PRACTICAL VOLUME GRAPHICS
実用的なボリュームグラフィックス

by

Shigeru Owada
大和田 茂

A Dissertation
博士論文

Submitted to
the Graduate School of
Information Science and Technology
the University of Tokyo
on December 16th
in Partial Fulfillment of the Requirements
for the Degree of Doctor of
Information Science and Technology

Abstract

All natural objects have volumetric structure and the structure strongly controls the behavior of objects. Therefore, medical applications and simulation systems have adopted volumetric data for a long time. On the other hand, surface oriented data structure is still widely used in films, virtual reality, and entertainment industries because of the light processing cost and the easiness to create nonexistent models. Surfaces are believed to be sufficient for such applications. However, in reality, surface representation itself limits the capability of graphical systems. This dissertation proposes systems by which the user can easily and intuitively create and manipulate volumetric models since we believe lack of intuitive user interface hampers frequent use of volume data for end-user applications.

We first propose a sketch-based modeling system by which the user can easily create shapes with holes or internal cavities, under favor of scalar volume representation and intuitive gestural operations such as temporary cuts. Through this work, we try to describe how beneficial volumetric representation is, even for traditional problems.

Despite recent advances of PC ability, large data size is still an undesired aspect of volumetric graphics for such applications as games or virtual world construction, where memory consumption and interactivity is the main concern, rather than precision or consistency of the model. For such applications, visual effects and necessary computational resource trades and therefore it is important to have various choices. We propose a new pseudo-volumetric data representation that is very memory efficient and generates realistic volumetric cross-sectional images. This system generates appropriate cross-sectional images on-the-fly, by using 2D images as references for texture synthesis technique, controlled by a 3D scalar volume. The data amount of 2D images is significantly smaller than a set of voxels in 3D and the 3D scalar volume is stored in a functional form, instead of a 3D voxel array. Therefore, this representation is compact and suitable for practical use. In addition, since 2D images are ubiquitous, this system is convenient for easy creation of volumetric models.

Currently, the most important source of volumetric data is the scanning of real-world objects using CT (Computed Tomography) scanners, MR (Magnetic Resonance) imaging devices or physical slicing machines. The result is usually stored as a set of cross-sectional images, each pixel

of which is then called *a voxel* that holds the property values at a regular 3D gridpoint. To use such data for practical purposes, we usually need to carve out the interesting region (region of interest, ROI). This carving operation is called *image segmentation* and there are wide varieties of applications such as enhanced volume rendering or intelligent user interfaces. Unfortunately, image segmentation is still one of the central topics in computer vision and no automated technique is yet available, especially for volumetric datasets. Therefore, we almost always have to provide additional information to obtain successful result. We propose two system to guide this task. One is a topology selection tool for contour-based segmentation. Currently, the most robust system is to let the user observe some of cross-sectional images and delineate the contours manually, which causes a difficulty in finding a correct correspondence between contours. We propose a system to enable to enumerate all possible correspondence patterns to find the global optimum of the shape objective function, while the user can interactively modify the false result by simply selecting the desired pattern from the list. We also propose a very simple user interface called *volume catcher*, to intuitively and quickly perform volume segmentation. The user should only trace the contour of the target region in the rendered image. These works allow the novice users to utilize existing volumetric data.

We believe that volumetric graphics has huge potentials to stimulate development of new contents. At the end of this dissertation, we show an interactive content that potentially uses all of the above-mentioned system as a data source. This is an interactive cooking content that allows the user to cut foodstuff by free form strokes and a virtual knife, using a standard mouse. This work not only shows that volume graphics can produce new contents but also shows the fact that we need to carefully design user interface to intuitively manipulate volumetric data. Although we currently concentrate on cooking interaction, the idea of cutting can be extended to support any kind of volumetric data.

This dissertation explores the capability of volumetric modelers, which is indispensable in the future advance of volume graphics. It is our hope to make volumetric graphics as popular as surface graphics and to make it an indispensable component for any computer graphics applications.

Contents

Acknowledgements

I would like to thank a number of people and organizations who have supported and continue to support this work.

In particular, I owe great thanks to Takeo Igarashi, my advisor at the University of Tokyo, for providing everything necessary for this work including precious advices, prompt feedback, encouragement, collaborators and excellent research environment. I cannot be too grateful to him because it was him who taught me how fun research is, how hopeful this work is, and how I can carry it out, when I was about to drop out from research community in 2002.

Besides my advisor, there has been one collaborator who is most responsible for helping me complete this work. Frank Nielsen in Sony Computer Science Laboratories, Inc. gave me numerous useful advices and continuously encouraged me doing this work any time, anywhere. He has always been extraordinarily helpful to me and it always pushed me to do more.

I would also like to thank my lab mates for their insightful comments, advices and other forms of help. Especially, Makoto Okabe's very stable CSG library is invaluable for most of my demo programs. Moreover, Hidehiko Abe, as a TeXnician, gave me useful information for formatting this dissertation.

My previous advisor Yoshihisa Shinagawa is the person who helped me most when I first stepped toward research community. He also gave me an opportunity to stay at the University of Illinois for a while. Although the period was quite tough to me because of my immaturity, the experience was really essential.

Finally, I would like to thank my parents, Hidemi and Makiko Owada for their endless encouragement, support, and devoted love.

This work was funded in part by grants from IPA (Information-Technology Promotion Agency, Japan) and ministry of education, culture, sports, science and technology (basic research bounty(C)(2) 16500311).

Chapter 1

Introduction

1.1 Generation of computer graphics

Since the invention of the term "Computer Graphics" by William Fetter in 1960, computerized image creation system has shown tremendous progress and nowadays computer graphics (in short, "CG") is widely recognized as one of the indispensable components for scientific, industrial, educational and entertainment fields (Figure 1.1). Why is CG so important? It is because visual information occupies a huge part of our perceptual ability. We believe it originated from the fact that visual sense has been crucial for survival. Actually, visual information not only conveys the approach of danger, but also strongly affects our mental status, thereby has a strong impact for humans.

Applications of CG technologies are countless. Scientific application domains include medical, biological, chemical, zoological, archaeological, astronomical or historical fields that mainly use CG for the purpose of information visualization (Figure 1.2). Raw data obtained by capturing the real-world or produced by simulation are just sets of numbers, which cannot be understandable for the observers without appropriately being visualized by CG. Most outstanding industrial application of CG is computer-aided design or manufacturing (CAD/CAM) that is nowadays a standard way of designing products such as machine parts, architectures, or clothes. Educational fields recently show much interest in 3D CG because CG can more closely represent real-world entities than static 2D pictures or illustrations. Examples of entertainment applications include movies or games, which have already produced huge profit. They try to imitate some aspects of the real-world in entertaining manner and represent their virtual world through 3D computer graphics. CG is also

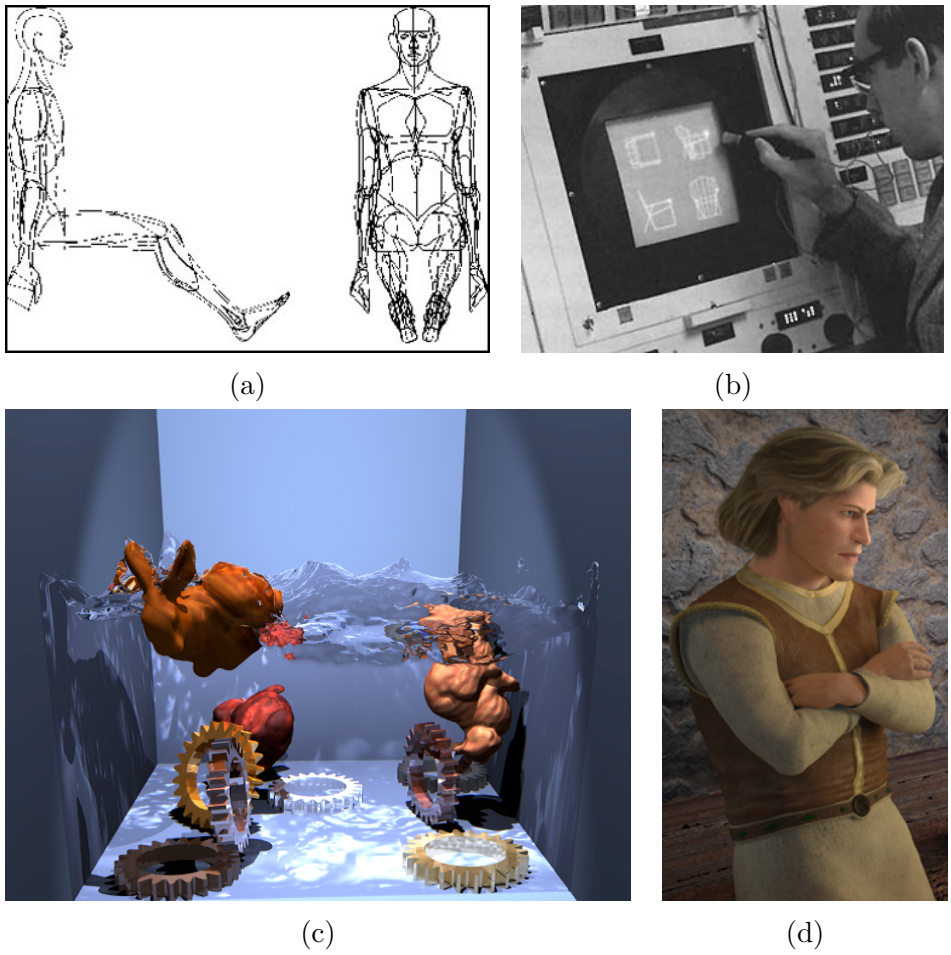


Figure 1.1: Earliest and contemporary computer generated images. (a) The first computer graphics by William Fetter in 1960. (b) Sketchpad: the first interactive computer graphics system invented by Ivan Sutherland in 1963 [135]. (c) The rendering of the result of fluid and rigid body simulation by Carlson et al.[16] (d) Rendering with global illumination by Tabellion et al.[136] (c) and (d) are taken from ACM Transactions on Graphics 23(3) (Siggraph 2004 proceedings).

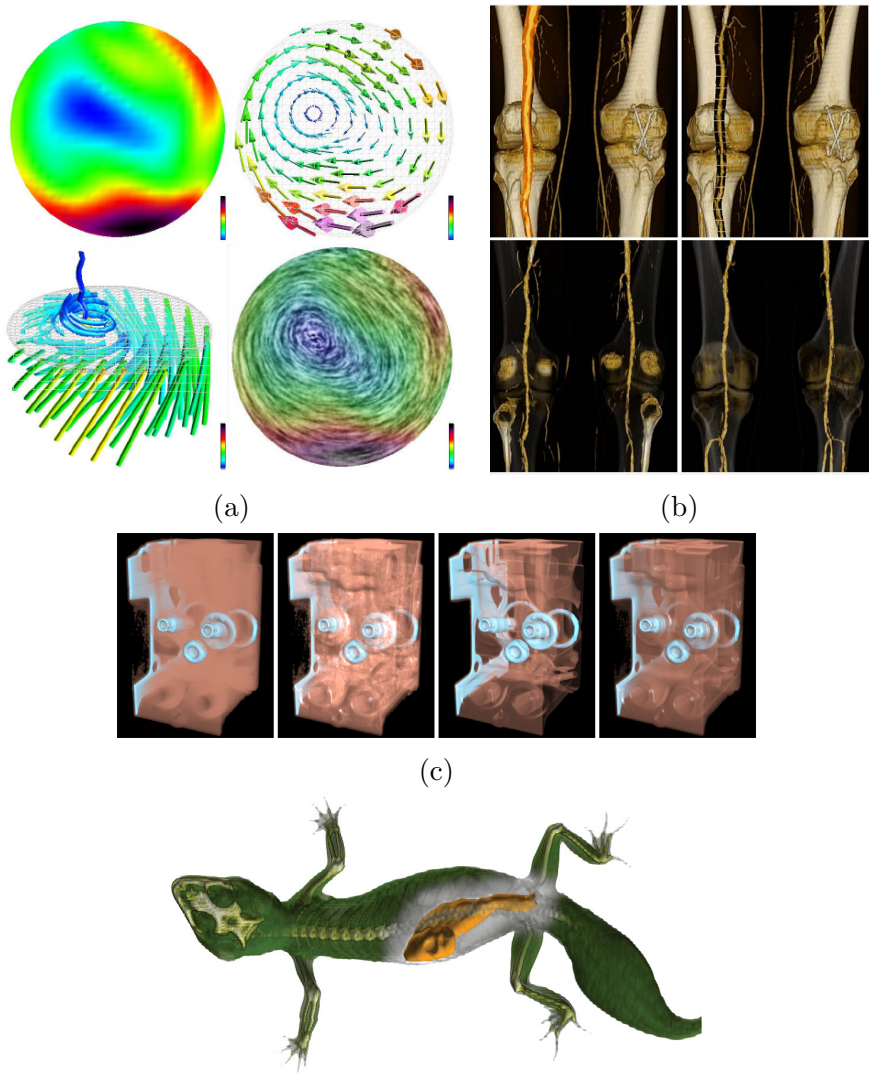


Figure 1.2: Applications of CG for visualization purpose such as (a) visualization of fluid simulation result (by Laramee et al. [84]), (b) medical visualization (by Straka et al.[133]), (c) reverse engineering (by Lum et al.[92]), (d) biological visualization with importance-driven enhancement (by Viola et al.[146]). All figures are taken from proc. IEEE Visualization 2004.

useful to cut down costs of movie production process because highly realistic shots can be taken by just sitting in front of a PC and operating on sophisticated CG softwares such as Alias MAYA [2] or Softimage [3], instead of hiring special actors, location, and huge facilities.

1.2 Shape model representation - Surfaces vs. Volumes

One of the most important purposes of computers is to model (in other words, encode) an interesting aspect of a real-world entity, do simulation and give back the result to again a real-world entity that is perceivable to human. The term "simulation" does not necessarily mean physical simulation such as finite element simulation. The simulation method strongly depends on which real-world aspect to be modeled. For example, the first computer ENIAC was designed to compute ballistic trajectories. In this case, the motion of a bullet is the modelled aspect of the real-world. Then, simulation is performed in the computer, taking into account the initial state, gravity, and other environmental effects. Eventually, the expected trajectory is output, which is the feedback from the computer to the real-world. CG clearly follows this process. Material properties that are related to the appearance of objects are modeled into the computer. Lighting simulation is then performed using reflectance or refraction models and the result is output to the final image through the rendering process. In this sense, CG is the "appearance simulation" of the real-world.

A great number of shape representations has been proposed until today. For most CG applications, the real-world is modeled only by the object boundary. Here the boundary forms a surface without thickness and such models are called *surface models*. Currently, surface models are widely used because they are compact, making them easy to construct, transmit, and render. Actually, the appearance of most 3D shapes is affected only by their surface properties. Therefore, surfaces are sufficient for most static scenes. On the other hand, lack of internal information causes several problems that are not present in real-world objects. Especially, creating dynamic and interactive scenes require internal structures because real-world behavior is sometimes driven by (possibly invisible) internal structures. For example, realistic animations can be generated only by simulation that requires fairly "realistic" representation of objects, say, volumetric data. More simply, if the model is split by cut operations in an interactive application, appropriate cross-sectional im-

ages should appear which is not possible with surface representations. Even if the scene is static, it is difficult to render translucent objects.[1] Another problem is self intersection. Self intersection is the situation that front faces and back faces go converse, which never happens in the real-world. This causes various problems such as inability to perform CSG operations [64].

Volumetric representations have complementary advantages and limitations. Since a volumetric representation stores internal information, it is straightforward to cut a model and observe the cross-sections. This is also suitable for generating realistic animations since simulation can also be performed for volume data. On the other hand, the amount of data generally far exceeds that of a surface representation, making storage, transmission, modeling, and rendering much more difficult. Modeling is especially problematic, although the other problems can eventually be mitigated by more memory, faster processors, and networks. We believe that this is the main reason why volume graphics is not so popular despite of the benefits. Since modeling problems are closely related to the limitations of human perception and manipulation, the design of appropriate user interfaces plays a critical role in addressing them.

Currently, the main sources of volume data are the capture and simulation. Since the invention of CT (Computed Tomography) scanners in 1973 [67], large-scale volume data became available. MRI (Magnetic Resonance Imaging) devices also generate volumetric data. These are called noninvasive capture devices and are now indispensable for medical diagnosis. Simulation is another source of volume data. Since the real-world is 3D, simulation is frequently performed in 3D, producing a 3D dataset as a result. These two data sources are actually the most standard and form mainstreams of volume graphics. However, the users of CG are not limited to medical doctors and academic/industrial researchers. As we mentioned before, such domains as educational or entertainment field are huge markets of CG and they have potential to benefit from volumetric graphics. Although Jim Kajiya's prevision in 1991 has been proven to be wrong[2], we still believe that there are a huge amount of unexplored benefit of volume graphics and the current

[1] If opacity of inside of an object is constant, the rendering can be precisely performed by processing only the surface. This computation can be performed on pixel shaders on contemporary graphics engines [38].

[2] T. Elvins's survey paper [34] cites the Jim Kajiya's words at Siggraph 91 "... in 10 years, all rendering will be volume rendering."

bottleneck lie in the fact that there are very few practical methods to create and manipulate volumetric data. In other words, the current user interface for volume graphics is amazingly poor. Our mission is to let all CG programmers benefit from volume graphics, even if they do not have specialized skills or devices to obtain and manipulate volumetric data that fit their requirement.

Unfortunately, this mission is not fully completed in this dissertation. However, we propose several interaction ideas that potentially make volume graphics tractable for end users- some of which are related to processing existing volume data and the others produces volumetric data from scratch.

1.3 Volume Data Representation

There are various types of volume data. In symbolic form, volume data is represented as a function $V(x, y, z)$, where (x, y, z) is a coordinate value. For example, if $V_b(x, y, z) \in \{0, 1\}$, V_b is called a *binary volume*. If $V_s(x, y, z)$ returns a single real number, V_s is called a *scalar volume*.

One important subset of scalar volume is an *implicit function*. Implicit function is developed in the context of surface modeling (See section 2.2.2). It usually consists of integral of weighted kernel functions and the surface is defined as the solution set of $V_s(x, y, z) = c$, where c is a user-defined threshold value (in most cases, $c = 0$). Usually, scalar volumes store auxiliary information of surface data. For example, distance field is an scalar volume that returns the closest distance to a surface. The distance field is easily computed from a surface model (unless the surface does not contain self intersection) and used to effectively find a central axis of the shape [5] or to provide an intuitive user interface [39]. Later we show an application of an scalar volume to a sketch-based modeling system, avoiding several problems seen in surface-oriented systems.

The function may also return a vector value. For example, the volume can return a tuple of three floating values, each of which contains the intensity of a color channel : $V_c(x, y, z) \in (r, g, b)$ where $r, g, b \in [0, 1]$. Then V_c is a *textured volume* or *colored volume*, which is a subset of more general *vector volumes*. Textured volumes are usually difficult to create since they usually contain detailed textures that are not necessarily related to the large-scale structure of the region. Some existing systems try to solve this by using a procedural approach that lets the user to di-

rectly program a volumetric modeling function using a specially designed language [111], possibly using help of an scalar volume [23]. However, previous systems are not user-friendly because a specialized skill is required to map desired geometric texture to a language code. In addition, it is not possible to generate highly detailed and unstructured textures using this interface. Another approach is the use of a reference volume and its seamless extension, which is mainly applied to video synthesis [80, 82, 131]. The drawback of this system is obviously the necessity of the reference volume. From the designer's point of view, this approach does not solve the essential problem because it ignores the most difficult part, the reference volume creation. There are some systems that tries to generate volume data from 2D cross-section [55, 148, 69]. However, none of them achieved both stability and flexibility at the same time.

Recently, a new kind of data called "Time-varying data" or "4-D volume" became accessible. This data take an additional variable t (time) as the input. Therefore, time-varying volume data is represented as $V_t(x, y, z, t)$. This form of data become more and more important for medical diagnosis of dynamic organs such as a heart. However, handing this data is beyond the scope of this dissertation.

1.4 Our contribution

In this dissertation, we propose several tools that make volume graphics tractable for end users. We mainly concentrate on modeling volumetric data, since we believe difficulty of modeling is the main obstacle for popularization of volume graphics. The overhead view of volume graphics and the position of this dissertation is shown in Figure 1.3. As in the figure, we focus mainly on user interface aspect of volume graphics. In Chapter 3 and 4, we propose two systems that manually create volumetric models from scratch. In Chapter 5 and 6, we propose two more systems that support segmentation of scanned or simulated volume data, represented as a set of voxels. The 3D model generated by any of these systems can be used for interaction system proposed in Chapter 7. We will explain each in detail.

In Chapter 3, we propose a sketch-based modeling system that can easily handle topological change, under the favor of intuitive user interface and the underlying scalar volume representation. Sketch-based modeling is a common technique for quick creation of rough 3D shapes. Since existing systems use surfaces as the primary shape representation,

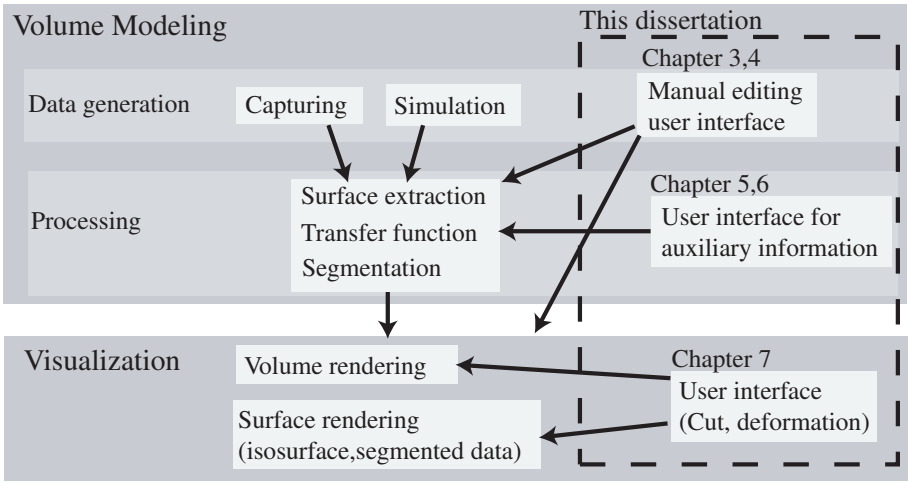


Figure 1.3: The domain of this dissertation in the context of volume graphics

implementing such systems was rather difficult (especially, implementing the CSG routine is difficult [64]) and the output can self-intersect, depending on the operations performed. The use of scalar volume representation is the most natural way to solve such problems. Through this work, we tried to show how we can easily and effectively benefit from volumetric shape representation.

In Chapter 4, we propose a system through which the user can easily create volumetric models using 2D example images. Here we introduce a new data representation that is a hybrid of 2D images (as an approximation of textured volume) and the control information in an scalar volume form. This representation is specifically designed for generating volumetric cross-section. If the user cuts the model, the 2D images is processed using an extension of pixel-based texture synthesis technique [149] to generate an appropriate target cross-sectional image. Since the data source is 2D images, it is easy to find and therefore suitable for end-user design. This is also extremely memory efficient compared to actually creating 3D textured volume.

In Chapter 5, we propose a topology enumerator and interactive selector, which enhances existing contour-based segmentation systems. Currently, the most robust volume segmentation system is to let the user observe each cross-sectional image and delineate the contours manually (possibly with the help of 2D image segmentation algorithms). But this

is extremely labor intensive and it is not realistic to ask the user to do this task for all slices. We rather ask the user to work on a few number of slices, resulting in a sparse set of parallel contours. Here a difficult problem arises: how can we interpolate these contours? This problem is rather traditional and a great number of approaches have already been proposed. The difficulty lies especially in the topology between contours, since once the topology is determined, spanning the smooth interpolating patch is easy by using well-known techniques such as thin-plate interpolation or a subdivision surface. Most existing techniques uniquely determine the topology from purely geometric relationship of contours, which may not be correct in general case, especially when slice interval becomes larger. We propose an algorithm to enumerate all possible bifurcation patterns to find the global optimal topology and corresponding user interface to modify the false result interactively.

