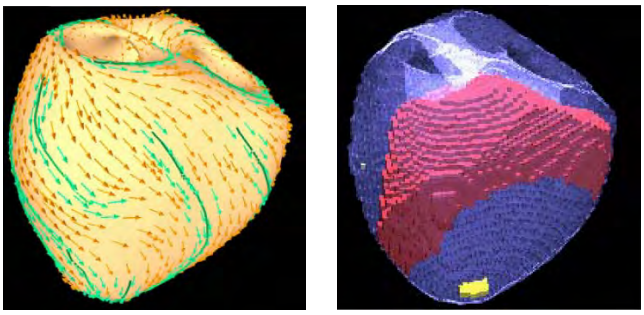

Figure 4.16: Myocardial fiber orientations created using our interface for modeling vector fields (left) and its corresponding electrophysiological simulation (right).

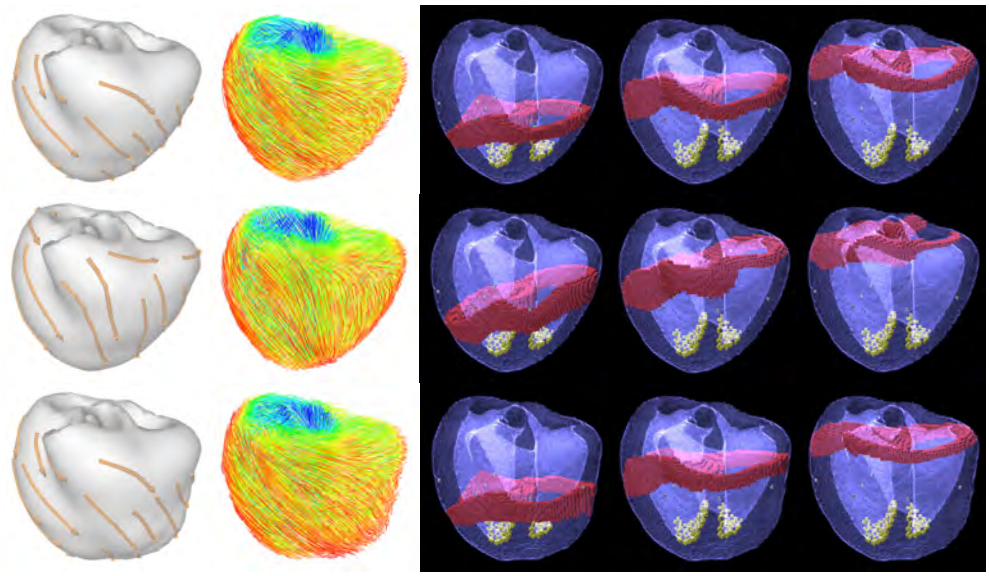

Figure 4.17: Myocardial fiber orientations created using our interface for modeling frame fields: (from left to right) user-drawn strokes, streamline-based visualization, and corresponding electrophysiological simulation.

vector field.

4.5.2 Results

Figures 4.16 and 4.17 show myocardial fiber orientations created by the cardiologist and corresponding simulation results. We can observe the influence of the difference of myocardial fiber orientations on the excitation propagation in the simulation results.

We then interviewed the cardiologist and obtained the following feedback. He evaluated our interfaces as important contributions to his area of research, because they

are the first ones that allow the user to directly create 3D fiber structures. Existing methods force the user to work on 2D slices or to rely on hard coding, which are both difficult and tedious. He was pleased with the myocardial fiber orientations he created using our interfaces, as they successfully represented the typical twisted structure of myocardial fibers. He preferred our interface for modeling frame fields to our interface for modeling vector fields, because an actual myocardial fiber consists of several layers parallel to the surface, and researchers usually associate myocardial fiber orientations with such layers and not with cross sections.

4.6 Application 2: Active Deformation of Unarticulated Objects

In this section, we present ProcDef, a procedural deformation framework for designing animations of unarticulated objects that are difficult to handle using existing skeleton- or keyframing-based methods. Since completely explaining all the technical details of the framework is beyond the scope of this thesis, we briefly explain the overview and how our interface for modeling volumetric orientation fields is useful for the framework. We refer the reader to the original article [42] for more details.

4.6.1 Background and Motivation

Most animation authoring methods are designed for rigged characters whereby the character body is divided into near-rigid parts such as arms and legs, and the user controls the joint angles between them. However, little work has dealt with the animation of flexible objects without specific skeletal structures such as jellyfish, snails, slugs, hearts, and stomachs. A typical approach to animating this type of object is to set a sequence of discrete key poses and interpolate them in time, but specifying individual key poses by manipulating many control points is a tedious work. Furthermore, making these objects respond to external forces such as contacts and collisions after all key poses are complete is very difficult.

We propose ProcDef to solve this problem. Our key observation is that the deformations of such flexible objects are driven by expansion and contraction of the local tissues. Local tissues receive some excitation signals and transform their shapes individually, and then the accumulation of the local deformations induces the global motion. For example, a heart is a muscular organ in which muscle fibers are aligned in a spiral direction. When beating, the heart muscles receive an electric signal arising

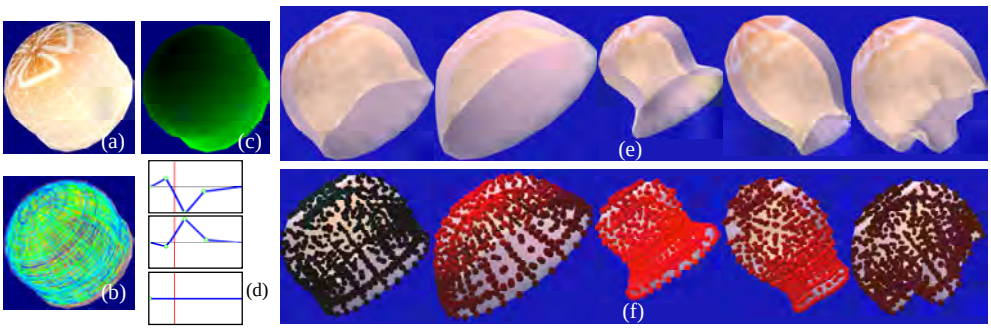

Figure 4.18: Overview of our ProcDef framework. Given an input 3D model (a), the user specifies parameters for local deformations, such as local orientation (b), phase-shift field (c), and deformation chart (d). The system computes global motion by accumulating local deformations (e). The propagation of deformation signals is visualized by highlighting the excited vertices in red (f).

in the sinoatrial node and locally contract along their fiber orientations, inducing a global twisting motion.

Based on this observation, we propose to create the global motion of a flexible object by controlling its local deformations (Figure 4.18). The user specifies local deformations such as contraction and expansion, and the system synthesizes the global motion by assembling the local deformations and taking the external forces into account. Stimuli-response deformations can be naturally handled within this framework.

An important parameter for controlling local deformations is a volumetric frame field that specifies the orientation of contraction and expansion for each local tissue, which is efficiently created using our interface for modeling frame fields described in Section 4.4. Other than the local orientation, there are several parameters for controlling deformations, such as one called "deformation charts" which define cyclic time-varying stretching and contraction of a local element in three directions (Figure 4.19).

4.6.2 Algorithm

Our algorithm for computing global motions induced by the accumulations of local deformations is based on shape matching dynamics [66, 91] which is unconditionally robust and computationally inexpensive, making ProcDef well suited for real-time interactive applications.

Figure 4.20 shows an overview of our deformation algorithm. Before the animation,

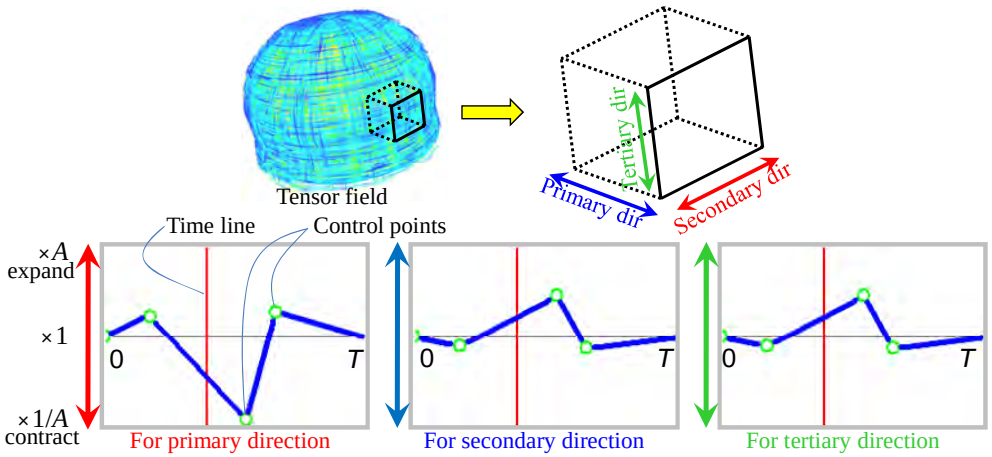

Figure 4.19: Deformation charts defining expansion and contraction rates. Top row shows an orientation field and a local region (cube). The three charts define the deformation rate in the primary, secondary, and tertiary directions of the orientation field.

we prepare a volumetric tetrahedral mesh and define a local region N_i around each mesh vertex i by connecting immediate (1-ring) neighborhood vertices. Neighboring local regions overlap each other (Figure 4.20a). In each animation step, we first deform each region N_i of the original mesh based on the user-specified deformation function $T_i(t)$, and then synthesize a new global shape that satisfies the deformed local regions as much as possible. For example, if we horizontally expand the upper regions and contract the lower regions (Figure 4.20b), the global shape bends downward (Figure 4.20c). If we contract the upper regions and expand the lower regions (Figure 4.20d), the global shape bends in the opposite direction (Figure 4.20e).

The local deformation function $T_i(t)$ varies depending on vertex i and time t. We define $T_i(t)$ as a linear transformation to simplify the problem. Note that even locally linear transformations can generate globally complicated nonlinear deformations. Since manually defining the individual $T_i(t)$ to design expressive motions is extremely difficult and time-consuming, our system provides an efficient scheme for defining $T_i(t)$ with a few global controls. We refer the reader to the original article [42] for more details.

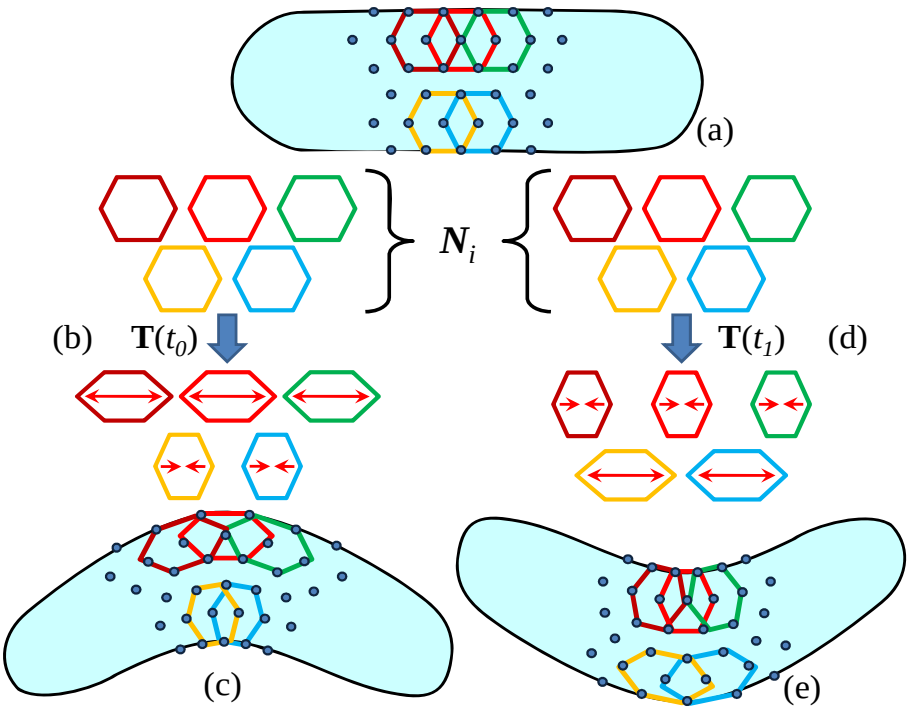

Figure 4.20: Overview of our deformation algorithm. We first deform individual local regions N_i (b)(d) of the original mesh (a) by the user-specified linear transformation $T(t)$ and synthesize the global shape that satisfies the deformed local shapes (c)(e).

4.6.3 Results

ProcDef supports general deformations that are induced by muscular tissues and it covers a large variety of possible motions. Figures 4.18e and 4.21 (left) show swimming jellyfish and crawling worms designed with ProcDef, respectively. The user can easily design these animations by setting the orientation field and other parameters using our interface. ProcDef can also easily handle stimuli response phenomena. In Figure 4.21 (right), a snail deforms its body when the user adds an external simulus to it. The deformation starts at the contact point and propagates through the entire volume.

Figure 4.22 shows scenes containing many moving objects. In these examples, the system computes both the deformations and collision avoidances between objects in real time. Note that these characters have not been frequently used in video games so far, because it was difficult and costly to design and compute their motions. We hope our method can make such currently unpopular characters to be heavily used in the future.

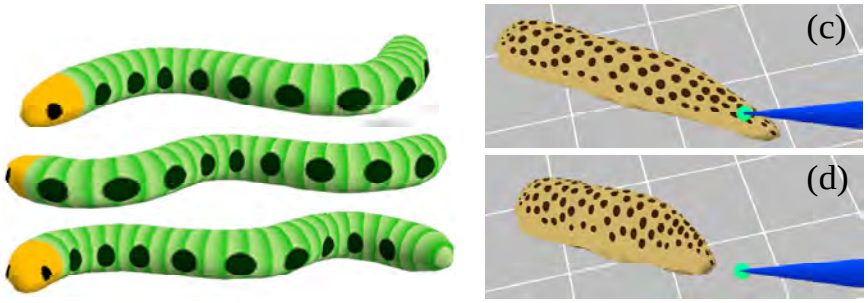

Figure 4.21: (Left) A crawling motion of a worm. (Right) A snail responding to a stimulus given by the user.

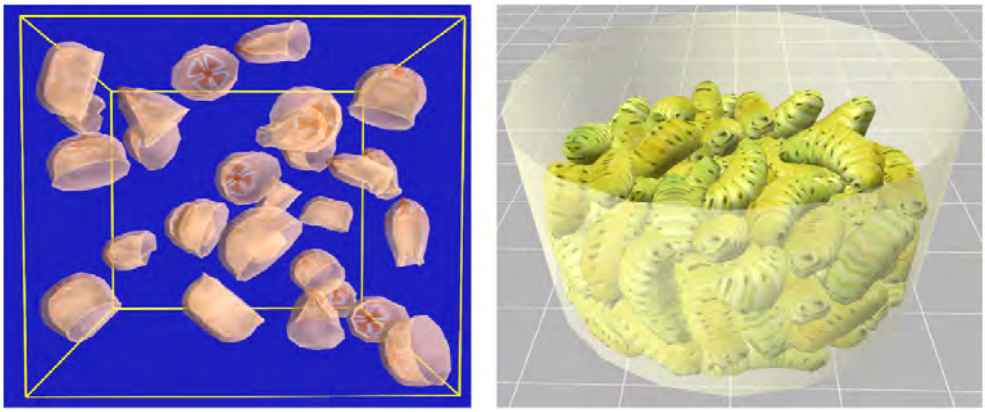

Figure 4.22: Many moving objects: 31 swimming jellyfishes (left) and 101 short worms (right).

Our system is also useful for animating organs (Figure 4.23). We use the same heart ventricle model and its volumetric frame field created in the previous section and set other parameters appropriately to generate highly realistic twisting motions of the heart. We also designed animations of a bowel. We defined a ring-shaped orientation field to generate peristaltic motions. Since these organ animations are computed in real time on a standard PC and the user can interact with the model, we believe ProcDef would be useful for medical applications such as surgery simulations or electronic charts.

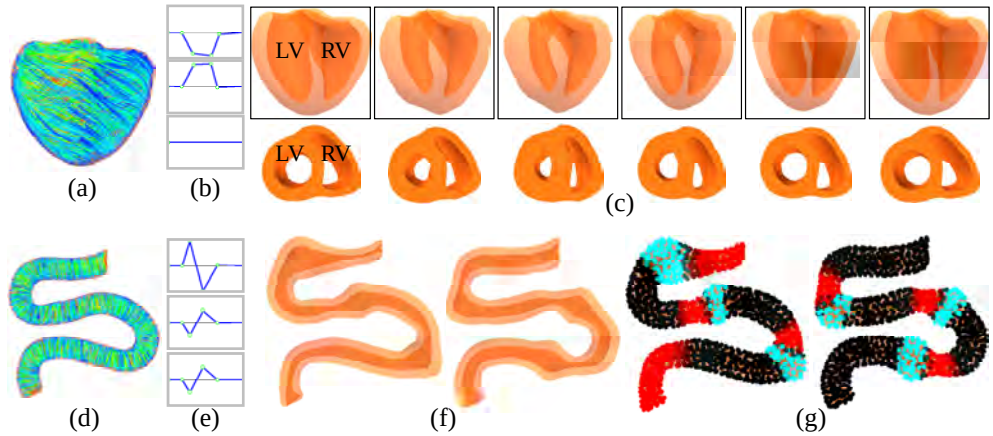

Figure 4.23: Animations of a heart and an S-shaped bowel. LV and RV indicate the left and right ventricles of the heart. We show the representative poses of the heart motion in the top row of (c). The bottom row of (c) is an overhead view of the sliced heart model. When beating, the left ventricle (LV) wall strongly thickens to reduce the size of the left ventricle (c).

Chapter 5

Raster-Based Method for Representing Detailed Internal Structures using Anisotropic Solid Textures

In this chapter, we present our raster-based method to create a detailed volumetric color distribution inside a 3D model by synthesizing anisotropic solid textures along a volumetric orientation field created using our interfaces for modeling volumetric orientation fields described in the previous chapter.

5.1 Overview

Our basic idea is to fill the model with overlapping patches of solid textures. This is inspired by the Lapped Textures approach proposed by Praun et al. [85], in which they synthesize anisotropic 2D textures on a 3D surface by covering the surface with overlapping patches of an exemplar 2D texture that align with a user-specified tangent orientation field (Figure 5.1 top). Our approach extends their approach from 2D surface textures to 3D solid textures, thus called *Lapped Solid Textures*, where we synthesize anisotropic solid textures inside a 3D model by filling the volume with overlapping patches of an exemplar solid texture that align with a user-specified volumetric orientation field (Figure 5.1 bottom). Seams among patches are made less noticeable by using alpha blending.

This approach is in accordance with our **Principle II** for volumetric modeling (i.e., "*exploiting structural regularities to achieve compact representations and fast algorithms*") in that the repetition of detailed structures is considered as the structural regularity. This regularity allows us to achieve a compact representation by reusing the

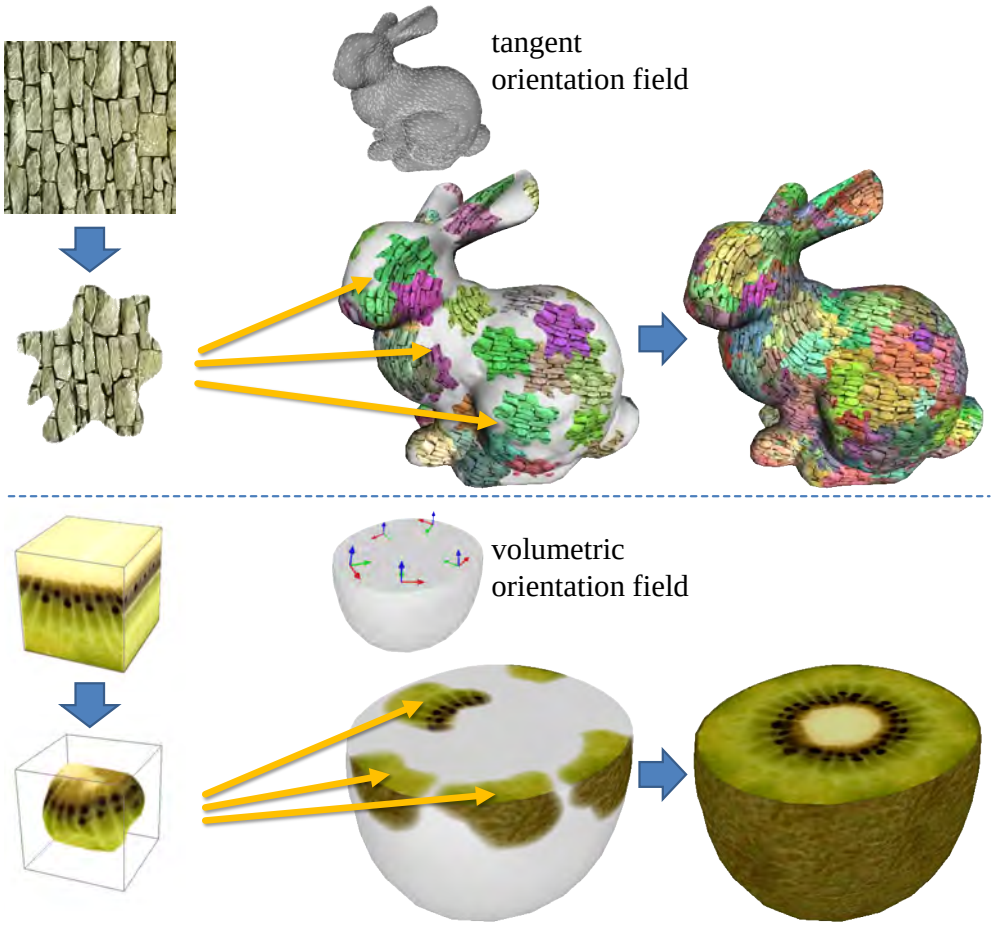

Figure 5.1: Conceptual similarities between Lapped Textures by Praun et al. [85] (top) and our Lapped Solid Textures (bottom).

same texture data many times to fill the model, as well as a fast patch-based texture synthesis algorithm as opposed to previous voxel-based texture synthesis algorithms.