In Chapter 6, we propose a simple user interface to segment traditional volume data. The most common volumetric data available today is in the form of regular voxels, which is essentially a set of 2D images across the 3D space. Although we have volume rendering hardware for such data readily available nowadays, it still requires much higher machine specification than that of surface graphics. For example, volume rendering is not desirable for most applications that work in multi tasking environment or in time critical systems. If the volume data is segmented and does not contain translucent voxels, the data is rendered by its boundary, say, surface models, and the volumetric aspect appears only when the model is cut, which can be performed on the main memory. In this sense, volume segmentation algorithm is a method to convert volumetric representation into surface models. However, the user interface for traditional volume segmentation task requires fair amount of interaction. We propose a very simple method: the user only need to trace the contour of the desired region in the rendered image. This interaction is significantly easier than any existing methods and we believe that this work makes existing volumetric data tractable even for end users.

In Chapter 7, we propose an interactive content that uses a volumetric user interface. In this Chapter, we show a cooking content, by which the user can interactively cut, peel, and deform volumetric models using free form strokes and a knife tool, controlled by a standard mouse. We tried to show that the volume technology has large potentials to

produce various interesting contents and also volume graphics require an effective and intuitive use interface.

Chapter 8 concludes the dissertation. This chapter summarizes the entire work and we discuss the current problems and future direction of volumetric computer graphics.

In this dissertation, we propose several tools that make volume graphics tractable for end users. We mainly concentrate on modeling user interface and corresponding techniques of volumetric data, since we believe difficulty in modeling volume data is the main obstacle for popularization of volume graphics. One limitation is that we do not propose a method to author textured volume data from scratch. This hampers our models to be used for volumetric simulation such as finite element simulation. However, we believe such simulation is only one of possible application of volume graphics and we can still benefit from volumetric representation such as scalar volumes (Chapter 3) and volumetric illustration (Chapter 4). It is our future work to author textured volume data from scratch, using sophisticated user interface.

Chapter 2

Related work

According to Arie Kaufman, "Volume graphics, which is an emerging subfield of computer graphics, is concerned with the synthesis, modeling, manipulation, and rendering of volumetric geometric objects, stored in a volume buffer of voxels [74]." This is fair but we consider volume graphics in more generic sense, including implicit function volumes or procedural volumes, which are not necessarily stored in voxels. Volume graphics is the most intensively explored in the context of medical diagnosis and scientific simulation. Medical doctors require volume graphics because medical devices such as CT or MR scanners are huge producers of volume data and effective visualization is crucial for their diagnostic task. Scientific simulation is another rich source of volume data since real-world is 3D and they need volume graphics to understand the real-world and the simulation result. However, the application area of volume graphics is not limited to those two. For example, entertainment industry also uses volumetric simulation in simplified form to represent amorphous phenomena such as clouds, fire, and smoke, which do not have distinct surfaces. In this chapter, we overview the history and current situation of volume graphics.

2.1 Volume data types

There are several ways to represent volume data, depending on the purpose or the property of data source.

The most rough classification of volume data is two-hold: a set of voxels (discrete sample points) and a continuous function form. Voxels are directly obtained from scanning real-world objects or simulation. Since these data sources are predominate, this form of data is the most

commonly used. A drawback of this representation is that it tend to require larger amount of memory for meaningful data. Large amount of data causes difficulty not only in storing and processing, but also in manual generation and transformation. Another drawback is that interpolation is necessary for various processing such as rendering or generation of arbitrary cross-sectional images. Function form volumes are less commonly used but from the point of view of generating volume data manually, this is virtually the only tractable and practical representation. This is mainly used for modeling amorphous phenomena or other organic textures such as marbles or tree rings [111, 23]. The advantage of this representation is that it is compact and continuous. On the other hand, the user has to define spatial distribution by function programing where the user interface cannot be very intuitive and the variety of possible textures is limited.

Voxels can be aligned either regularly or irregularly. Regularly aligned voxels is most commonly used and most volume rendering hardwares (eg. VolumePro [112], NVidia GeForce series) only support this form of data. Since this is a natural extension of 2D images into 3D, a great number of 2D imaging algorithms such as compression or segmentation, are directly applicable to voxels. A common source of irregular set of voxels is the finite element meshing for simulation. Irregular voxels can be stored either as an independent set of points or tetrahedron. Although this type of data holds flexibility in representing arbitrary layout of sample points, it is more difficult to visualize than regular voxels because determining the correct visibility is not straightforward.

2.2 Volume data sources

Currently, the source of volume data is limited. It is a clear difference from surface graphics since there are a great number of 3D shape modelers designed for surface graphics both in commercial and research levels [2, 3, 157, 68]. To obtain volume data, we need to have access to special scanning device or generate the data through simulation or program the texture by a scripting language. Scanning is easier and more typical method but such input devices are usually extremely costly. On the other hand, most manual creation methods do not require special input device but the user interface is usually not very sophisticated and the variety of resulting data is limited. We believe this poor variety of data sources narrows the application domain of volume graphics.

2.2.1 Scanning real-world objects

Currently, the main source of volumetric data is the scanning of real-world objects. Scanning is often performed in the fields of medical or biological research. The existing scanning devices can be categorized into two types: non-invasive and invasive ones.

Non-invasive methods Non-invasive devices capture 3D volume without destroying the subject. The most common such devices are CT (Computed Tomography) and MR (Magnetic Resonance) scanners. CT scanner has been recognized as one of indispensable devices in such fields as medical, biological and engineering. This device is based on the theory of projection/back-projection which was discovered by J. Radon in 1917 [120]. This theory is called Radon and inverse Radon transformation. The first X-ray CT scanner is invented by Hounsfield in 1973 [66]. He received Nobel Prize for this achievement. MR scanners also non-invasively explore inside of subjects. However, in contrast to CT scanners, MR scanners use magnetic field. The principle for this device was found in 1946 by Broch et al. and Purcell et al, independently [13, 119]. They were also awarded the Nobel Prize in 1952. Lauterbur could generate MR image in 1973 [85] and in mid 1980s, MR scanners began to widespread. MR scanners have several differences from X-ray CT scanners. First, MR scanners do not cause X-ray bombing. Second, the spatial resolution of MR scanners is usually less than that of CT scanners. MR imaging is appropriate to detect hydrogen (say, water) while X-ray is good for heavy atoms such as calcium.

Invasive methods The principle of invasive methods is simple: the observer actually cuts the subject and takes pictures of the cross-section. The benefit of this method over non-invasive methods is that this can directly capture visual property (color and opacity) of the cross-section. In addition, other properties such as stiffness or humidity can be captured because the subject is actually opened up and arbitrary processing is possible on the cross-section. On the other hand, obvious drawback is that this is invasive. The subject is irreversibly destroyed and the original shape cannot be recovered in most cases. Therefore, it is impossible to capture living subject as it is.

A good example data that is captured invasively is Visible Human Dataset [10]. Dead bodies are physically sliced at $\frac{1}{3}$ to 1 millimeter intervals. This dataset is obtained by NLM (National Library of Medicine, USA).

Although Visuble Human Project developed a special slicing device, a most popular slicing device is called a microtome which can cut only small objects. On the other hand, there are some groups that develop microtomes for mid-sized objects [105].

2.2.2 Scalar volume, implicit surface modeling

Scalar volumes return a scalar value for each spatial point (See section 1.3). If the spatial function $V_s(x, y, z)$ is interpreted to define a surface which is extracted as the solution set of $V_s(x, y, z) = c$ where c is a constant value, the function V_s is called an implicit function and the corresponding surface is called an implicit surface.

Implicit surface modeling has more than 20 years of history [106]. The pioneer to introduce implicit surfaces is Blinn [12]. Earlier systems represent the implicit function by a set of points that span a spherical profile (the field value is determined only by the distance from the center of the kernel: radial basis function) [46, 155], while later systems employ more sophisticated primitives such as integration of kernel function along a line or a part of a surface [14, 6].

Implicit modeling has several advantages. One is that if the kernel function is smooth, the smoothness of the surface is automatically guaranteed, making it convenient for creating smooth, organic shape. Another benefit is that the user has no need to care about the topological change explicitly. This is because of its volumetric nature and this is the main aspect we focus on in Chapter 3.

One drawback is the difficulty to visualize the surface. There are mainly two ways to do this: direct rendering and polygonization. Direct rendering method casts rays from the viewpoint to the scene, looking for a intersection to the target surface and sample the normal vector at each location. This is appropriate to achieve precise visualization of the surface in the sense that the resolution of sampling implicit function is aligned to the screen resolution [12, 46]. There is a good review of this technique in [52]. The main drawback of this technique is that it is almost impossible to achieve interactive frame rate.

More commonly used technique is the surface extraction. It is not straightforward because there is a gap between continuous implicit function and discrete (piecewise linear) mesh representation. The most popular polygonization algorithm is the Marching Cubes method, which samples the space regularly and locally fit a triangular mesh [90]. Im-

plementing this method is fairly easy but there are some drawbacks such as computational complexity and loosing topological consistency. Some other techniques do not have such problems [153, 25].

2.2.3 Vector (textured) volume modeling

Vector volumes are defined to be a spatial function $V_v(x, y, z)$ that returns more than one scalar values. The most important subclass of this type of volume data is a textured (or colored) volume, which returns the intensity of a tuple of three or four scalar values that represent red, green, and blue channels (optionally with an opacity channel), respectively. Textured volume modeling is much more difficult than scalar volume modeling because the texture that should be modeled has fine structures that usually has no (or little) connection to the geometric feature of the object. There are few methods proposed in research domain but only procedural methods are used in practical. Unfortunately, procedural methods lack flexibility and intuitive user interface, which we think is the biggest limitation.

Procedural Procedural methods enable the user to design various 3D structures using specially designed scripting language [111, 77]. This method is also called solid texturing and widely used for modeling 3D texture such as marble, clouds, and fire. The key feature of this technique is the use of multi-scale noise function called "Perlin noise function", which generates natural structural texture efficiently. Cutler et al. also proposed a scripting language for volumetric modeling [23]. However, it is difficult for most people to obtain desired textures by programming or adjusting parameters.

Special input devices There are systems that use 3D pointing devices to create 3D textures [43, 37, 94]. However, they are designed mainly to convey the overall shape of a model and it is still difficult to design the detailed internal textures of 3D volumes. Another drawback is that such input devices are usually expensive, compared to standard devices such as mice.

2D texture examples to 3D Usage of 2D images is one solution for explicit volume modeling. This approach has potential to offer a nice user interface because 2D images are very much easier to obtain than 3D example volumes. Heeger et al.[55] and Dischler et al.[26] tried to analyze an original texture sample using frequency decomposition and applied the information to generate 3D texture. These techniques work only

for isotropic, noisy textures. Wei extended the pixel-based 2D texture synthesis method to generate 3D textures [148]. This work returns much better result than frequency-based method but still is not sufficient for practical use. Jagnow et al. recently proposed a stereological technique to generate high quality 3D texture from 2D sample image [69]. However, [69] can only handle isotropic textures that consist of set of elements and to make matters worse, the shapes of elements should be predefined.

Other techniques Wang et al. and Mizuno et al. proposed a carving technique to author 3D shape with textures [147, 96]. Their system focuses on mimicking a carving process, rather than actively constructing 3D texture.

2.3 Visualization of volumes

The history of volume graphics had almost been identical to the history of volume visualization, because comprehension of volume data is the most primary and important interaction but is not as easy as expected.

2.3.1 Binary volumes

The simplest method is to convert general volume data into binary form by setting a threshold value and then render each opaque voxels by six quadrangular faces [58]. This method is later improved by isosurface extraction of the volume, which is related to the polygonization of implicit surfaces (Section 2.2.2). The notion of isosurface is later extended as the limit of an interval volume [41]. The isosurface of binary volume usually becomes jaggy but the effect is mitigated by computing the surface normal from the gradient of the original volume data [62]. Binary volumes are also produced by hard segmentation of volumes (Section 2.5).

2.3.2 Volume rendering

Volume rendering, which is now a standard way to visualize volume data, directly uses each voxel's nonbinary opacity. A number of algorithms are proposed.

Ray casting Ray casting algorithm casts rays and integrate illumination values along the rays [70, 86]. Drebin et al. used fuzzy classification of voxels for coloring the target volume [27]. This work has

a great impact and is considered to be a pioneer of volume rendering technique. Cube-4 is a special-purpose volume rendering hardware that renders 1024^3 voxels at 30 frames per second, using variant of ray casting algorithm [113]. This technique is extended to a commercial system called VolumePro [112].

Splatting Splatting is an algorithm to map point primitives to image plane [152]. The shape and the size of the mapped primitive is approximated and displayed as a translucent blob. This is a forward mapping technique, which is different from standard ray casting algorithm where the access to the volume data is aligned by pixel locations in the final image. Forward mapping is usually faster than backward mapping since the access to the original data is more cache coherent. Recent work achieve faster and higher quality result with GPU (graphics processing unit) computation using extension of splatting [17]

Shear-warp / 3D texturing Shear-warp factorization is another forward-mapping technique [81]. Instead of point primitives, shear-warp factorization essentially uses texture mapping and can be efficiently implemented by standard texture mapping hardware. Despite its easiness for implementation and fast rendering speed, shear-warp factorization has several drawbacks: resampling is performed only within the slices (therefore the interpolation is bilinear, instead of trilinear) and also sampling rates are affected by viewing angles, both cause significant degradation of the rendered image quality. Although these drawbacks are mitigated by careful implementation using the multi-texturing and the multi-stage rasterization facility [122], more direct and fast volume rendering using a contemporary consumer level graphics hardware unit is 3D texturing [44, 100]. Rendering quality is further enhanced by adopting GPU functionalities [79].

Evaluation in 2000 shows splatting produces quality result comparable to ray casting and shear-warp/3D texturing are fast but the image quality is low [101].

Non-photorealistic rendering Yet another emerging volume rendering technique is non-photorealistic volume rendering. Non-photorealistic rendering (in other words, expressive rendering) arose from the context of simulation of traditional media and human drawing [134, 47]. The significance of computerized non-photorealistic rendering is not just imitating real-world media, but efficiently conveying visual information by intentional emphasis and suppression, rather than pursuing realism

[45, 124]. For volume rendering, a variety of methods are already proposed such as silhouette enhancement [28, 117], pen-and-ink style rendering [140], and stippling [91]. Importance driven volume rendering also falls into this category, where the designer specifies the region of interest and the information affects the final rendering [146].

2.3.3 Other visualization techniques

Another primary method is the simulation of x-ray imaging, that is, parallel rays are cast to the volume, averaging the intensities along the ray [51]. MIP (Maximum Intensity Projection) is yet another visualization method, which just takes the largest value along the cast ray. Since it is easy to achieve interactive frame rate using MIP, this technique is still widely used and actively explored [54, 97]. 3D magic lenses is a technique to locally control rendering parameters in the 3D space [145]. Magic mirrors proposes a multimodal rendering environment where the user can browse target object in multiple styles while maintaining visual coherency by using the metaphor of mirrors [78].

Visualization of arbitrary (curved) cross-section is an important technique to precisely observe an aspect of volumetric object. Earlier systems just display planar and axis-aligned cross-sections. Some recent works try to generate special cross-sections that most clearly represent the structure of volume data [72]. Interactive browsing with cutting and deformation is another novel and promising approach [95].

2.4 Transfer functions

Volume classification, which splits raw data into two or more semantic regions, plays a crucial role in making information clearly visible to the viewer. The most important and frequently used volume classification tool is the transfer functions [114]. They map raw volume data (usually grayscale) to color and opacity values (in our terminology, mapping from scalar to textured volumes, or, possibly textured to textured volumes). The spirit is essentially the same as the image segmentation, which will be discussed in detail in the next section. The key difference from image segmentation is that the target volume data scarcely contain color channels. Compared to sophisticated image segmentation algorithms, setting transfer functions looks rather simplistic. In reality, identifying a good transfer function proves difficult. Actually, this is

counted as one of 10 unsolved problems in computer graphics by Pat Hanrahan in 1992 [114]. The reason is that voxels are frequently occluded by other voxels and also the region of interest (ROI) may not hold distinct features. Because of the importance in volume visualization, there are great number of approaches are already proposed and tested.

Traditionally, transfer function directly maps the original voxel values to color and opacity values [27]. He et al. proposed a system that directly walks through the parameter space. They used a stochastic technique (such as hill-climbing or simulated annealing) to generate parameter values, driven by a predefined or user-specified objective function. Marks et al. proposed the *Design Galleries* system, which allows the user to interactively select general rendering parameter space with the help of pre-rendered thumbnail images [93].

The above-mentioned 1D histogram clustering has a limited preformance since location and texture are not well considered. Bajaj et al. proposed a system called *Contour spectrum* that displays the surface area, volume, and gradient integral as functions of scalar voxel values and allows the user to select a desired isovalue [9]. Kniss et al. take gradient and second directional derivative of the original voxels and use them as input of the transfer function [76]. Since increase of the dimension of transfer function causes complication of user interface, [76] also introduced the sophisticated user interface called direct manipulation widgets. Fujishiro et al. and Takahashi et al. presented a distinctive system to analyze topological structure of volume data and reflect the information to manually or automatically define the transfer function [42, 137].

2.5 Segmentation

Image segmentation is the task of extracting regions corresponding to perceptually distinct regions. This is a fundamental topic in vision that has received a lot of attention and been intensively explored in the 2D domain. A large number of different approaches have been proposed, such as thresholding, k-means clustering, deformable models, watershed segmentation, graphcut algorithms, level-set methods, and the Hough transform, all of which can be applied to 3D voxels with no or slight modification. We will not give an overview of all existing methods here. Instead, we explain a few promising approaches. The most common

algorithms optimize graph partitions of weighted neighborhood graphs [156, 36, 127]. Solving these graph partitioning problems can either be done locally by fast greedy decision heuristics [36, 103], or globally by computing decompositions [156, 127] of matrices induced from these graphs. Segmentation algorithms designed primarily for 2D data are often applicable to 3D with little or no modification.

Unfortunately, completely automatic image segmentation algorithm cannot exist because segmentation is dependent on semantic interpretation of image, which is very difficult to emulate by computer. Therefore, recent systems seek to incorporate human control into segmentation task. Examples of such systems for 2D imaging are the Lazy Snapping system [87] or the Crayons system [35]. On the other hand, it is especially difficult to control 3D segmentation because typical input device is a 2D mouse. The most reliable method is to isolate a slice of the volume and manually specify the contour of the ROI [53]. This information is then propagated to adjacent slices using a region growing technique. Alternatively, the user may place seed points for region growing on the cross-sectional plane [126]. These techniques require fair amount of user interaction. Setting transfer functions is indirect from the viewpoint of user interaction [114]. Tzeng et al. recently presented a user interface for providing high level classification information through roughly drawing freeform strokes on a cross-sectional plane [144]. The user cuts the data perpendicular to each axis and specifies foreground and background regions using a painting tool. By observing the gradient, location, and neighbouring voxel values, as well as the voxel value itself, this system captures local texture and positional information of a voxel. Nock and Nielsen describe a fast and provably good region-merging algorithm [104] based on statistical analysis of regions. Their method easily generalizes [103] to incorporate user-defined constraints and is directly applicable to 3D.

Chapter 3

Sketch-based modeling of scalar volumes

Figure 3.1: Examples created by our system

This chapter presents a sketch-based modeling system for creating objects that have internal structures. Using hand-drawn sketches and gestural operations, the system automatically generates a volumetric model. The underlying volumetric representation solves any self-intersection problems and enables the creation of models with a variety of topological structures, such as a torus or a hollow sphere. To specify internal structures, our system allows the user to cut the model temporarily and apply modeling operations to the exposed face. In addition, the user can draw multiple contours in the Create or Sweep stages. Our system also allows automatic rotation of the model so that the user does not need to perform frequent manual rotations. Our system is much simpler to implement than a surface-oriented system because no complicated mesh editing code is required. We observed that novice users could quickly create a variety of objects using our system.

3.1 Background

Geometric modeling has been a major research area in computer graphics. While there has been much progress in rendering 3D models, creating 3D objects is still a challenging task. Recently, attention has focused on sketch-based modeling systems with which the user can quickly create 3D models using simple freehand strokes rather than by specifying precise parameters for geometric objects, such as spline curves, NURBS patches, and so forth [157, 68]. However, these systems are primarily designed for specifying the external appearance of 3D shapes, and it is still difficult to design freeform models with internal structures, such as internal organs. Specifically, the existing sketch-based freeform modeling system [68] can handle 3D models only with spherical topology. This paper introduces a modeling system that can design 3D models with complex internal structures, while maintaining the ease of use of existing sketch-based freeform modelers. We used a volumetric data structure to handle the dynamically changing topology efficiently. The volumetric model is converted to a polygonal surface and is displayed using a non-photorealistic rendering technique to facilitate creative exploration. Unlike previous systems [68], our system allows the user to draw nested contours to design models with internal structures. In addition, the user can cut the model temporarily and apply modeling operations to the exposed face to design internal structures. The underlying volumetric representation simplifies the implementation of such functions. Moreover, our system actively assists the user by automatically rotating the model when necessary.

The heart of our technique is automatic "guessing" of 3D geometry from 2D gestural input, and it is done by making certain assumptions about the target geometry. To be specific, the system assumes that the target geometry has a rotund, smooth (low curvature) surface [68] other than the places where the user explicitly defined the geometry by the input strokes. In other words, the user specifies the information about important features (silhouette, intersection, and sweep path) and the system supplies missing information based on the above assumption.

Our system is designed to facilitate the communication of complicated geometric information, such as surgical plans. Like other sketch-based modeling systems, however, our system is not suitable for creating the final output of any serious production, because of its lack of accuracy.

3.2 Prior art: sketch-based modeling

Sketch-based modeling using standard mouse operations became popular in the past decade. Instead of creating precise, large-scale objects, a sketching interface provides an easy way to create a rough model to convey the user's idea quickly. One of the earliest sketching systems was Viking [118], which was designed in the context of prototypic CAD models. Later works include SKETCH [157] and Teddy [68]. The SKETCH system is intended to sketch a scene consisting of simple primitives, such as boxes and cones, while the Teddy system is designed to create rotund objects with spherical topology. Although improvements to the original Teddy system is proposed later [73], extending the topological variety of creatable models is still an unsolved problem.

3.3 User Interface

The entire editing operation is performed in a single window. Modeling operations are specified by freeform strokes drawn on the screen and by pressing buttons on a menu bar. The freeform strokes provide necessary geometric information and the buttons apply specific modeling operations using the strokes as input. The drawing of strokes is assigned to the left mouse button and rotating the model is assigned to the right mouse button. The current implementation uses four buttons, as shown in Figure 3.2. The leftmost button is used to initialize the current scene; the second one is to create items; the third is for the extrusion/sweep function; and the last is for undo.

Figure 3.2: Buttons in our system

3.3.1 Create

Objects are created by drawing one or more contours on the canvas and pressing the "Create" button. This operation inflates the intermediate region between the strokes leaving holes (Figure 3.3).

Figure 3.3: Nested contours are allowed in the Create operation.

3.3.2 Extrusion

Extrusion is an operation that generates a protuberance or a dent on a model. The user draws a single closed stroke on the object's surface specifying the contour (Figure 3.4 (b)) and presses the "Extrusion/sweep" button. After rotating the model (Figure 3.4 (c)), the user draws a second stroke specifying the silhouette of the extruded area (Figure 3.4 (d, f)). The user should place each end of the silhouette stroke close to each end of the projected surface contour (otherwise the second stroke is interpreted as a sweep path; see Section 3.4.) A protuberance is created if the second stroke is drawn on the outside of the object (Figure 3.4 (d,e)). The user can also create a hole by drawing a stroke into the object (Figure 3.4 (f,g)). Volumetric representation automatically prevents self-intersection problems, where specialized care must be taken when using a polygonal representation. A hidden silhouette is rendered as broken lines.