Our approach inherits the same advantages with Praun et al.'s approach as follows. Our representation is very compact because we can reuse the input bitmap texture data many times and only need to store the texture mapping information (i.e., 3D texture coordinates), instead of the synthesized color at every 3D point. Our approach is computationally inexpensive; the precomputation of patch placement takes only tens of seconds, and the resulting model displays in real time using only standard features of the GPU. Other general color channels in the texture such as displacements can be trivially incorporated. Finally, efficient implementation of our approach is straightforward.

A unique difference of our method from Praun et al.'s is that our method takes advantage of our classification of solid textures presented in Chapter 2. Depending on the texture type, texture patches are placed along different kinds of volumetric fields, such as vector field, frame field, and depth field. Our system provides an intuitive modeling user interface and an efficient synthesis algorithm tailored for each texture type. Another notable difference is that we are able to handle layered textures (i.e., textures with variation level 1) by considering the layer depth during the patch-pasting process, while the original Praun et al.'s technique was limited to homogeneous textures (i.e., textures with variation level 0). Thanks to this extension, we can create volumetric models of objects with layered internal structures such as kiwi fruits, carrots, and trees, whose appearance changes gradually in the depth direction.

Note that our method assumes the availability of the input solid texture exemplar; various ways for creating different types of solid texture exemplars are detailed in Appendix A.

5.2 User Interface

The user first loads a tetrahedral mesh representing the 3D model's geometry and an exemplar solid texture. The system then shows a dialog box to allow the selection of a texture type. We explain the modeling process for each texture type in the following.

5.2.1 Texture Type 0-A

This type corresponds to isotropic textures, such as sponge and concrete. In this case, the user only needs to specify a volumetric scalar field representing the spatially-varying texture scaling. The user first puts a solid texture onto the model by clicking, and moves it interactively by dragging with the mouse (Figure 5.2a). The user can change the texture scale interactively using the mouse wheel (Figure 5.2b). When satisfied, the user can set the local texture scale by double-clicking on the desired position of the model. After the texture scaling is set appropriately (Figure 5.2c), the system fills the model with the texture taking into account such user-specified texture scaling (Figure 5.2d).

5.2.2 Texture Type 0-B

This type represents textures with flow or fiber orientation, such as bamboo and muscle. The user first specifies a volumetric vector field over the model using one of our interfaces presented in Chapter 4. After the volumetric vector field is created, the

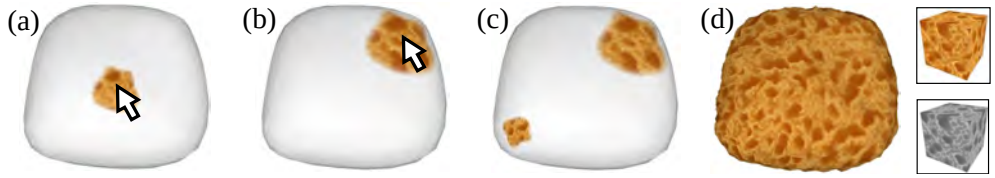

Figure 5.2: Modeling process for texture type 0-A. (a) Moving the texture patch by dragging with the mouse. (b) Changing the texture scale with the mouse wheel. (c) User-specified texture scaling. (d) Result of automatic filling (rendered with displacement mapping).

Figure 5.3: Modeling process for texture type 0-B. (a) Drawing strokes to specify local vector fields. (b) Setting the texture scaling. (c) Result of automatic filling.

user then sets the texture scaling (Figure 5.3b) as in the previous section. Finally, the system fills the model with the texture (Figure 5.3c).

5.2.3 Texture Type 0-C

This type represents homogeneous textures whose cross-sections have three different appearances depending on their relative orientations with respect to the local frame field. Such textures can be seen in materials such as flattened fibers. In this case, the user creates a frame field over the model using our interface for modeling volumetric frame fields presented in Section 4.4 (Figure 5.4 a, b). After the frame field is set appropriately, the user moves on to the process of setting the texture scaling (Figure 5.4c), followed by automatic filling (Figure 5.4d).

5.2.4 Texture Type 1-A

This type represents layered textures, such as cakes and watermelons. The user specifies a depth field over the model using our interface for modeling depth fields presented in Section 4.4 (Figure 5.5 a, b). After the depth field is set appropriately, the system then fills the model with the texture while considering the layer depth (Figure 5.5c).

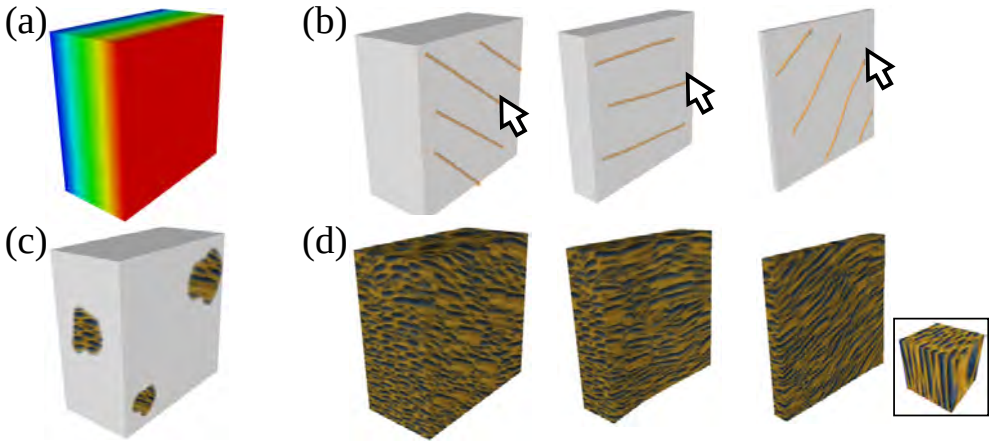

Figure 5.4: Modeling process for texture type 0-C. (a) Specifying the depth field. (b) Specifying the secondary directions by drawing strokes on layers. (c) Setting the texture scaling. (d) Result of automatic filling.

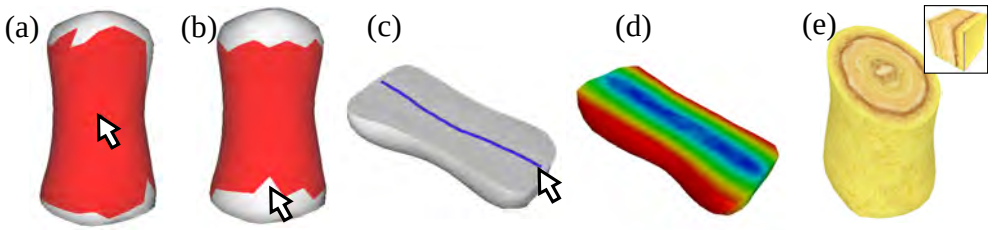

Figure 5.5: Modeling process for texture type 1-A. (a) Painting interface for modeling a depth field. (b) Computed depth field. (c) Result of automatic filling.

Texture scaling and orientation are derived automatically from the gradient of the depth field.

5.2.5 Texture Type 1-B

This type also represents layered textures, as in type 1-A, but the two perpendicular cross-sections parallel to the layer depth direction appear different. Such textures can be seen in objects such as kiwi fruit, carrots, and trees. The modeling process is identical to that for texture type 0-C, except that the texture scaling is derived automatically from the gradient of the depth field (Figure 5.6).

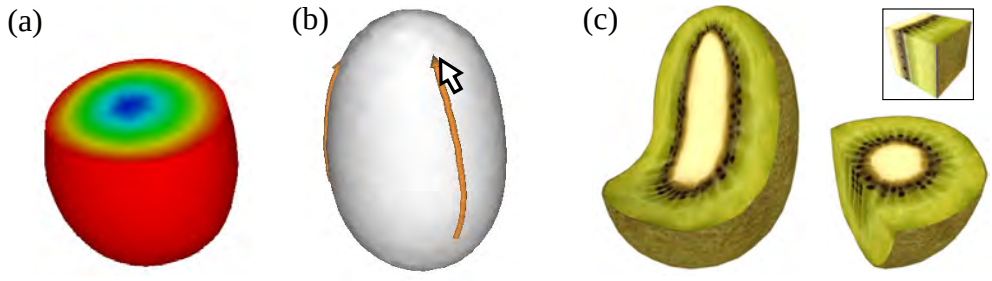

Figure 5.6: Modeling process for texture type 1-B. (a) Specifying the depth field. (b) Specifying the secondary directions by drawing strokes on depth layers. (c) Result of automatic filling.

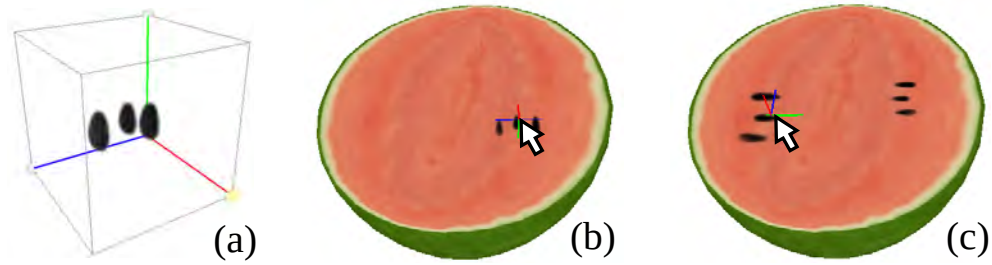

Figure 5.7: Manual pasting of additional textures. (a) Solid texture exemplar to be pasted. (b) Moving and rotating the texture patch by dragging with the mouse. (c) Changing the texture scale with the mouse wheel.

5.2.6 Manual Pasting of Textures

After the system generates a solid textured model, the user can also manually paste additional solid textures onto the model. The user first loads an additional solid texture (Figure 5.7a), which can then be moved and rotated on the model by dragging the mouse (Figure 5.7b). The user can also change the texture scale interactively with the mouse wheel (Figure 5.7c). Finally, the texture can be pasted onto the model by double-clicking.

5.3 Algorithm

The input to our system consists of a tetrahedral mesh representing the 3D model's geometry and a solid texture exemplar. The output is a lapped solid textured (LST) model; many overlapping patches of solid texture are pasted inside the mesh.

We used a tetrahedral mesh to represent solid models because this representation

has certain advantages over voxel representation for our purposes. First, it can approximate 3D shapes well with a smaller number of elements. Second, the tetrahedral mesh naturally corresponds to a triangular surface mesh when we extend the original 2D technique [85] to 3D. Finally, cross-sectioning and iso-surface extraction can be performed easily using marching tetrahedra [105], which is similar to marching cubes [62] except that it is faster and easier to implement.

We first describe how to render an LST model created in our system and then describe the process of constructing an LST model in detail.

5.3.1 Rendering an LST Model

Each tetrahedron in an LST model has a list of 3D texture coordinates assigned to each of its four vertices. To render such a model, we first convert it into a polygonal model that consists of surface triangles with a list of 3D texture coordinates assigned to each of its three vertices. We can then render this polygonal model using the same run-time compositing algorithm as Praun et al.'s method [85]. Each surface triangle is rendered multiple times (approximately 10 to 20 times in most of our results) using the texture coordinates in its assigned list, with alpha blending enabled.

Cutting

When the user cuts the model by drawing a freeform stroke (Figure 5.8a), the system constructs a scalar field over the tetrahedral mesh vertices, which takes negative and positive values on the left- and right-hand sides of the stroke, respectively (Figure 5.8b). We used RBF interpolation [107] to construct such a scalar field. The cross-sectional surface is then obtained by extracting the iso-surface of value 0 from the mesh (Figure 5.8c). The texture coordinates for each triangle on the cross-section are obtained by linearly interpolating the texture coordinates of the original tetrahedron. The tetrahedral mesh is subdivided near the cross-section to allow subsequent cutting operations.

Volume Rendering

We can also perform volume rendering on an LST model using the same approach as described above. We first construct a scalar field over the mesh vertices to give the distance between the camera and each vertex. We then calculate a large number of slices of the model perpendicular to the camera direction by iso-surface extraction.

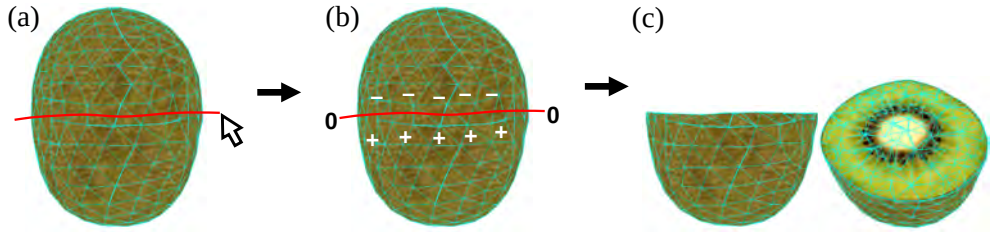

Figure 5.8: Cutting operation. (a) User-drawn stroke across the 3D model. (b) Scalar field computed from the stroke. (c) Resulting cross-sectional surface mesh.

5.3.2 Construction of an LST Model

The overall procedure closely follows the original [85], but each process contains non-trivial extensions, which we describe in detail in the following subsections. We first create a 3D alpha mask for solid textures to make the resulting seams between pasted texture patches less noticeable. We then construct a frame field over the mesh based on user input. The direction and magnitude of the frame field specify the orientation and scaling of the texture, respectively. Finally, textures are pasted repeatedly onto the model while aligning with the frame field.

The texture pasting process is as follows. First, a seed tetrahedron is selected. Then, we grow a clump of tetrahedra around the seed until it is large enough to cover the texture patch being pasted. Next, we perform texture optimization which warps the pasted texture so that it aligns locally with the frame field. Finally, we update the coverage of textures for each tetrahedron.

We also propose a method to create depth-varying solid models which was not addressed by Praun et al.'s original work [85]. We prepared several exemplar textures with different alpha masks and pasted them according to the depth.

Creating a 3D Alpha Mask

In the original 2D case, Praun et al. [85] created alpha masks for 2D texture patches using standard image editing tools. For a less-structured texture, they used a "splotch" mask independent of the content of the texture. For a highly structured texture, they created an appropriate alpha mask that preserved the important features of the texture as much as possible.

In our 3D case, we manually created a 3D alpha mask for solid textures by modeling a 3D shape of the mask using existing 3D modeling techniques, such as that reported by Nealen et al. [69] (Figure 5.9a). This mask is the 3D version of the "splotch" mask

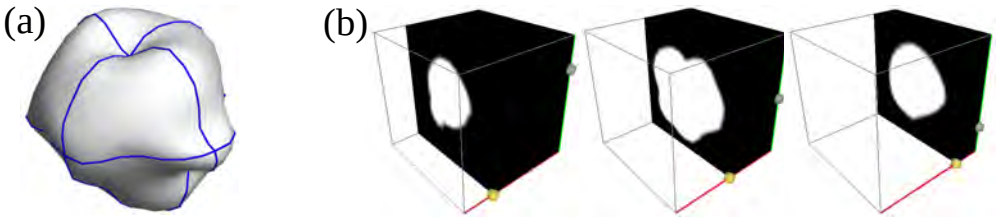

Figure 5.9: Manual creation of a 3D alpha mask. (a) 3D shape of the mask. (b) Cross-sections of the alpha mask.

in the 2D case, which can be applied to a less-structured solid texture (Figure 5.9b). The alpha value drops off around the boundary of the mask, which makes the resulting seams between pasted textures less noticeable. We found that an appropriate width of this drop-off is about 5–10% of the texture size in our experiments.

It is difficult, however, to create an appropriate alpha mask manually for a highly structured solid texture that preserves the important features of the texture. For now, we assume all the textures in our examples are less structured, and therefore we use a constant "splotch" mask shown in Figure 5.9 for all the textures. This assumption often causes artifacts when using highly structured textures, which will be discussed later.

Constructing a Frame Field

Regardless of the texture type, it is necessary to have a frame for each tetrahedron in order to compute the final texture coordinates using the texture optimization algorithm as described later. In the case of texture types 0-C and 1-B, we can naturally use the user-specified frame field. In the case of texture types 0-B and 1-A, however, the user has specified only a vector field (as the gradient of a depth field in the case of texture type 1-A), leaving an unresolved freedom about the rotation around that direction. Therefore, for each patch-pasting operation, we randomly choose an arbitrary 3D vector and orthogonalize it with the user-specified vector field at every location, resulting in a temporary frame field. In the case of texture type 0-A, we just randomly choose an arbitrary frame and regard it as a constant frame field.

The magnitude of the frame (i.e., the length of its three 3D vectors) determines the spatial extent of the resulting texture in the object space. Thus, a local frame defined as above, which has unit length, needs to be multiplied by an appropriate scaling factor, which is obtained as a user-specified texture scaling field in the case of texture types 0-A, 0-B, and 0-C, or derived automatically from the depth field in the

case of texture types 1-A and 1-B (as discussed later).

The frame field is initially stored on the tetrahedral mesh vertices. For each tetrahedron, we simply assign an average of frames of its four vertices. In the case of texture types 1-A and 1-B, the original depth field is kept and used as the final texture coordinate in the depth direction, as described later.

Selecting a Seed Tetrahedron

We first initialize a list of "uncovered" tetrahedra with all the tetrahedra in the mesh. For each pasting operation, one is selected at random from this list as a seed tetrahedron. After the pasting operation, tetrahedra are removed from the list if they are completely covered by the previously pasted textures. We repeat this process until the "uncovered" list becomes empty. In the case of manual pasting of the textures, the seed tetrahedron is set to the one clicked by the user.

Growing a Clump of Tetrahedra

We first map the seed tetrahedron from the object space into the texture space, so that its mapped frame axes align with the standard axes of the texture space, and its transformed central position is located in the center of the texture.

Let $(\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T})$ be the three orthogonal vectors of the frame associated with the seed tetrahedron $\mathcal{T}$. We first compute barycentric coordinates $r_1, \ldots, r_4$ on $\mathcal{T}$ to represent $\boldsymbol{R}$ as

$$\begin{aligned} r_1\boldsymbol{v}_1 + r_2\boldsymbol{v}_2 + r_3\boldsymbol{v}_3 + r_4\boldsymbol{v}_4 &= \boldsymbol{R} \\ r_1 + r_2 + r_3 + r_4 &= 0 \end{aligned}$$

where $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_4$ are the four vertices of $\mathcal{T}$. We do the same with $\boldsymbol{S}$ and $\boldsymbol{T}$. We then compute the transformed vertex positions $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_4$ in the texture space by solving the following equations

$$\begin{aligned} r_1\boldsymbol{w}_1 + r_2\boldsymbol{w}_2 + r_3\boldsymbol{w}_3 + r_4\boldsymbol{w}_4 &= (1,0,0)^T \\ s_1\boldsymbol{w}_1 + s_2\boldsymbol{w}_2 + s_3\boldsymbol{w}_3 + s_4\boldsymbol{w}_4 &= (0,1,0)^T \\ t_1\boldsymbol{w}_1 + t_2\boldsymbol{w}_2 + t_3\boldsymbol{w}_3 + t_4\boldsymbol{w}_4 &= (0,0,1)^T \\ c_1\boldsymbol{w}_1 + c_2\boldsymbol{w}_2 + c_3\boldsymbol{w}_3 + c_4\boldsymbol{w}_4 &= (0.5,0.5,0.5)^T \end{aligned}$$

where $c_1, \ldots, c_4$ are the barycentric coordinates on $\mathcal{T}$, which represent the position inside $\mathcal{T}$ where the center of the texture should be. In the case of automatic filling,

the position is set to the barycenter of $\mathcal{T}$ (i.e., $c_1 = \cdots = c_4 = \frac{1}{4}$), while it is set to the user-specified position in the case of manual pasting. After appropriate transformation of vertex positions, we finally compute an affine transform matrix M that maps $\boldsymbol{v}_i$ to $\boldsymbol{w}_i$.

Next, we grow the clump by adding adjacent tetrahedra. We visit each tetrahedron around the clump and add it to the clump if the tetrahedron satisfies the following two conditions: its frame is not markedly different from that of the seed, and it is partially inside the alpha mask in the texture space when transformed by M.

Texture Optimization

The purpose of texture optimization is to warp the texture so that it aligns locally with the frame field. More precisely, for each tetrahedron in the clump, we minimize the difference between the frame axes of the tetrahedron transformed into the texture space and the standard texture coordinate axes.

The input to this process is a clump of tetrahedra $\{\mathcal{T}_i\}$ and its associated frames $\{(\boldsymbol{R}_i, \boldsymbol{S}_i, \boldsymbol{T}_i)\}$ $(i = 1, \ldots, n)$. The output is the 3D texture coordinates $\{\boldsymbol{w}_j\}$ for all the vertices $\{\boldsymbol{v}_j\}$ $(j = 1, \ldots, m)$ in the clump.

For each T_i, we first compute the barycentric coordinates r_k^i, which represent $\boldsymbol{R}_i$ in the same way as in growing clump of tetrahedra described above. We then define the difference vector $\boldsymbol{d}_r^i$ between the transformed frame axis $\boldsymbol{R}_i'$ and the standard texture axis $\hat{\boldsymbol{r}}$ as

$$\boldsymbol{d}_r^i = r_1^i \boldsymbol{w}_{j_1} + r_2^i \boldsymbol{w}_{j_2} + r_3^i \boldsymbol{w}_{j_3} + r_4^i \boldsymbol{w}_{j_4} - (1, 0, 0)^T$$

where $j_1, \ldots, j_4$ are the indices of the four vertices of T_i. We do the same for the s and t directions (Figure 5.10). We minimize all these difference vectors, while satisfying the positional constraint given to the seed tetrahedron in the same way as growing clump of tetrahedra described above. The optimized solution $\{\boldsymbol{w}_j\}$ can be obtained quickly in a least squares sense by solving a sparse linear system.

Note that the optimization may warp the texture coordinates such that the image of the clump in the texture space no longer fully covers the splotch mask. In such cases, we add the lacking tetrahedra to the clump and re-compute the optimization.

Coverage Test of Tetrahedron

The original 2D method [85] used a rasterization technique to test the coverage of overlapping textures. We perform a similar computation over sampling points inside the tetrahedra. We first create several predefined discrete sampling points (165 points

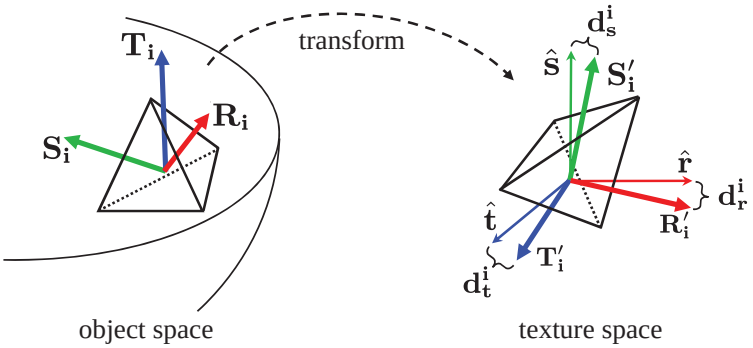

Figure 5.10: The optimization minimizes the difference vectors $\boldsymbol{d}_r^i, \boldsymbol{d}_s^i, \boldsymbol{d}_t^i$ between the texture coordinate axes $(\hat{\boldsymbol{r}}, \hat{\boldsymbol{s}}, \hat{\boldsymbol{t}})$ and the transformed frame axes $(\boldsymbol{R}_i', \boldsymbol{S}_i', \boldsymbol{T}_i')$.

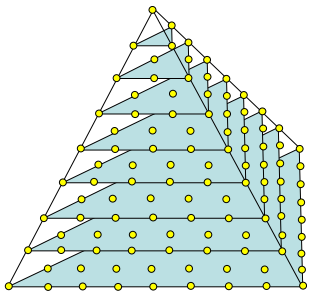

Figure 5.11: A set of 165 sampling points inside a tetrahedron for testing the texture coverage information.

in our current prototype) inside each tetrahedron in the mesh (Figure 5.11). Each time a texture is pasted, we sample the alpha values of the mask at these discrete points of each tetrahedron in the clump, which are then accumulated. If the accumulated alpha values of all the sampling points of a tetrahedron reach 255, we assume that the tetrahedron is completely covered by the overlapping texture patches.

Creation of Depth-Varying Solid Models

We create depth-varying solid models by arranging a depth-varying solid texture so that it aligns with the depth field defined over the target 3D model. The basic concept is to map the clump of tetrahedra into the corresponding depth position in the texture space instead of the central position. To achieve this, we alter the positional constraints in the process of growing a clump of tetrahedra and in the texture optimization process from $(0.5, 0.5, 0.5)^T$ to $(0.5, d_{seed}, 0.5,)^T$, where d_{seed} is the depth value assigned to $\mathcal{T}_{seed}$ assuming the s-axis corresponds to the depth orientation. However, a problem

Figure 5.12: (a) A problem occurs if we use only a single alpha mask. (b) The use of three types of alpha mask solves this problem.

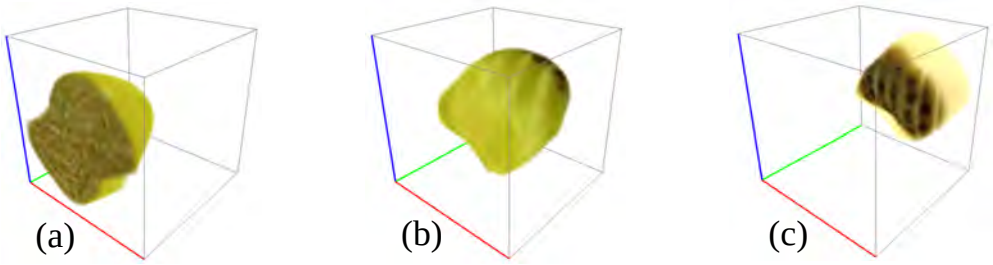

Figure 5.13: Three types of alpha mask: (a) outer part, (b) middle part, and (c) inner part.

occurs when we paste textures onto the inner and outer parts of the model (Figure 5.12a), because the alpha mask covers only the middle part of the texture.

To solve this problem, we prepared solid textures corresponding to different layers of the original texture, each with different alpha masks (Figure 5.13). These were created by simply applying the same alpha mask to different places (outer, middle, and inner). In the texture pasting process, an appropriate texture is chosen from these three according to the depth value of the seed tetrahedron (Figure 5.12b).

The depth values defined over the 3D model can be used directly as the texture coordinates of the depth direction, so we only solve for texture coordinates of the other two directions. We have also found that the appropriate texture scaling is the inverse of the magnitude of the depth gradient vector. This can be explained as follows. Suppose we have a large depth gradient vector at a certain position in the 3D model. This means that the depth value changes rapidly there, which also implies that the region corresponds to the thin part of the 3D model. Therefore, the texture scale should be small.

5.4 Results

Figure 5.14 shows some examples of volumetric models created using our approach. We can observe consistency among different cross-sections of these models, which was not possible in Owada et al.'s volumetric illustration method [76] as their method performs randomized 2D texture synthesis every time the model is cut. Models of kiwi fruit and watermelon contain many seeds, which would cause artifacts in the work of Pietroni et al. [83] as their method spatially interpolates a few 2D cross-sectional images placed in 3D space. Texture type 1-B is used to create models of kiwi fruit, carrot, and tree. Appearances of their cross-sections differ depending on the slicing orientation with respect to the central axes of their depth fields. Most textures of types 1-A and 1-B in our results have a thin (1–3 voxel thickness) slice of outer skin, and our depth adjustment technique arranges solid textures successfully so that such outer skin regions align precisely with the surfaces of the models. Cross-sectioning is faster than run-time synthesis approaches because the computation only involves linear sampling of texture coordinates. We can create more interesting solid models such as models of strata and cake by combining several LST models together. We can also perform volume rendering on translucent LST models, as shown in the model of a tube made of white oriented particles, which is impossible in Owada et al.'s volumetric illustration method [76]. This result was obtained by taking 200 slices from the model, a process that took about 3s. Our method can be extended easily to support other channels of textures, and we show the displacement mapping results in Figures 5.14c and 5.2d where the grayscale displacement map channel is shown next to the RGB texture. This is done by first subdividing the surface mesh and then moving each vertex along its normal direction according to the displacement value sampled there.

We implemented our prototype system using C++ and OpenGL on a notebook PC with a 2.3-GHz CPU and 1.0 GB of RAM. The statistics of our results are summarized in Table 5.1, which shows that our method is fairly inexpensive in terms of both computation and memory for representing large-scale solid models. It took a relatively long time to fill the cake model, because the model has a large thin sponge region that requires pasting a large number of texture patches. However, the rendering and cross-sectioning can still be performed in real-time.

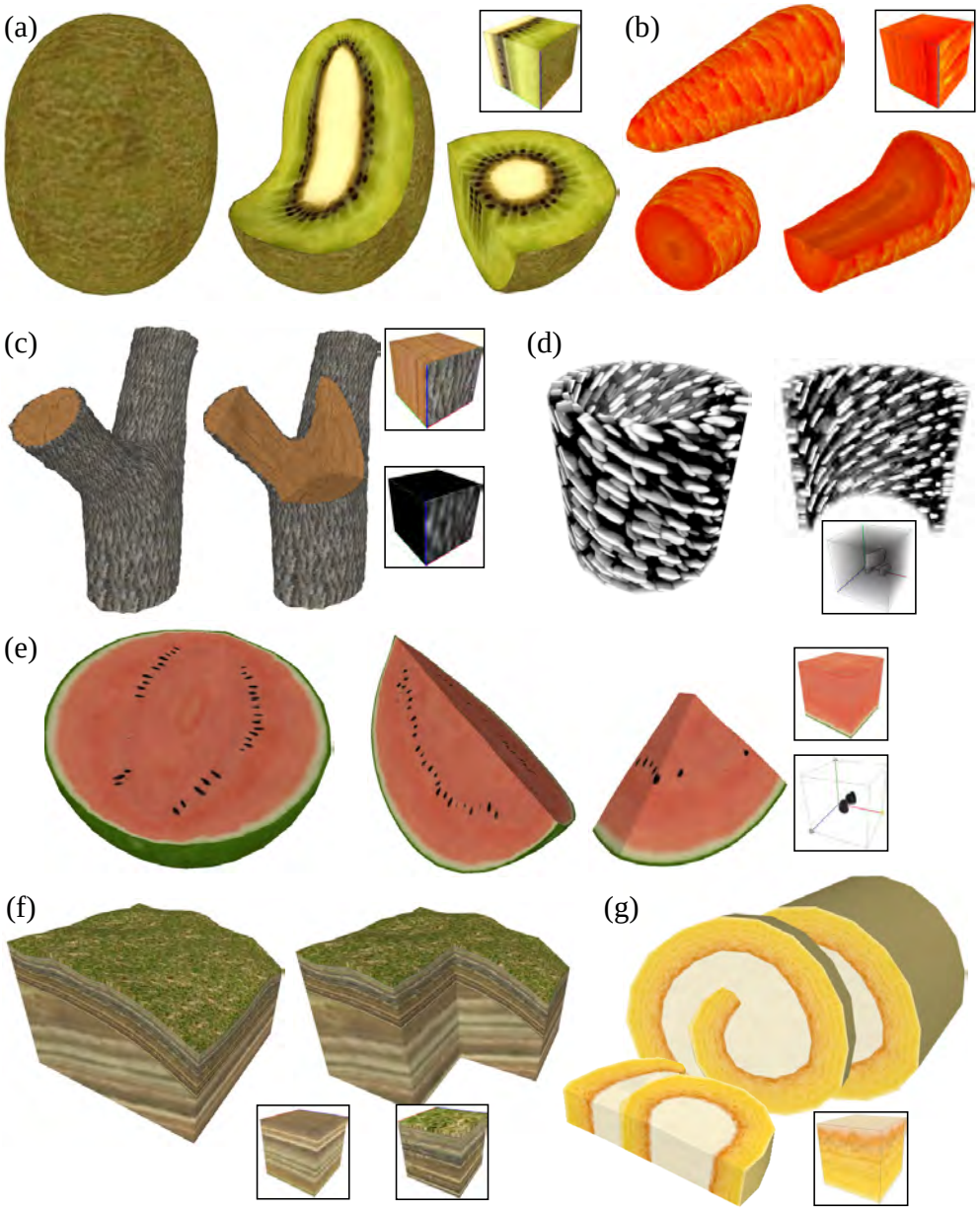

Figure 5.14: Resulting volumetric models filled with overlapping patches of solid textures: (a) kiwi fruit, (b) carrot, (c) tree (the grayscale texture represents the displacement map channel), (d) volume rendering of a fibrous tube, (e) watermelon, (f) strata, and (g) cake. Note that the input solid textures include surface textures as well as interior textures.

Title	Tetra	Ori [sec]	Fill [sec]	Cut [msec]	Size [MB]
Kiwi fruit	4126	29	39	78	9.1
Carrot	2313	38	31	63	7.1
Tree	5012	76	104	125	12.2
Watermelon	2717	17	25	63	9.0
Tube	1089	27	18	31	2.7
Strata	2827	113	77	110	10.4
Cake	2734	34	416	187	14.5

Table 5.1: Statistics of our results. Column describe (from left to right): title, number of tetrahedron, time for modeling orientation field, time for automatic filling, time for cross-sectioning (without subdivision), and total data size of LST model (including texture exemplars). The size of texture exemplars was 64^3 throughout.

5.5 Limitations

Our method inherits the limitations of the original method [85]. First, the patch seams become noticeable when using a texture with strong low-frequency components. Second, artifacts appear around singularities of the frame field, such as the center of a depth-varying object with a radial axis (Figure 5.15). This can be alleviated by locally subdividing tetrahedra in such areas. Finally, as we use a constant "splotch" mask for all the textures, blurring artifacts appear when a highly structured texture is used (Figure 5.16). It is necessary to create an appropriate alpha mask that preserves the structure of the texture as much as possible, and this may be achieved by extending the existing 2D contour detection technique [51] to 3D. It is also necessary to consider the alignment between texture patches to avoid misalignment artifacts (Figure 5.16c). Soler et al. [95] proposed hierarchical pattern mapping, which considers the coherence between texture patches on surfaces, but extending their technique to 3D solid appears to be non-trivial.

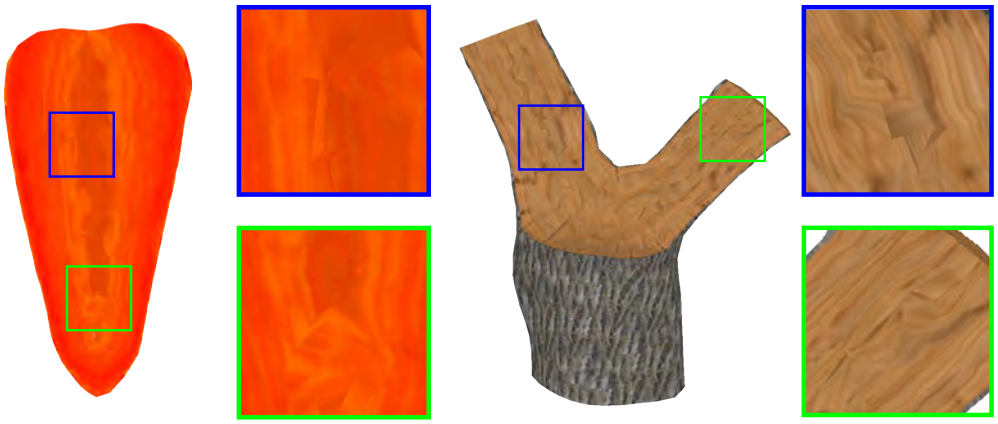

Figure 5.15: Artifacts occurring at singularities of volumetric frame fields.

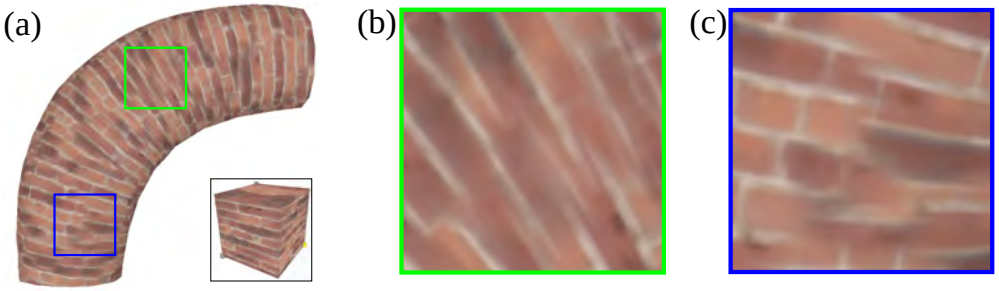

Figure 5.16: Failure case with a highly structured texture. (a) A curved cylinder filled with bricks shows (b) blurring and (c) misalignment artifacts.

Chapter 6

Vector-Based Method for Representing Smooth Color Transitions using Colored 3D Surfaces

In our raster-based method presented in the previous chapter, we assumed that the input texture exemplar is less structured, making the approach suited for modeling objects that can be well described as a collection of small-scale texture details, such as carrots and tree stumps. However, the approach is problematic when using structured texture exemplars such as brick wall, causing artifacts such as blurring and misalignment of structural features, as shown in Figure 5.16. Furthermore, it is unclear how the approach can be used to model volumetric objects that have global and distinct internal structures distributed non-uniformly, such as tomatoes, strawberries, and apples (Figure 6.1).

To enable volumetric modeling of such objects, we propose a vector-based method

Figure 6.1: Examples of modeling target in this chapter which are difficult for our raster-based method described in the previous chapter.

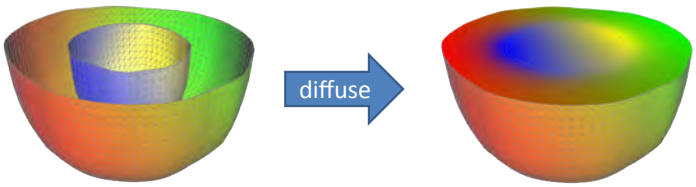

Figure 6.2: The concept of the DSs representation. A set of colored surfaces (left) defines a volumetric color distribution (right) by diffusing colors in 3D space.

in which we do not use either volumetric orientation field or 3D texture exemplars. Instead, we model volumetric objects using a new representation called *Diffusion Surfaces* (DSs). In our DSs representation, a model is defined as a set of 3D surfaces with colors assigned to both of their front and back sides. A volumetric color distribution is defined as a diffusion of colors from these surfaces through the volume. This way, global and distinct volumetric structures can be well represented (Figure 6.2).

This approach is in accordance with our **Principle I** for volumetric modeling (i.e., "*propagating user-specified information on the surfaces through the volume*") in that we let the user paint colors on such 3D surfaces which are then propagated though the volume. We also make use of our **Principle II** for volumetric modeling (i.e., "*exploiting structural regularities to achieve compact representations and fast algorithms*") by regarding the smoothness of volumetric color transitions as the structural regularity. This regularity allows us to compactly represent a volumetric model with only a sparse set of colored 3D surfaces. In addition, it also enables a fast algorithm for diffusing colors through the volume, which will be detailed later.

The DSs representation is conceptually an extension of Diffusion Curves (DCs) [72], a vector graphics representation for 2D images, to 3D volumes. Thus, our representation inherits the same advantages of the DCs representation: compactness and editability. We emphasize that these advantages are even more pronounced when extended to volumetric modeling in our case. If we use a raster-based representation (i.e., voxel grid) to store the volume, the amount of data needed grows rapidly as the resolution increases, easily using up all the available memory. From the user's point of view, manually painting colors over a 3D voxel grid is fundamentally much more difficult than painting over a 2D pixel grid, and thus sparse editable vector representation is much desired for volumetric modeling.

In the DCs approach, color diffusion is computed by solving a Poisson equation over a 2D pixel grid, where the colors specified along the curves serve as Dirichlet

boundary conditions. Directly extending this method to 3D volumes is costly both in terms of computation time and memory consumption; to achieve high-quality results, high-resolution voxel grids or tetrahedral meshes are required (especially in order to faithfully represent complex volumetric models), and precomputing and storing the resulting color information is computationally expensive. Instead, we propose to interpolate colors only locally at cross-sectional locations using a modified version of the positive mean value coordinates (PMVC) algorithm [60], enabling us to generate high-resolution cross-sections efficiently. Our method produces consistent volumetric color distributions comparable to Poisson solutions, and at the same time saves the effort of computing the entire color volume at once, eliminating the necessity of precomputation. The resulting framework is suitable for trial-and-error modeling processes where volumetric structures can be interactively added, removed, edited, and explored, which is indispensable for creative interactive modeling.

The process of modeling the shapes and colors of the DSs is independent of the representation, such that any general-purpose modeling tools can be used to create DSs for arbitrary objects (e.g., fruits, vegetables, organs, or geological structures). However, modeling and spatially positioning the required number of intricate and nested geometric components using existing tools is often difficult and time-consuming. Thus, we leverage the fact that many objects exhibit rotational symmetries in their internal structure, and design a sketch-based interface tailored for this purpose. Fruits, vegetables, and other familiar objects fall into this category of volumetric structures. Our interface assumes and exploits rotational symmetries, which greatly simplifies the modeling process. The interface also enables generating random variations of models, which is particularly useful for scenes containing a large variety of similar objects.

Using DSs, we are able to create a variety of volumetric models not possible with our raster-based method. As we ignore all the small-scale texture details, the plain rendering of DSs produces rather too smooth appearances. Thus we apply several simple non-photorealistic rendering (NPR) techniques to DSs to increase the expressivity of the plain rendering of DSs. Note that the information about internal structures encoded by DSs is helpful for such NPR effects.

6.1 Related Work on Vector Graphics

Our work is partly inspired by the recent developments of vector graphics techniques. Vector graphics approach essentially aims at representing objects in 2D images with only a sparse set of information about the overall structure and color. Our basic moti-

vation is that such vector graphics approach should also be well suited for volumetric modeling, because sparseness is a desired property for volumetric representations.

There are a few different approaches in 2D vector graphics. Here we briefly review two representative techniques that became popular recently: gradient meshes (GMs) and diffusion curves (DCs), and discuss their strengths and weaknesses to justify why we chose to extend DCs to 3D volumes. In addition, we also mention other recent approaches to vector representations for 3D volumes.

6.1.1 Gradient Meshes

The GMs technique was first introduced in Adobe Illustrator in 1998 and subsequently in Corel CorelDraw in 1999, without its technical details being reported publicly. Since then, surprisingly, no research about the GMs technique has been conducted until Sun et al.'s work in 2007 [97]. Because there was no publicly available documentation about the definition of GMs, they first gave the formal mathematical definition for GMs as follows. A gradient mesh is a regular quad mesh consisting of Ferguson patches [29] which is a cubic function $m(u,v)$ defined on the 2D parameter domain as $m : [0,1]^2 \mapsto \mathbb{R}$. $m(u,v)$ can be used to represent general quantities, and in this case it is used to represent geometry (i.e., x and y coordinates) and color (i.e., RGB color channels). The user specifies its value and partial derivatives m_u, m_v, and m_{uv} at each of its four control points $\{(i,j) \mid i,j \in \{0,1\}\}$ as m^{ij}, m_u^{ij}, m_v^{ij}, and m_{uv}^{ij}. Then, its value at an arbitrary parametric location $(u,v) \in [0,1]^2$ can be computed as:

$$m(u,v) = (1 \; u \; u^2 \; u^3) C Q C^T \begin{pmatrix} 1 \\ v \\ v^2 \\ v^3 \end{pmatrix}$$

where $C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 3 & -2 & -1 \\ 0 & -2 & 1 & 1 \end{pmatrix}$ and $Q = \begin{pmatrix} m^{00} & m^{01} & m_v^{00} & m_v^{01} \\ m^{10} & m^{11} & m_v^{10} & m_v^{11} \\ m_u^{00} & m_u^{01} & m_{uv}^{00} & m_{uv}^{01} \\ m_u^{10} & m_u^{11} & m_{uv}^{10} & m_{uv}^{11} \end{pmatrix}$.