3.3.3 Loop Extrusion

In addition, it is also possible to create a hollow object using extrusion. To do this, the user first cuts the model to expose the internal region (Figure 3.5 (a-c)), then draws a contour on the exposed plane (Figure 3.5 (d)), and finally draws a circular stroke that entirely surrounds the contour (Figure 3.5 (e)). We call this operation "Loop Extrusion". The cutting operation that we use differs from the standard Cut operation in the Teddy system [68] in that the removed region is just deactivated temporarily. The system distinguishes these two operations by checking whether there is a corner at the end of a stroke. The system performs a standard cutting operation when there is no corner, while the system deactivates a region when there is a corner. The direction

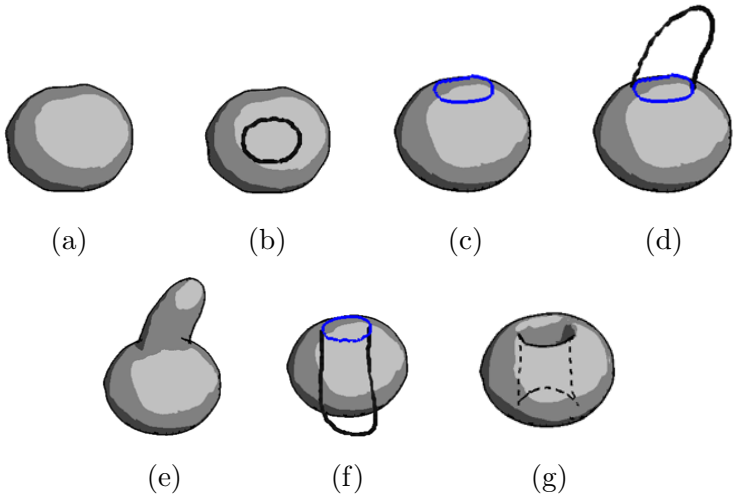


Figure 3.4: Examples of Extrusion

of the stroke end is used to determine which area to deactivate. The silhouette of the deactivated parts is rendered as broken lines.

Deactivation is provided in order to make the inside of an object accessible. The user can draw a contour and have it extrude on an internal surface in exactly the same way as done on an external surface (Figure 3.6). The following sweep operation can also be used in conjunction with deactivation.

3.3.4 Sweep

After pressing the "Extrusion/Sweep" button, the user can also draw an open stroke specifying the sweep path. If a single contour is drawn in the first step, both ends are checked to determine whether they are close to the projected contour. Unlike extrusion, the user can draw multiple contours to design tube-like shapes (Figure 3.7).

3.3.5 Animation Assistance

In extrusion or sweep, the model must be rotated approximately 90 degrees after pressing the "Extrusion/Sweep" button to draw the last stroke. To automate this process, our system rotates the model after

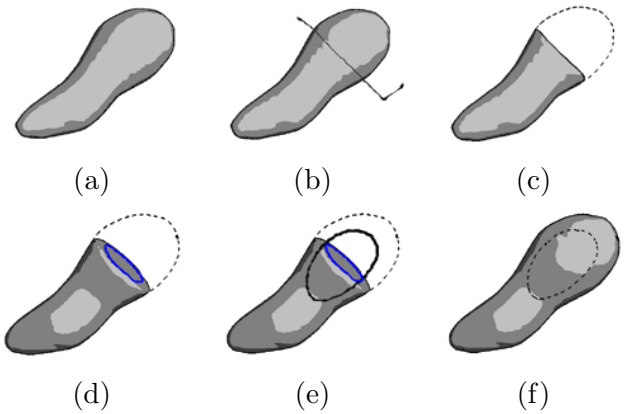


Figure 3.5: An example of creating a hollow object: first, the user defines the desired cross-sectional plane by deactivating part of the object (a-c). Then, the user draws a contour on the cut plane (d). Finally, the user draws a extruding shape surrounding the contour, which we call "Loop Extrusion" (e). This creates a hollow object (f).

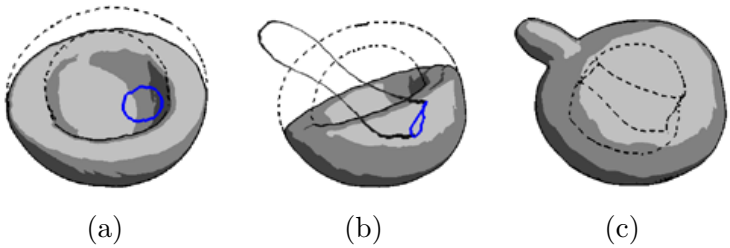
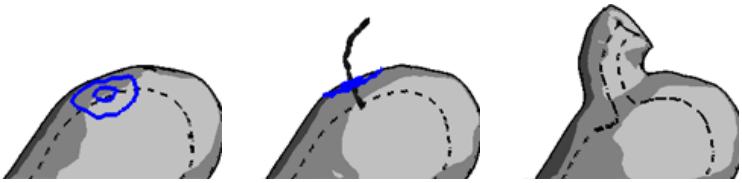


Figure 3.6: A extrusion from an internal surface of an object using deactivation

the "Extrusion/Sweep" button is pressed; the contours are then moved so that they are perpendicular to the screen (Figure 3.8 (a-c)). This animation assistance is also performed after a Cut operation, because it is likely that a contour will be drawn on the cut plane in the next step. When a model is cut, it is automatically rotated so that the cut plane is parallel to the screen (Figure 3.8 (d-f)).



(a) (b) (c)

Figure 3.7: Sweeping double contours: drawing contours on the surface of an object (a) and sweeping them (b) produces a tube (c).

3.4 Implementation

We use a standard binary volumetric representation. The examples shown in this paper require approximately 400^3 voxels. The volumetric data are polygonized using the Marching Cubes algorithm [90]. The polygonized surface is then smoothed [139] and displayed using a non-photorealistic rendering technique [83]. The silhouette lines of invisible or deactivated parts are rendered as broken lines.

The Create operation is different from that of the original Teddy system [68]. Basically, the algorithm approximates the user-drawn closed stroke by a set of circles. Then each circle is converted to a ball (Figure 3.9).

It is performed by first finding the medial axis of the user-drawn stroke. The approximated medial axis is computed by taking ridge lines in the signed distance field of the input stroke, which is efficiently computed by the vector distance transformation [98]. This process is equal to finding the approximating set of circles because the centers of the circles lie on the medial axis and the distance values on the medial axis represent the radius of the circles.

Conversion to 3D balls is again efficiently performed by slightly modifying the 3D vector distance transformation algorithm [98]. The original algorithm sets the input surface locations as distance 0 constraints. Instead, we set centers of balls as nonzero distance constraints. We locate the 2D medial axis (with distances) into 3D space as constraints and then perform the 3D vector distance transformation. The zero set of the resulting volume is the output surface.

The shape tend to be more round than the original algorithm proposed in the Teddy system 3.10.

In Extrusion, our system adds the additional geometry to the orig-

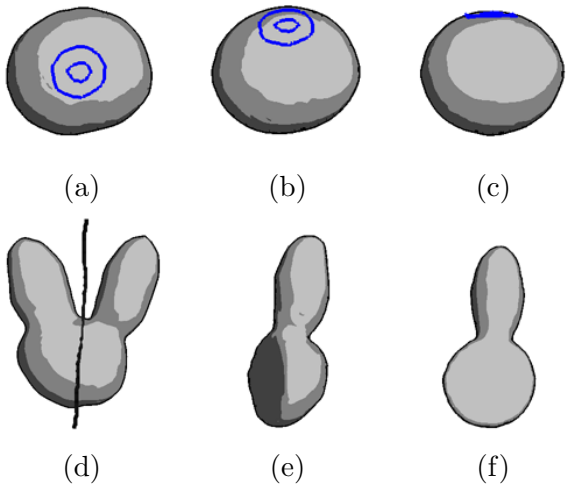


Figure 3.8: Examples of animation assistance: as soon as the user presses the "Extrusion/Sweep" button, the model is rotated so that the contours are perpendicular to the screen (a-c). When the user cuts a model, the /model is automatically rotated so that the cut plane is parallel to the screen (d-f).

inal model when an outward stroke is drawn and subtracts it when an inward stroke is drawn. Note that complex "sewing" of polygons is not necessary and no self-intersection will occur because of the volumetric data structure. Loop Extrusion applies the standard inward (subtract) extrusion in both directions. The Sweep operation in our system requires two-path CSG operations to add a new geometry to the original model. First, the sweep volume of the outermost contour is subtracted from the original model (Figure 3.11 (a-c)). Then, the regions between the contours are swept and the sweep volume is added to the model (Figure 3.11 (d)). This avoids the inner space being filled with the original geometry.

The volumetric representation significantly simplifies the implementation of the Cut operation and enables the change in topology. A binary 2D image is computed from the cutting stroke in the screen space to specify a "delete" region and a "remain" region. Both ends of the cutting stroke are extended until they intersect or reach the edges of the screen. Then, one of the separated regions is set as the "delete" region (usually the region to the left of the stroke, following the original Teddy convention). Each voxel is then projected to the screen space to check

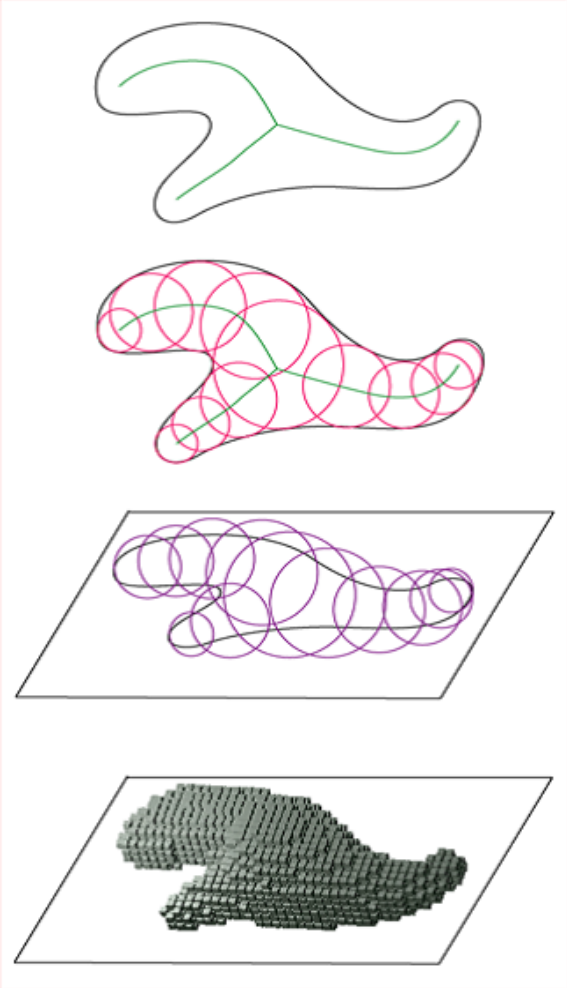


Figure 3.9: Our Teddy-like Create algorithm

whether it is in the deleted region; if so, the voxel is deleted. This process is significantly simpler than traversing the polygonized surface and remeshing it.

3.5 Results

We used a Dell Dimension 8200 computer that contained a Pentium 4 2-GHz processor and 512 MB of RAM. The graphics card was an NVIDIA GeForce3 Ti500 with 64 MB of memory. Users can create models interactively on this machine. We also used a display-integrated tablet as an input device, with which the user can edit an object more

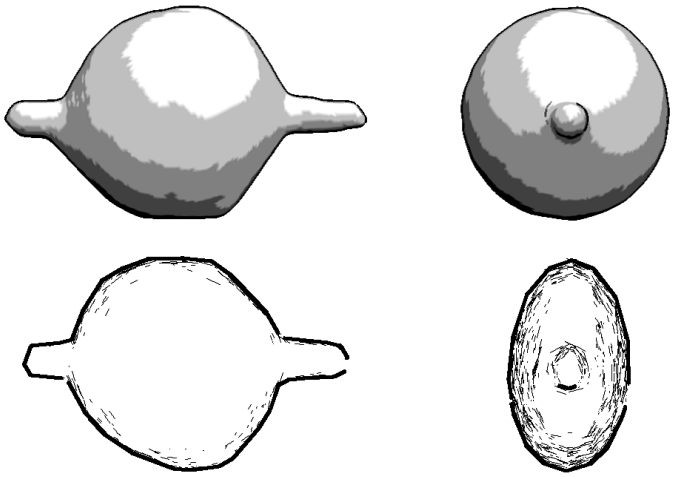


Figure 3.10: Difference of Create algorithm between our system (top row) and the original Teddy system [68](bottom row).

intuitively. However, some users found it difficult to rotate an object because they needed to press a button attached to the side of the pen and move the pen without touching the display.

Figure 3.13 shows some models created using our system. Figure 3.13 (a-c) were created by novices within fifteen minutes of an introductory fifteen-minute tutorial; the others were created by an expert. Our observations confirmed that users could create models with internal structures quickly and easily.

3.6 Discussions

In this section, we presented a sketch-based modeling system for creating objects with internal structures. The underlying volumetric data structure simplifies the handling of a dynamically changing topology. The user can modify the topology easily in various ways, such as by cutting an object, forming a extrusion, specifying multiple contours with create or sweep operations, or specifying internal structures in conjunction with temporal deactivation. In addition, automatic rotation of the object frees the user from tedious manual labor.

However, the limitations also became clear. The users occasionally found the behavior of Extrusion unpredictable because there was no depth control. Specifically, when a user tried to create a cavity in an

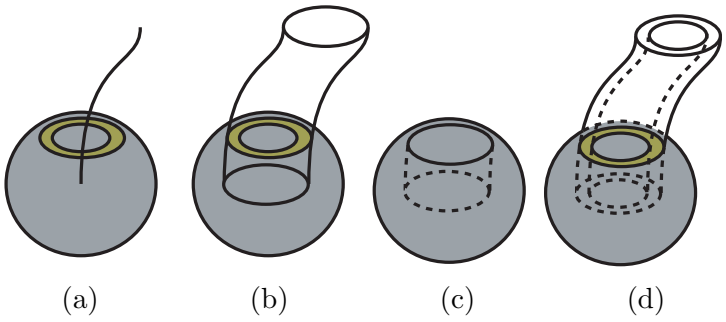


Figure 3.11: Handling the sweep operation. The outmost contour is swept along the specified path (a,b) and extracted from the original model (c). Then, every contour is swept and added to the model.

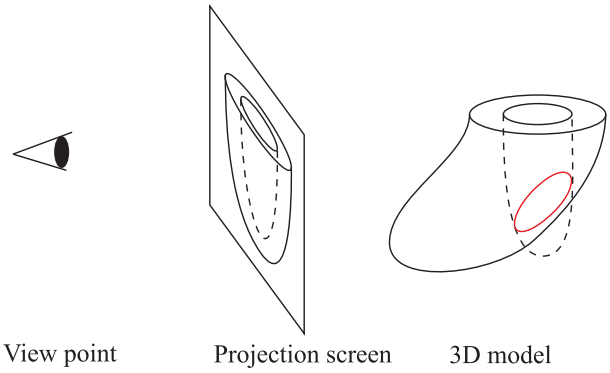


Figure 3.12: An undesired effect caused by the lack of depth control. Since there is no depth information in the original model, the newly created cavity can pierce the wall.

object, the hole sometimes penetrated the wall of the original model (Figure 3.12). This suggests the necessity of explicit topology control for the system. In addition, it is difficult to modify shapes locally and we are exploring ways to add small details. One promising approach to solve this problem is introduction of multi-resolution data representation [39]. It is our future work to explore along these directions.

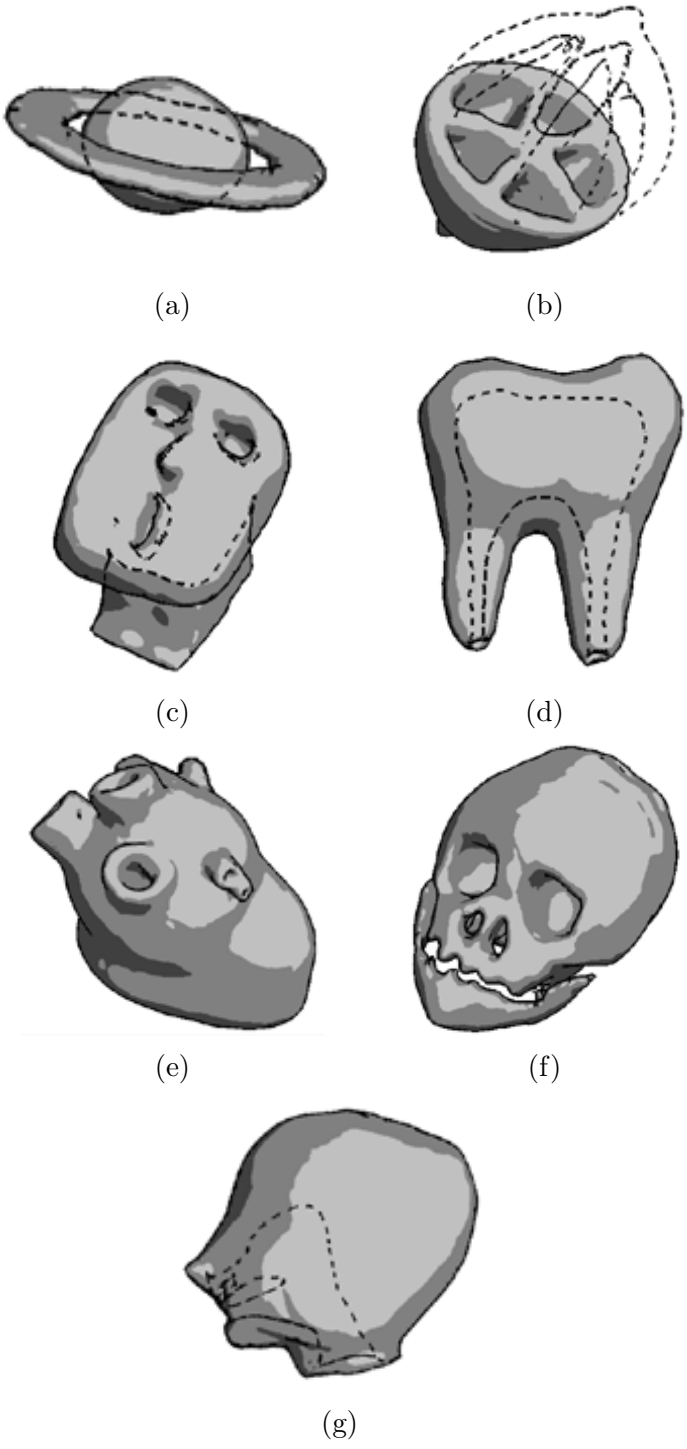


Figure 3.13: Results. (a-c) were created by novices, while (d-g) were created by an expert.

Chapter 4

Volumetric Illustration: 2D volumetric texture synthesis of cross-section

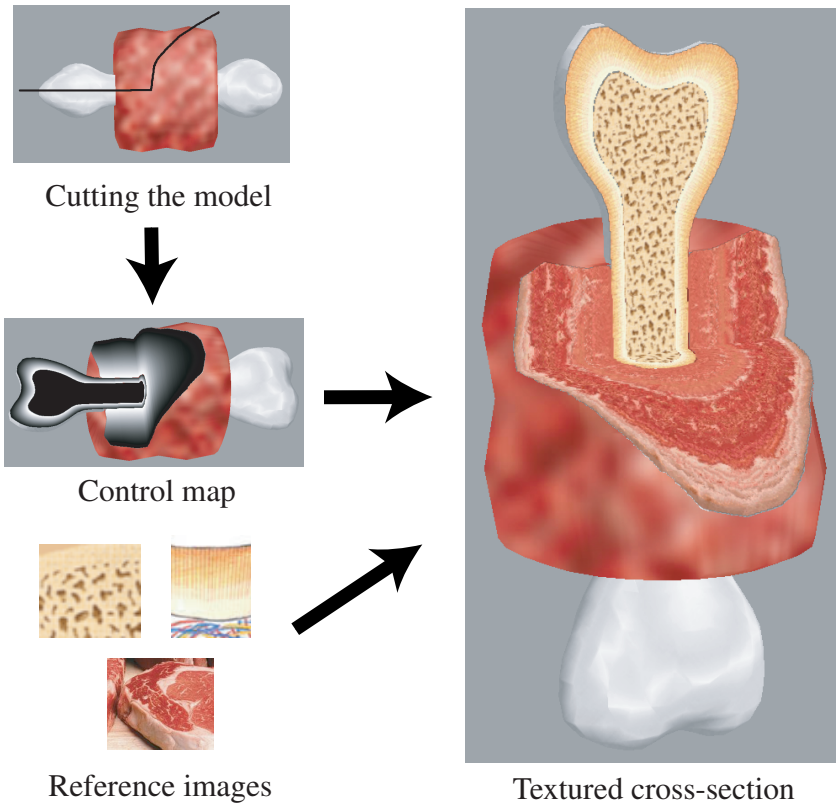


Figure 4.1: System overview. The user can cut the model anywhere and observe internal textures. Internally, the system first computes a control map using predefined guiding information and then synthesizes the texture using the control map and the reference textures.

This chapter presents an interactive system for designing and browsing volumetric illustrations. Volumetric illustrations are 3D models with internal textures that the user can browse by cutting the models at desired locations. To assign internal textures to a surface mesh, the designer cuts the mesh and provides simple guiding information to specify the correspondence between the cross-section and a reference 2D image. The guiding information is stored with the geometry and used during the synthesis of cross-sectional textures. The key idea is to synthesize a plausible cross-sectional image using a 2D texture-synthesis technique, instead of sampling from a complete 3D textured volume directly. This simplifies the design interface and reduces the amount of data, making it possible for non-experts to rapidly design and use volumetric illustrations. This data structure is an approximation of explicit volume representation, which offers easy-to-use volumetric authoring system since 2D information is tractable even with standard 2D input devices (e.g. a mouse, a tablet).

4.1 Background

Our goal of this work is to develop an interactive designing and browsing system that allows the user to add interesting textures to surface meshes manually by using existing 2D reference images (Figure 4.1). The basic property of opaque volumetric data is that we cannot see all of the 3D volumetric information simultaneously; because of occlusion, we can see only one 2D cross-section at a time. For example, illustrations in biology textbooks or scientific magazines often show cross-sections of a volumetric object to explain internal structures (Figure 4.2). It is also important to note that these illustrations are the result of careful design processes rather than a literal simulation of reality. In Figure 2, for example, nuclei are seen in all the cells in the illustration, while an actual cross-section would contain cells whose nuclei were not visible.

Based on this observation, we propose a new representation for 3D models with internal textures, namely one in which the system synthesizes the internal textures for a cross-section by using 2D reference images instead of maintaining all the 3D volumetric data (Figure 4.1). To assign internal textures to a model, the designer specifies the correspondence between a geometric cross-section and a reference 2D image by providing guiding information, such as flow orientation. This approach significantly reduces the amount of data that the model requires.

It also allows designers to add 3D internal texture to a model without specifying each voxel manually.

Our technical contributions are the interfaces that are used to assign internal textures to a given surface mesh and the algorithms that synthesize textures on a cross-section. On a larger scale, our contribution is a modeling structure in which the specification and viewing of simple volumetrically textured models is easy and convenient, allowing non-experts to create volumetric illustrations rapidly.

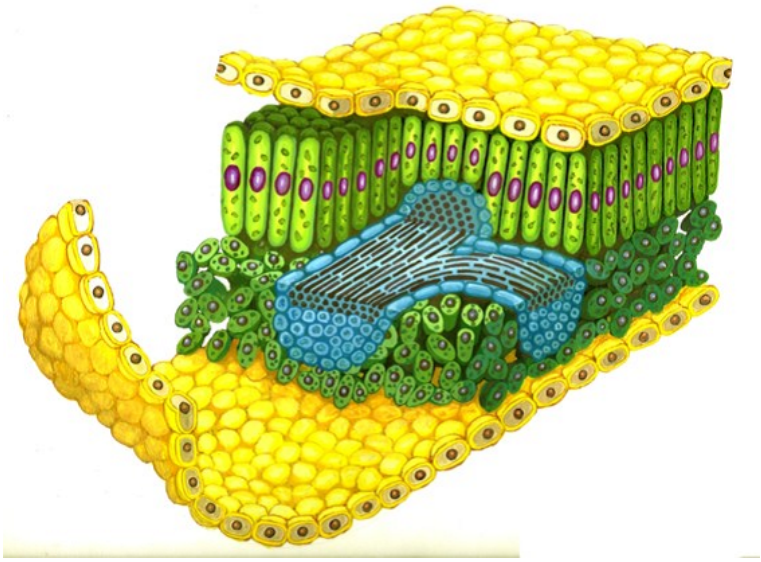


Figure 4.2: An example of an illustration that reveals internal structures (courtesy of Saeko Sato)

4.2 Related work

4.2.1 Texture synthesis

Texture synthesis algorithms take reference images as input and synthesize new images that appear similar. The technique is originated from the context of stochastic machine learning of image pattern [50, 115]. They synthesized a texture to see how successful their model learned the texture. Probabilistic models used for synthesis of such patterns include Markov random fields [22]. This model is later introduced to CG community [32] and inspired a great number of related work. Texture synthesis algorithms can roughly be categorized into four types:

frequency domain, pixel-based, patch-based, and non-periodic tiling.