For controlling the geometry, the user moves control handles displayed on the screen to specify m^{ij}, m_u^{ij}, and m_v^{ij}. For controlling the color, the user specifies color values (i.e., m^{ij}) of control points while the derivatives (i.e., m_u^{ij} and m_v^{ij}) are set automatically according to the color values of adjacent control points. The freedom to specify m_{uv}^{ij} is not much useful in practice, so m_{uv}^{ij} is always set to zero. Since any pair of adjacent Ferguson patches share the same control points, the function is smoothly

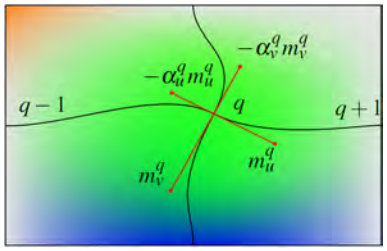

Figure 6.3: A simple example of gradient mesh consisting of four Ferguson patches.

continuous across the whole mesh. The user carefully designs the topologies and layouts of GMs such that they align well with the target objects in the image, which is often an extremely time-consuming process. Figure 6.3 shows a simple example of gradient mesh consisting of four Ferguson patches.

Sun et al. [97] proposed a method for semi-automatically generating high-quality optimized GMs based on user-specified initial layouts of GMs by solving nonlinear least-squares problem (Figure 6.4 top). Later, Lai et al. [57] proposed a fully automatic method for generating GMs for a given input image (Figure 6.4 bottom). Their representation is topologically more flexible than the standard GMs representation allowing holes in the mesh. Their key idea is to utilize recent mesh parameterization techniques for image vectorization.

6.1.2 Diffusion Curves

The approach of diffusion curves was proposed by Orzan et al. in 2008 [72]. Their basic idea, which is inspired by Elder's seminal work in 1999 [26], is to represent objects in images using edges instead of regions as in the GMs approach. In the DCs representation, the user directly specifies properties of each edge: the two colors on its left and right sides, and the amount of blur that controls its sharpness. Specifically, the user can draw a set of curves freely (Figure 6.5a), and for each curve the user can specify a set of three continuously varying values along the curve: the left and right side color (Figure 6.5b) and the blur (Figure 6.5c). Given such information, the system first diffuses the curve colors over the entire image, and then performs spatially-varying blur on the resulting image to obtain the final image (Figure 6.5d).

Mathematically, a diffusion curve is defined over the 1D parameter space $\mathbb{T} := [0, 1]$ with associated attributes consisting of:

- its geometry (i.e., x and y coordinates) $P : \mathbb{T} \mapsto \mathbb{R}^2$,

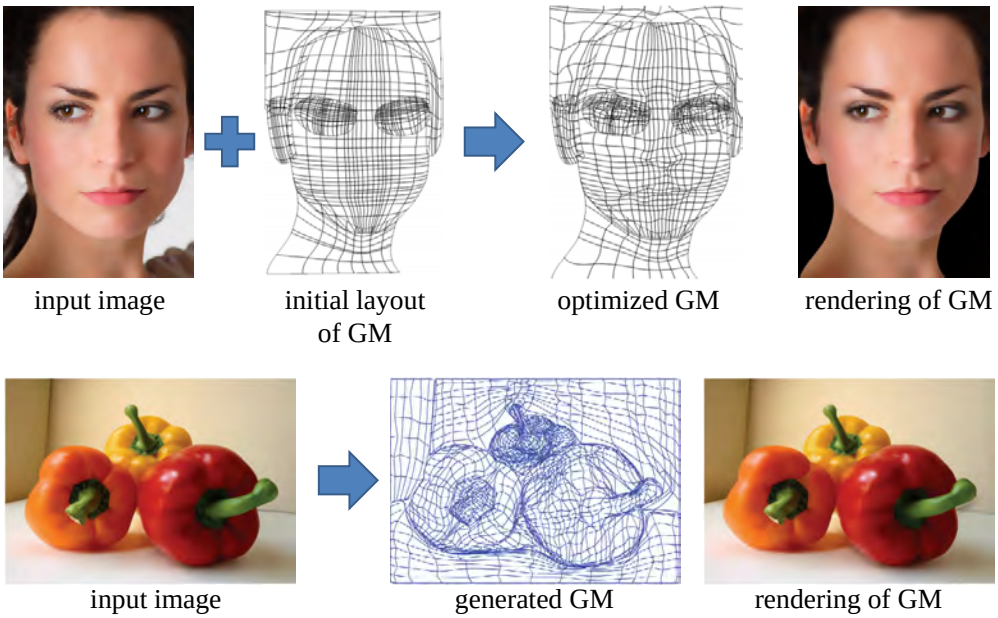

Figure 6.4: Two recent approaches by Sun et al. [97] (left) and Lai et al. [57] to image vectorization using gradient meshes.

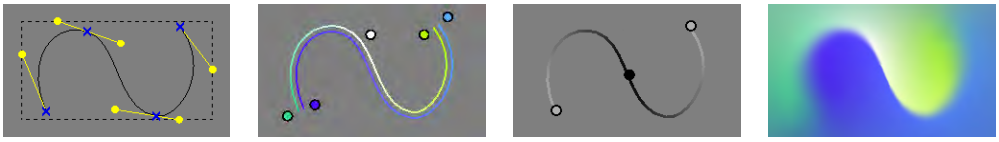

Figure 6.5: Definition of a diffusion curve. In addition to its geometry in the form of standard Bezier splines (a), it also has continuously varying values of colors on its left and right sides (b) and blur (c). The final image is obtained by diffusing the curve colors and then performing spatially-varying blur (d).

- its left and right side colors (separate for each of RGB color channels) $C_L : \mathbb{T} \mapsto \mathbb{R}$ and $C_R : \mathbb{T} \mapsto \mathbb{R}$,
- and its blur $\Sigma : \mathbb{T} \mapsto \mathbb{R}$.

These parametric functions can be expressed in any forms such as piecewise linear segments or cubic Bezier splines.

Since the processes of color diffusion and spatially-varying blur over the entire image are both computationally expensive, Orzan et al. propose to implement these on the GPU to achieve real-time performance that enables interactive drawing by

the user. There is an issue when implementing a color diffusion solver on the GPU; ideally, a point on a diffusion curve should define two different color constraints at the same location, which cannot be handled well by the standard GPU's rasterization functionality. Therefore, they propose to slightly move the two color constraints apart from each other along the curve normal, and set a color gradient constraint on the original curve point. Because of this strategy, the color diffusion can now be computed by solving a Poisson equation.

Figure 6.6 shows an overview of the entire process. The curve geometry is first duplicated and displaced along its normal with a small distance, resulting in a pair of left and right side curves as $P_L : \mathbb{T} \mapsto \mathbb{R}^2$ and $P_R : \mathbb{T} \mapsto \mathbb{R}^2$, respectively (Figure 6.6a). An image $I(x, y)$ resulting from color diffusion, called sharp color image, can be obtained by solving the following Poisson equation:

$$\Delta I = \mathrm{div} w$$

subject to

$$I|_{P_L(t)} = C_L(t), \quad I|_{P_R(t)} = C_R(t), \quad \forall t \in \mathbb{T},$$

where Δ and div are the Laplacian and divergence operators, respectively, and $w(x, y)$ is a 2D vector field defined as

$$w(x,y) = \begin{cases} (C_R(t) - C_L(t)) \frac{P_R(t) - P_L(t)}{\|P_R(t) - P_L(t)\|} & (x,y) = P(t),\ t \in \mathbb{T} \\ 0 & \text{otherwise} \end{cases},$$

as shown in Figure 6.6b. Next, an image $B(x, y)$ resulting from diffusing the blur over the entire image, called blur map, can be obtained by solving the following Laplace equation:

$$\Delta B = 0$$

subject to

$$B|_{P(t)} = \Sigma(t),\ \forall t \in \mathbb{T},$$

as shown in Figure 6.6c. The final image is obtained by applying spatially-varying blur to the sharp color image I with the blur kernel size proportional to the blur map B, as shown in Figure 6.6d. Orzan et al. also proposed a simple method for automatic image vectorization which generates DCs from a given raster image by extracting and merging the image edges using the scale-space approach.

Orzan et al.'s work was so influential that many researchers subsequently explored various extensions. Jeschke et al. proposed variable stencil size Laplacian solver [47]

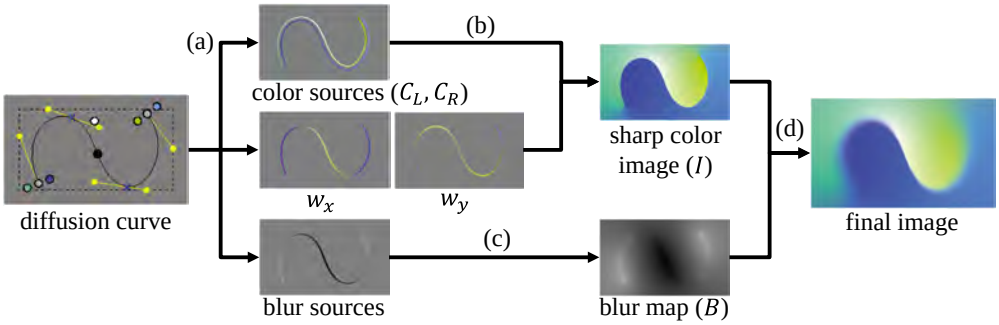

Figure 6.6: An overview of the entire process of rendering a diffusion curve.

which makes the color diffusion much more robust and simpler to implement, and also proposed a method to use DCs for texturing 3D surfaces [48]. Bezerra et al. modified the Poisson formulation to achieve more control over the color diffusion process [3]. Bowers et al. proposed a ray tracing approach to diffusing colors [7]. Finch et al. demonstrated the ability of higher-order biharmonic interpolant for providing more detailed and intuitive controls over the color diffusion [30]. A few researchers explored other curve attributes in addition to colors. Winnemoller et al. proposed to add 3D normal vector and 2D texture coordinate to the curve attributes to achieve relighting and texture draping, respectively [115]. Jeschke et al. proposed a combination of DCs and procedural Gabor noise [56] along with a simple scheme for estimating noise parameters from an input image [46].

6.1.3 Gradient Meshes vs. Diffusion Curves

In terms of topological flexibility, the DCs approach is more desirable than the GMs approach. In the GMs approach, the user needs to carefully design the topologies of regular quad meshes, while in the DCs approach she can draw freeform curves in arbitrary ways.

In terms of computational cost, the GMs approach is more favorable than the DCs approach. In the GMs approach, a Ferguson patch defines its color distribution explicitly as cubic interpolation which is easy to compute. In contrast, the DCs representation defines its color distribution implicitly as the solution to the color diffusion problem which is more sensitive to small differences in particular solver implementations and configurations of boundary conditions. Although several alternative solvers have been proposed subsequently [47, 7, 30], the high computational cost involved in computing the color diffusion is inevitable. This makes it difficult to render vector

images in the DCs representation on devices with limited computational resources such as mobile phones. Partly because of this high computational cost, the DCs representation has not yet been integrated so far into any of the popular vector graphics packages such as Adobe Illustrator and Corel CorelDraw.

We chose to extend the DCs approach to 3D volumes because of its advantage of topological flexibility. To alleviate the high computational cost which becomes even higher when extended to 3D volumes, we propose a method to obtain approximate solutions locally with low computational cost based on the positive mean value coordinates algorithm [60].

6.1.4 Vector Graphics for Volumes

As mentioned in Chapter 3, a few researchers recently explored vector representations for 3D volumes [111, 110, 120]. All of these methods divide the space into disjoint regions using signed distance fields, and fill each region with a smoothly-varying color distribution. Our main difference from these is that we do not necessarily partition the volume into disjoint regions, thus allowing open-ended boundaries and not needing to store signed distance fields. Also, we define the volumetric color distribution as the color diffusion from 3D surfaces, which we believe is more intuitive and easier to control than Wang et al.'s approach using RBF [111, 110]. Note that both of Wang et al.'s two techniques [111, 110] are not extensions of any existing 2D vector graphics representations, while Zhang et al.'s technique [120] is an extension of the GMs approach to 3D volumes.

6.2 Diffusion Surfaces

In the following, we describe our basic primitive for volumetric modeling, called a *diffusion surface* (DS), and describe an efficient algorithm for computing and rendering cross-sections of volumetric objects represented by DSs.

6.2.1 Definition

DSs extend diffusion curves (DCs) to 3D volumes by replacing 2D curves with 3D surfaces. A DS is a triangle mesh in which each mesh vertex has the following attributes: colors on its front and back sides and a blur value. A DS diffuses its colors on either side, and the softness of the color transitions between the front and the back sides of the surface is controlled by the blur values (Figure 6.7).

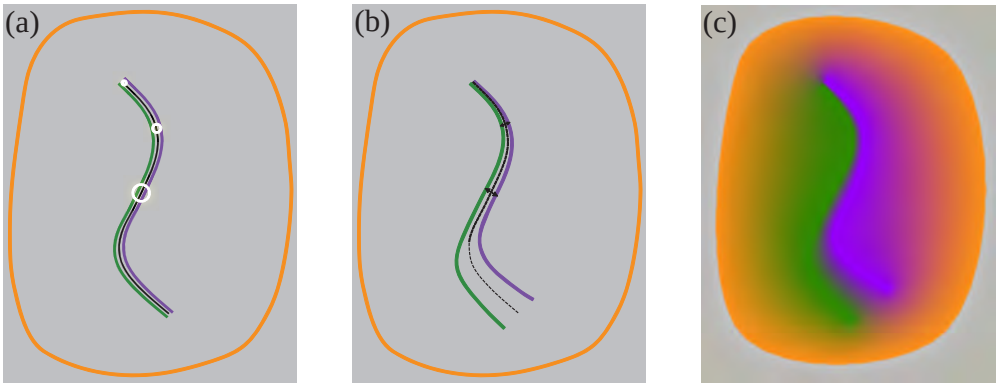

Figure 6.7: A DS is a surface with colors on either side and blur values (a). The blur radii are depicted as white circles. (b) The surface is duplicated into two sheets, which are moved apart along the normals by the blur radius distance. (c) Colors from both sheets are smoothly diffused in space.

Unlike DCs in which the blur is applied after diffusion as a post process, the blur in DSs is achieved by duplicating each DS into two surface sheets (one for the front and one for the back side), and moving the sheets apart along the surface normal directions by the distances explicitly specified as the blur values (Figure 6.7). This is because DSs are not a voxel-based representation, and the post-process blur of DCs cannot be applied to DSs. Our blur may be less smooth, but we found it sufficiently smooth in practice.

6.2.2 Algorithm for Generating Cross-Sections

Our goal is to generate a cross-section with colors diffused from DSs when the user cuts the model at an arbitrary location. A straightforward way to achieve this is to compute the entire volumetric color diffusion by solving a Poisson equation over a voxel grid or a tetrahedral mesh, similar to DCs. However, this approach lacks scalability and becomes too expensive for complex models, as discussed in detail in Section 6.2.3. Thus, we propose to compute color diffusion locally at cross-sectional locations using positive mean value coordinates [60] (PMVC). This is inspired by the work of Farbman et al. [27] where the Poisson solution was replaced by mean value coordinates [32] to accelerate various image editing tasks. The actual process of our algorithm consists of two steps: cross-sectional mesh generation and color diffusion using PMVC.

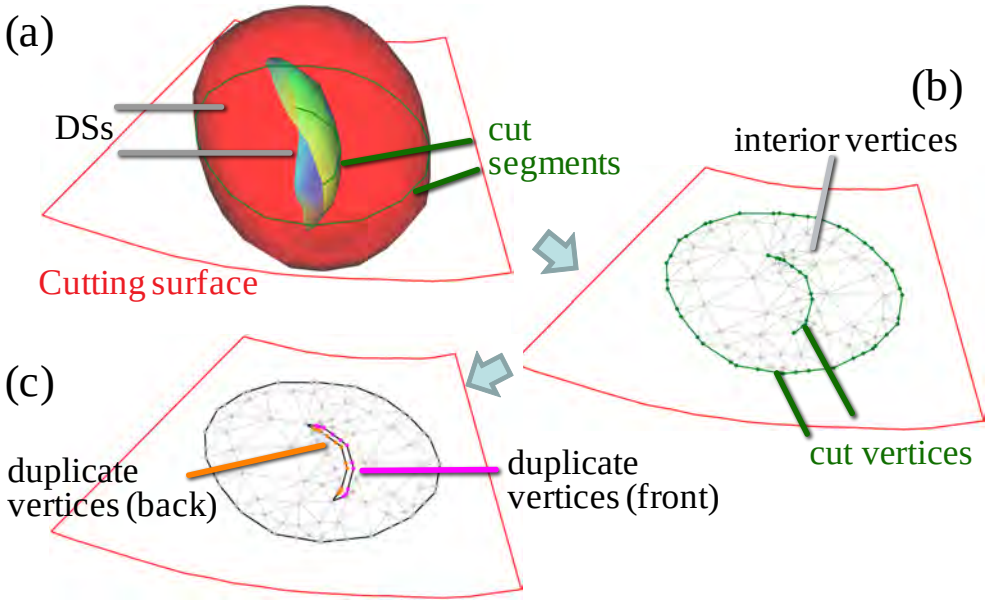

Figure 6.8: Cross-sectional mesh generation. Given DSs and a cutting surface (a), the system triangulates the cutting surface (b) and splits the mesh connectivities (c).

Cross-Sectional Mesh Generation

The system first generates a cross sectional mesh when the user cuts the model. Given a 3D cutting surface, all of its intersections with DSs are computed, forming a set of line segments called *cut segments* (Figure 6.8a). The cutting surface is then tessellated while preserving these cut segments using conforming 2D Delaunay triangulation [93], which nicely adapts the mesh density to the size and placement of the DSs to alleviate discretization artifacts. The cross-sectional mesh vertices that are not part of the cut segments are called *interior vertices* (Figure 6.8b); their colors are computed by diffusing colors from the DSs using PMVC, as explained later. Mesh vertices that lie on the cut segments are called *cut vertices*; their colors are sampled directly from the respective DSs. A cut vertex has two different colors (front and back) at the same location, so it is necessary to split each cut vertex into *duplicate vertices* (Figure 6.8c), divide the mesh connectivity along the cut segment, and assign the front and back colors of the corresponding DS to each duplicate vertex appropriately. The rendering is done simply by drawing mesh triangles with associated colors.

Color Diffusion Using PMVC

Once the mesh is generated on the cross section, the system diffuses colors on the mesh using PMVC. Given an interior vertex, our goal is to compute its color by interpolating the colors of the DSs using PMVC. To do so, we first render all the DSs viewed from the interior vertex to a cube map. This efficiently determines which parts of the DSs are visible and how far they are from the interior vertex. The system then sums up all the colors rendered on the cube map pixels, weighted by the inverse of the distances obtained from the depth buffer. Note that the diffusion is *consistent* for any cut surface, since the color information is obtained by integrating over the entire set of 3D DSs.

While Lipman et al.'s original method [60] renders the mesh triangles into the cube map using carefully tailored colors to compute coordinate values, our method renders the actual colors of the DSs directly to the cube map since we only need interpolated colors instead of coordinate values. This allows the integration of the cube map pixels to be performed entirely on the GPU, significantly reducing the cost of both reading the GPU memory back to the CPU and integrating cube map pixels on the CPU. Our modification means that the cost of integrating cube map pixels is negligible and the bottleneck becomes the cost of rendering DSs (roughly 99% of the total). Our current unoptimized implementation renders all the mesh triangles of the DSs naïvely per each interior vertex, but it could be greatly accelerated by parallelizing processes for all interior vertices and using efficient spatial data structures [90].

Note that our technique of efficiently diffusing colors using PMVC is due to our **Principle II**; it is made possible only by assuming the structural regularity of the smoothness of the volumetric color distribution.

6.2.3 Comparison of PMVC and Poisson Diffusion

We compared our PMVC diffusion to a standard Poisson diffusion to demonstrate the effectiveness of our approach. For the discretization of the Poisson equation, we used unstructured tetrahedral meshes generated using TetGen [94] instead of regular voxel grids, since tetrahedra can easily conform to DSs while voxels lead to rasterization artifacts.