Frequency domain Early systems used frequency domain techniques [55, 24]. Frequency domain approach is technically sound, but the quality is usually low and they can handle only specific types of texture. In addition, it is difficult to extend these algorithms.

Pixel-based approach The pixel-based approach [32, 149] uses a simpler strategy, which copies and pastes one pixel at a time. This algorithm is a direct application of Markov random field theory. Observing the neighboring pixels around the target pixel in the destination, the pixel that has the most similar neighboring pixels in the source image is retrieved and the central pixel is copied to the target. Its simplicity draws a great number of extensions. Ashikhmin proposed a method to restrict the search domain, resulting in more consistent image [7]. Wei et al. [148] and Turk et al. [143] applied pixel-based technique to synthesize textures on a surface mesh. Hertzmann et al. applied this technique to learn arbitrary filter pattern from example images and applied the filter to other images [59]. This work is called the *Image Analogies* system and further extended to generate geometric texture using volumetric data structure [11].

Patch-based approach The patch-based algorithm copies much wider area (called a patch) simultaneously [116, 31, 80]. The appropriate offset of the patch is computed by minimizing the image difference of the overlapped region and then the patch is fused to the target image by finding the optimal cutting line (optimal in the sense that the boundary is the least visible.) Such cutting line is found either by dynamic programming [31] or by computing maximum cut of the pixel graph [80]. This scheme produces much higher quality images than pixel-based algorithm, mainly due to its larger scale consistency. The main drawback is that the distinct features of the texture may be discontinuous around patch boundaries. Recent work try to overcome these problems by combining pixel-based algorithm [99] or locally deforming the patch [154, 99].

Non-periodic Tiling Non-periodic tiling approaches quickly generate texture patterns [132, 102, 19]. This algorithm first generates finite number of tiles that satisfy specific boundary conditions and then fill the target region by just laying out the tiles. The user just need to care that the neighboring tiles have matching boundary. The drawbacks of this technique is that it is difficult to compute an appropriate size of the tile (if the size is inappropriate, many tiles are necessary to have wide

variety of structure.)

4.2.2 Non-photorealistic modeling and rendering

This system draws inspiration from various non-photorealistic rendering (NPR) systems that focus on the communication of particular information rather than the simulation of light transport [45]. Our system is particularly influenced by the stylized rendering of 3D models that synthesize interesting 2D pictures by adding details to simple 3D geometries on the fly [83, 71]. In a similar spirit, we synthesize detailed textures on cross-sections of simple surface models.

Some systems also address the problem of authoring. [59] introduced a painting interface for directing the texture-synthesis process for various artistic expressions; [71] proposed an interface for painting a 3D model to specify rendering styles directly. Like these systems, our system provides a tailored user interface for intuitively designing volumetric illustrations.

4.3 User interface

4.3.1 Browsing interface

The system comprises two functions: browsing and modeling. The browsing interface is a subset of the modeling interface. The browsing interface is a standard 3D model viewer with an extension that allows inspection of internal textures using a cut operation. Rotating and translating the model are assigned to the right mouse button, and cutting the model is assigned to the left mouse button. If the right button is pressed on the model, the model rotates. If the right button is pressed elsewhere, the model translates parallel to the screen.

The user can cut the model by drawing a freeform stroke that crosses the model on the screen [68] (Figure 4.3). The cut object then opens automatically with animation [107] and the user can see the internal textures on the cross-section. The model closes when the user clicks an empty space.

Although our current implementation uses a cut operation that is based on a freeform stroke, the framework can easily be extended to support other interfaces for specifying cross-sections, such as 3D magic lenses [145] and two-handed operations using a prop and a plate [61].

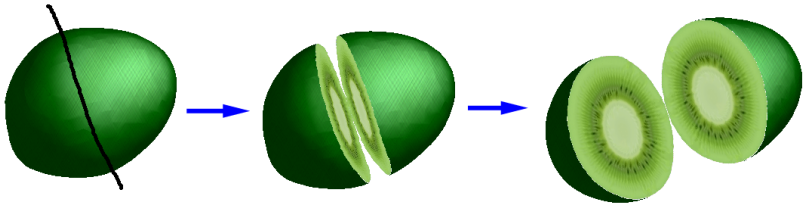


Figure 4.3: Inspecting internal structures using a freeform cut operation.

4.3.2 Modeling interface

The modeling operation starts by loading a predefined surface mesh (i.e., a surface mesh that delimits volumetric regions) and predefined 2D images. The goal of the modeling operations is to specify how given images are to be mapped to the interior of the given surface mesh, while existing methods usually specify textures on surfaces [49, 110]. The modeling process has the following steps.

- Cut the target 3D model and specify a 3D region to be textured by clicking on the cross-section.
- Choose one of the three texture types to use for the region.
- Import a reference 2D image.
- Establish the correspondence between the 2D reference image and the cross-section of the 3D model by providing the necessary guidance information.
- Repeat steps 1-4 for each 3D region.

We explain each step in turn.

4.3.3 Specifying a region to be filled

First, the user cuts the target surface mesh using the freeform cut operation. The cross-section reveals the internal structure of the model and can be divided into several closed regions. For example, a model of an egg might consist of a spherical surface mesh representing the yolk enclosed by a larger sphere representing the egg white. In this case, the cross-section has two regions. The user can simply click on the target region on the cross-section to specify the 3D region to be textured.

4.3.4 Selecting a texture type

When the user clicks on the target region on the cross-section, the system opens a dialog box that is used to specify the texture type (Figure 4.4a). Once the user has specified the texture type, the system opens a separate pane that shows the reference image and a reference cube (Figure 4.4b). The reference cube is an intermediate representation that visualizes the relationship between the 2D reference image and the 3D region. Steps 3 and 4 differ slightly for each texture type. Therefore, we first introduce the three texture types and then explain the steps for each.

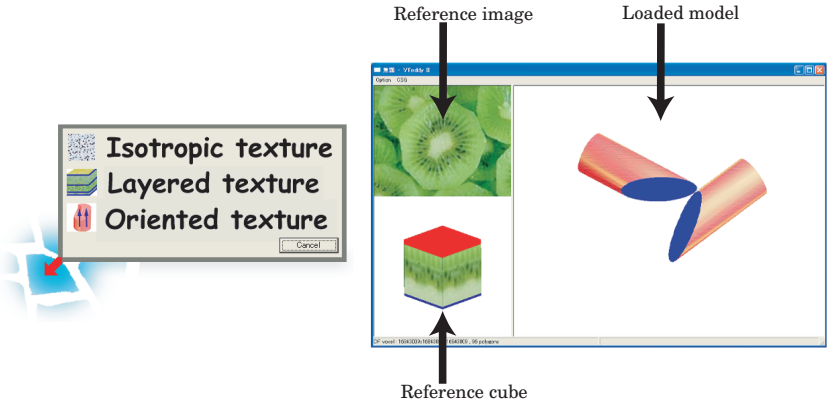
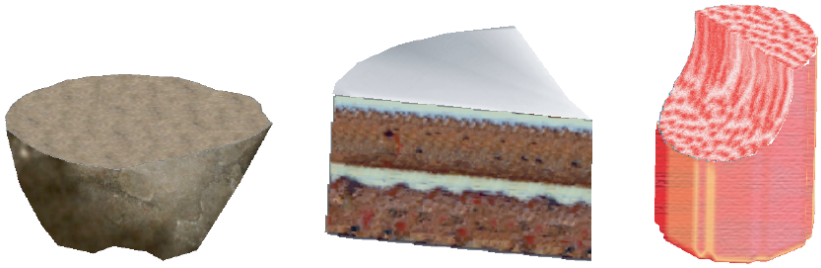


Figure 4.4: Window layout of the system for assigning textures to a model.

The current system supports three types of textures: isotropic, layered, and oriented (Figure 4.5). Isotropic textures have a uniform distribution in the 3D space with no dependency on position or orientation. All of the cross-sections of an isotropic texture look similar, regardless of their location or orientation. Examples include a sausage, a sponge, and any other material that consists of isotropic elements. Layered textures have varying appearances according to their position in the axial or radial direction. Examples include kiwi fruits, human skin, tree trunks, and leaves. Layered textures require depth information for the target 3D region. Finally, oriented textures are defined by both a reference image and a flow direction; the appearance of an oriented texture depends on the orientation of the cut-plane relative to the flow-direction (Figure 4.5c). Examples include muscle, plant stems (of monocotyledonous

plants), and any other material that consists of bundled long fibers. The oriented texture requires that the flow orientation in the target 3D region be specified.

We do not claim that these three types cover all possible real-world textures. Some textures can be combinations of layered and oriented textures, and some have more complicated structures. We support these three types in the current implementation because they are relatively easy to understand, they can be specified with a simple interface, and they cover a wide range of interesting textures that are commonly seen in organic materials. Future work will investigate other types of texture filling.



(a) Isotropic (b) Layered (c) Oriented

Figure 4.5: Examples of texture types

4.3.5 Isotropic textures

The user imports a reference image by dragging an image file and dropping it onto the source image area in the main window. The user can choose a specific region of the reference image by rubber banding. The selected region is immediately transferred to all faces of the reference cube and to the cross-section of the surface mesh using a texture-synthesis technique. No guidance information is required in this case (Figure 6).

4.3.6 Layered textures

A layered texture requires additional guidance information in the surface mesh to specify the mapping between the image and the model. The user first draws two freeform strokes that correspond to the upper and lower bounds of the layer on the imported reference image (Figure

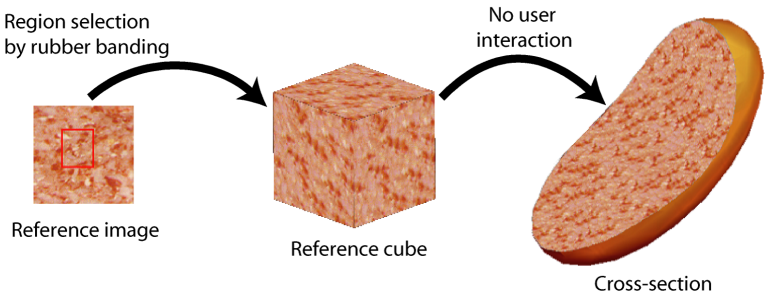


Figure 4.6: Using an isotropic texture. The user specifies a region to use by rectangular rubber banding, and it is then transferred to all faces of the reference cube and the cross-section.

4.7, left). The first and last points of the two strokes are connected by straight lines to carve out a portion of the input image. Then, texture-synthesis techniques fill the side faces of the reference cube using the portion of the image as a reference (Figure 4.7, middle). The user then specifies two corresponding upper and lower bounds in the surface mesh by clicking a boundary or drawing a stroke on the cross-section (Figure 4.7, right). Clicking on the boundary selects the associated surface region and the stroke becomes the constraint on the cross-section (Figure 4.8).

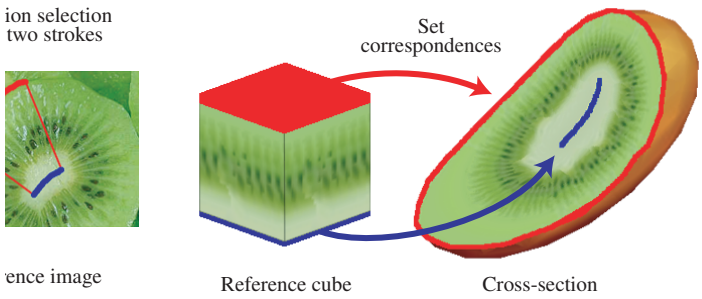


Figure 4.7: Using a layered texture. The red and blue marks on the reference image define the upper and lower bounds for the algorithm. The texture-synthesis techniques fill the side faces of the reference cube, using the image between the two marks as a reference. Similarly, the red and blue marks on the surface mesh define the upper and lower bounds. The texture on the cross-section is synthesized using the reference image as an example.

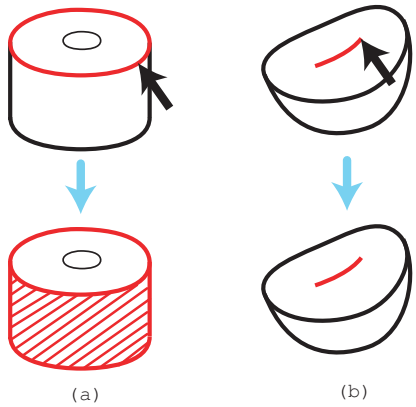


Figure 4.8: Clicking a boundary on the cross-section selects the associated surface region as the upper or lower bound (a). We assume that the user has predefined the correspondence between a boundary and a surface region (provided as a surface mesh). A stroke drawn in the interior of a cross-section becomes a bound on the cross-section (b).

4.3.7 Oriented textures

An oriented texture has distinct appearances in cross-sections that are perpendicular and parallel to the flow orientation. Our current implementation asks the user to provide a reference image for the cross-sections perpendicular to the flow orientation. The reference image must be isotropic. The user specifies a rectangular region in the reference image using rubber banding. The system then synthesizes the top face of the reference cube using the selected region as a reference. Then, it generates a reference volume by sweeping the image vertically and shows textures for the cross-sections parallel to the flow orientation on the side faces of the reference cube.

We experimented with other strategies for specifying the reference volume. One let the user specify the images for the side faces of the volume, and the other let the user specify the images on both the top and side faces. We used Wei's volumetric texture-synthesis technique to synthesize the reference volume in these cases [148]. We did not pursue this direction further in our current implementation because it was too slow and the quality of the resulting volume was unsatisfactory. However, it is sometimes desirable to specify the appearance of side faces, and in the future we will investigate efficient supporting strategies.

An oriented texture requires the user to specify the flow field across

the target region as guidance information. This is done by drawing short arrows that represent local flow directions on the cross-section and surface of the region (Figure 4.9, right) [143]. Our current implementation does not allow the user to draw arrows that are not parallel to the cross-section. Therefore, the user should cut the model parallel to the desired flow orientation.

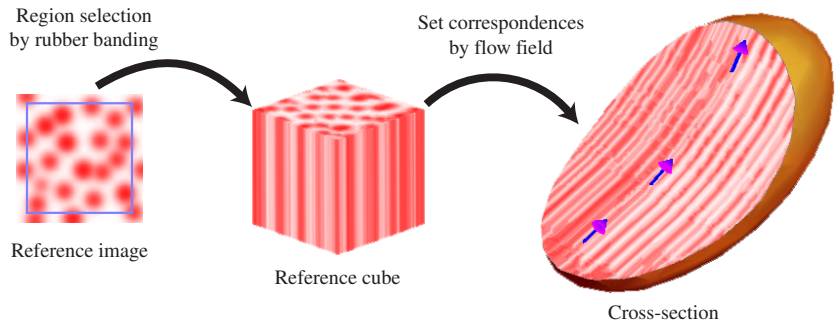


Figure 4.9: Using an oriented texture. The user specifies a region to use by rectangular rubber banding, and it is then transferred to the top face of the reference cube. The system generates the side faces by sweeping the image on the top face vertically. The user draws short arrows on the cross-section to specify the flow orientation.

4.4 Algorithms

This section describes the algorithms and implementation details for synthesizing a texture for a given cross-section using the reference images and the associated guidance information. The cutting operation cuts the model by sweeping a user-drawn 2D stroke in the direction perpendicular to the screen (we use orthogonal projection). Using this curved plane, a model is divided by CSG operations [64]. The parametrization of the cross-section is given as follows: the y-axis is defined along the cutting stroke and the x-axis is defined along the sweeping direction. The starting point of the stroke becomes y=0 and the corner of the model's bounding box nearest the screen becomes x=0. The imported surface mesh is scaled so that the size of the bounding box equals 1, and the pixel size of the synthesized texture is 1/150 1/400 in our current implementation. A cross-sectional bitmap is obtained by synthesizing the pixel color at each grid point on this parameterized cross-section within a rectangular region that covers the model's bounding box. If

the user wants to change the scale of the synthesized texture, the original image must be scaled beforehand. We describe our texture-synthesis algorithms for each of the three texture types.

4.4.1 Isotropic textures

An isotropic texture has no dependency on position or direction. Therefore, we simply use a standard 2D texture-synthesis algorithm to construct a 2D texture image for the cross-section [149, 19, 80]. The system uses the selected region in the original reference image as the reference for the synthesis directly. The reference cube exists only to give feedback to the user. Note that there is no guarantee of obtaining exactly the same image when a model is cut twice at the same cross-section. However, since our aim is to convey a volumetric impression rather than to generate consistent volumetric data, we believe that this simple approach is sufficient.

4.4.2 Layered textures

A layered texture has an appearance that varies according to the depth. Therefore, the algorithm needs depth information for each pixel in the reference image and target cross-section. Figure 4.10 illustrates the overall process. The system generates a reference control map for the reference image and a target control map for the cross-section. A control map is a grayscale 2D image in which the floating-point pixel values indicate associated depth values.

The grayscale values of the reference control map represent a smooth 2D depth field constrained by the two bounds provided in the reference image. The mark for the upper bound (red curve in Figure 4.10) is associated with depth value 0 and that for the lower bound (blue curve in Figure 4.10) is associated with depth value 1. We use a 2D thin-plate interpolation technique [142] to compute this smooth 2D depth field (reference control map in Figure 4.10). The depth field is given as a continuous function that returns a scalar value for a given 2D position. This function is sampled on each pixel location on the reference control map, which has the same resolution as the reference image.

To construct the target control map, the system first computes a 3D scalar field using the user-defined upper and lower bound 3D geometries as constraints. The upper-bound geometry (a surface region or a line in 3D space) is associated with depth value 0 and the lower-bound geometry

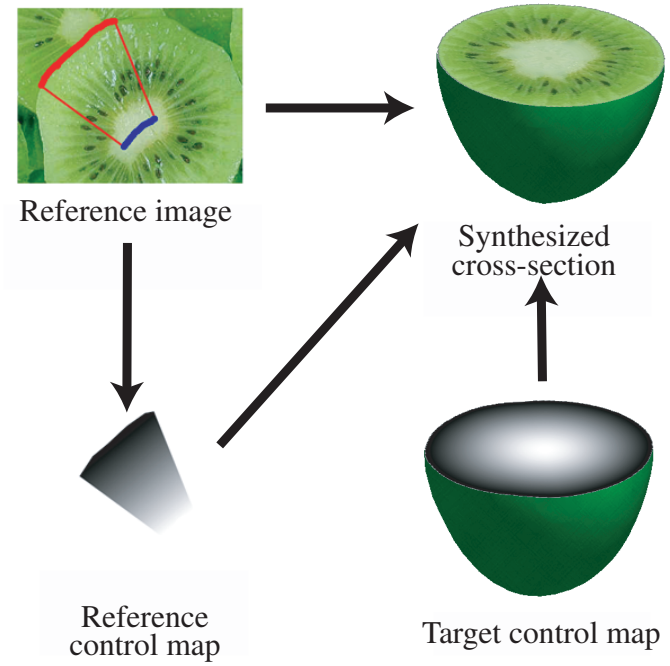


Figure 4.10: Overview of texture synthesis for layered textures. The system synthesizes the texture bitmap for a cross-section by using the reference image, the reference control map associated with the reference image, and the target control map associated with the target texture.

is associated with depth value 1. Again, we use Turk and O'Brien's 3D thin-plate interpolation technique to construct this smooth 3D scalar field [142]. When the user cuts the model, the system generates a 2D target control map by sampling the aforementioned 3D depth field on the cross-section.

Given the reference image, reference control map, and target control map, it is now possible to start the texture synthesis process using a pixel-based technique. This synthesis process is similar to field distortion synthesis, which is used for texture synthesis on surfaces [143, 158]. The differences are as follows (also see Figure 4.11):

1. An orientation field is computed from the target control map as the gradient direction in the target control map. 2. Each pixel in the reference image also has an orientation computed as the gradient direction in the reference control map. The neighboring structures are computed according to this orientation. 3. The order of synthesis is determined using the depth value of pixels. 4. When synthesizing a pixel

in the target image, the search space in the reference image is restricted by the depth value of the pixel being synthesized; pixels whose depth value equals that of the synthesizing pixel are subject to the search. In our implementation, real-valued depth values are discretized into 32 levels and the pixels are indexed according to the discretized scalar level during preprocessing.

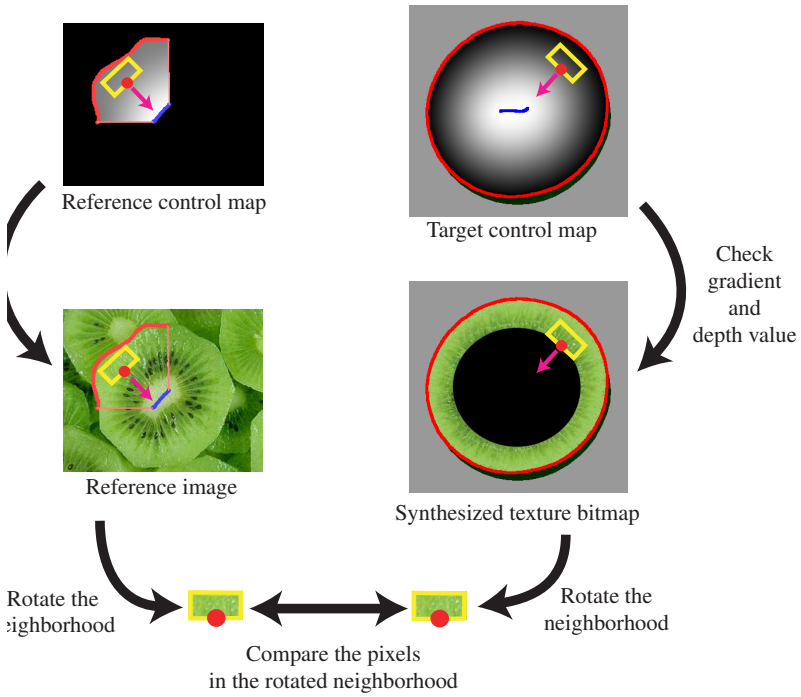


Figure 4.11: Computing the pixel color in the synthesized texture. First, the system rotates the neighborhood of the target pixel so that the gradient of the target control map matches that of the reference control map. Then, the system compares the rotated neighborhood of the target pixel with that of the reference image around a pixel whose gray-scale value in the reference control map is identical to the gray-scale value in the target control map.

4.4.3 Oriented textures

The synthesis process for an oriented texture requires a 3D reference volume and a flow field defined in the 3D region in the surface mesh. As already discussed, the 3D reference volume is obtained by sweeping the top face of the reference cube vertically. The top face is synthesized

using the selected region in the reference image as the example. The reference volume is oriented vertically (each pixel in the reference volume is associated with a vertical flow vector) and the size of the reference volume is $64 \times 64 \times 64$.

The construction of the 3D flow field again uses the thin-plate interpolation technique [142], employing user-defined arrows as constraints. A smooth scalar-valued interpolation function is constructed for each xyz component. The flow field is constructed by combining them and normalizing the resulting flow vectors. This approach can produce singular points where no flow is defined. However, singular points rarely appear on the cross-section in our system because the user cuts the model using a freeform stroke. When singular points do appear, we assign a random orientation to the pixel. After obtaining the 3D flow field, the system generates a 2D target control map for the given cross-section (each pixel is associated with a flow orientation) by sampling the flow vectors along the cross-section.