Figure 6.9 shows some comparison results. The quality of the Poisson diffusion heavily depends on the resolution of the tetrahedral mesh. For relatively simple models (e.g., onions), low resolution meshes are sufficient to produce smooth diffusions, but this does not hold for more complex models (e.g., persimmons). For such models

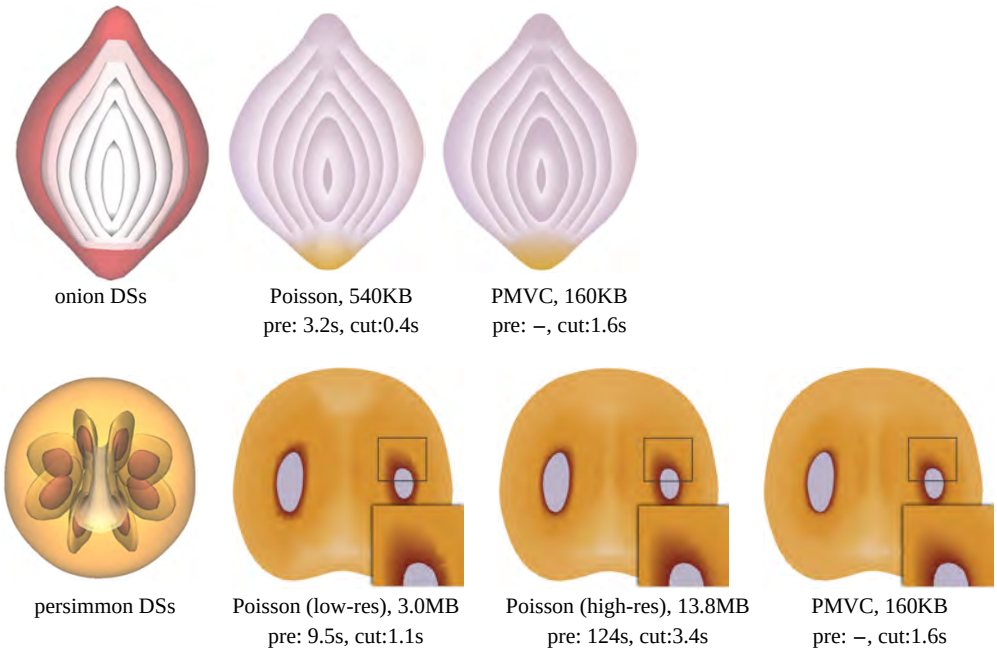

Figure 6.9: Comparisons of our PMVC diffusion and Poisson diffusion. The precomputation time (denoted "pre:") includes the tetrahedralization and solving the Poisson equation. The storage sizes refer to the precomputed tet mesh with colors for Poisson and the cross-section mesh with colors for PMVC.

we need to use denser volumetric meshes, which lead to a rapid increase in computational cost. In contrast, PMVC diffusion involves no volumetric meshing and no large systems of equations, and thus scales well with the complexity of the model. The visual differences between PMVC diffusion and Poisson diffusion are noticeable but not significant. The run-time cutting using PMVC diffusion takes a bit longer than that with Poisson diffusion, but this is mainly due to our unoptimized implementation which can be further accelerated. PMVC diffusion also allows frequent switching between designing and browsing of DSs since it requires no precomputations for browsing, which is highly desirable for trial-and-error modeling processes. This way, surfaces can be interactively added, removed, and edited, and the results can be viewed without delay. Additionally, Poisson diffusion has high memory consumption, since the entire precomputed color volume needs to be stored, whereas PMVC diffusion does not suffer from this problem. In summary, Poisson diffusion has some benefits but lacks scalability, and therefore our PMVC diffusion process is better suited for interactive modeling.

6.3 Creating Diffusion Surfaces

We focus on creating DSs from scratch rather than converting from scanned color volume data, since such data is not yet abundant. DSs are simple and basic primitives, and therefore any existing 3D modeling tools can be used to create DSs representing arbitrary objects. However, modeling volumetric structures composed of many nested surfaces using traditional tools can be difficult, time-consuming, and unintuitive. Thus, we chose to focus on a case study of objects with rotational symmetries, and designed a sketch-based modeling interface especially tailored to such objects. Particularly important classes of this kind are fruits and vegetables, since they are one of the most common volumetric objects in our daily lives, and they often possess intricate volumetric structures including both sharp and smooth color transitions. Our assumption of rotational symmetry additionally enables synthesis of random variations of models, which is useful for populating scenes with many similar objects, such as the one shown in Figure 6.21.

The entire modeling process with our interface works as follows (Figure 6.10):

1. The user first sketches several 2D curves on both vertical and horizontal cross-sectional images, and the system generates 3D surfaces by sweeping.

2. Optionally, the user can distribute predefined small grains over these sweep surfaces by specifying distribution parameters (the region to be populated by grains and their density). The distribution proceeds automatically by dart throwing while respecting a user-defined Poisson disc radius.

3. Next, the user paints colors and blur values on surface mesh vertices in the 3D view. Colors can be sampled from the reference images.

4. Finally, the system instantly synthesizes random variations of the DSs model on request.

We detail each of these steps below.

6.3.1 Symmetry-Aware Sketching Interface

We assume that representative horizontal and vertical cross-sections of the object to be modeled with DSs are available as photographs or illustrations, to make it easier on the user by providing a rough reference of the object's shape and structure. The user provides the system with a few guidance sketches that delineate the salient structures of the object, and DSs are then generated from these sketches. Note that

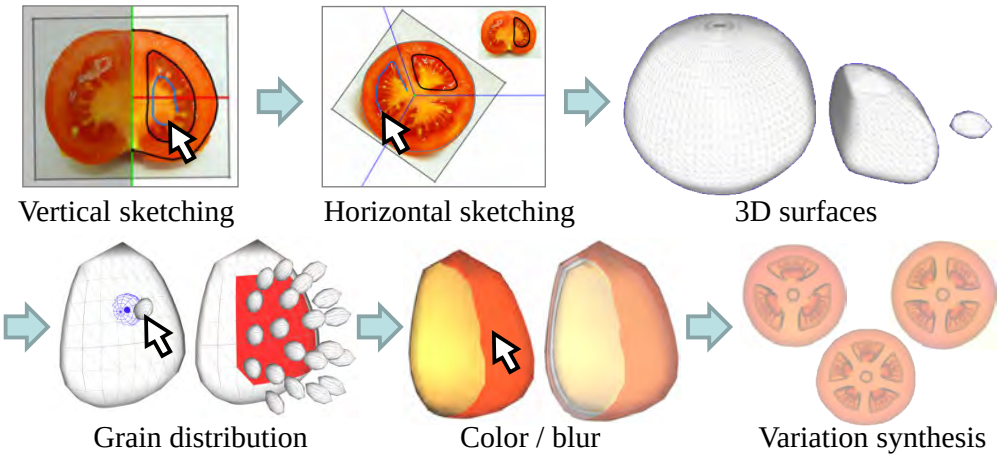

Figure 6.10: Overview of our modeling interface for creating DSs.

our sketching interface also utilizes our **Principle I** for volumetric modeling in a sense, because the user sketches on two surfaces characteristic to the internal structures (i.e., parallel/perpendicular to the symmetry axis) and then the system propagates such information through the volume to form 3D surfaces.

We chose to let the user trace image features by sketching, instead of relying on automatic feature detection algorithms, such as a Canny detector [9]. This is because the feature detection would be very challenging especially for subtle features on noisy cross-sectional images of fruits and vegetables, and also because we regard this feature extraction task as the user's creative design process, rather than an automatic reconstruction problem. For the similar reasons, we let the user specify the information about rotational symmetries, instead of using automatic algorithms for detecting them [58].

Symmetry Types

We consider two types of rotational symmetry: *cylindrical symmetry* that refers to objects the structures of which remain mostly the same after any amount of rotation around the axis of symmetry (e.g., strawberries and avocados), and *N-fold symmetry* that refers to objects having N repetitive structures around the axis of symmetry (e.g., tomatoes and okra) (Figure 6.11). We provide a similar, but separate, user interface for these two types of symmetry as described below.

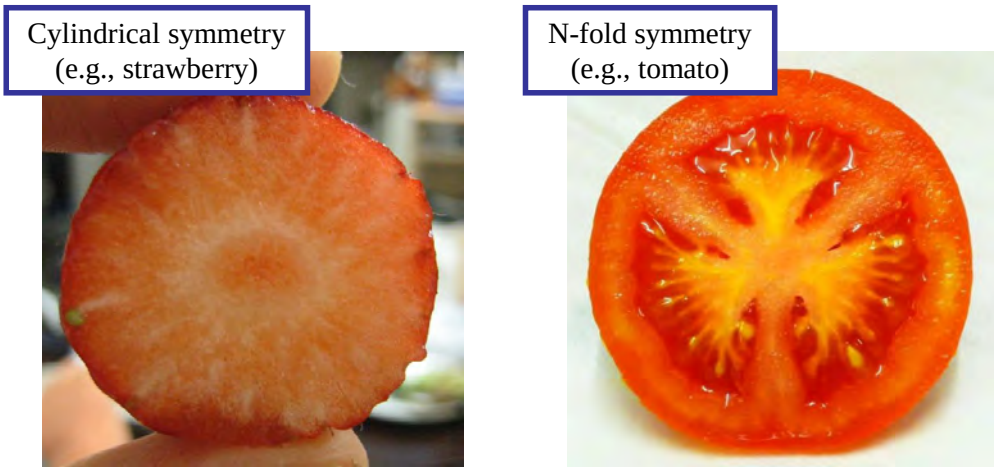

Figure 6.11: Two types of rotational symmetry in our user interface.

Cylindrical Symmetry

The user first loads an image of a vertical cross-section (i.e., a plane parallel to the axis of symmetry) and adjusts its position. We assume that the axis of symmetry is always straight, depicted as a green line in Figure 6.12a. The user then draws several curves representing geometric features on the image. The user is allowed to draw only on the right-hand side of the axis. If an endpoint of a curve is close enough to the axis, it is snapped to the axis. Several curve-editing tools, such as deformation and smoothing [40], are provided for convenience. An automatic curve fitting algorithm [51] would assist the user's sketching even more, which is not yet implemented.

The user next loads an image of a horizontal cross-section (i.e., a plane perpendicular to the axis of symmetry) and adjusts its position. Then, the user draws curves on the horizontal cross-sectional image corresponding to the curves on the vertical cross-section drawn previously (Figure 6.12b). The axis of symmetry is shown as a point, and the user must draw closed loops surrounding this point. Given pairs of vertical and horizontal curves, the system generates 3D surfaces by sweeping horizontal curves along the axis of symmetry with an appropriate scale and position, according to the vertical curves (Figure 6.12c).

We made the assumption of the axis of symmetry being straight in order to simplify the user interface. The modeling of objects with curved axes of symmetry could be possible by, for example, simply applying space warping or shape-preserving deformations as a post-process.

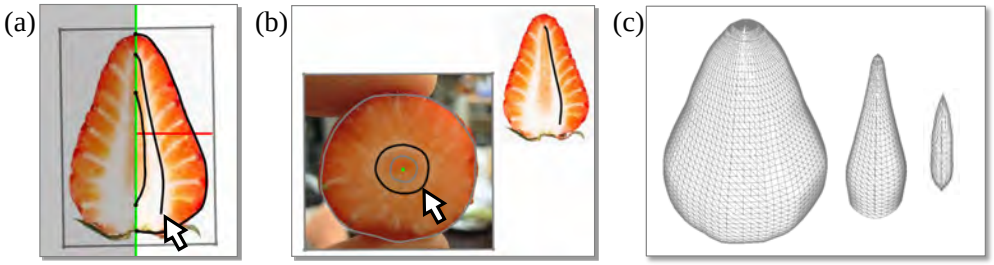

Figure 6.12: Sketching interface for the case of cylindrical symmetry.

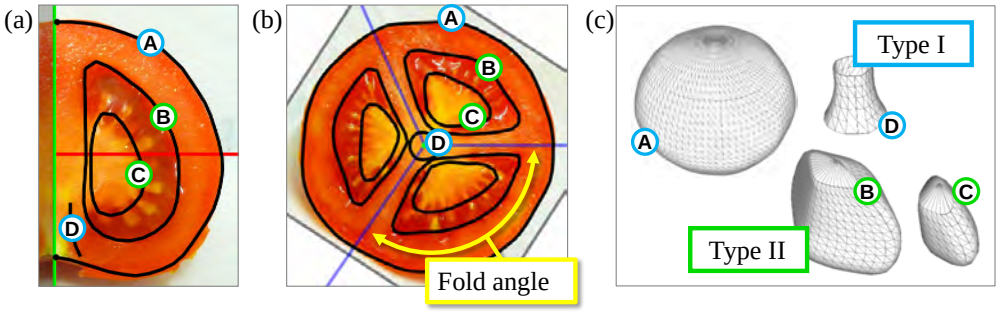

Figure 6.13: Sketching interface for the case of N-fold symmetry.

N-fold Symmetry

In this case, we classify features into two types: *Type I*, which surrounds the axis of symmetry (e.g., the outer skin of a tomato); and *Type II*, the geometry of which is included entirely within a single fold (e.g., an inner chamber of a tomato), as depicted in Figure 6.13. The modeling process for Type I geometry is almost the same as for the case of cylindrical symmetry described above, except that the user must specify the number of folds, N, and adjust the fold angles appropriately when drawing horizontal curves (Figure 6.13b).

For the Type II geometry, the user draws a closed loop in the vertical cross-section, as well as a set of N closed loops, each of which is entirely included within a single fold in the horizontal cross-section. In this case, the system generates 3D surfaces by first dividing each vertical curve into two at its top and bottom, and then sweeping the corresponding horizontal curve along these two curves (Figure 6.13c).

Partial Deletion: We sometimes require only portions of the surfaces generated by sweeping (Figure 6.14). Our system allows the user to remove any unwanted parts by simply sketching a cutting stroke as depicted in Figure 6.14.

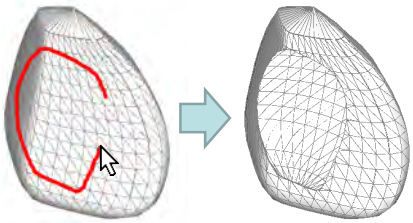

Figure 6.14: Partial deletion of surfaces generated by sweeping.

6.3.2 Modeling of Small Grains

Fruits and vegetables often contain many small grains covering some base surface regions (e.g., the seeds in a tomato). These cannot be modeled well with the interface described above, because their arrangement is not actually governed by the rotational symmetry. Thus, our system provides a specialized interface for such objects. The user first selects a vertical curve corresponding to the base surface over which grains will be distributed, and then draws a pair of curves near the selected curve (Figure 6.15a). The user then draws a closed loop representing the swept profile of the grain geometry (Figure 6.15b). Given these curves, the system generates a 3D surface by simple sweeping, which is transformed to a canonical position (Figure 6.15c). Next, the user specifies the region of the base surface where grains are distributed using a sketching interface (Figure 6.15d), and the radius of Poisson disk sampling as the distribution density (Figure 6.15e). The system can then distribute grains over the specified base surface region with the specified density by dart throwing (Figure 6.15f) [12]. The coordinate frame of each distributed grain is determined by the normal orientation of the base surface and the axis of symmetry orientation. The user can also adjust either the scale of the grain geometry or its offset from the base surface, if necessary.

6.3.3 Surface Attributes

Each mesh vertex of DSs is associated with two attributes: color and blur. While extracting colors and blur values along feature curves on images could be possible [72], we do not opt for this approach because it is not clear how to assign colors and blur values to 3D mesh vertices based on those values assigned to the pairs (vertical and horizontal) of 2D curves. Therefore, we chose to let the user paint these attributes directly on 3D mesh vertices.

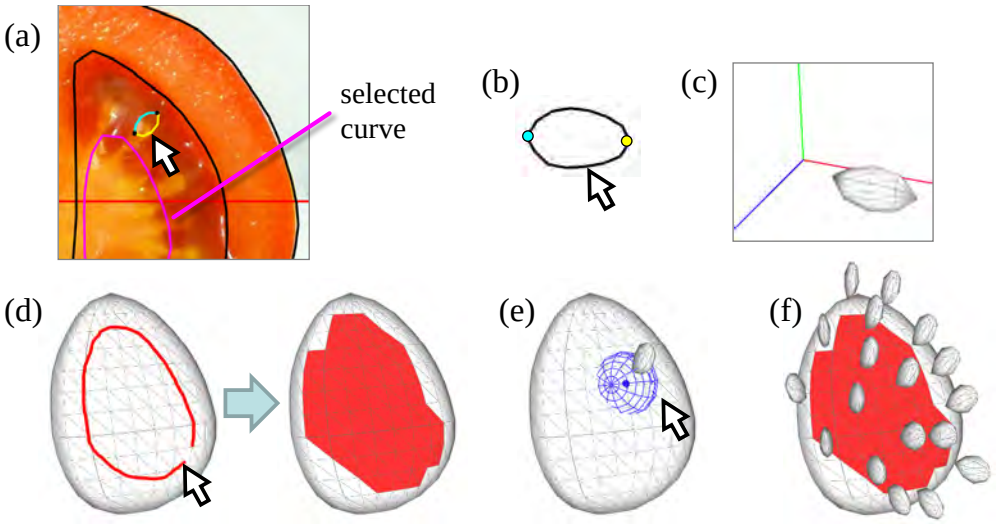

Figure 6.15: Modeling of small grains.

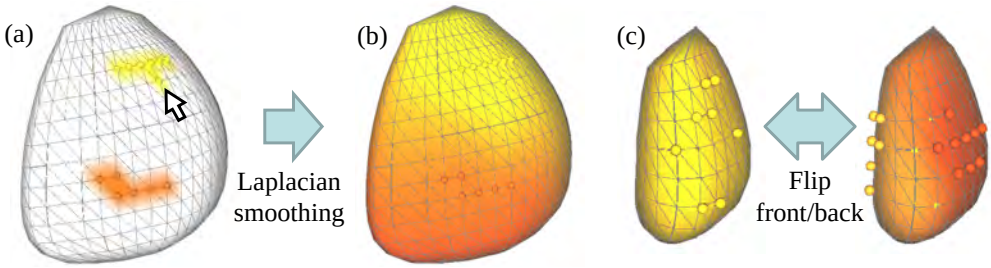

Figure 6.16: User interface for painting colors.

Colors: The user can choose the current color using either a dialog box or picking up a pixel color from an image. The user can assign the current color to surface mesh vertices by dragging the mouse over them (Figure 6.16a). These vertices are treated as being constrained, and colors at unconstrained vertices are obtained using Laplacian smoothing (Figure 6.16b). The user can switch the flag of two-sided colors (i.e., whether different colors can be assigned to the front and back side of the surface) on and off. The user can flip the front and back colors for surfaces with two-sided colors (Figure 6.16c) for convenience. Note that the DSs representation does not allow vertices at open boundaries to have two-sided colors, since they are adjacent to both the front and back side.

Blur values: Unlike the case of diffusion curves where the blur effect is achieved by applying spatially-varying blur kernels [72], our system achieves a similar effect

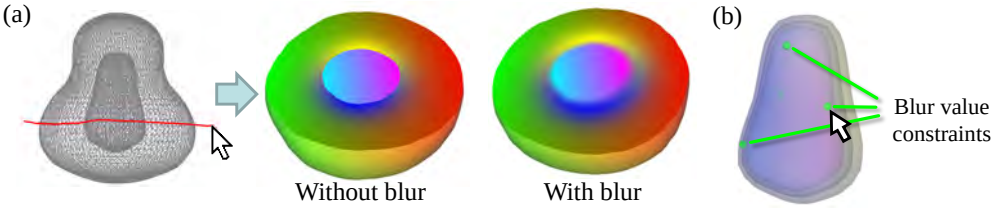

Figure 6.17: Blur effects.

Figure 6.18: Variation synthesis results for cylindrical symmetry (top, onion) and N-fold symmetry (bottom, tomato).

in 3D by duplicating the DSs and moving them in the surface normal directions by some distance explicitly specified by the user (Figure 6.7). Similar to the case of specifying colors as described above, the user can specify blur values at several vertices as constraints, and the system performs Laplacian smoothing to obtain blur values at unconstrained vertices (Figure 6.17b). Note that our system currently does not support a mix of blurred and unblurred regions within a single surface, because this would lead to a non-manifold surface.

6.3.4 Synthesis of Random Variations

The system synthesizes the horizontal curve geometries, while other information, such as the vertical profile curves, remain fixed. We analyze and synthesize the horizontal curves in a 2D polar coordinate system whose origin is the axis of symmetry. The process differs depending on the type of symmetry.

In the case of cylindrical symmetry (Figure 6.18 top), the curve geometry is first uniformly resampled. Then for each curve point, we sample its radius along with the angular difference from its next curve point. This forms a 1D cyclic array each element

of which is a pair of radius and angular difference. This array can be regarded as a 1D texture sample. We thus simply apply a texture synthesis algorithm [17] on this sample to generate a new randomized 1D array, resulting in a new randomized curve geometry.

In the case of N-fold symmetry (Figure 6.18 bottom), we use an approach similar to the morphable face model [5]. The system first takes N samples of feature vectors, each representing geometries within a fold, then performs a principal component analysis over these samples, and finally blends the principal vectors linearly with random coefficients.

The feature vectors are computed as follows. For the Type I geometry, the curve is split at every fold line, forming a set of N open curves. These open curves are then uniformly resampled with the same number of points and transformed into a canonical space (i.e., a pie-shaped space with the angle of $2\pi/N$). Finally, the curve points are concatenated into the feature vectors. For the Type II geometry, the set of N curves is evenly resampled and then transformed into the canonical space. Then the correspondences among curve points are obtained based on the sums of Euclidean distances. Finally, the curve points are concatenated into the feature vectors. Note that after synthesizing feature vectors, the Type I geometries (open curves) are blended linearly across each fold line to ensure the continuity among adjacent folds.

All synthesized geometries share the same mesh connectivity with the original so that they can use the same attribute data, such as colors and grain distribution parameters associated with surface meshes.

6.4 Results

We have created numerous DSs models using the proposed interface (Figure 6.19). Photorealistic rendering of real-world objects using DSs is challenging, since the current representation lacks texture detail information. That said, since DSs explicitly represent volumetric structures via 3D surfaces and blur values, a variety of rendering styles can be easily applied. We have experimented with simple non-photorealistic rendering (NPR) techniques (Figure 6.20) that highlight the internal structures and are more illustrative and expressive than the plain color rendering. We have employed artistic silhouettes [71] and color modulation based on 3D Perlin noise [81] with manually adjusted weights on different frequency bands based on the unmodulated color. Note also that the readily available structure representation allows rendering cross-sections with concavities or hollow spaces, as visible in the okra and pepper models,

Figure 6.19: DSs models created using our user interface. Here we visualize internal structures using alpha blending in addition to rendering cross-sections.

for example. This is simply done by tagging the respective side of the DSs representing those regions as invisible and hiding them when rendering cross-sections.

Figures 6.21, 6.22, and 6.23 show scenes consisting of many cut pieces of objects displayed using NPR. In such scenes, our algorithm for synthesizing random variations of models proved very effective. All objects contain global distinct structures, spatially-varying materials, and smooth color transitions, which would be difficult to handle with

Figure 6.20: Increasing the expressivity of plain color rendering (a) by applying simple NPR techniques: artistic silhouette [71] (b) and Perlin noise [81] (c).

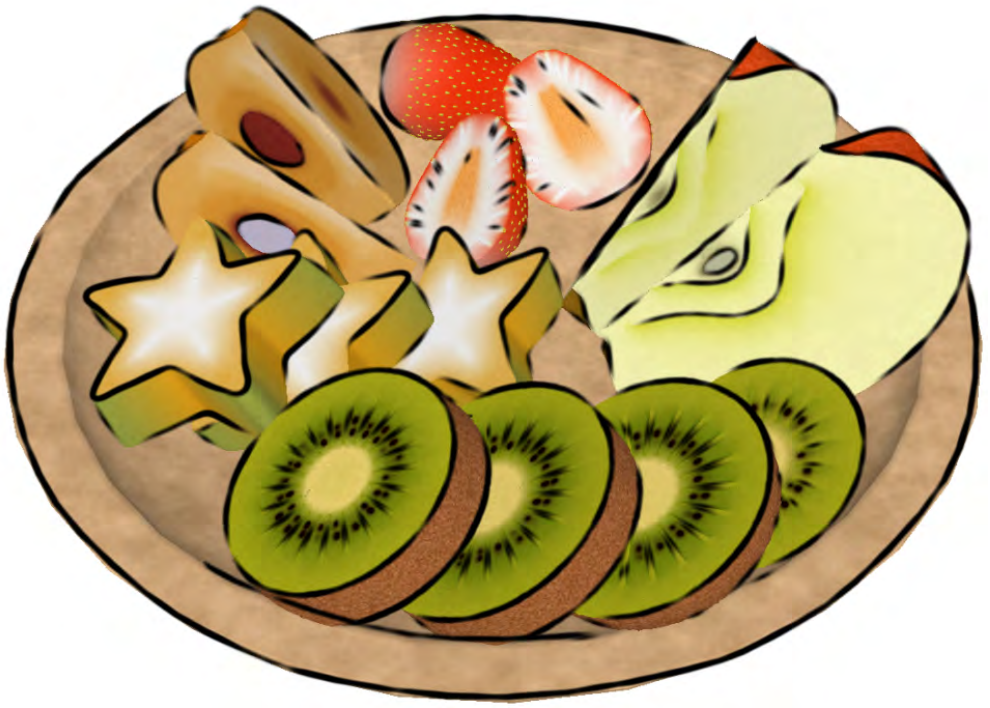

Figure 6.21: A scene of assorted fruits displayed using NPR.

previous methods [21, 74]. Also note that most examples (e.g., tomatoes, strawberries, star fruits, and more) make use of open surfaces, which cannot be represented by isosurfaces of signed distance fields [111]. We emphasize that the DSs representation is rather general and can be used to model a variety of volumetric objects. For instance, we have successfully created a geological model (Figure 6.24); our user interface was instrumental in designing the different layers and veins in this case, since they generally follow symmetry around the main lava axis. The kidney model (Figure 6.25) is another

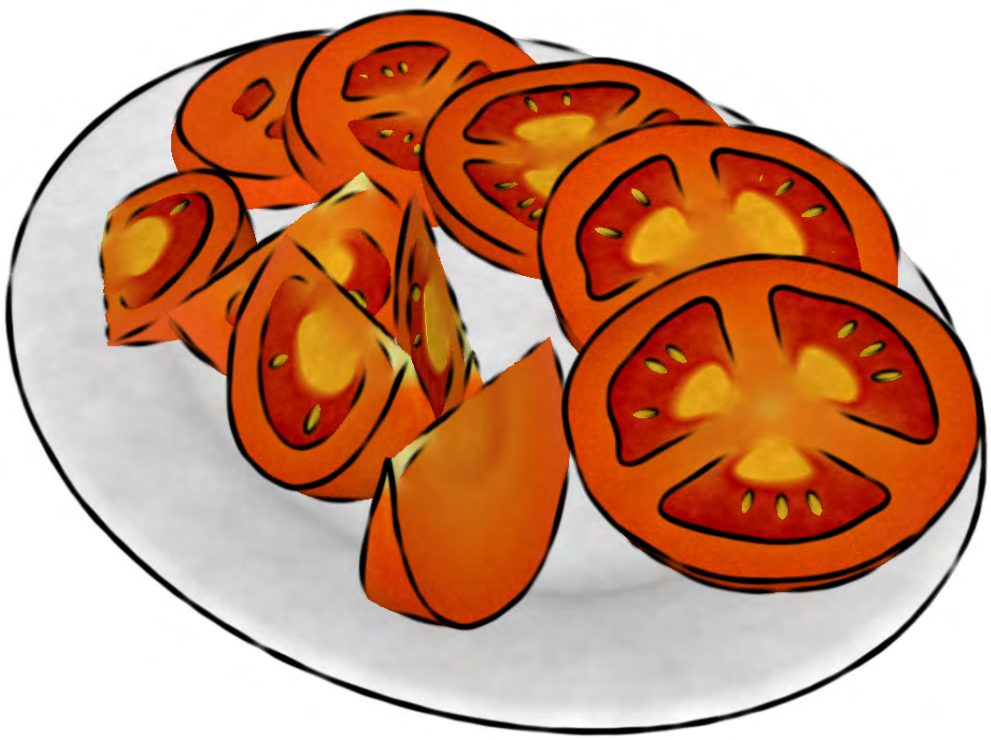

Figure 6.22: A tomato salad.

example of a natural object whose inner structure is modeled with DSs (its shape was modeled manually).

Our prototype system is implemented using C++, OpenGL and GLSL on a laptop with a 2.6 GHz CPU, 3.0 GB of RAM, and an NVIDIA Quadro FX 570M GPU. Table 6.1 shows our result statistics, indicating the efficiency of the DSs representation in terms of both storage and computation, as well as the usefulness of our user interface for creating DSs, which leads to fairly low design times. All results were designed by the authors, and the design times typically took several minutes. Some of the more complex models were created through a lot of experimentation; the creative exploration took several hours in those cases. Most of the modeling processes required much trial and error, and our simple sketching interface was effective for this purpose. A formal user study is a subject for future work.

Figure 6.23: A vegetable salad.

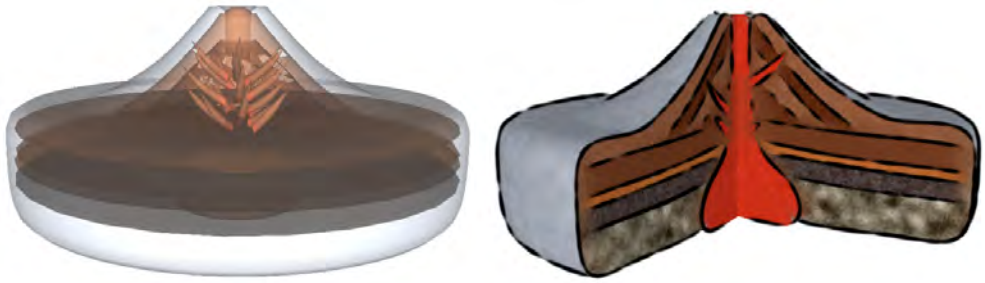

Figure 6.24: A DSs volcano model created with our interface.

6.5 Limitations

The main current shortcoming of DSs is the lack of information about texture details. Combining DSs with texture synthesis, in which smooth color distributions defined by DSs guide the synthesis of spatially-varying textures, is an interesting future direction. Additionally, we currently ignore translucency which plays an essential role in photorealistic rendering of volumetric objects. Translucency could be introduced by

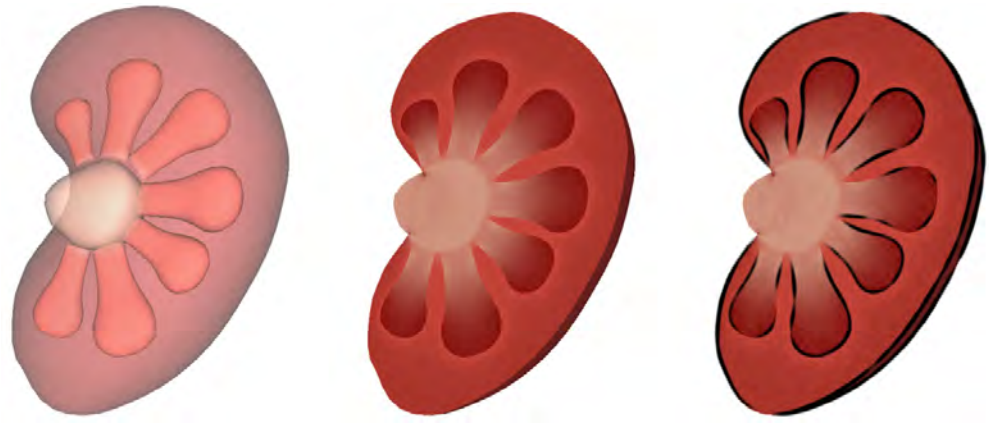

Figure 6.25: A kidney model represented as DSs.

Title	Type	Design (min)	Cut (sec)	Vtx #	Face #	Size (KB)
Tomato	N-fold	25	6.1	10619	20412	669
Onion	Cylndr	10	1.9	4184	8110	264
Persimmon	N-fold	16	8.1	9162	18208	584
Pepper	N-fold	20	6.8	9317	18328	592
Avocado	Cylndr	8	0.8	2699	5341	172
Apple	N-fold	25	3.3	4074	7685	255
Okra	N-fold	17	3.7	9400	18480	597
Star fruit	N-fold	15	3.2	6357	12190	400
Strawberry	Cylndr	141	7.7	18158	32049	1110
Cucumber	N-fold	184	7.9	14984	28592	942
Kiwi	N-fold	162	27.7	21438	38688	1321
Volcano	Cylndr	187	10.9	11057	20673	690
Kidney	–	362	3.9	2939	5694	185

Table 6.1: Result statistics. We report the type of rotational symmetry in column 2; *Design* refers to the total design time including sketching and painting; *Cut* is the typical time for cross-sectioning; *Size* is the total data size of the DSs model.

assigning translucent material information to DSs in addition to colors, but realistic rendering of such highly translucent and heterogeneous materials is still a challenge.

Chapter 7

Conclusion

In this chapter, we conclude this thesis by first summarizing our contributions, then discussing the limitation of our methods, and finally depicting future directions for further investigation.

7.1 Summary of Contributions

The goal of this thesis is to allow the user to interactively and efficiently create volumetric models of natural objects that have complex internal structures using compact and consistent representations. We identified two inherent difficulties in volumetric modeling. The first difficulty is that the user cannot specify volumetric information directly, since volumetric information cannot be perceived directly by humans. The second difficulty is that the computational cost increases much more rapidly in volumetric modeling than in surface modeling. To deal with these difficulties, we proposed two general principles for volumetric modeling. The first principle is to let the user specify some attribute values such as color and orientation on some surfaces that are characteristic to the object's internal structures, which we refer to as *characterizing surfaces*, and then propagate such values specified on the characterizing surfaces through the volume (**Principle I**). The second principle is to exploit the regularities of the object's internal structures to achieve compact representations and fast algorithms (**Principle II**). In order to comprehensively discuss regularities of objects' internal structures, we classified solid textures into different types based on the scale and anisotropy of the structures represented by solid textures. We further realized that raster-based and vector-based representations are better suited for modeling smaller and larger scale structures, respectively, corresponding to different types of solid textures in our classification. Based on this analysis, we proposed two methods for volumetric modeling,

one in the raster-based approach and the other in the vector-based approach, both of which make use of our two principles for volumetric modeling.

For the raster-based approach, we proposed a method to represent detailed internal structures of objects using anisotropic solid textures. The basic idea is to repeatedly paste patches of an input anisotropic solid texture to a model's interior according to a user-specified volumetric orientation field, filling the model with the solid texture. We proposed two sketch-based user interfaces that allow the user to quickly create volumetric orientation fields inside the model. Our user interfaces for modeling volumetric orientation fields are based on **Principle I** in that either the model's exterior surface or the model's depth layers are regarded as characterizing surfaces, and the user draws strokes on these surfaces to specify desired orientations which are then propagated through the volume. **Principle II** is also utilized in our raster-based method in that we regard the repetition of the same detailed structures as the structural regularity, which allows us to reuse the same texture data leading to a compact representation, and to use a patch-based texture synthesis algorithm which is generally much faster than voxel-based texture synthesis algorithms.

For the vector-based approach, we proposed a method to represent a smooth volumetric color distribution using a set of colored 3D surfaces. The smooth volumetric color distribution is obtained by diffusing colors from the surfaces over the volume. This method utilizes **Principle I** in that the boundaries of sharp color transitions in the volume are regarded as characterizing surfaces, and user-specified colors on the characterizing surfaces are propagated through the volume to define a smooth volumetric color distribution. **Principle II** is also utilized in our vector-based method in that we regard the smoothness of the volumetric color distribution as the structural regularity, which allows us to efficiently compute the diffusion of colors only locally on cross-sectional points. This further allows us to represent a volumetric model compactly as a set of colored 3D surfaces without storing the entire volumetric color distribution.

We demonstrated that it is possible to quickly and easily create a number of volumetric models of various natural objects that have complex internal structures using our methods, suggesting the effectiveness of our two principles.

7.2 Limitations

While we have so far only demonstrated cases where our methods are successful, in this section we discuss cases where our methods are inadequate. Most of natural

Figure 7.1: Examples of natural objects that are difficult to model using our methods.

objects modeled using our methods have rotationally symmetric internal structures (e.g., kiwi fruits, carrots, trees, tomatoes, strawberries, etc.) with a few exceptions (e.g., strata, roll cakes, and kidney), although both of our raster-based and vector-based representations per se do not assume rotationally symmetric internal structures. On the contrary, volumetric modeling of objects with internal structures that are not rotationally symmetric (e.g., meat, fish, brain, tooth, etc., see Figure 7.1) using our methods proved to be either impossible or very time-consuming. We discuss the reasons below.

In the case of our raster-based method presented in Chapter 5, the system repeatedly pastes patches of a solid texture with variation level 0 or 1 to a model's interior. It turned out that textures with variation level 0 (i.e., homogeneous textures) are most of the time only useful for creating artificial models such as those shown in Figures 5.2, 5.3, and 5.4. To create more interesting models of natural objects, textures with variation level 1 (i.e., layered textures) are always used. Such layered textures can represent either planar layered structures (e.g., strata, roll cakes) or rotationally symmetric layered structures (e.g., kiwi fruits, carrots, trees, etc.). Note that the modeling user interface and the synthesis algorithm are identical for both types of structures and are unaware of rotational symmetry. It is, unfortunately, impossible to create models with more complex internal structures than such layered structures using this method.

In the case of our vector-based method presented in Chapter 6, the representation itself is general since a model simply consists of a set of colored 3D surfaces. It is therefore possible in theory to create models representing arbitrary kind of objects by placing an arbitrary number of colored 3D surfaces at arbitrary 3D locations using standard 3D modeling packages such as Autodesk Maya. Such a time-consuming modeling process is, however, undesirable for our purpose of achieving efficient volumetric modeling. Modeling 3D object's internal structures efficiently is still a challenging problem for which only a few methods have been explored [75]. In order to enable

quick and easy volumetric modeling of natural objects, we proposed our sketch-based 3D modeling interface that assumes rotational symmetry in the object's internal structures, based on our observation that many natural objects actually have rotationally symmetric internal structures.

In short, most natural objects modeled successfully using our methods have rotationally symmetric internal structures. Volumetric modeling of objects that have more complex internal structures than rotationally symmetric ones would be either impossible using our raster-based method, or very time-consuming using our vector-based method. In fact, the kidney model shown in Figure 6.25, which does not have rotationally symmetric internal structures and therefore was modeled using standard 3D modeling techniques, took much longer time to create than others, despite its relatively fewer number of surfaces. It is subject of our future work to achieve efficient volumetric modeling of objects that have more complex internal structures than rotationally symmetric ones.

7.3 Future Directions

There are a number of interesting and exciting future directions that lie beyond this thesis as described below, in addition to the one mentioned in the previous section. We believe the field of volumetric modeling will receive even more attention in the future.

7.3.1 Combining Raster-Based and Vector-Based Approaches

As described earlier, our raster-based and vector-based methods are suited for representing detailed and global structures, respectively, and thus are complementary to each other. It would be desirable if we could combine these two approaches to enable a hybrid approach.

One possible idea is to incorporate detailed structures represented by our raster-based method into global structures represented by our vector-based method. Since internal textures of many natural objects vary spatially (e.g., a tomato has very different textures at its fleshy part and the part close to its stem end), it would be necessary to extend our raster-based approach to using continuously morphable textures instead of using the same texture as in the original, such that the transitions of the synthesized textures will follow the smooth volumetric field defined by our vector-based method. For interpolating textures, Matusik et al.'s technique to define a continuous space of morphable textures [64] could be utilized. An interesting question then would be how

to define an appropriate space of morphable textures suited for our purpose of volumetric modeling. Another important question is how to integrate our raster-based method into our vector-based method while ensuring the compactness achieved by not storing a tetrahedral mesh in our vector-based method.

7.3.2 Use of Scanned Volume Data

This thesis focused on creating volumetric models from scratch, rather than from volume data obtained using scanning methods such as CT or MRI. This is because such volume data are currently not yet very abundant, and it seems more desirable for artists to be able to create volumetric models without relying on expensive scanning facilities. In the future, however, it is likely that such scanning facilities become less expensive and more common, and thus volumetric modeling techniques based on scanned volume data will be needed.

Recently, Wang et al. [110] used segmented volume data of human brain to extract information about geometry for multi-scale volumetric modeling. We could use the same approach for our vector-based method. However, information about color cannot be obtained from CT or MRI volume data, as each voxel has only a scalar intensity value. In Wang et al.'s approach [110], material properties for each segmented region is manually specified by the user. It is very challenging to automatically obtain information about color from CT or MRI volume data. We believe it is necessary to combine such volume data with other data sources such as photographs to estimate information about color.

Note that there is another approach to scanning volumes by directly slicing objects and taking their photographs sequentially, resulting in volume data with colors. Because of its invasive nature, this approach is currently used almost exclusively for scientific research purposes such as biology, making it less popular compared to CT and MRI. It has several limitations such as low resolution along the depth direction, the fixation process (e.g., freezing) that might affect the structure and appearance of objects, and destruction of objects' tiny structures during the cutting process. If these limitations are alleviated in the future, this approach will be viable for volumetric modeling in the context of computer graphics.

7.3.3 Physically-Based Volume Rendering

In the field of physically-based volume rendering, the main challenge nowadays is how to create volumetric models to render. Recently Zhao et al. [121] presented a

method to utilize very high-resolution volume data of pieces of fabric obtained by micro CT scanning to achieve extremely realistic physically-based volume rendering of thick cloths. As mentioned above, CT volume contain only scalar intensity values, and cannot be used directly for physically-based rendering. In order to deal with this issue, after some preprocessing that reduces noise and extracts local fiber orientation, they estimate material parameters for fabrics (i.e., albedo and standard deviation of flake distribution) by solving a radically simplified inverse rendering problem.

One limitation of their approach is that most of the parameters (other than the fiber density and orientation) are assumed to be globally constant, which means that the fabric consists of the same kind of tiny fibers distributed with varying density and orientation. This assumption does not hold true for other types of fabrics (or natural objects in general) that contain various materials with different optical parameters. The problem of how to construct heterogeneous volumetric models suited for physically-based volume rendering remains a challenge.

7.3.4 Biologically-Motivated Procedural Modeling

In Chapter 6, we presented a simple sketch-based user interface for modeling internal structures of natural objects such as fruits and vegetables. This approach, however, did not allow us to easily create realistic geometries of natural objects containing complex and intricate internal structures, and required time-consuming trial-and-error artistic experiments.

An interesting approach to this problem would be to use biologically-motivated computational models such as L-systems [86] which has been successfully used for modeling trees [78] and other kinds of organic structures such as Purkinje fibers [41]. To our knowledge, there is no previous work that applies the same approach to volumetric modeling, which seems an interesting area for future research. In particular, it would be interesting from the scientific perspective as well if we could simulate the growth processes of volumetric internal structures of natural objects.

7.3.5 Techniques for Interacting with Volumetric Models

In this thesis we have focused on creating volumetric models while letting the user browse them by simply cutting. We believe, however, that techniques for interacting with volumetric models are also equally important for providing the user with more rich and intuitive experiences. Examples of such interaction techniques include more sophisticated cutting [73, 65, 13], physically accurate deformations accounting for ma-

terial heterogeneity [28], and cooking simulation that handles change of appearances due to heating [52]. Note that consistent volumetric information created using our approaches is essential for realizing these applications. Although several works exist as above, the area of interaction techniques for volumetric models has not yet been fully explored, and we believe it deserves further research.