At this point, we have the 3D reference volume (associated with vertical flow orientation) and the 2D control map for the cross-section that contains the flow vectors for each pixel. Given this information, the system computes the color for each pixel in the cross-section by finding a pixel in the reference volume that has a similar neighborhood [149]. The similarity of neighbors is computed as follows. The neighborhood of the pixel on the cross-section is approximated by a small flat rectangle. For each pixel in the reference volume, the system samples the neighborhood in a corresponding small flat rectangle whose slant angle (angle between the rectangle and the vertical flow vector) equals that of the rectangle on the cross-section (Figure 4.12). Rotation about the flow vector does not matter, because we assume that the reference volume has an isotropic structure on cross-sections perpendicular to the flow orientation.

Given the rectangle on the cross-section and that in the reference volume, the system can now compare the similarity between the two.

Ideally, the system should sample every possible small rectangle in the entire reference volume. For performance reasons, however, the system samples pixels in a slanted 2D square region at the center of the reference volume and uses the resulting image as a reference for standard 2D texture synthesis. In our current implementation, the size of the square is 45×45 (to fit within the reference volume completely). We further reduce the computation time by caching the sampled image

using discretized slant angles as keys (a discretization step is $\pi/32$).

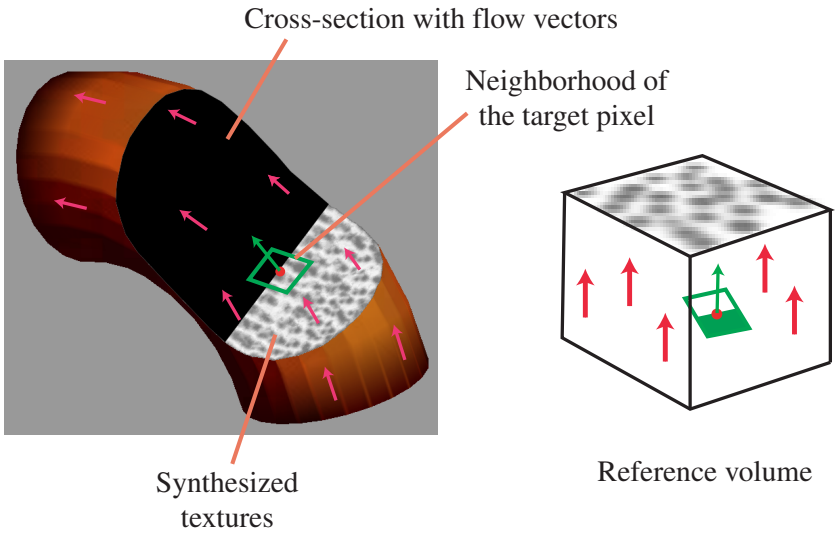


Figure 4.12: Finding pixels that have a similar neighborhood in the reference volume for each pixel in the cross-section.

4.5 Results

Figure 4.13 shows some volumetric illustrations that were created using our system. The amount of data in the models, time for design, and time for synthesizing the cross-section are summarized in Table 1. For isotropic and oriented textures, we used a three-level multi-resolution pyramid with a 3×3 square neighbor for lower resolution and a 5×5 L-shaped neighbor for higher resolution. For layered textures, we used a 3×3 square neighbor for lower resolution and a 5×3 rectangular neighbor for higher resolution. We used a laptop computer with a Pentium M 1.6-GHz processor and 1 GB of RAM. Although in some cases it took more than 10 seconds to obtain the result for the finest resolution, this did not impede the interactive modeling process because progressive synthesis of cross-section images frees the user from waiting for the final result each time. Since the resolution of the cross-section is approximately 300×300, the quality is comparable to that of 300^3 colored voxels, which would require approximately 80 MB of storage.

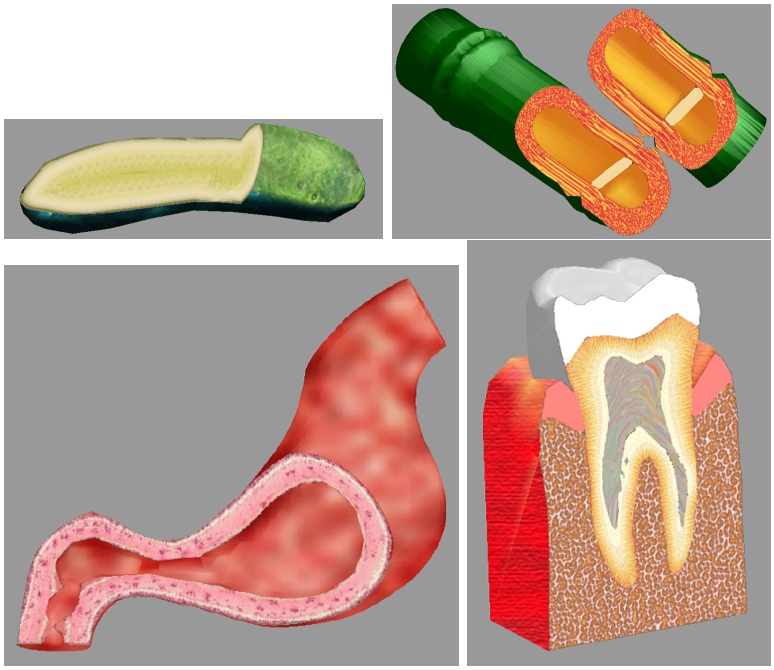


Figure 4.13: Results (Cucumber,Bamboo,Stomach,Tooth)

Title	Amount of data (without compression)	Modeling time (excluding mesh editing)	Synthesis time
Meat (Figure 4.1)	622 kb	90 sec	22 sec
Cucumber	53 kb	40 sec	4 sec
Bamboo	291 kb	30 sec	14 sec
Stomach	402 kb	15 sec	18 sec
Tooth	307 kb	120 sec	32 sec

Table 1 Statistics for the example models

4.6 Discussions

In this chapter, we described a modeling and browsing system that adds internal textures to a surface mesh. The user provides 2D reference images and a surface mesh with simple guidance information that specifies the correspondences between them. When the user cuts the model, the system synthesizes cross-sectional images using 2D texture-synthesis

techniques. This system would be useful for conveying volumetric information, such as between a teacher and students, a doctor and a patient, or a virtual-reality content-provider and consumers. Although the lack of real volumetric data makes some applications impossible, such as translucent rendering or volumetric simulation, our lightweight data representation may well be useful in many applications such applications as games or virtual world construction, where precision or volumetric consistency is not the main concern. It is because such application domains, visual effects and necessary computational resource trades.

Nevertheless, our system has several limitations. One is the computational cost for synthesis. Although the required memory is much less than that of 3D textured volume, necessary computation for generating quality cross-section is rather high. More specifically, a layered texture rotates the neighbors so that the gradient of the target control map matches that of the reference control map, while an oriented texture generates a 2D reference image by slicing the reference volume at an appropriate angle for each pixel. These processes require more computation than standard texture synthesis. The quality of the synthesized image also requires improvement, in part because of the cascading resampling and resulting distortion. In future work, we will improve both the performance and quality of the overall process. In the Appendix A, we propose an idea to improve the quality of the cross-sectional image by introducing patch-based technique (with significant loss of computation time, though).

One interesting avenue for future research is to enhance images of the cross-section by using the information that is embedded in the pixels of the reference map. For example, we can add text annotation and displacement mapping on the cross-section. The pixels of the reference image are associated with annotations and displacements, and the system adds them to the corresponding pixels on the cross-section. Text annotation allows designers to add textual explanations to the internal material of 3D models, which would be useful for educational and communication applications. Displacement mapping can add realism to a cross-section, as cross-sections of real objects cannot be perfectly flat. Adding such additional information to 3D regions directly can be very difficult, but is straightforward in our framework.

Chapter 5

Contour-based segmentation interface

Volume segmentation is the process to carve out the region of interest (ROI) of the volume data. This is one of the most important operations for volume data since there are a great number of applications such as volume computation (by counting the number of the voxels in the ROI), enhanced volume rendering, and intelligent interaction to volumes. The image segmentation technique has been and still is a central topic of computer vision research and volume segmentation can be treated as a straightforward extension of 2D image segmentation. Unfortunately, successful and fully automatic general-purpose image segmentation still does not exist. Therefore, it is important to efficiently supply necessary information to the system manually, while keeping the maximum automaticity.

In the previous section, we presented a system that generates implicit volume data from scratch using a sequence of simple gestural operations. On the other hand, as described in Section 2.2.1, the main source of volumetric data is the scanning of real-world objects. The typical scanning devices capture the target objects as a set of 2D cross-sectional images. A naive but most robust, and therefore still used segmentation method is to segment regions in each 2D cross-sectional image, resulting in a set of parallel cross-sectional contours. One drawback of this approach is that the number of images is usually large. Therefore, the user cannot help but work on a part of the images. In this case, how contours between cross-sections should be connected is often ambiguous. In this chapter, we propose an algorithm that enumerates all possible cases of the correspondence of contours, aiming at a smart system by which the user can specify the topology of the contour easily. The significance of listing all cases lies in the possibility to automatically find the

global optimal solution with respect to the interpolating shape objective function.

5.1 Background

Computed tomography (CT) and magnetic resonance (MR) devices enable us to easily obtain cross-sectional images from physical objects. When polygonal meshes are constructed from sparse cross-sectional contours, each interspace between adjacent cross-sectional planes has to be filled by interpolation. If the shape of the object is simple, the interpolation is trivial. However, handling real-world medical data, such as human skeleton or brain, becomes difficult as the spacing between slices increases and when the numbers of contours in adjacent cross sections differ.

There has been many previous work in this field [40, 121, 141]. However, the existing work focuses mainly on how to obtain a smooth transition rather than on the correspondence of topological structures. For example, in Figure 5.1, it is not easy to determine the correct topological structure for the interspace. Suppose that the six contours are obtained from a blood vessel. One may speculate that two streams meet between the two slices as shown in Figure 5.1(a). However, if *shape-based interpolation*[121] is used, for example, the result is four pillars, as shown in Figure 5.1(b) (arising from the four areas of overlap in a top view).

It is usually difficult to determine topological structure without additional knowledge about the object. In Figure 5.1, the four pillars could actually be the correct interpolation for some other physical object. Although topological structure is important, a user's knowledge is not limited to topology. The topology of an object describes just one part of the full shape information - the skeleton information. Examples of such information are the integral of the surface curvature, volume of the object, or its interference with other objects' surfaces. However, these quantities cannot be computed or modified, until the topological structure is determined.

In this chapter, we propose a method that enumerates all possible topological structures. A polygonal mesh is then constructed for each result in the enumeration. The patterns are then sorted by an objective function that evaluates how good the interpolated shape is. We view this algorithm as a basis for pseudo-automatic topology generator. Although

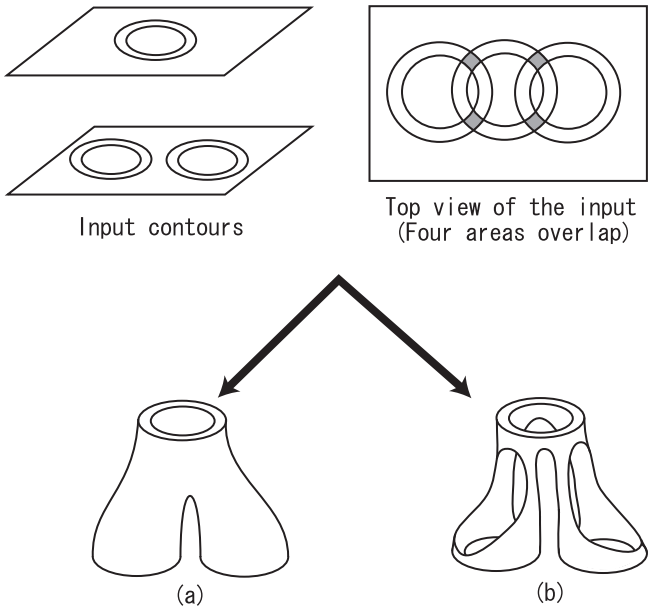


Figure 5.1: Ambiguity in interpolating contours where the interpolated object could be either a bifurcating pipe (a) or four pillars (b)

our current system generates a huge number of possible patterns, listing all possible cases enables to find the global optimum, which probably be the best guess of the system. The objective functions bring out the full performance and the user interaction becomes simple. We tried to sort enumerated results by a simple objective function and observed that it works fine in many cases.

5.2 Related work

Over two decades have been devoted to research on interpolation of contours [75]. Fuchs et al. presented an algorithm to convert the problem of finding the correspondence between points on contours into the problem of finding the minimum cost path in a directed toroidal graph, although this work did not deal with bifurcation [40]. Later work has taken bifurcation into account. The original toroidal graph was extended to deal with bifurcations and holes by Shantz [125]. Shinagawa et al. extended the toroidal graph to be continuous.[129] Christiansen et al. proposed an algorithm based on the connection of the nearest points [18]. Ekoule et al. proposed another method that handles highly

complex bifurcations and convex contours [33]. The Delaunay triangulations can also be used for solving the correspondence problem [15]. These methods typically use contours defined by pieces of straight lines as input, calculate the correspondence between junction points, and output triangulated meshes.

There is yet another approach where a 2D function is defined for each cross section and is interpolated in 3D. In this framework, each cross-sectional contour is first converted to a binary image and then converted to a grayscale image where the intensity of a pixel is computed as the distance from the contours [121, 57]. It calculates the gray value as the shortest signed distance from the contours (a positive value for the interior of the contours and a negative for the exterior) and is linearly interpolated in 3D. The final surface is then extracted as the isosurface of the 3D field distance function. Recently, shape-based interpolation has been further extended by using information about the correspondence of contours [141]. Distance field manipulation is similar to these approaches [109].

If the spacing between slices is quite narrow, it is possible to solve the contour interpolation problem as the interpolation of unorganized points [65, 8]. In this scenario, each contour is converted to a set of unorganized points having no connectivity information. There is some work involved in the reconstruction from this set of unorganized points, mainly in the context of approximating the object surface obtained by range scanners or stereo matching algorithms.

Other approaches include work by Cong et al., in which a numerical solver was used that directly calculates the functional value in 3D [20]. Yet another method uses singularity and the distance between contours in determining bifurcation, [128] which has similarities to our algorithm. However, there is still a possibility with this method of rejecting a correct result, since it only uses the distances between contours and the genus as the knowledge of an object. Our method just outputs possible results, giving further information about an object such as the curvature limit or the volume of the closed area.

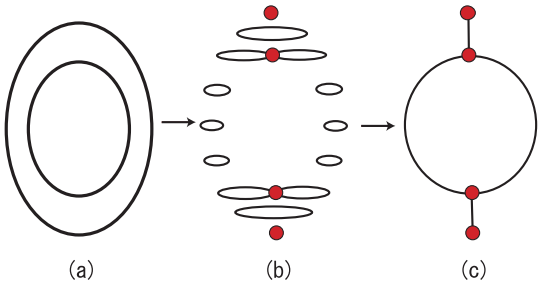


Figure 5.2: Construction of a Reeb graph where the original object (a) is sliced (b) and the corresponding Reeb graph (c) is constructed by handling each contour as a point

5.3 Algorithm

5.3.1 Notation

Reeb graphs

The first tool we use is the so-called *Reeb graph*. A Reeb graph is a graph that gives a simple representation of the topology (bifurcation status) of a continuous function defined onto an object. One of the simplest functions is the height function h. The height function h simply returns the height (e.g. the z value) of the point of the surface.

A Reeb graph of an object is built as follows. As depicted in Figure 5.2, the object is sliced and each cross-sectional contour is represented as a point of the Reeb graph. The points where two (or more) contours meet or a contour disappears are called the *singular points* (see Figure 5.3). Mathematically, the singular points are defined as the points where

$$\frac{\partial h}{\partial x} = \frac{\partial h}{\partial y} = 0 \tag{5.1}$$

holds (see Morse theory).

The importance of the number of singular points is as follows. If the input contours are as shown in Figure 5.4(a), then from traditional assumptions of bifurcations and from the distance between contours, the topology shown in Figure 5.4(b) results. However, if the user specifies knowledge about the number of singular points, (in this case, "two",) the topology shown in Figure 5.4 (c) should be the correct answer.

The singular points are represented as the nodes of the Reeb graph [128]. Singular points where more than two contours meet are said to

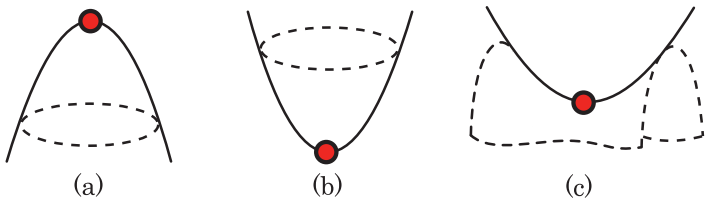


Figure 5.3: Singular points: a peak point (a), a pit point (b), and a saddle point (c)

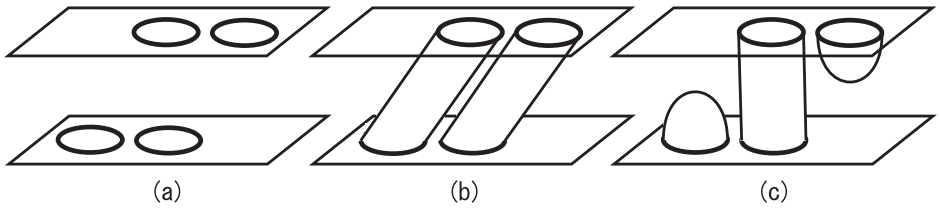


Figure 5.4: An example in which the number of singular points is important where the input contours(a) are interpolated either using general assumptions only(b), or with knowledge about the number of singular points(c)

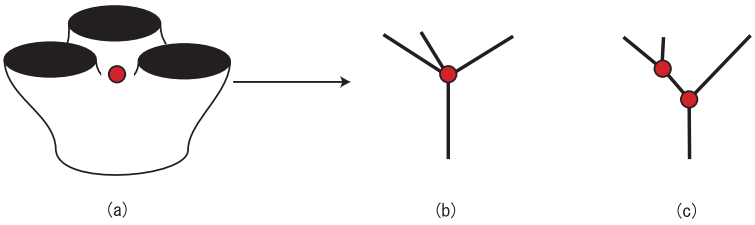


Figure 5.5: An example of a degenerate singular point (a) where the singular point connects three contours to one contour (b) which can also be interpreted as the combination of two 2-to-1 singular points that coincide at this point (c)

be *degenerate* (see Figure 5.5). These degenerate singular points can be decomposed into a sequence of simple 2-to-1 singular points. Although it is possible to use other functions,[60] the Reeb graph of the height function is sufficient for this system.

Contour trees

A *contour tree* represents the circumscription relationship of contours in a slice, and similar notions have been used previously [56]. Each node in a contour tree represents one contour and each edge represents the circumscription relationship between two contours. A contour circumscribes all contours which are its descendants in the contour tree. In general, there is more than one separate object in an image. In this case, more than one connected component exists in a contour tree. For convenience, we assume that there is a virtual contour that circumscribes all the contours in the image and the corresponding root node is added to the contour tree. Such a contour is called a *Virtual Hollow Contour (VHC)* (see Figure 5.6). A singular point exists at the height where the topology of the contour tree changes [130].

5.3.2 Outline of the proposed algorithm

We assume that contours are already extracted from input images and the contour trees are built. The outline of the algorithm is as follows.

- Transform a contour tree of each slice by the operations that merge or eliminate nodes. This corresponds to the transformation of the Reeb graph. Each elementary transformation generates a singu-

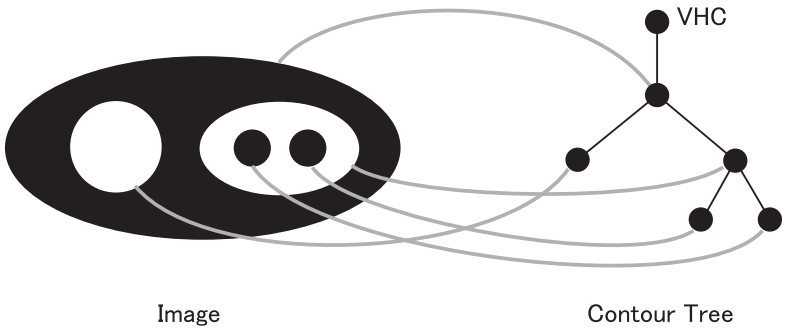


Figure 5.6: An example of a contour tree where each contour in an image corresponds to a node in the contour tree and a parent-child relationship in the contour tree implies a circumscription relationship in the image. The root node of the contour tree is the VHC

lar point. At this stage, all possible transformation patterns are enumerated. We limit ourselves to the cases where nodes are only merged or eliminated rather than divided or added because the increase in the number of nodes in these cases leads to an infinite number of solutions. This restriction is further discussed later on.

- Two of the transformed contour trees are compared, each of which is one of the enumerated contour trees from adjacent slices. If both trees have the same structure, the transformation operations applied in the previous stage are validated.

Transformation of a contour tree

According to the Morse theory, there are three types of non-degenerate singular points. In what follows, we denote them by e^0,e^1 and e^2 using the same notations as the cells corresponding to the singular points [130]. There are twelve types of Morse operators defined [130]. If operations that increase the number of contours are allowed, the number of possible Reeb graphs becomes infinite. Moreover, it causes an unnatural result as in Figure 5.7, where topological structure in the interspace is complicated (more specifically, when the object is sliced by a plane at the middle of the original upper and lower cross sections, the number of cross-sectional contours is larger than that on either of the original planes.)

For this reason, we assume the number of contours will never in-

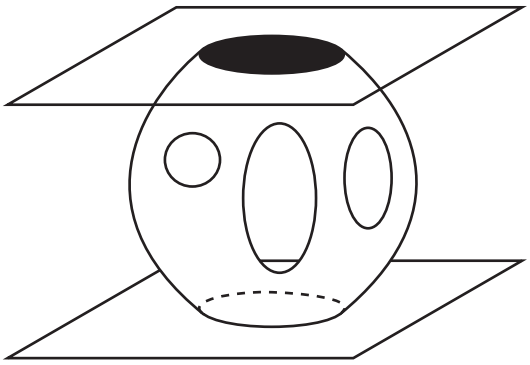


Figure 5.7: An example of an unnatural result caused by an increase in the number of contours

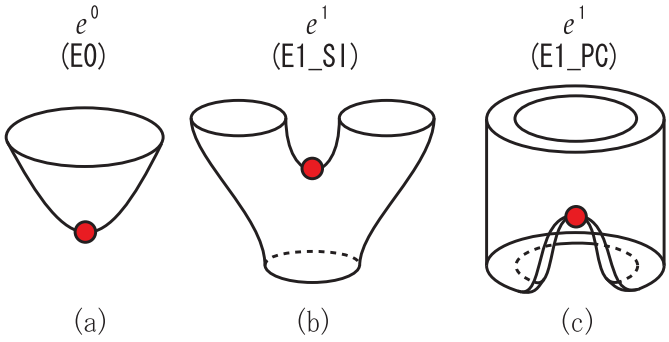


Figure 5.8: Three types of Morse operators which correspond to the singular points e^0 and e^1 where e^0 removes a contour and e^1 connects two contours

crease at the enumeration stage. Thus, we adopt only three types of Morse operators. The singular point e^0 is considered to be the point where a contour disappears as shown in Figure 5.8(a) (if seen from top to bottom); i.e., this removes a node in a contour tree. The corresponding operator is **E0**. The singular point e^1 is a point where two contours are connected. Since there are two types of operations in the case of the singular point e^1 (that is, it connects either sibling contours as shown in Figure 5.8(b) or the parent and child contours as shown in Figure 5.8(c)), the corresponding two operators (**E1_SI** and **E1_PC**) are defined. Note the difference between the singular points and the Morse operators. E0 is the operator that corresponds to the singular point e^0, whereas E1_SI and E1_PC correspond to e^1.