References

[1] Takashi Ashihara, Tsunetoyo Namba, Takanori Ikeda, Makoto Ito, Masahiko Kinoshita, and Kazuo Nakazawa. Breakthrough waves during ventricular fibrillation depend on the degree of rotational anisotropy and the boundary conditions: A simulation study. *Journal of Cardiovascular Electrophysiology*, 12(3):312–322, 2001. 51

[2] Takashi Ashihara, Tsunetoyo Namba, Takenori Yao, Tomoya Ozawa, Ayaka Kawase, Takenori Ikeda, Kazuo Nakazawa, and Makoto Ito. Vortex cordis as a mechanism of postshock activation: Arrhythmia induction study using a bidomain model. *Journal of Cardiovascular Electrophysiology*, 14(3):295–302, 2003. 51

[3] Hedlena Bezerra, Elmar Eisemann, Doug DeCarlo, and Joëlle Thollot. Diffusion constraints for vector graphics. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, pages 35–42, 2010. 83

[4] Bernd Bickel, Moritz Bächer, Miguel A. Otaduy, Wojciech Matusik, Hanspeter Pfister, and Markus Gross. Capture and modeling of non-linear heterogeneous soft tissue. *ACM Trans. Graph.*, 28(3):89:1–89:9, 2009. 2

[5] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3D faces. In *Proc. SIGGRAPH 99*, pages 187–194, 1999. 96

[6] David Bommes, Henrik Zimmer, and Leif Kobbelt. Mixed-integer quadrangulation. *ACM Trans. Graph.*, 28(3):77:1–77:10, 2009. 39

[7] John C. Bowers, Jonathan Leahey, and Rui Wang. A ray tracing approach to diffusion curves. *Computer Graphics Forum*, 30(4):1345–1352, 2011. 83

[8] S. Bruckner and M. E. Groller. Style transfer functions for illustrative volume rendering. *Computer Graphics Forum*, 26(3):715–724, 2007. 35, 36

[9] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986. 90

[10] Guoning Chen, Gregory Esch, Peter Wonka, Pascal Müller, and Eugene Zhang. Interactive procedural street modeling. *ACM Trans. Graph.*, 27(3):103:1–103:10, 2008. 39

[11] Yanyun Chen, Xin Tong, Jiaping Wang, Stephen Lin, Baining Guo, and Heung-Yeung Shum. Shell texture functions. *ACM Trans. Graph.*, 23(3):343–353, 2004. 34, 35

[12] D. Cline, S. Jeschke, K. White, A. Razdan, and P. Wonka. Dart throwing on surfaces. *Computer Graphics Forum*, 28(4):1217–1226, 2009. 93

[13] Carlos Correa, Deborah Silver, and Min Chen. Feature aligned volume manipulation for illustration and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1069–1076, 2006. 35, 36, 107

[14] Keenan Crane, Mathieu Desbrun, and Peter Schroder. Trivial connections on discrete surfaces. *Computer Graphics Forum*, 29(5):1525–1533, 2010. 39

[15] Barbara Cutler, Julie Dorsey, Leonard McMillan, Matthias Müller, and Robert Jagnow. A procedural approach to authoring solid models. *ACM Trans. Graph.*, 21(3):302–311, 2002. 15, 29, 31

[16] Timothy A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):165–195, 2004. 43

[17] Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Proc. SIGGRAPH 97*, pages 361–368, 1997. 96

[18] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proc. SIGGRAPH 99*, pages 317–324, 1999. 43

[19] J.M. Dischler and D. Ghazanfarpour. A survey of 3d texturing. *Computers & Graphics*, 25(1):135–151, 2001. 23

[20] J.M. Dischler, D. Ghazanfarpour, and R. Freydier. Anisotropic solid texture synthesis using orthogonal 2d views. *Computer Graphics Forum*, 17(3):87–95, 1998. 23, 120

[21] Yue Dong, Sylvain Lefebvre, Xin Tong, and George Drettakis. Lazy solid texture synthesis. *Computer Graphics Forum*, 27(4):1165–1174, 2008. 26, 29, 41, 98, 120

[22] Craig Donner and Henrik Wann Jensen. Light diffusion in multi-layered translucent materials. *ACM Trans. Graph.*, 24(3):1032–1039, 2005. 35

[23] Craig Donner, Tim Weyrich, Eugene d'Eon, Ravi Ramamoorthi, and Szymon Rusinkiewicz. A layered, heterogeneous reflectance model for acquiring and rendering human skin. *ACM Trans. Graph.*, 27(5):140:1–140:12, 2008. 35

[24] Julie Dorsey, Alan Edelman, Henrik Wann Jensen, Justin Legakis, and Hans Køhling Pedersen. Modeling and rendering of weathered stone. In *Proc. SIGGRAPH 99*, pages 225–234, 1999. 34, 35

[25] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steve Worley. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann, 3 edition, 2002. 17

[26] James H. Elder. Are edges incomplete? *International Journal of Computer Vision*, 34(2/3):97–122, 1999. 80

[27] Zeev Farbman, Gil Hoffer, Yaron Lipman, Daniel Cohen-Or, and Dani Lischinski. Coordinates for instant image cloning. *ACM Trans. Graph.*, 28(3):67:1–67:9, 2009. 85

[28] François Faure, Benjamin Gilles, Guillaume Bousquet, and Dinesh K. Pai. Sparse meshless models of complex deformable solids. *ACM Trans. Graph.*, 30(4):73:1–73:10, 2011. 108

[29] James Ferguson. Multivariable curve interpolation. *J. ACM*, 11(2):221–228, 1964. 79

[30] Mark Finch, John Snyder, and Hugues Hoppe. Freeform vector graphics with controlled thin-plate splines. *ACM Trans. Graph.*, 30(6):166:1–166:10, 2011. 83

[31] Matthew Fisher, Peter Schröder, Mathieu Desbrun, and Hugues Hoppe. Design of tangent vector fields. *ACM Trans. Graph.*, 26(3):56:1–56:56:9, 2007. 39

[32] Michael S. Floater. Mean value coordinates. *Comput. Aided Geom. Des.*, 20(1):19–27, 2003. 85

[33] Hongbo Fu, Yichen Wei, Chiew-Lan Tai, and Long Quan. Sketching hairstyles. In *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling*, pages 31–36, 2007. 39, 40

[34] D. Ghazanfarpour and J.M. Dischler. Spectral analysis for automatic 3-d texture generation. *Computers & Graphics*, 19(3):413–422, 1995. 22, 23, 24

[35] D. Ghazanfarpour and J.M. Dischler. Generation of 3d texture using multiple 2d models analysis. *Computer Graphics Forum*, 15(3):311–323, 1996. 23, 120

[36] R. Haraguchi, T. Igarashi, S. Owada, T. Yao, T. Namba, T. Ashihara, T. Ikeda, and K. Nakazawa. Electrophysiological heart simulator equipped with sketchy 3-d modeling. In *Complex Medical Engineering*, 2005. 51

[37] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *Proc. SIGGRAPH 95*, SIGGRAPH '95, pages 229–238, New York, NY, USA, 1995. ACM. 21, 23, 24, 26

[38] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *Proc. SIGGRAPH 2000*, SIGGRAPH '00, pages 517–526, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 38, 39

[39] Jin Huang, Yiying Tong, Hongyu Wei, and Hujun Bao. Boundary aligned smooth 3d cross-frame field. *ACM Trans. Graph.*, 30(6):143:1–143:8, 2011. 41, 42

[40] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. *ACM Trans. Graph.*, 24(3):1134–1141, 2005. 91

[41] Takashi Ijiri, Takashi Ashihara, Takeshi Yamaguchi, Kenshi Takayama, Takeo Igarashi, Tatsuo Shimada, Tsunetoyo Namba, Ryo Haraguchi, and Kazuo Nakazawa. A procedural method for modeling the purkinje fibers of the heart. *The Journal of Physiological Sciences*, 58(7):481–486, 2008. 107

[42] Takashi Ijiri, Kenshi Takayama, Hideo Yokota, and Takeo Igarashi. Procdef: Local-to-global deformation for skeleton-free character animation. *Computer Graphics Forum*, 28(7):1821–1828, 2009. 6, 53, 55

[43] Robert Jagnow, Julie Dorsey, and Holly Rushmeier. Stereological techniques for solid textures. *ACM Trans. Graph.*, 23(3):329–335, 2004. 23

[44] Wenzel Jakob, Adam Arbree, Jonathan T. Moon, Kavita Bala, and Steve Marschner. A radiative transfer framework for rendering materials with anisotropic structure. *ACM Trans. Graph.*, 29(4):53:1–53:13, 2010. 35

[45] Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. A practical model for subsurface light transport. In *Proc. SIGGRAPH 2001*, pages 511–518, 2001. 35

[46] S. Jeschke, D. Cline, and P. Wonka. Estimating color and texture parameters for vector graphics. *Computer Graphics Forum*, 30(2):523–532, 2011. 83

[47] Stefan Jeschke, David Cline, and Peter Wonka. A gpu laplacian solver for diffusion curves and poisson image editing. *ACM Trans. Graph.*, 28(5):116:1–116:8, 2009. 82, 83

[48] Stefan Jeschke, David Cline, and Peter Wonka. Rendering surface details with diffusion curves. *ACM Trans. Graph.*, 28(5):117:1–117:8, 2009. 83

[49] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. *SIGGRAPH Comput. Graph.*, 23(3):271–280, 1989. 34, 35

[50] James T. Kajiya and Brian P Von Herzen. Ray tracing volume densities. *SIGGRAPH Comput. Graph.*, 18(3):165–174, 1984. 34, 35

[51] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988. 74, 91

[52] Fumihiro Kato, Mina Shiina, Takashi Tokizaki, Hironori Mitake, Takafumi Aoki, and Shoichi Hasegawa. Culinary art designer. In *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, pages 398–398, 2008. 108

[53] Johannes Kopf, Chi-Wing Fu, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski, and Tien-Tsin Wong. Solid texture synthesis from 2d exemplars. *ACM Trans. Graph.*, 26(3):2:1–2:9, 2007. 25, 26, 120

[54] Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. Texture optimization for example-based synthesis. *ACM Trans. Graph.*, 24(3):795–802, 2005. 26

[55] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D.S. Ebert, J.P. Lewis, K. Perlin, and M. Zwicker. A survey of procedural noise functions. *Computer Graphics Forum*, 29(8):2579–2600, 2010. 17

[56] Ares Lagae, Sylvain Lefebvre, George Drettakis, and Philip Dutré. Procedural noise using sparse gabor convolution. *ACM Trans. Graph.*, 28(3):54:1–54:10, 2009. 17, 83

[57] Yu-Kun Lai, Shi-Min Hu, and Ralph R. Martin. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM Trans. Graph.*, 28(3):85:1–85:8, 2009. 80, 81

[58] Seungkyu Lee, R.T. Collins, and Yanxi Liu. Rotation symmetry group detection via frequency analysis of frieze-expansions. In *Proceedings of CVPR 2008*, pages 1–8, 2008. 90

[59] Sylvain Lefebvre and Hugues Hoppe. Parallel controllable texture synthesis. *ACM Trans. Graph.*, 24(3):777–786, 2005. 26

[60] Yaron Lipman, Johannes Kopf, Daniel Cohen-Or, and David Levin. GPU-assisted positive mean value coordinates for mesh deformations. In *Proc. Eurographics Symposium on Geometry Processing 2007*, pages 117–123, 2007. 78, 84, 85, 87

[61] Yang Liu, Weiwei Xu, Jun Wang, Lifeng Zhu, Baining Guo, Falai Chen, and Guoping Wang. General planar quadrilateral mesh design using conjugate direction field. *ACM Trans. Graph.*, 30(6):140:1–140:10, 2011. 39

[62] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, 1987. 65

[63] Aidong Lu, David S. Ebert, Wei Qiao, Martin Kraus, and Benjamin Mora. Volume illustration using wang cubes. *ACM Trans. Graph.*, 26(2):11:1–11:18, 2007. 35

[64] Wojciech Matusik, Matthias Zwicker, and Frédo Durand. Texture design using a simplicial complex of morphable textures. *ACM Trans. Graph.*, 24(3):787–794, 2005. 105

[65] Michael J. McGuffin, Liviu Tancau, and Ravin Balakrishnan. Using deformations for browsing volumetric data. In *Proc. IEEE VIS'03*, page 53, 2003. 35, 107

[66] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM Trans. Graph.*, 24(3):471–478, 2005. 54

[67] K. Nakazawa, T. Suzuki, T. Ashihara, M. Inagaki, T. Namba, T. Ikeda, and R. Suzuki. Computational analysis and visualization of spiral wave reentry in a virtual heart model. In T. Yamaguchi, editor, *Clinical application of computational mechanics to the cardiovascular system*. Springer, 2000. 51

[68] Andrew Nealen. *Interfaces and Algorithms for the Creation, Modification, and Optimization of Surface Meshes*. PhD thesis, Technische Universität Berlin, October 2007. 42

[69] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Fibermesh: designing freeform surfaces with 3d curves. *ACM Trans. Graph.*, 26(3):41:1–41:9, 2007. 66

[70] P. M. Nielsen, I. J. Le Grice, B. H. Smaill, and P. J. Hunter. Mathematical model of geometry and fibrous structure of the heart. *American Journal of Physiology - Heart and Circulatory Physiology*, 260(4):H1365–H1378, 1991. 47

[71] J. D. Northrup and Lee Markosian. Artistic silhouettes: a hybrid approach. In *Proc. NPAR 2000*, pages 31–37, 2000. 96, 98

[72] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: a vector representation for smooth-shaded images. *ACM Trans. Graph.*, 27(3):92:1–92:8, 2008. 77, 80, 93, 94

[73] Shigeru Owada, Ayumi Akaboya, Frank Nielsen, Fusako Kusunoki, and Takeo Igarashi. Kiru ("cut", in japanese). In *12th Workshop on Interactive Systems and Software (WISS 2004)*, pages 1–4, 2004. 2, 107

[74] Shigeru Owada, Takahiro Harada, Philipp Holzer, and Takeo Igarashi. Volume painter: Geometry-guided volume modeling by sketching on the cross-section. In *Proceedings of Eurographics Symposium on Sketcy-Based Interfaces and Modeling*, pages 9–16, 2008. 27, 28, 29, 98

[75] Shigeru Owada, Frank Nielsen, Kazuo Nakazawa, and Takeo Igarashi. A sketching interface for modeling the internal structures of 3d shapes. In *Proceedings of the 3rd international conference on Smart Graphics*, pages 49–57, 2003. 104

[76] Shigeru Owada, Frank Nielsen, Makoto Okabe, and Takeo Igarashi. Volumetric illustration: designing 3d models with internal textures. *ACM Trans. Graph.*, 23(3):322–328, 2004. 18, 19, 39, 40, 72

[77] Jonathan Palacios and Eugene Zhang. Rotational symmetry field design on surfaces. *ACM Trans. Graph.*, 26(3):55:1–55:10, 2007. 39

[78] Wojciech Palubicki, Kipp Horel, Steven Longay, Adam Runions, Brendan Lane, Radomír Měch, and Przemyslaw Prusinkiewicz. Self-organizing tree models for image synthesis. *ACM Trans. Graph.*, 28(3):58:1–58:10, 2009. 107

[79] Darwyn R. Peachey. Solid texturing of complex surfaces. *SIGGRAPH Comput. Graph.*, 19(3):279–286, 1985. 16

[80] K. Perlin and E. M. Hoffert. Hypertexture. *SIGGRAPH Comput. Graph.*, 23(3):253–262, 1989. 34, 35

[81] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, 1985. 16, 17, 96, 98, 125

[82] Nico Pietroni, Paolo Cignoni, Miguel Otaduy, and Roberto Scopigno. Solid-texture synthesis: A survey. *IEEE Computer Graphics and Applications*, 30(4):74–89, 2010. 16

[83] Nico Pietroni, Miguel A. Otaduy, Bernd Bickel, Fabio Ganovelli, and Markus Gross. Texturing internal surfaces from a few cross sections. *Computer Graphics Forum*, 26(3):637–644, 2007. 19, 20, 72

[84] Pixar. Ratatouille (motion picture), 2007. 2

[85] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In *Proc. SIGGRAPH 2000*, pages 465–470, 2000. 38, 59, 60, 65, 66, 69, 74

[86] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants (Virtual Laboratory)*. Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 1990. 107

[87] Xuejie Qin and Yee-Hong Yang. Aura 3d textures. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):379–389, 2007. 26, 120

[88] Nicolas Ray, Bruno Vallet, Laurent Alonso, and Bruno Levy. Geometry-aware direction field processing. *ACM Trans. Graph.*, 29(1):1:1–1:11, 2009. 39

[89] Nicolas Ray, Bruno Vallet, Wan Chiu Li, and Bruno Lévy. N-symmetry direction field design. *ACM Trans. Graph.*, 27(2):10:1–10:13, 2008. 39

[90] T. Ritschel, T. Engelhardt, T. Grosch, H.-P. Seidel, J. Kautz, and C. Dachsbacher. Micro-rendering for scalable, parallel final gathering. *ACM Trans. Graph.*, 28(5):132:1–132:8, 2009. 87

[91] Alec R. Rivers and Doug L. James. Fastlsm: fast lattice shape matching for robust real-time deformation. *ACM Trans. Graph.*, 26(3):82:1–82:6, 2007. 54

[92] Michael P. Salisbury, Michael T. Wong, John F. Hughes, and David H. Salesin. Orientable textures for image-based pen-and-ink illustration. In *Proc. SIGGRAPH 97*, pages 401–406, 1997. 38, 39

[93] Jonathan Richard Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry*, 22(1–3):21–74, 2002. 86

[94] Hang Si. On refinement of constrained delaunay tetrahedralizations. In *Proc. of the 15th International Meshing Roundtable*, pages 509–528, 2006. 47, 87

[95] Cyril Soler, Marie-Paule Cani, and Alexis Angelidis. Hierarchical pattern mapping. *ACM Trans. Graph.*, 21(3):673–680, 2002. 74

[96] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 175–184, 2004. 42

[97] Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. Image vectorization using optimized gradient meshes. *ACM Trans. Graph.*, 26(3):11:1–11:7, 2007. 79, 80, 81

[98] Kenshi Takayama, Takashi Ashihara, Takashi Ijiri, Takeo Igarashi, Ryo Haraguchi, and Kazuo Nakazawa. A sketch-based interface for modeling myocardial fiber orientation that considers the layered structure of the ventricles. *The Journal of Physiological Sciences*, 58(7):487–492, 2008. 6

[99] Kenshi Takayama and Takeo Igarashi. Layered solid texture synthesis from a single 2d exemplar. In *ACM SIGGRAPH 2009 posters*, 2009. 7

[100] Kenshi Takayama, Takeo Igarashi, Ryo Haraguchi, and Kazuo Nakazawa. A sketch-based interface for modeling myocardial fiber orientation. In *Proceedings of the 8th international symposium on Smart Graphics*, pages 1–9, Berlin, Heidelberg, 2007. Springer-Verlag. 6

[101] Kenshi Takayama, Makoto Okabe, Takashi Ijiri, and Takeo Igarashi. Lapped solid textures: Filling a model with anisotropic textures. *ACM Trans. Graph.*, 27(3):53:1–53:9, 2008. 6

[102] Kenshi Takayama, Ryan Schmidt, Karan Singh, Takeo Igarashi, Tamy Boubekeur, and Olga Sorkine. Geobrush: Interactive mesh geometry cloning. *Computer Graphics Forum*, 30(2):613–622, 2011.

[103] Kenshi Takayama, Olga Sorkine, Andrew Nealen, and Takeo Igarashi. Volumetric modeling with diffusion surfaces. *ACM Trans. Graph.*, 29(6):180:1–180:8, 2010. 6

[104] Xin Tong, Jiaping Wang, Stephen Lin, Baining Guo, and Heung-Yeung Shum. Modeling and rendering of quasi-homogeneous materials. *ACM Trans. Graph.*, 24(3):1054–1061, 2005. 34, 35

[105] G. M. Treece, R. W. Prager, and A. H. Gee. Regularised marching tetrahedra: improved iso-surface extraction. *Computers & Graphics*, 23(4):583–598, 1999. 50, 65

[106] Greg Turk. Texture synthesis on surfaces. In *Proc. SIGGRAPH 2001*, pages 347–354, 2001. 18, 38, 39

[107] Greg Turk and James F. O'Brien. Shape transformation using variational implicit functions. In *Proc. SIGGRAPH 99*, pages 335–342, 1999. 42, 50, 65

[108] Nobuyuki Umetani, Kenshi Takayama, Jun Mitani, and Takeo Igarashi. A responsive finite element method to aid interactive geometric modeling. *IEEE Computer Graphics and Applications*, 31(5):43–53, 2011.

[109] Jiaping Wang, Shuang Zhao, Xin Tong, Stephen Lin, Zhouchen Lin, Yue Dong, Baining Guo, and Heung-Yeung Shum. Modeling and rendering of heterogeneous translucent materials using the diffusion equation. *ACM Trans. Graph.*, 27(1):9:1–9:18, 2008. 35

[110] Lvdi Wang, Yizhou Yu, Kun Zhou, and Baining Guo. Multiscale vector volumes. *ACM Trans. Graph.*, 30(6):167:1–167:8, 2011. 15, 32, 34, 84, 106

[111] Lvdi Wang, Kun Zhou, Yizhou Yu, and Baining Guo. Vector solid textures. *ACM Trans. Graph.*, 29(4):86:1–86:8, 2010. 15, 31, 84, 98

[112] Li-Yi Wei. Texture synthesis from multiple sources. In *ACM SIGGRAPH 2003 Sketches*, 2003. 25, 120

[113] Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. State of the art in example-based texture synthesis. In *Eurographics '09 State of the Art Reports (STARs)*. Eurographics, 2009. 21

[114] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proc. SIGGRAPH 2000*, pages 479–488, 2000. 18

[115] H. Winnemoller, A. Orzan, L. Boissieux, and J. Thollot. Texture design and draping in 2d images. *Computer Graphics Forum*, 28(4):1091–1099, 2009. 83

[116] Steven Worley. A cellular texture basis function. In *Proc. SIGGRAPH 96*, pages 291–294, 1996. 17

[117] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph.*, 23(3):644–651, 2004. 42

[118] E. Zhang, J. Hays, and G. Turk. Interactive tensor field design and visualization on surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):94–107, 2007. 39

[119] Guo-Xin Zhang, Song-Pei Du, Yu-Kun Lai, Tianyun Ni, and Shi-Min Hu. Sketch guided solid texturing. *Graphical Models*, 73(3):59–73, 2011. 40, 41

[120] Guo-Xin Zhang, Yu-Kun Lai, and Shi-Min Hu. Efficient synthesis of gradient solid textures. Technical report, Tsinghua University, 2011. 84

[121] Shuang Zhao, Wenzel Jakob, Steve Marschner, and Kavita Bala. Building volumetric appearance models of fabric using micro ct imaging. *ACM Trans. Graph.*, 30(4):44:1–44:10, 2011. 35, 106

Appendix A

Methods for Creating Solid Texture Exemplars

In this appendix, we describe methods for creating solid texture exemplars that can be used by our raster-based volumetric modeling method presented in Chapter 5. We describe three different methods below: synthesis of homogeneous solid textures, synthesis of layered solid textures, and manual creation of solid textures.