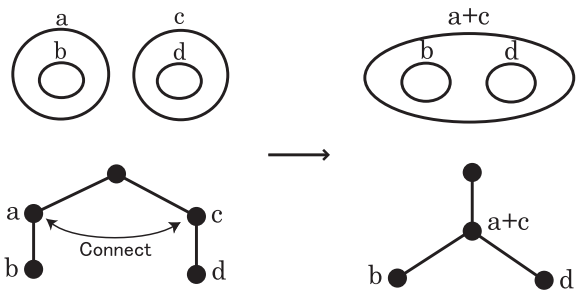


Figure 5.9: An example of E1_SI where descendant nodes **b** and **d** are merged after the connection

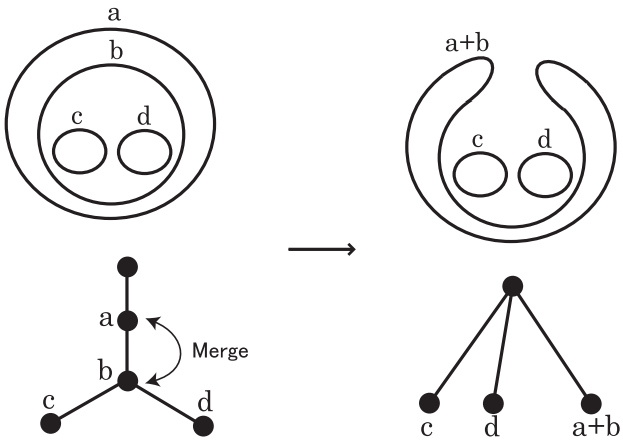


Figure 5.10: An example of E1_PC where after the contour **a** and **b** are merged, **c**,**d** and **a+b** become sibling nodes

When one of the three operators is applied to a part of a contour tree, the remaining part of the contour tree is affected accordingly [130]. If E0 is applied, all the descendant nodes must be removed simultaneously to avoid self intersection that never occurs in the case of natural solid objects. When E1_SI is applied, the descendants of the connected two contours are simply merged and become the descendants of the newly created node (see Figure 5.9). The E1_PC case is more complicated (see Figure 5.10). In this case, contour **a**(parent) and contour **b**(child) are connected. The newly created contour **a+b** and the descendants of **b** (**c,d**) become siblings. The general case is shown in Figure 5.11.

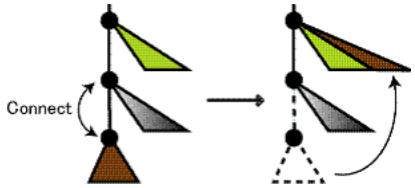


Figure 5.11: When a parent and a child are connected by E1_PC, the descendant nodes of the child become sibling nodes of the parent node.

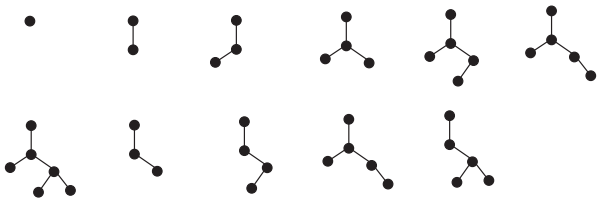


Figure 5.12: All the enumerated patterns by the application of E0 to the contour tree shown in Figure 5.6

Enumeration

Enumerating every possible correspondence of contours amounts to enumerating all the possible transformations of contour trees and matching them, which in turn amounts to seeking all possible combinations of the aforementioned operators. A problem is the complex behavior of the remaining parts of the tree when the operators are applied. We propose the following algorithm.

1. Applying E0

   At first, we enumerate all the combinations of E0. To do this, we traverse the contour tree from the root(VHC) and set a flag at each node indicating whether to apply E0 in breadth-first order or not. As an exception, the root node (VHC) is never marked. If a node is marked as "apply E0", all the descendants are deleted. An example of the result at this stage is shown in Figure 5.12, where the contour tree shown in Figure 5.6 is used as input.

2. Applying E1_SI and E1_PC

   Next, E1_SI and E1_PC are applied. These operators merge nodes in the contour tree. As we stated previously, E1_PC creates a

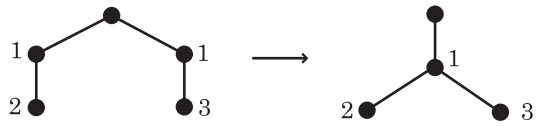


Figure 5.13: An example of E1_SI application by indexing, where both children of VHC have index "**1**" and these two nodes are connected by E1_SI

complex structure in the remaining part of the tree. To handle this, we again mark the nodes. At this stage, both operators connect nodes several times and eventually a certain number of nodes remain, regardless of the applied operators. To describe this, we assign the same index number to the original nodes which merge to one node. Figure 5.13 depicts an example of the correspondence between indexing and the actual application of the operators. In this case, nodes that have index "**1**" are connected. The applied operator is E1_SI because the original two nodes with index "**1**" are siblings. Every time a merge operation is applied, it is necessary to move the other part of the tree according to the type of the operator. This is repeated until no two neighboring nodes have the same index. If any two nodes still have the same index, the index assignment is invalid (e.g., Figure 5.14). This test is performed for every combination of index assignments.

By converting the combination of operators to the indexing problem, the order of operators is lost; e.g., the difference between Figure 5.15(a) and Figure 5.15(b) is not distinguished. However, we do not regard this difference as important because in many cases, the height of the singular points depends on what kind of smoothing algorithm is used in the final mesh reconstruction. In this work, the constructed Reeb graph is just used for determining the initial mesh. The final surface shape is not necessarily the same as the original Reeb graph.

Matching the contour trees

After the possible contour trees are enumerated, the two trees enumerated from adjacent cross sections are compared. If they have identical structure, the two contour trees can be matched. In this case no

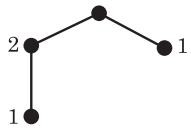


Figure 5.14: Impossible assignment of indices where two nodes with index "**1**" are contained and these nodes cannot be connected because they are neither siblings nor parent-child nodes

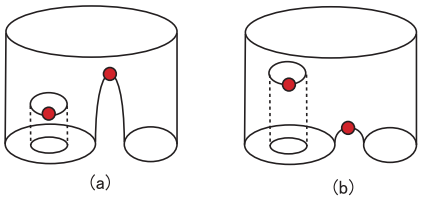


Figure 5.15: Examples of singular points at different heights

more transformation of contour trees (singular points) is necessary to connect the contours.

5.4 Implementation

5.4.1 Initial mesh construction

Once the correspondence of contours is determined, the surface of the object is constructed. Although construction of the surface is not essential in this system, we propose a simple procedure here. At first, the position of each singular point is calculated by the following two steps.

1. Determine the two dimensional positions of the singular points on the cross-sectional plane. If the singular point is e^0, the position is simply the center of gravity of the contour connected to the singular point. If the type of singular point is e^1, the position is calculated as the center of the closest two points[18] (see Figure 5.16).

2. Determine the height of each singular point within the corresponding interspace. The height order of singular points corresponds to the application order of the Morse operators (see section 3.2). The

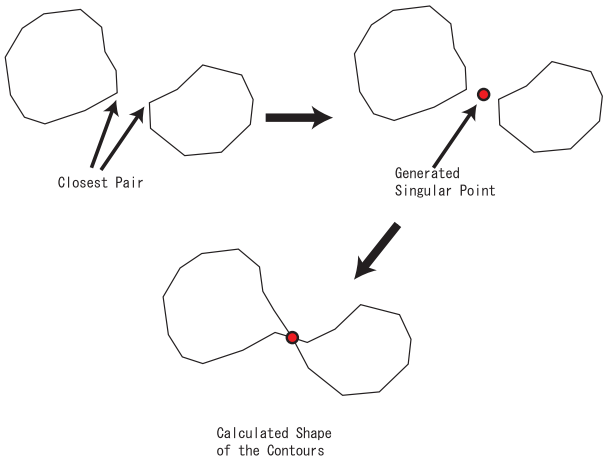


Figure 5.16: Calculation of contour shapes where a new singular point is generated at the center of the closest pair of contours

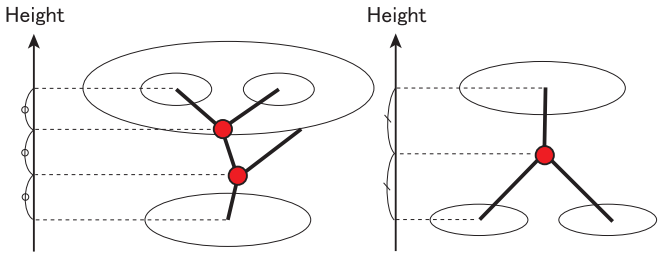


Figure 5.17: Calculation of the height of each singular point, where each singular point is located evenly in the height direction retaining the height order within the connected component

singular points are located evenly in the height direction retaining the height order (see Figure 5.17). Note the height difference between singular points is calculated for each connected component of the Reeb graph within an interspace.

Finally, the mesh is built. It is an easy and well-studied problem because all the bifurcation points and the corresponding contour shapes are already calculated. It means that the topological shape is already known and only the one-to-one correspondence of contours should be considered. In addition, because newly created contours have similar shapes to the neighboring contours as shown in Figure 5.16, it is trivial to find the correspondence of points between the new contour and the

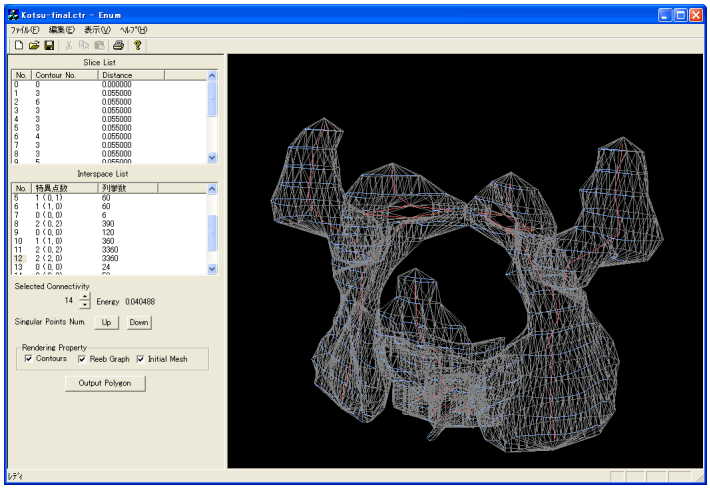


Figure 5.18: A screen shot of our interactive topology selector

neighbors. When the contours on adjacent slices vary widely in shape, we use the toroidal graph[40] to calculate the correspondence.

5.4.2 Experimental Results

We have implemented an interactive topology selector as shown in Figure 5.18. When the contour data is input (Figure 5.19 (a)), possible Reeb graphs are automatically enumerated. This software also has an ability to sort the enumerated results using an objective function. This function is designed to minimize the number of singular points[48] and the distance between connected contours. If the Reeb graph at the top of the list is not satisfactory, the user can select another Reeb graph (Figure 5.19 (b)). After the topology is set, an initial mesh is constructed (Figure 5.19 (c)). Finally the mesh is smoothed and output (Figure 5.19 (d)).

We used the Loop subdivision scheme [89] to smooth the mesh. The final shape does not exactly "interpolate" the original contour data because the Loop subdivision scheme is an "approximating" subdivision scheme in comparison to "interpolating" subdivision schemes such as the Butterfly or the Kobblet schemes. We do not consider this problem crucial in this paper, but if the final surface must strictly passes through the input contours, interpolating subdivision schemes or, alternatively, other smoothing methods such as the global thin plate energy minimizing method [151] or the non-shrinking Gaussian smoothing method [139] should be used.

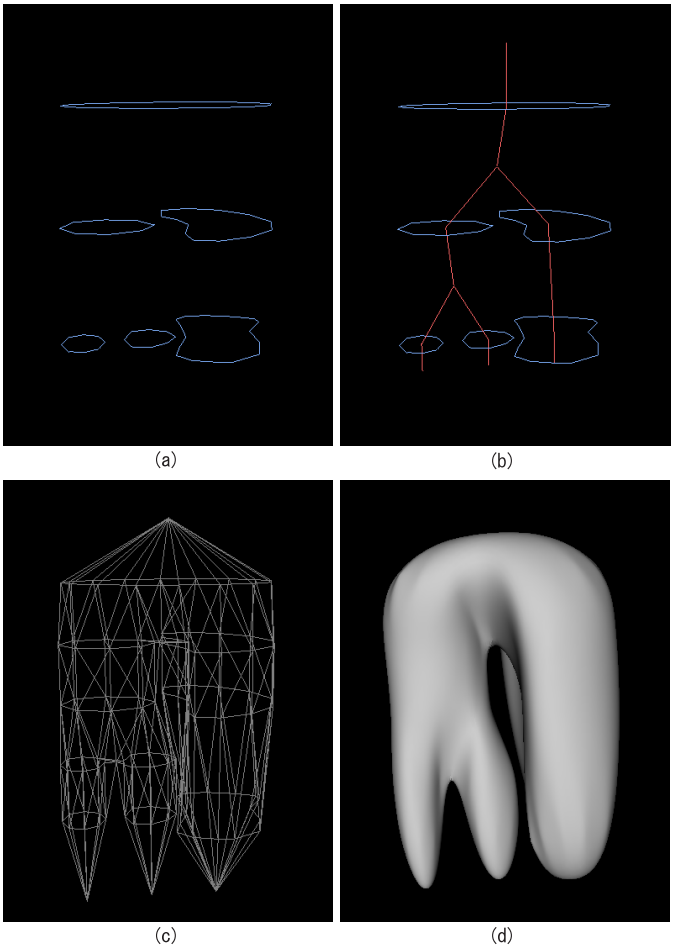


Figure 5.19: An example of input contours (a), the Reeb graph which has the smallest objective function value (b), the constructed initial mesh (c), and the smoothed surface (d)

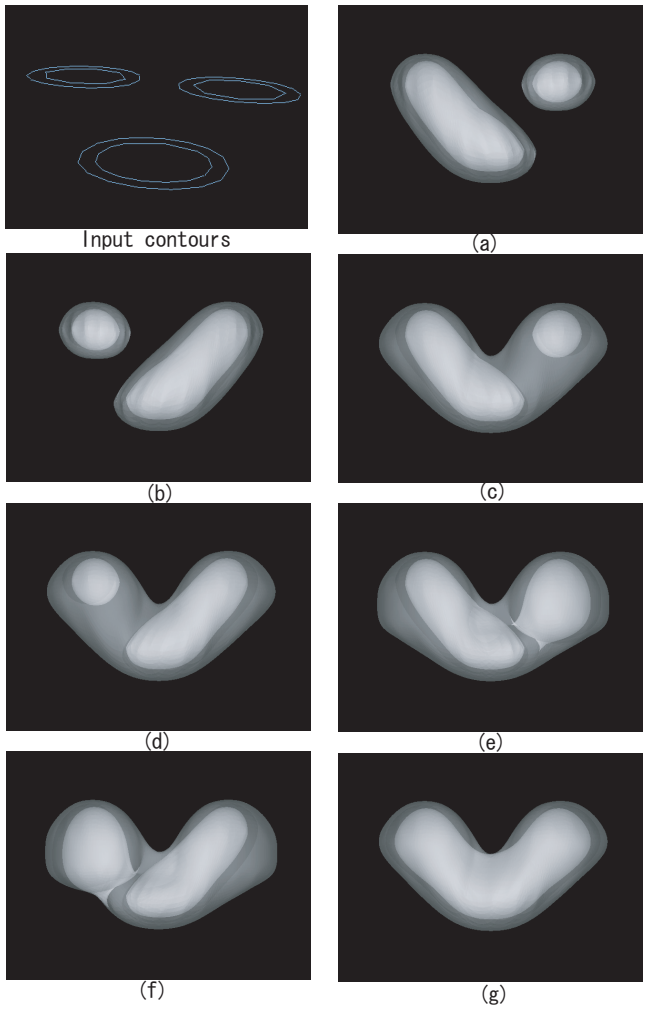


Figure 5.20: Seven enumerated results from six contours on four slices rendered by translucent polygons

All the enumerated patterns when contour images with four slices are input are shown in Figure 5.20. In this example, the topmost image and the bottommost image contain no contour while the second image contains four contours and the third image contains two contours. In this case, only seven patterns are possible (Figure 5.20 (a)-(g)). The final mesh is rendered as translucent polygons. (e) and (f) may need more explanation. The difference between (c) and (e) is that in (e), the top-right small contour is connected to its exterior contour while in (c), it is deleted. The difference between (d) and (f) is the same.

Figure 5.21 shows the results of our method. The four columns show the input contours, the Reeb graph, the reconstructed initial mesh and the final smooth mesh, respectively. In the pelvis and the bronchus data, the contours are extracted manually by Bézier curves and then converted to point lists. Table 5.4.2 and 5.4.2 show the enumeration results. The leftmost column shows the indices of the interspace. The second column is the number of contours in the slices at the upper and the lower regions of the interspace. The third column is the number of correct singular points. The fourth column is the number of the enumerated Reeb graphs. The last column is the manually chosen correct indices of the Reeb graph. If this number is 0, the top candidate of the sorted result was correct and no manual selection was required. In most cases, this is just the smallest possible number under the given input contours which can automatically be calculated. However in other cases, this is manually specified.

5.5 Discussions

We have proposed an algorithm to enumerate every possible correspondence of contours when interpolating cross-sectional images. This allows us to explicitly handle topological ambiguity and avoids falling into local minima by finding the answer which best matches the user's knowledge about the object.

However, the current implementation has some room for further improvements in order to achieve a fully automatic reconstruction. To begin with, we need to find an appropriate method to model the user's knowledge about an object. Our final goal is to develop a system which outputs a correct result without any interaction by the user. Although the current implementation automatically optimizes the number of singular points and average distances between contours, it is not, by itself, sufficient to achieve our goal. One possible improvement of the algorithm

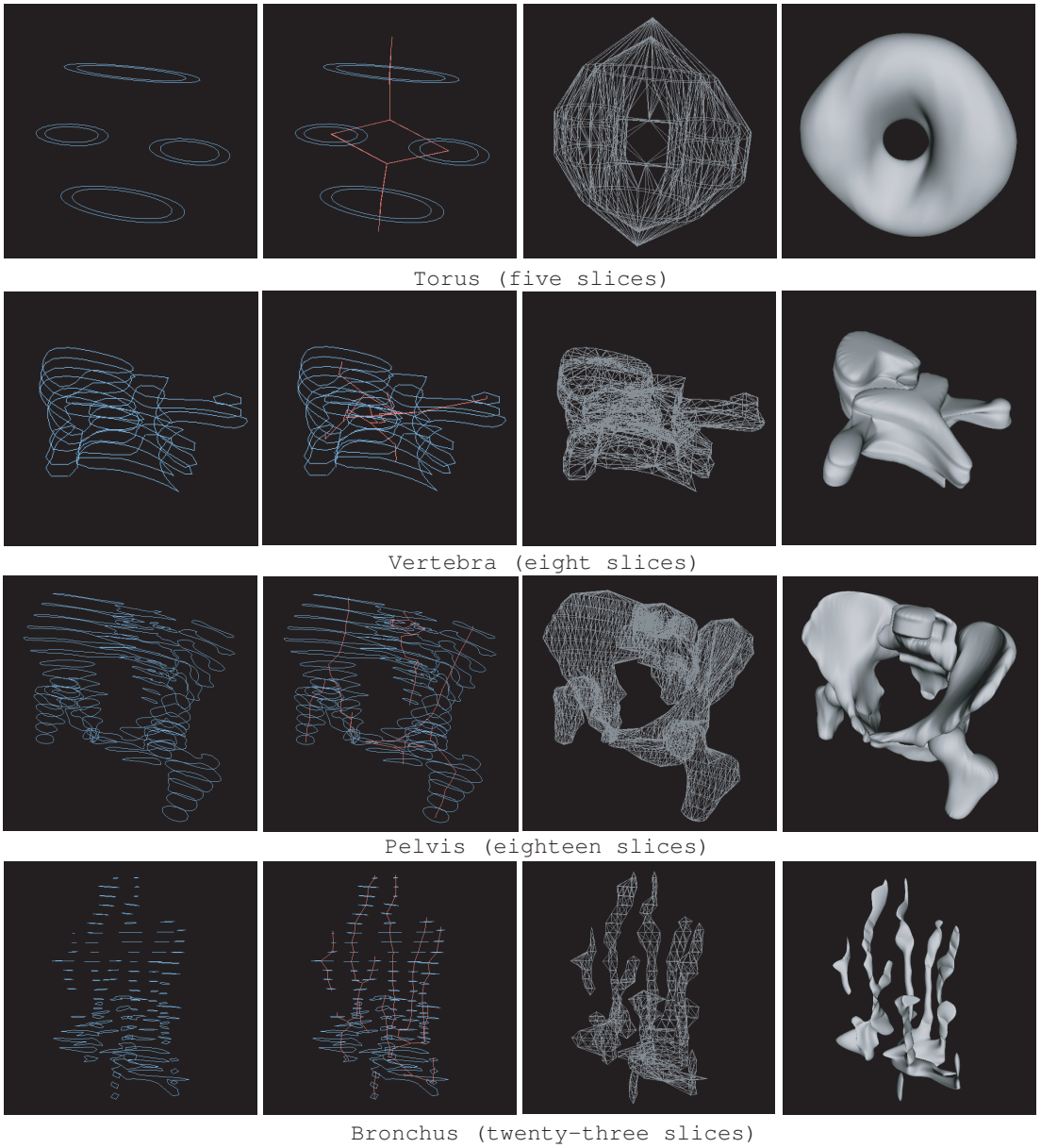


Figure 5.21: Results of the application of our algorithm to various data

Table 5.1: The result of enumeration for pelvis data

Id.	#Contours	#Singularities	#Candidates	Correct Id.
0	0-3	3	1	0
1	3-6	3	2100	35
2	6-3	3	2100	63
3	3-3	0	6	0
4	3-3	0	6	0
5	3-4	1	60	2
6	4-3	1	60	1
7	3-3	0	6	0
8	3-5	2	390	5
9	5-5	0	120	0
10	5-4	1	360	0
11	4-6	2	3360	36
12	6-4	2	3360	14
13	4-4	0	24	0
14	4-2	2	50	0
15	2-2	0	2	0
16	2-0	2	1	0

Table 5.2: The result of numeration for bronchus data

Id.	#Contours	#Singularities	#Candidates	Correct Id.
0	0-3	3	1	0
1	3-4	1	60	11
2	4-5	1	360	3
3	5-5	2	5400	8
4	5-7	4	378000	7
5	7-8	1	181440	2
6	8-7	1	181440	0
7	7-7	0	5040	0
8	7-6	3	670320	0
9	6-6	2	52920	1
10	6-6	0	720	0
11	6-6	0	720	0
12	6-5	1	2520	0
13	5-5	0	120	0
14	5-5	0	120	0
15	5-3	2	390	0
16	3-2	1	12	0
17	2-2	0	2	0
18	2-2	0	2	0
19	2-2	0	2	0
20	2-2	0	2	0
21	2-0	2	1	0

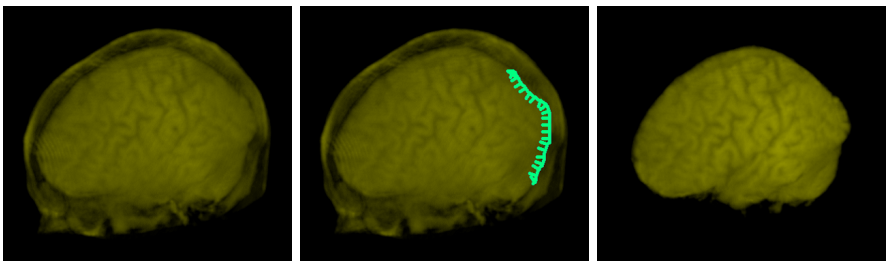
is the consideration of more global knowledge such as the entire integral of the surface curvature or the number of bifurcations or holes. A more straightforward approach would be to input a complete 3D model into the system as the knowledge about an object, which is the so-called model fitting. In this framework, our enumeration strategy is also useful to avoid finding local minima.

In our algorithm, every contour is used during calculations. To handle larger size data , the grouping of contours will be essential because of its computational complexity. If the average number of contours in a slice is N, the upper bound on the number of enumerated cases in an interspace is $O(N^{2N})$.

Finally, how to handle contours on planes which are not parallel is not discussed in this paper. However, our method can easily be applied, provided that each cross section has only two adjacent cross-sectional slices on each side.

Chapter 6

Volume catcher: a simple user interface for volume segmentation



(a) (b) (c)

Figure 6.1: (a) $105 \times 73 \times 73$ head MRI data. (b) Drawing a 2D stroke along the contour of the brain. (c) Resulting 3D region. The system automatically computes the depth of the stroke and applies constrained segmentation.

We proposed an extension of existing contour-based segmentation algorithms in the previous chapter. However, this work requires the result of 2D segmentation on some of the cross-sectional images as a set of contours. This chapter proposes a simple and intuitive user interface for volume segmentation: instead of working on the cross-sectional images, the user traces the contour of the target region using a 2D free form stroke directly on the screen, and the system instantly returns a plausible 3D region inside the stroke by applying a segmentation algorithm. The main contribution is that the system automatically infers the depth information of the ROI by analyzing the data, while existing systems require the user to explicitly provide the depth information. Our system first computes the 3D location of the user-specified 2D stroke based on

the assumption that the user traced the silhouette of the ROI, that is, the curve where the gradient is perpendicular to the view direction. The system then places constraint points around the 3D stroke to guide the following segmentation. Foreground constraints are placed inside of the stroke and background constraints are placed outside of the stroke. We currently use Nock et al.'s statistical region merging algorithm [103] for the segmentation. We tested our system with real-world examples to verify the effectiveness or our approach.

6.1 Background

Volume segmentation is the process of splitting volume data into several perceptual or semantic units. It is a fundamental procedure that is required to obtain useful information from the volume region such as its shape, topology, and various measurements (cubic volume, number of components, etc.). The importance of volumetric segmentation has been widely recognized for about 30 years and many sophisticated segmentation algorithms have been proposed.

However, no fully automated method is yet available (Section 2.5). The reason is that segmentation is dependent on the observer's subjective interpretation, which is impossible to obtain without user intervention. Most segmentation methods have focused on low-level features such as edge detection and texture analysis, and have achieved some degree of success. The difficult part is high-level recognition that is related to the semantics of data. For example, suppose we have a scene that contains a bunch of grape on a dish. "A bunch of grape" or "a dish" are both semantic elements, which may consist of more than one low level feature. The user may want to carve out only one grape, or the entire bunch, or even the entire bunch along with the dish. These options are all probable depending on the intent of the user. Therefore it is crucial for the user to give appropriate guiding information to get the desired segmentation result.

From the user's point of view, one problem with 3D volumetric segmentation is that 3D guiding information is hard to specify since the typical input device is a mouse, which provides only 2D information. Recent work tries to deal with this problem by allowing the user to draw strokes on the cross-section of volume data that roughly indicate foreground and background regions [144]. This stroke information is used to train a classifier that is designed for segmenting voxels. One

drawback of this system is that it requires a lot of user interaction: the user has to specify the cutting plane and provide several strokes to train the classifier.

We propose a simple interface for specifying a region of interest. The user simply delineates the ROI on the rendered image using a 2D free form stroke. The system automatically computes the missing depth information and returns a volumetric region inside the stroke by performing segmentation inside of the stroke. The user no longer needs to manually specify foreground and background constraints in 3D space. This chapter describes the user interface and the implementation details of our prototype system. We also show some segmentation examples on the real-world dataset.

6.2 User interface

We first describe the system from the user's point of view. After the user loads a volumetric model, the system renders it using a traditional volume rendering method. Here the user can apply any rendering techniques to enhance the appearance of the model. Our system currently allows the user to modify an opacity transfer function, a color transfer function, and a gradient enhancement function [88]. The user can interactively change the view direction, scale, and these transfer functions, to locate a target ROI in the volume.

To select a ROI, the user simply traces at least a part of the ROI's contour using a 2D freeform stroke on the screen (Figure 6.1b). We currently assign dragging of the left mouse button down to drawing freeform strokes. The current implementation requires that the user traces the contour in a clockwise direction. In other words, the right hand side of the stroke's drawing direction is recognized as the target region. The system shows hatching along the stroke to indicate this constraints. The system automatically computes the depth of the stroke so that the stroke is on a region boundary in the 3D space and applies segmentation based on the fact that right hand side of the 3D stroke is inside of the region. Finally, the system returns a volumetric region as either voxels in the volume (Figure 6.1c) or boundary surface computed by the Marching Cubes algorithm [90] (Figure 6.7). The algorithm is based on an assumption that the user's stroke closely traces the contour of the ROI. Concequently, it may fail if the stroke is far away from the contour or if the contour is too fuzzy.

Our current system supports two types of strokes: open and closed strokes. If the distance between both ends is close (20 pixels in our current implementation), the system automatically connects the end points and processes it as a closed stroke. Otherwise, the stroke is treated as open. Closed strokes are useful when the entire boundary is visible and open strokes are useful when part of boundary is hidden from the user.

6.3 Algorithm

6.3.1 From 2D freeform stroke to 3D path

Our algorithm first converts the 2D stroke on the screen into a 3D path by adding depth information. We assume that the ROI is visually distinct and the user follows its contour. If this assumption is valid, the 3D path should be close to the silhouette of the ROI. In other words, the gradient vector of the volume data near the 3D stroke should be almost perpendicular to the view direction. We find such 3D path as follows.

1. Sweep the 2D freeform stroke along the depth direction to create a 3D curved surface (Figure 6.2a). The sweep extent is set to cover the entire volume. This curved surface is called the *sweep surface*.

2. Parameterize the sweep surface. The system assigns the x coordinate axis to the depth direction and the y coordinate axis to the direction parallel to the stroke on the screen (Figure 6.2b). That is, x is 0 along the curved edge of the sweep surface near the camera and y is 0 along the straight edge corresponding to the starting point of the stroke.

3. Set sampling points on the sweep surface. Sampling points are lattice points in the parameter space. We set the interval of the lattice as 0.3% of the diameter of the volume data's bounding sphere[1]. In the following context, each lattice point is denoted by L_{ij} (i, j are indices along the x and y directions in the parameter space where $1 \leq i \leq X_{max}, 1 \leq j \leq Y_{max}$.).

4. On each lattice point L_{ij}, compute $S_{ij} =\mid N_{ij} \cdot G_{ij} \mid$ where N_{ij} is a unit normal vector while G_{ij} is a unit volume gradient (Figure

[1]Strictly speaking, this parameterization is not uniform if we use perspective transformation. The space is slightly stretched in the y direction where $x = 1$.

6.2c). S_{ij} is called a *silhouette coefficient*, which indicates how much the point looks like a silhouette from the current viewpoint.

The volume gradient G_{ij} is computed as follows. If the original data contains color channels, the system first converts the color value to a grayscale value that represents the perceptual brightness of the color using the standard equation $Gray = Alpha \times (0.299 \times Red + 0.587 \times Green + 0.114 \times Blue)$ [123]. If the volume renderer filters the original data (eg. by the opacity and color transfer functions), the post-filtered color should be used because we assume the user wants to select a visually distinct structure in the current rendered image. Furthermore, to handle the user's imprecise input stroke and to suppress the effect of noise, the system computes G_{ij} from a blurred version of the original data. Any kind of blur filter can be used. We chose a discrete approximation of The Gaussian filter. The radius R of the filter in the data's coordinate system is matched to the extent of stroke's error margin in the screen coordinate system[2]. The stroke's error margin is set as 5 pixels in our implementation. To blur and differentiate the data at the same time, we convolve the data with the derivative of a blur filter. We construct our blur/differential filter kernel by first computing a blur filter and taking the forward difference along the x direction. The result can be immediately used for the y, z directions by just rotating by 90 degrees.

5. The problem of computing the depth of the user-drawn 2D stroke is now the problem of finding a path in the parameter space that starts from a point on the edge $y = 1$ and ends at a point on the edge $y = Y_{max}$ where the sum of the silhouette coefficients along the path is maximized (Figure 6.2d). Since this path has only one x value for each y, it is represented as a function $x = f(y)$. To maintain the continuity of the path, we impose the condition that $\mid f(y_i) - f(y_{i+1}) \mid \leq c$ where c is an integer constant value that controls the continuity of the path. We currently use $c = 1$. We introduce an additional condition of $\mid f(y_1) - f(y_{max}) \mid \leq c$ for a closed stroke. We use dynamic programming to solve this problem [21]. The optimal path on the parameter space is then converted to a 3D path by connecting the corresponding 3D lattice points

[2]Strictly speaking, R should be dependent on the depth in perspective projection. However, we use a constant value computed at the center of the voxels.

(Figure 6.2e). The matrix S_{ij} and the optimal path for Figure 6.1 is shown in Figure 6.3.

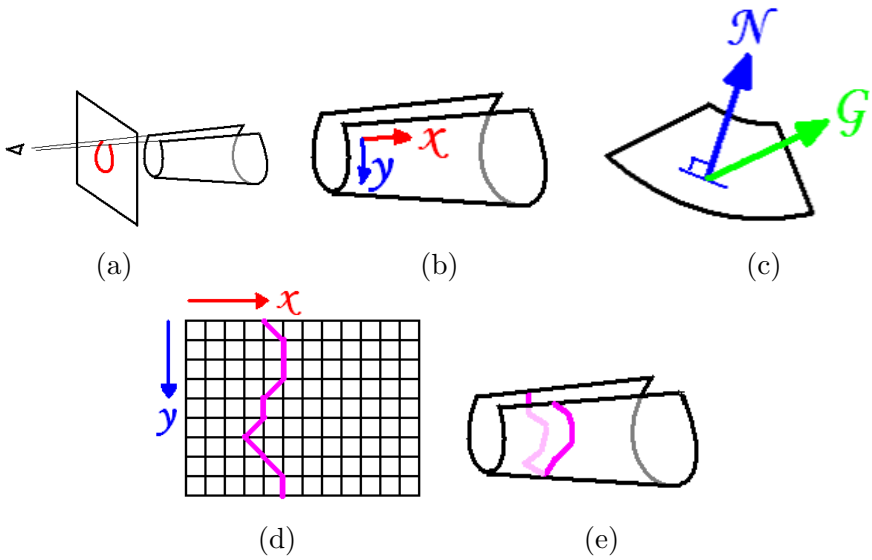


Figure 6.2: From a 2D stroke to a 3D path. (a) Construction of the sweep surface. (b) Parameterization of the sweep surface. (c) Computation of a silhouette coefficient. (d) Finding the optimal path. (e) Resulting 3D path.

6.3.2 Generating constraints and segmentation

The next task is to generate constraints to guide the segmentation. We currently use Nock et al.'s region merging algorithm [103] for segmentation. It takes a set of constraint points that specify foreground and background regions as input and separates the volume into a foreground region and a background region that contain the corresponding constraints. The constraints can be directly used for other segmentation algorithms such as those which use Graphcut technique [87]. We generate the constraints by offsetting the 3D path. The offset direction D is obtained simply by computing a cross-product of the view vector and the tangent vector of the path (Figure 6.4a). The displacement extent e is proportional to the radius of the blur kernel R computed in the previous section, therefore, it is also proportional to the stroke's error margin ($e = 2R$, in our implementation). Each point in the 3D path is offset by $\pm e\frac{D}{|D|}$ and the points on the right hand side become foreground con-

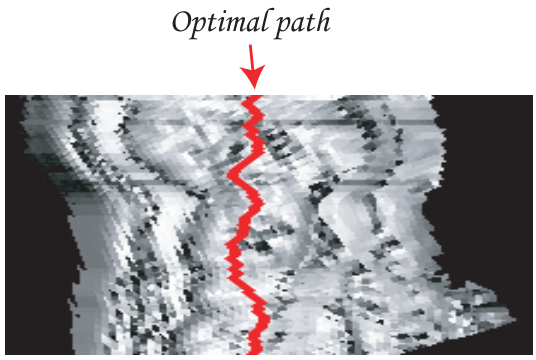


Figure 6.3: Silhouette coefficient matrix and computed optimal path of Figure 6.1.

straints and those on the left hand side become background constraints (Figure 6.4b). Constraints generated for Figure 6.1 are shown in Figure 6.5.

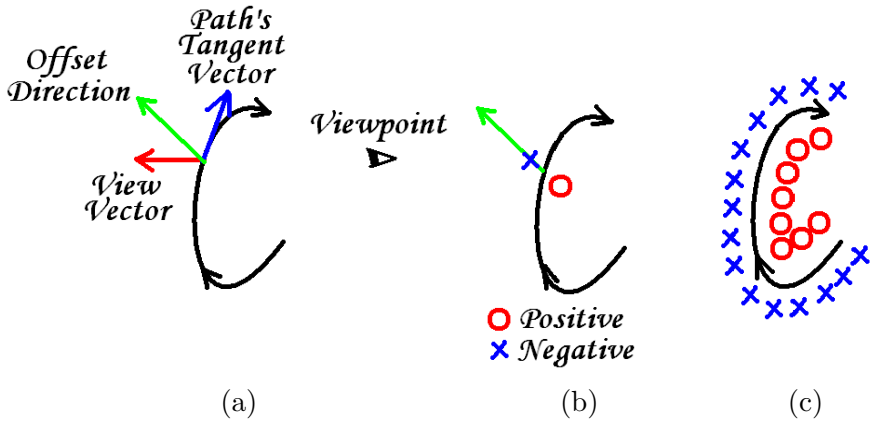


Figure 6.4: Constraint locations

We perform the actual volume segmentation using these constraints. If the user-given stroke is a closed stroke, only those voxels inside of the sweep surface are returned.

6.4 Results

We applied our technique to several data sets. Figure 6.6 shows an example of a high-potential iron protein dataset. Note that our system works even when only a part of the target region is visible (Figure 6.6b). We also applied our system to color data (Figure 6.7). We captured this

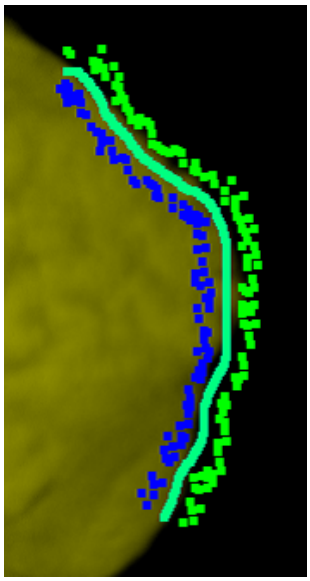


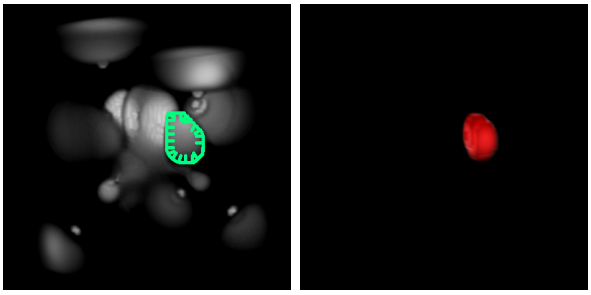
Figure 6.5: Constraints generated for Figure 6.1

full color volumetric data using the slicer system developed by Ogawa et al. [105] (Figure 6.8). This slicer can cut a $3cm \times 5cm \times 2cm$ frozen in OCT compound or paraffin-embedded subject into $10\mu m$ thick slices. The green matter in Figure 6.7 is OCT compound and the slice interval for the chocolate crisp data is $42\mu m$ thick.

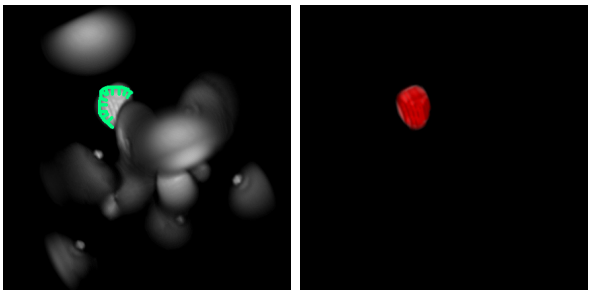
6.5 Discussions

In this chapter, we showed a successful example of using information on the intrinsic structure of target data to reduce user intervention. The objective of our system is to extract the ROI from existing (unsegmented) volume data. The application domains of this technique are countless. For example, Tzeng. et al. showed that segmentation (according to them, classification) is an important information to control browsing parameters such as the transfer functions [138]. McGuffin et al. proposed a set of interesting ideas to interactively explore pre-segmented volumetric data [95].

Although our system works for many real-world examples, there are several cases in which our system does not work properly. For example, if the ROI is not round near the 3D stroke, generated positive constraints can miss the ROI. Another limitation is that when two objects have almost the same contour location from one viewpoint, the system can



(a)



(b)

Figure 6.6: An application to a 66^3 high-potential iron protein data. The segmented region is rendered as opaque, red voxels.

only carve out one of them. In addition, our system does not have a mechanism to modify failed result. For future work, we want to add a user interface for fixing segmentation errors. We are also seeking a way to apply this technique to surface graphics by using implicit representation of an object.

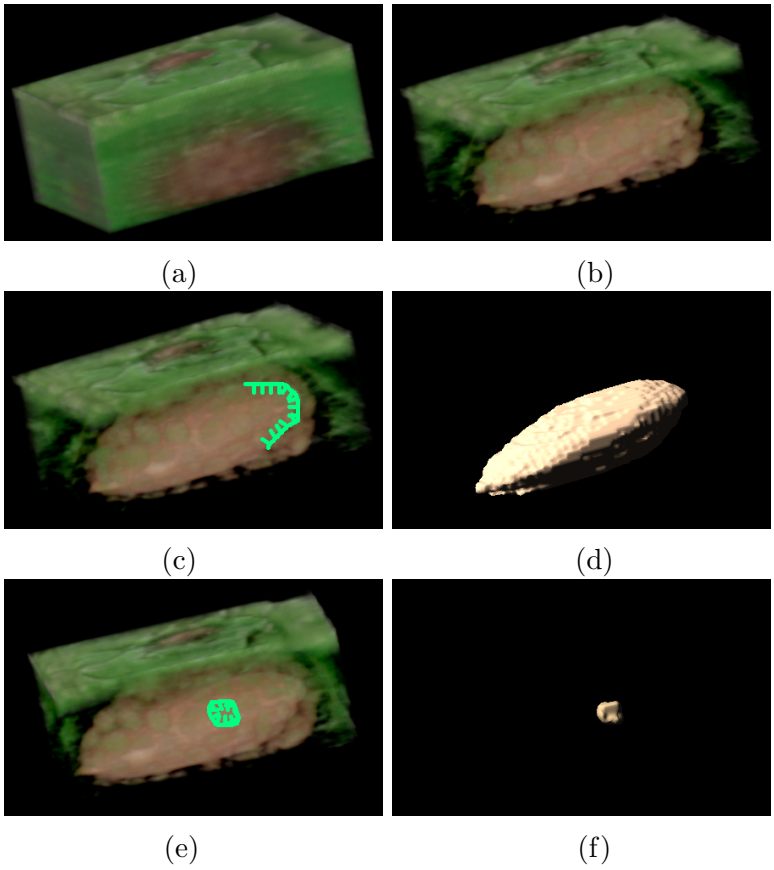


Figure 6.7: Application to a chocolate crisp dataset($126 \times 89 \times 53$, color data). The opacity transfer function is applied to the original data (a) to make the almond distinguishable (b). When the user draws a stroke (c), the almond region is segmented and rendered as a surface (d). The surface color is set to the mean color of the selected region. The user can also pick small masses (e,f).

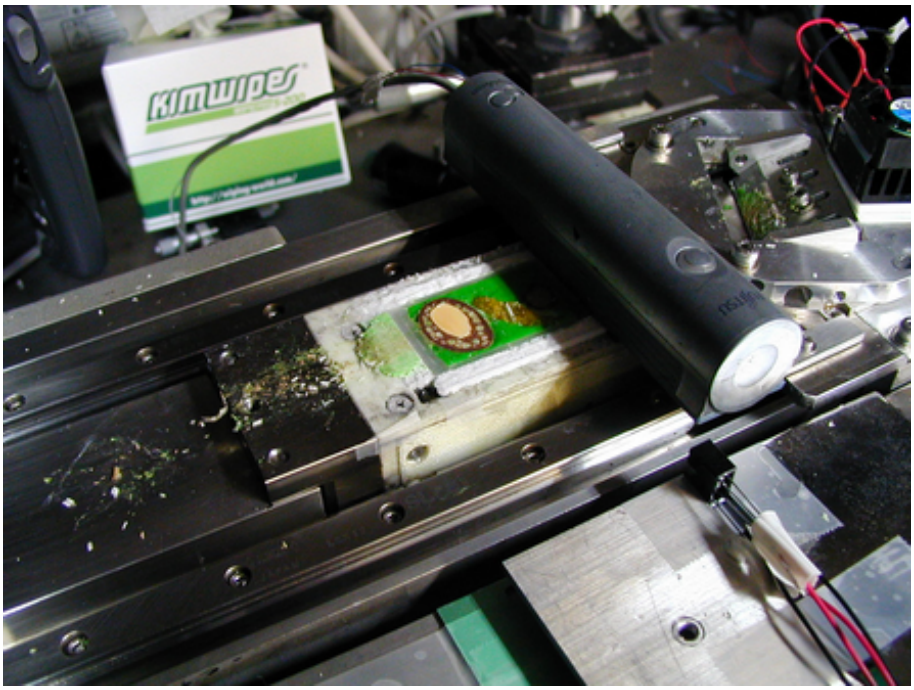


Figure 6.8: A cutting device

Chapter 7

Interacting with volumes

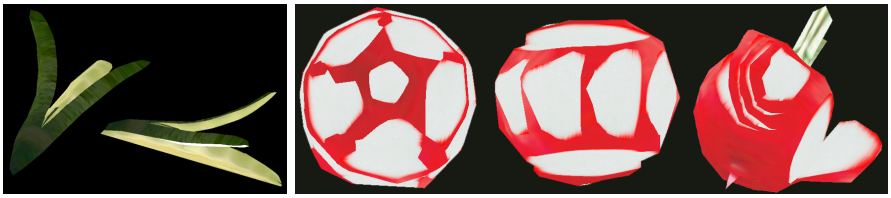


Figure 7.1: Food model created by our system.

This chapter presents an interactive system that enables the user to cut 3D food models using free form strokes and a knife controlled by standard mouse operations. We try to show how volume graphics have potential to produce new kinds of interactive contents and convince the readers that volume graphics may open up the new age of computer graphics.

Cutting is one of the most primary interactions with volume data. Actually, as we have shown in Chapter 4, well-designed cross-sectional images convey an idea of volumetric structures, although the cross-section is 2D. Therefore, it is important to explore various cutting operations. Most previous systems, however, just provide planar cuts and no intelligent interface support was provided. Here we pay notice to real-world cutting interaction of foods. There are a great number of non-planar and artistic cut patterns in cooking domain. We designed a new cut interface by which the user can easily create artistic food models using simple mouse operations. Our system allows the user to cut along a surface of an object and also a part of a model is deformed, if desired. We show various models created by only cutting an original food model. Currently this is just a cooking interaction. However, we believe that

we can further apply these cutting interaction ideas to general volume datasets.

7.1 Background

The progress of 3D CG technology produced a great number of unique and interesting contents. Accordingly, the noble user interfaces for such contents are also proposed. However, the standard input device equipped on a standard PC is a mouse, which essentially inputs only 2D information. Therefore, in general, it is difficult to interact with a 3D space.

Interacting with volume data is especially problematic since the dimension of such data is higher than that of surface models. There are mainly two way to browse volume data. One is volume rendering and the other is cutting the target and observing the cross-section. Volume rendering is a well-studied topic and there are many sophisticated algorithms for effective control of browsing parameters [114]. We are more interested in cutting volume data because there are surprisingly few works tried to explore this interaction, regardless of its importance. The existing systems are not sufficient for practical use since some requires predefinition of the cut geometry [150], while another requires a special device [61].

We are currently working on a new CG educational content that mainly target at foods and cooking. The objective of this project is to let the user to cook in a computer and enjoy observing the dynamically changing appearance of foodstuff. We chose this application domain since it requires variety of cutting patterns. Food cutting has a long history and there are a great number of artistic and surprisingly complicated cutting patterns using only one or two knives (Figure 7.2). We believe exploring this domain will eventually produce fairly intuitive user interface since this is one of the most daily activities.

We introduce several interface ideas and show that various cutting operations are possible by manipulating a standard mouse. The user can intuitively cut the target model by moving a knife in the system. The key ideas are: (1) cutting along the surface of the model, (2) controlling the extent of knife motion, and (3) deformation of the model. We obtained these ideas from observing the usage of a knife for real-world cooking [4].