A.1 Synthesis of Homogeneous Solid Textures

Homogeneous solid textures (i.e., texture types 0-A, 0-B, and 0-C) can be created using one of the existing solid texture synthesis algorithms [35, 20, 112, 87, 53, 21], many of which allow the user to specify different exemplar texture images for different directions of cross-sections perpendicular to the x, y, and z coordinate axes. If we assign a single exemplar texture image to all of the three coordinate axes, the resulting texture will be of type 0-A (Figure A.1 left). If we assign an exemplar texture image to the x axis and another different exemplar texture image to the other y and z axes, the resulting texture will be of type 0-B (Figure A.1 middle). If we assign three different exemplar texture images to the three coordinate axes, the resulting texture will be of type 0-C (Figure A.1 right).

A.2 Synthesis of Layered Solid Textures

Synthesis of layered solid textures has not been addressed previously. Here we present an algorithm for synthesizing layered solid textures from 2D texture exemplars (Figure A.2) by extending existing non-parametric solid texture synthesis algorithms [53, 21].

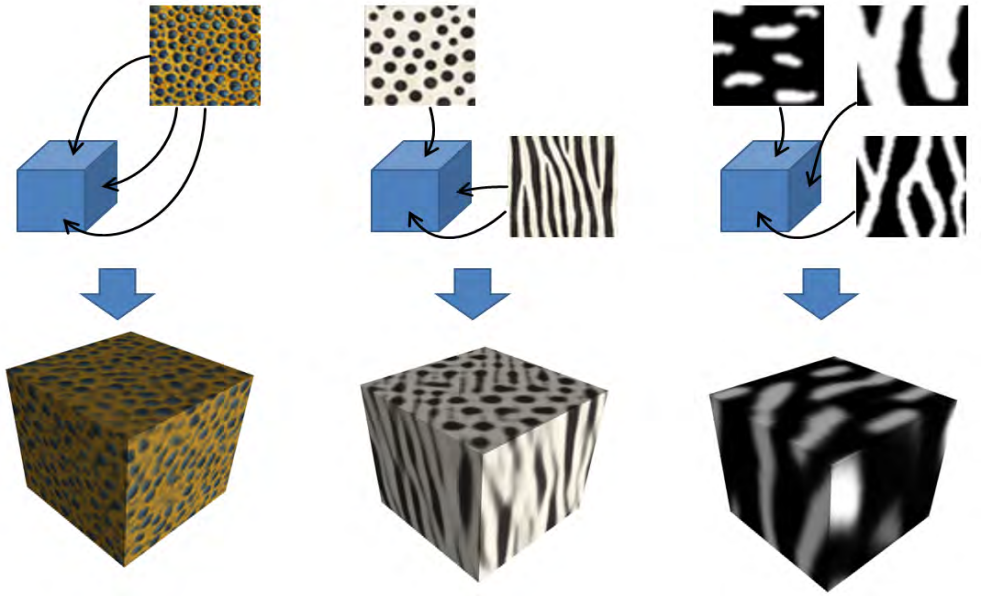

Figure A.1: Synthesis of homogeneous solid textures corresponding to texture types 0-A (left), 0-B (middle), and 0-C(right).

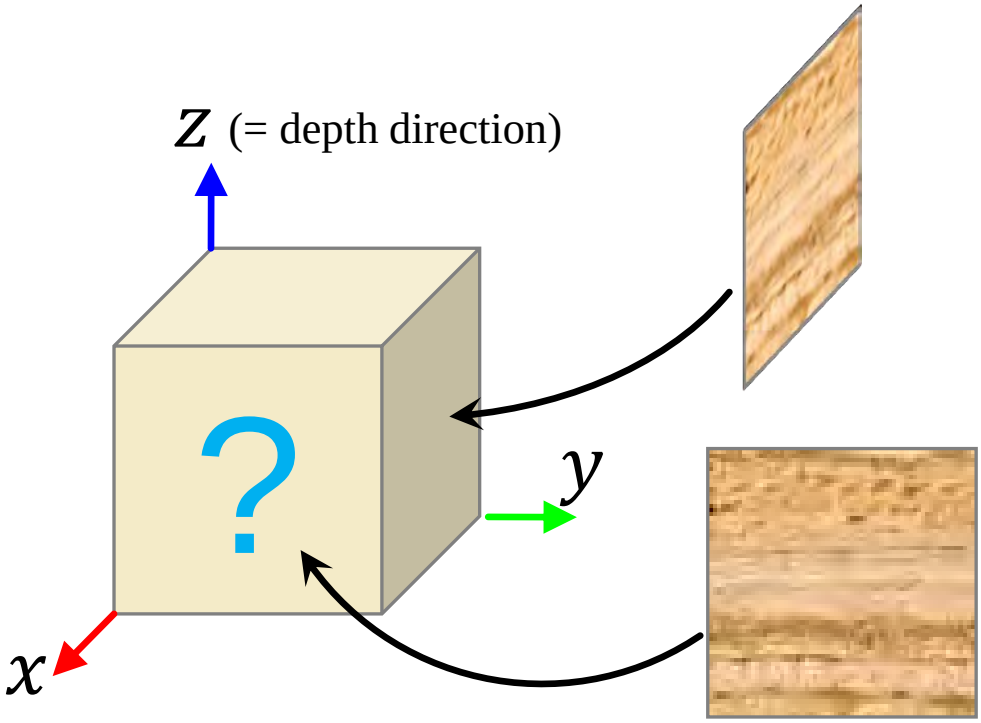

Figure A.2: The definition of a problem of synthesizing layered solid textures from 2D texture exemplars parallel to the depth direction.

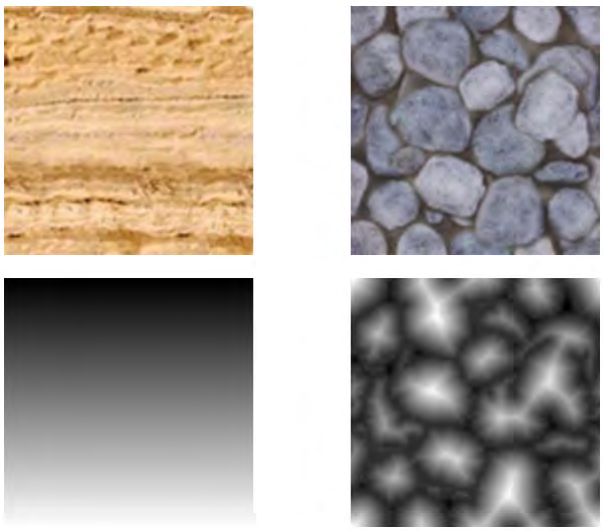

Figure A.3: Synthesis of layered solid textures by considering an additional layer depth map channel (left), similar to previous methods of using an additional feature map channel to preserve structures (right).

Our basic idea is to consider the layer depth in the exemplar texture as an additional channel (Figure A.3 left), much like feature maps are used to preserve distinct structures in the previous works (Figure A.3 right). Note that, since we do not have any 2D exemplars orthogonal to the depth direction, we perform the neighborhood matching and the texture optimization only in the other two directions. We also modify the process of initializing the synthesis volume at the coarsest level; instead of initializing each voxel with a pixel sampled from the exemplar in a totally random manner, we initialize each voxel with a pixel randomly sampled from the 2D exemplar which has the same layer depth value (Figure A.4b).

This simple extension is sufficient for the synthesis of textures of type 1-B where we have two different 2D texture exemplars of two orthogonal cross-sections. For the synthesis of textures of type 1-A, however, this simple extension leads to a severe sweeping artifact as shown in Figure A.5. The reason of this problem is that the above extension accepts a simple sweep of the single 2D exemplar in a direction orthogonal to the depth direction and oblique to both of the other two directions, as an optimized synthesis result. In other words, for each voxel, the two neighborhoods of the current synthesis volume in each of the two directions are very likely to best match with the same neighborhood of the 2D exemplar (Figure A.6).

We solve this problem using the following simple scheme. In the neighborhood

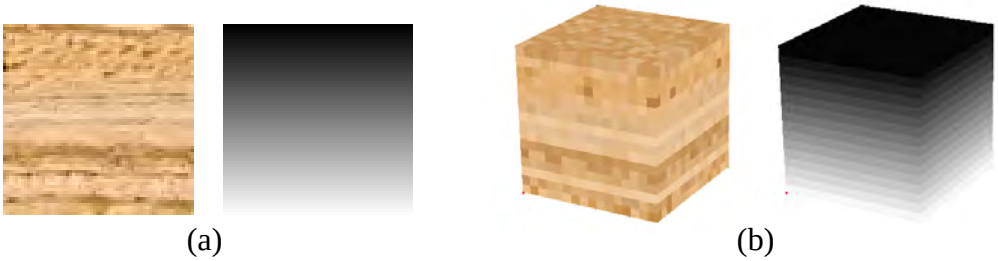

Figure A.4: Additional control map channel encoding layer depth information fed to the synthesis algorithm. (a) A layer depth control map augmented to the 2D exemplar. (b) The depth-aware initialization of the synthesis volume at the coarsest level.

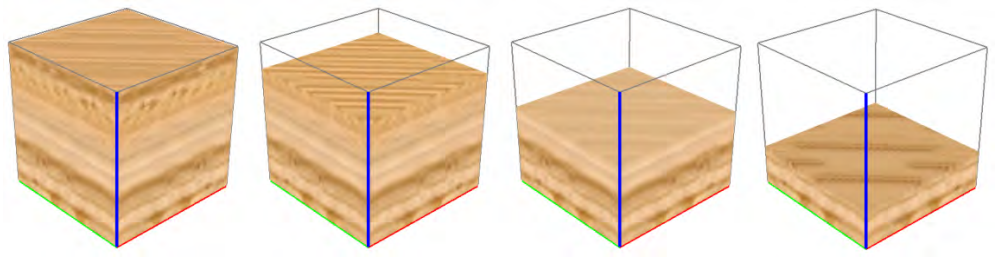

Figure A.5: A problem of severe sweeping artifact appears with our naive extension of only using additional depth map channel.

search phase, for each voxel, we collect the two best matching neighborhoods in each of the two directions (Figure A.7(middle)). If the first best matching neighborhoods in both of the two directions point to the same location in the exemplar, we select one of the two based on the matched distance and assign that pixel as the best matching neighborhood for the selected direction. The neighborhood for the other direction is chosen to the second best matching neighborhood (Figure A.7(right)).

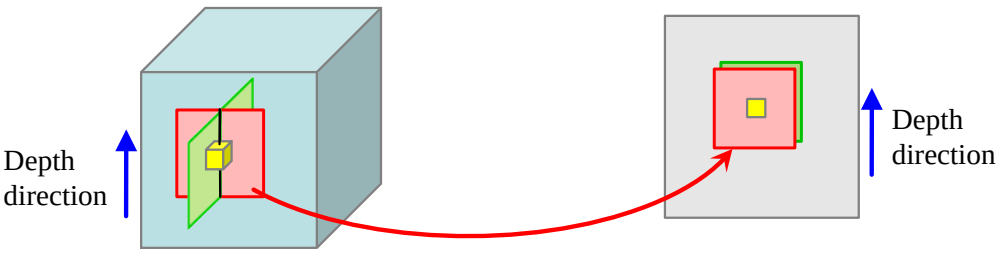

Figure A.6: The cause of the sweeping artifact. Neighborhoods of the synthesis volume in the two directions tend to match the same neighborhood in the 2D exemplar.

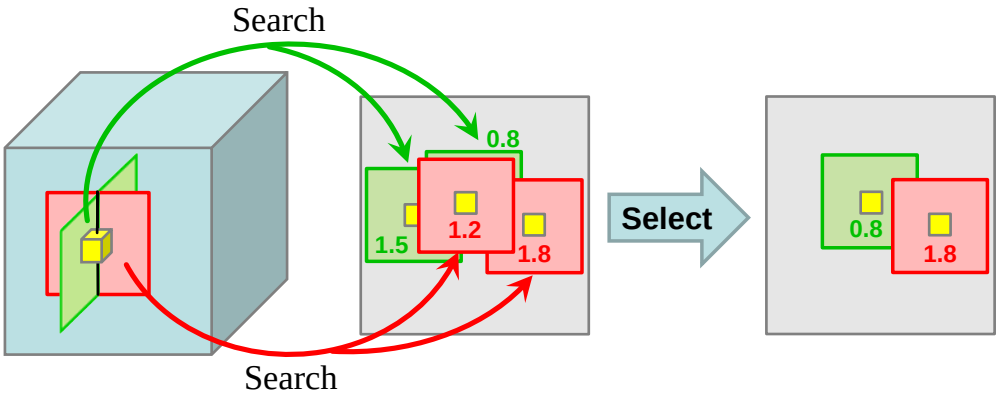

Figure A.7: A modified version of the neighborhood matching process to deal with the sweeping artifact. (middle) We collect the two best matching neighborhoods for each of the two directions. Numbers displayed near the neighborhood rectangles represent matching distances. (right) A simple selection is performed to avoid the matched neighborhoods in the two directions pointing to the identical location in the 2D exemplar.

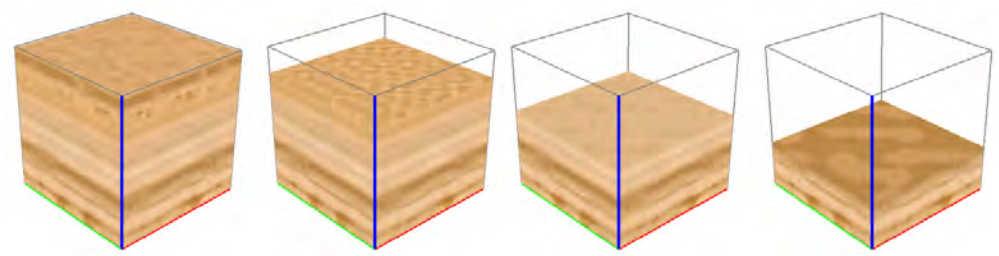

Figure A.8: The effect of our simple scheme removing the severe sweeping artifact seen in Figure A.5. Notice how the appearances on the cross-sections orthogonal to the depth direction are well synthesized solely from a 2D exemplar parallel to the depth direction.

Comparison: Figure A.8 shows that this simple scheme works well to remove the severe sweeping artifact seen in Figure A.5.

Figure A.9 left shows two examples of volumetric models created using the synthesized layered solid textures for our raster-based method presented in Chapter 5. We demonstrate the benefit of our texture synthesis algorithm by comparing it with a naive approach of using solid textures created by sweeping 2D images in the two orthogonal directions, shown in Figure A.9 right.

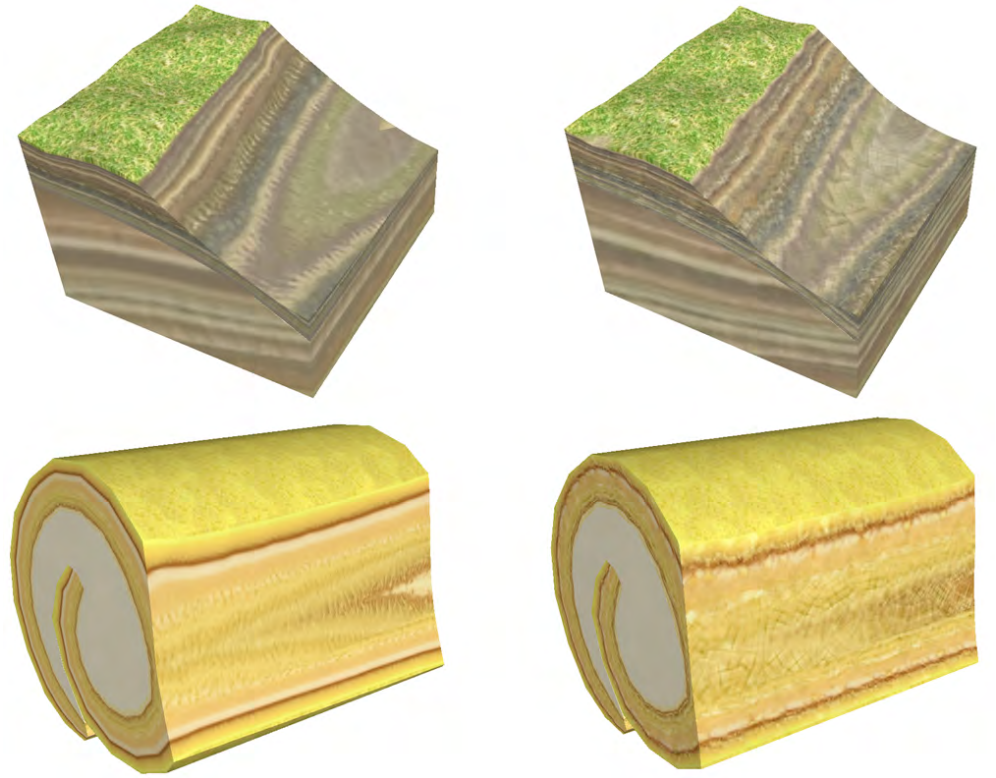

Figure A.9: (left) Models filled with layered solid textures created using our algorithm. (right) Artifacts appearing on some cross-sections when using layered solid textures created naïvely by sweeping 2D images.

A.3 Manual Creation of Solid Textures

In practice, it is also possible to create plausible solid textures without relying on solid texture synthesis techniques by, for example, simply sweeping a 2D image, adding some noise [81], and manually placing a few colored particles representing distinct elements (Figure A.10). It is unrealistic to manually create an entire volume of a large solid object, but since our raster-based method only requires small exemplars, manual creation is a viable option.

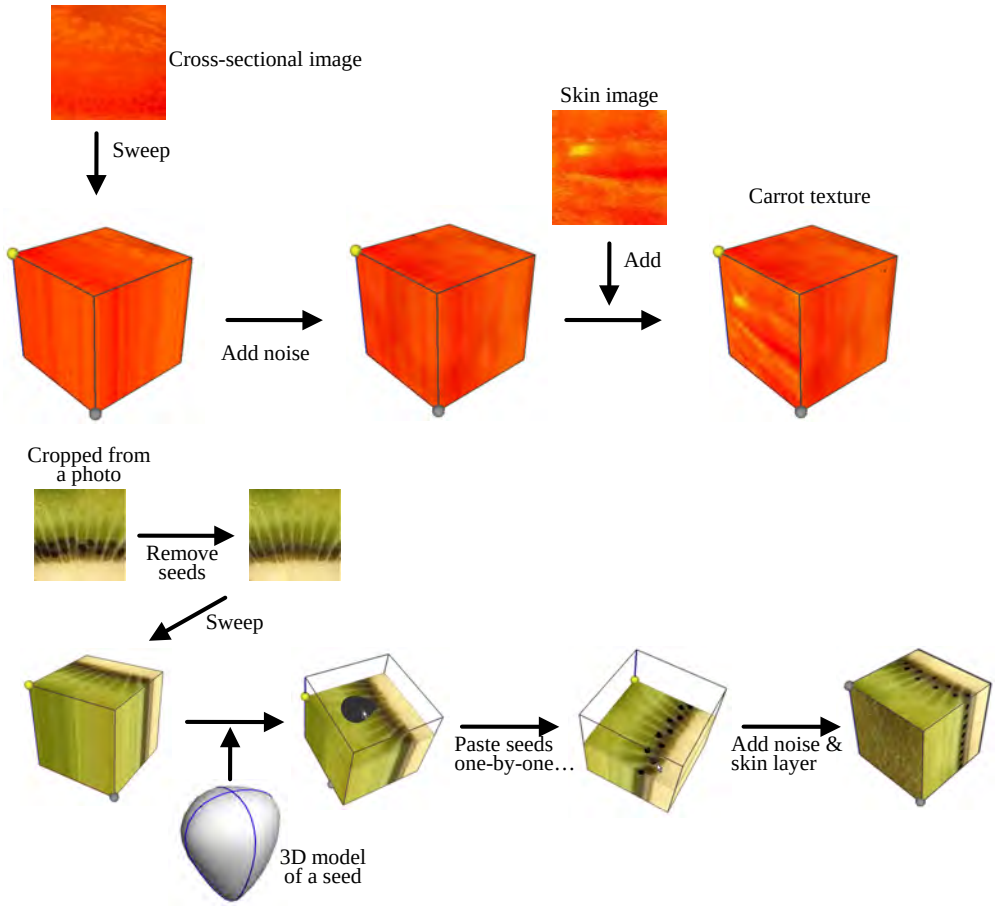

Figure A.10: Manual creation of solid texture exemplars for carrots (top) and kiwi fruits (bottom).