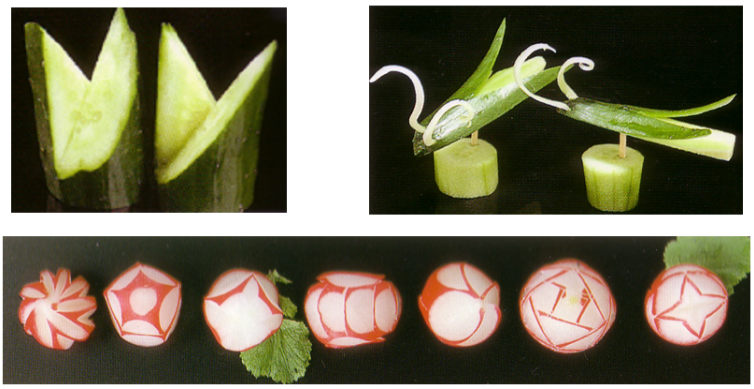


Figure 7.2: An example of complicated cutting pattern

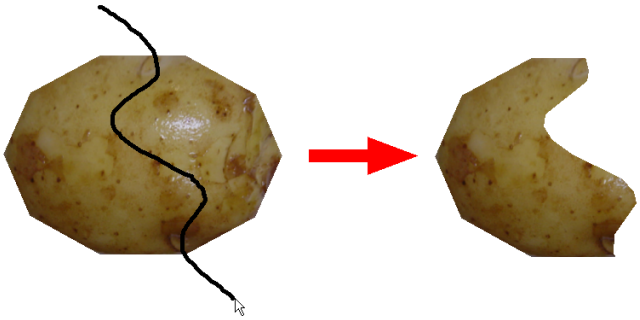


Figure 7.3: Cutting by a freeform stroke

7.2 User interface

First of all, the user can cut the model by a freeform stroke [68](Figure 7.3).

Our new cutting interaction supports variety of patterns by manipulating a knife. Since our system has high degree freedom, we split the whole process into several steps. We explain each in turn. For ease of explanation, we define a local coordinate system for a knife. We set the x axis as the orientation of the handgrip and the y axis as the orientation of the blade (direction of the knife motion) (Figure 7.4). "Clicking a button" means pushing down the button and release the button instantly. "Pressing a button" means pushing down the button but keep pushing. Stopping "pressing a button" is "releasing a button". "Dragging" is the motion of a mouse while pressing a button.

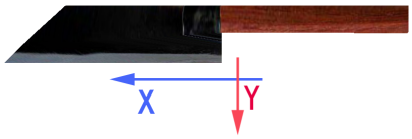


Figure 7.4: Local coodinate system of a knife

Figure 7.5: Putting a knife (1)

Putting a knife

The user first puts a knife on the target model. There are two ways to do this. One is to click the left button of the mouse on the model. The other is to press the left button on the model, wait a while, drag, and then release.

The first method results in the knife to be put as Figure 7.5. The x axis of the knife is upward and the y axis goes the viewer's view direction. An animation is added to supply a feedback that the object is a knife.

The orientation of the knife can be modified by the second method through the dragging operation after waiting. The x-y plane of the knife is visualized as a translucent plane to supply the feedback of the

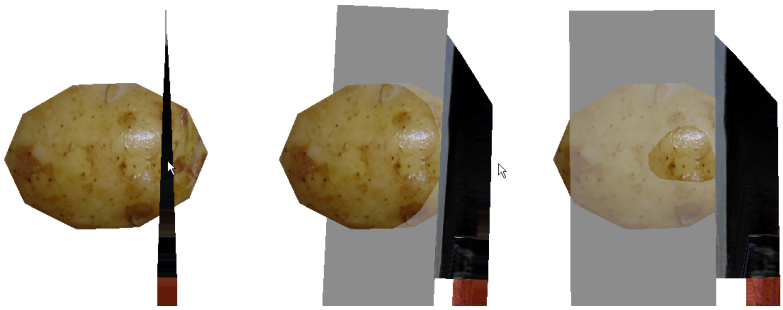


Figure 7.6: Putting a knife (2)

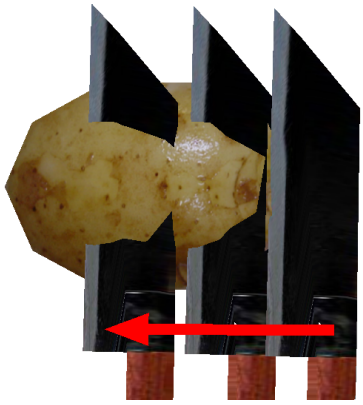


Figure 7.7: Dragging a knife

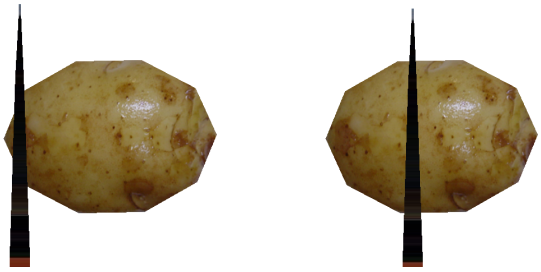


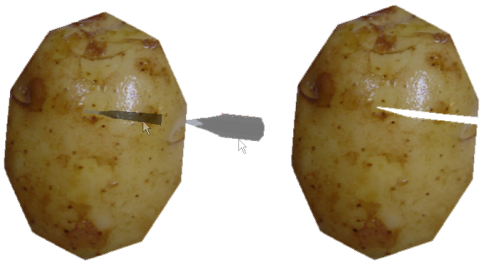
Figure 7.8: Skin peel mode (left) and normal mode (right)

orientation of the knife (Figure 7.6). When the user releases the button, the orientation of the knife is fixed.

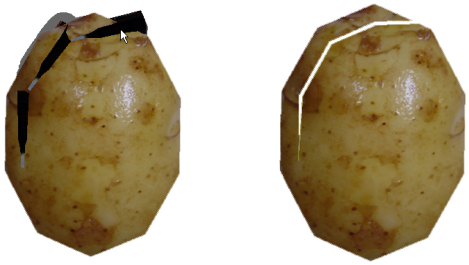
Moving a knife

Next, the user moves the knife by left-dragging it (Figure 7.7). If the knife is put by left click in the previous step, the model and the knife should be rotated beforehand by right-dragging since the y axis of the knife is parallel to the view direction.

Internally, there are two modes for the knife motion. If the initial y direction of the knife goes close to the surface of the model, the mode becomes a special mode call "skin peel mode". Otherwise, the mode is normal (Figure 7.8). The blade goes straight in the normal mode while it follows the surface of the model (Figure 7.9). However, the orientation of the x axis does not change.



Normal mode



Skin peel mode

Figure 7.9: Knife motion of each mode (top view)

Deformation (Optional)

If desired, the cut model can be deformed. If the mouse motion is stopped while dragging (Figure 7.10(a,b)), the model is snicked along the knife's motion path and the system goes into "deformation" mode (Figure 7.10(c)).

If the user moves the mouse again, a part of the model is deformed (Figure7.10(d)). If the mouse is dragged along the positive y direction of the knife, the closer part of the snicked model is deformed. Otherwise the farther part is deformed. Figure 7.10(e) is a different view of Figure 7.10(d).

Completion of cut

If the left mouse button is released, the cutting sequence is completed. If the model is split into two parts, one of them is thrown away (Figure 7.11). In skin peel mode and no deformation is performed, thinner part is automatically thrown away. Cutting by free form stroke results in the left side of the stroke to be thrown away [68].

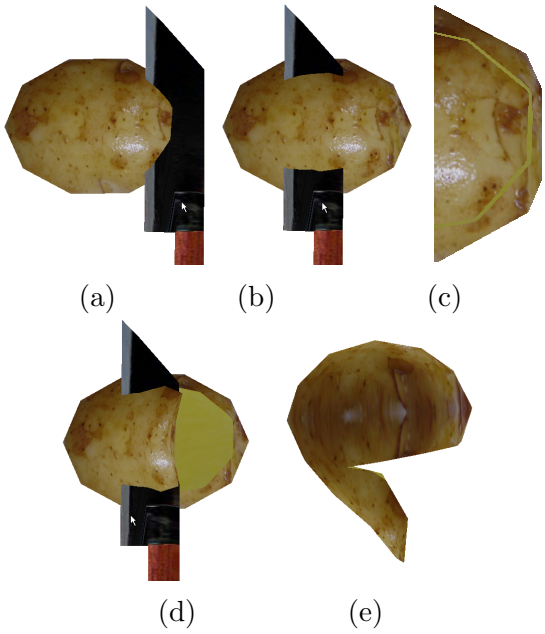


Figure 7.10: Deformation

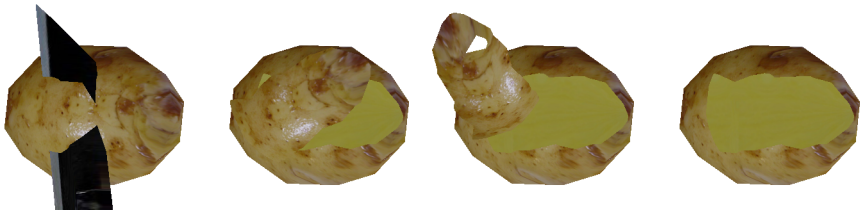


Figure 7.11: Removing a part

7.2.1 Implementation

We implemented our system using C++ and OpenGL. Rendering is performed by a triangular mesh and CSG algorithm is used for cutting the mesh model [64]. There are several ways to generate volumetric cross-section. We used several models for this purpose, including pseudo volumetric data [108](Chapter 4) and segmented scan data (Chapter 5 and 6).

Deformation algorithm

The deformation algorithm is as follows:

- When the user stops the mouse motion, vertices on the newly created cross-section is rotated around the blade of the knife at

that moment. The rotation angle is proportional to the distance from the axis (Figure 7.10(e)).

- Move other vertices. For each of other vertices, find the closest vertex on the cross-sectional plane and apply the same rotation of the found vertex.

Computing the knife motion path in skin peel mode

Computing the knife motion path in skin peel mode is not very straightforward. Basically the knife should follow the surface of the object. But if we have a closer look, we observe that the blade first runs from the surface to some depth, then moves parallel to the surface, and finally goes up to the surface. To generate such motion, we first generate a binary image using the following algorithm.

1. When the knife it put(Figure 7.12(a)), render the scene into off-screen buffer from the viewpoint where the x axis of the knife goes parallel to the view vector and the y axis goes right hand (Figure 7.12(b). The knife is not rendered in practical).

2. Divide the image into foreground and background (Figure 7.12(c)).

3. Choose one of the upper bound or lower bound of the foreground, which is closer to the knife and let the height difference r (Figure 7.12(d)). If r is larger than user-defined threshold value, the cut mode is normal. In this case, the knife just goes straight. Otherwise, go to the next step.

4. Erode[1] the foreground of the binary image by a radius r circle (Figure 7.12(e,f)). The foreground shape shrinks by r.

This binary image is computed when the knife is put. When the user drags the knife, the knife moves straight until the blade touches the edge of the foreground shape and then follows the outline of the foreground (Figure 7.12(g)).

If the user release the mouse button, the system has to generate the remaining path. Since the knife follows the contour of the eroded surface, the blade must be inside of the object. We must make sure that the knife robustly split the object into two parts and the stroke

[1]Morphology operator.

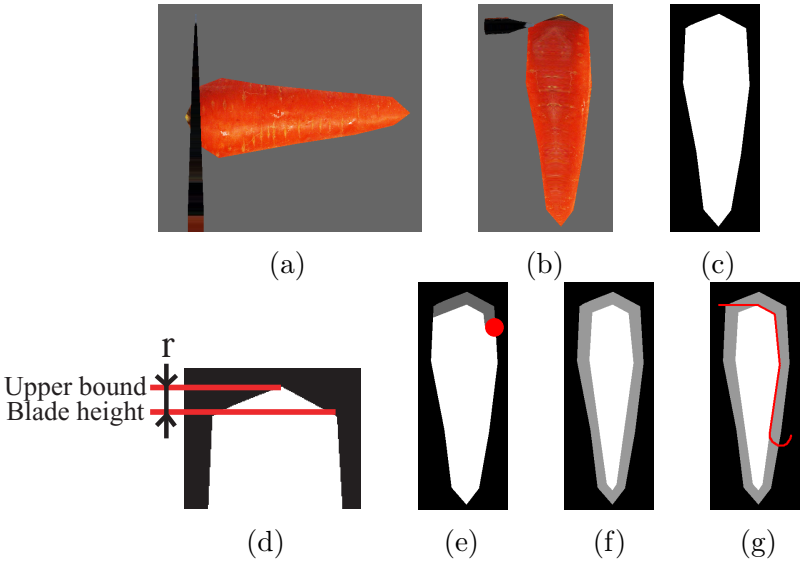


Figure 7.12: Knife path computation

is smooth. The system first attempts to move the knife straight ahead and see if the blade goes outside of the model. We search until the knife moves $4r$. If the blade goes outside, this motion is adopted. Otherwise the system adds a circular arc whose radius is r, at the end of the knife path. The above-mentioned procedure generates smooth and splitting path (Figure 7.12(g)).

7.2.2 Results

Using our system, we could easily create models shown in Figure 7.13. As a reference, we also list pictures of real-world foodstuffs. The snapshot of our contents are shown in Figure 7.14. Figure 7.14 left shows an educational content that works with Macromedia Flash, aiming at elementary school students. Figure 7.14 left is a so-called "cutting game". The user has to create the target shape shown in the bottom left window within some amount of time show in in the upper left window. Such games helps quickly study our user interface.

7.3 Discussions

We proposed an user interface to freely cut a model using a standard mouse. The proposed operations such as cutting along the surface, controlling the cut extent, and deformation extended the variety of possible

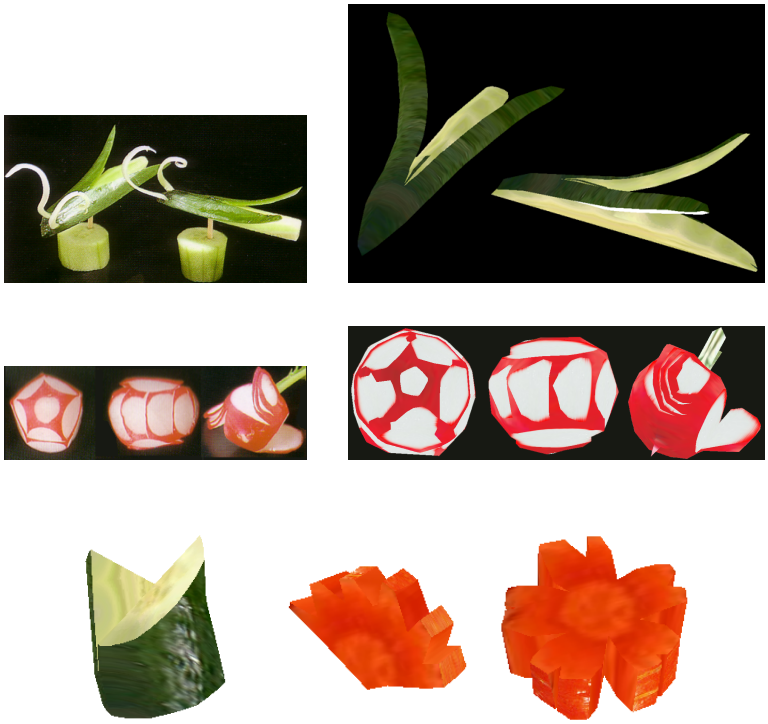


Figure 7.13: Results. The upper two rows are comparison to real-world foodstuffs. The left is the real models)

cut patterns. We are interested in further extension of the proposed user interface and application to more general models such as CT-scanned data or simulated data.

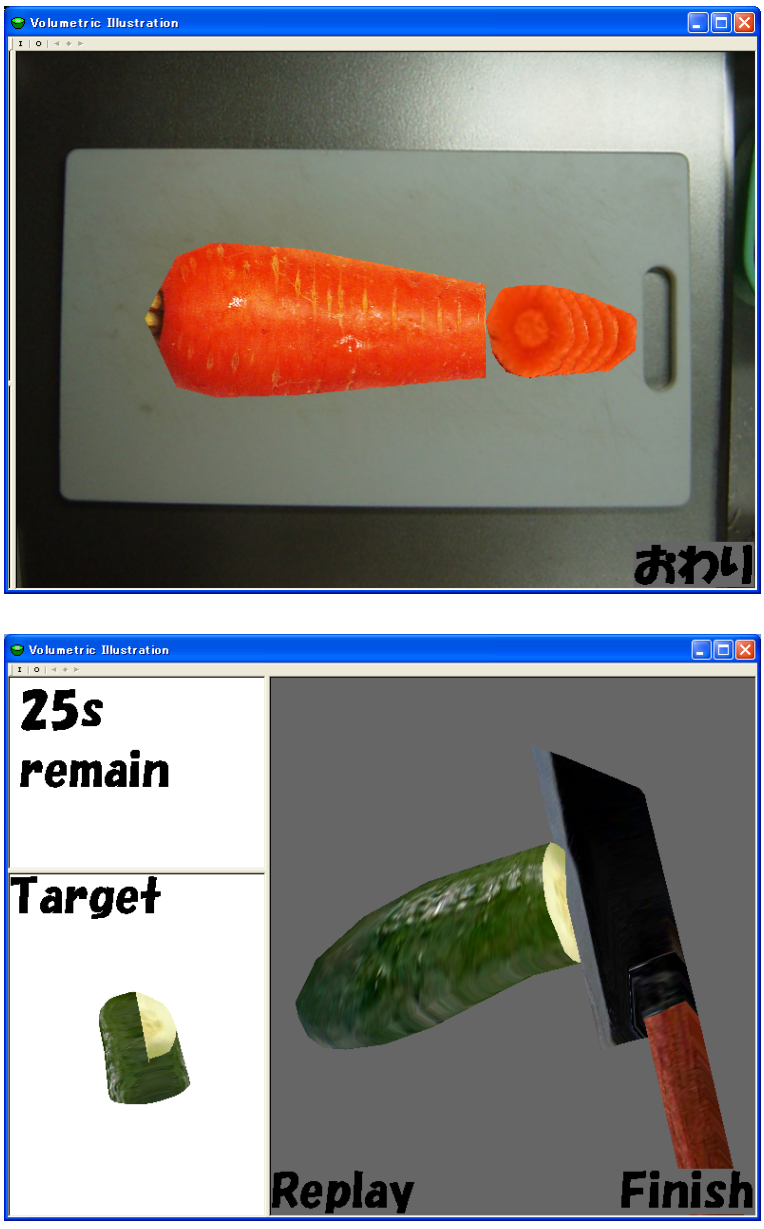


Figure 7.14: Application examples

Chapter 8

Conclusion

In this dissertation, we proposed several tools that make volume graphics tractable for end users. We mainly concentrate on modeling volumetric data, since we believe modeling user interface is the main obstacle for popularization of volume graphics.

First, we proposed a sketch-based shape modeler that naturally handles topological change through intuitive user interface. The easiness of implementation is mainly due to its underlying volumetric representation that automatically maintains solidity of the shape. In this work, we tried to emphasize that volumetric representation has potential to provide simple solution for traditional problem such as self-intersection. One of the main obstacles of volume graphics has been its high computational cost. However, because of the rapid progress of both software and hardware technologies, problems of storage and processing time are not as critical as they used to be. Therefore, we believe that we should be more ready to use volumetric graphics than before for simple and natural processing.

Second, we proposed the volumetric illustration system that focuses on cutting interaction and ignores other aspects of volumetric objects such as rendering with transparency or volumetric deformation. Instead of authoring "real" volumetric data, this work utilizes 2D example images and processes the images when the user cuts the object using the texture synthesis technique. For most 3D graphics programmers and designers, 2D reference images are much easier to obtain than 3D reference volumes. Therefore, this work makes it possible to utilize volumetric contents for those who do not have access to special 3D capturing devices. This system is efficient not only because necessary storage is small but also because the user interface is simplified because of the 2D nature of

input data.

Next, we proposed new systems to segment volume data. Medical scanning devices are the main source of explicit volume data today. The data are provided as a set of cross-sectional images and stored in regularly aligned samples (voxels), which do not have any regional information, thereby further processing is difficult. Voxel segmentation is an image processing algorithm to divide raw volume data into several semantic elements. Since volume data is purely a 3D entity, it is not straightforward to specify information necessary for segmentation. We proposed two methods to support this process. One is based on manually segmenting sparse set of cross-sectional images and then construct 3D regions. We proposed a method to automatically enumerate all possible correspondence of contours and then the user selects the desired pattern from the list. The other proposes a new interface that allows the user to directly work on projected image on the screen. By just tracing the outline of the target region, point constraints are generated and by applying an existing segmentation technique, the target region is carved out. The significance of these works is that we now have a convenient tool to utilize volume data, since segmentation leads numerous applications such as fast rendering or deformation.

Finally we showed one possible application of volumetric computer graphics, which explores a variety of cutting patterns for volumetric models. Although cutting is one of the most important methods to visualize volumetric structure, there have not been much work that specially explore cutting techniques. We tried to explore cooking interaction because in cooking domain, there are a great number of complicated and artistic cutting patterns that are possible with only planar knives. We believe that the proposed idea for cutting is applicable to more generic volumetric data such as CT-scanned data.

8.1 Significance of volume graphics

Throughout the dissertation, we tried to propose easy-to-use volumetric systems and to show potential abilities which volumetric data processing have. Our aim is not to replace current graphics systems by volumetric graphics. Instead, we believe that volumetric graphics can generate more realistic and plausible behavior of objects with minimum burden to both graphics engineers and designers, because all real-world objects have volumetric structure. Computer graphics is essentially an

appearance simulation of the real-world. Ideally, computer generated scenes should be animated by physical simulation or its approximation and rendered with lighting simulation. Therefore, if we pursue supreme reality to computer graphics, we unavoidably need to introduce volumetric data representation. We believe that our works stimulate frequent use of volumetric graphics.

8.2 Future direction

We hope that our systems convince the reader that volumetric graphics have potential to solve traditional problems easily and broaden application domains. However, this work is not a complete solution for all problems of volumetric graphics. In this Section, we discuss several unsolved problems that can make volume graphics still difficult.

8.2.1 Modeling of explicit volume data

Although we propose several tools to author volumetric data, a method to generate explicit volumetric data from scratch is missing in this dissertation. The volumetric illustration system (Chapter 4) inputs 2D images and generates 2D "volumetric" cross-sections. Although this is a benefit from the viewpoint of necessary data amount, this cannot be used for realistic simulation such as deformation or translucent rendering. Dispite the several volume modeling methods as shown in Section 2.2, we have not yet attained a complete solution.

8.2.2 Handy scanning system

Even if we have a successful method to create volumetric model from scratch in future, the importance of scanning devices will never decrease. It is obvious if we view the 2D case: 2D drawing software and capturing devices such as digital cameras or image scanners are compatible because the application is different.

Currently we do not have many channels to obtain volume data. Medical devices such as CT scanners or MR scanners are virtually the only source of the data. These devices are extremely expensive and ordinary graphics designers cannot afford to buy them for their activities. In a research level, there are some devices that scans daily objects such as foods, small animals, and so on [105]. Unfortunately, such devices

still require annoying setup of the machine and fair amount of capturing time.

There is another problem that exist in current scanning devices: we cannot obtain the surface and the internal structure at the same time. The success of surface graphics proves that the surface attribute is indispensable for realistic representation of objects. Current systems can capture only one of those (or can capture both but in separate paths, which requires very difficult registration step afterwards.)

8.2.3 Rendering capability

Another bottleneck of the current volume graphics is that the consumer-level graphics card can render only a small size of volume data. The current high-end graphics card is equipped with only 256MB of video memory, that can store approximately 400^3 3D texture. If we want to render volumetric data that is compatible with contemporary high-end surface model, the resolution should be around 1024^3 or more, which requires more than 4 gigabites. Currently, we need a special purpose hardware to render such a large-scale dataset [113].

However, we are optimistic with this problem since the progress of device technology is so fast. Actually, the ability of GPUs has been progressing much faster than what Moore's law indicates.

References

[1] Avs. advanced visual systems. *http://www.avs.com/*.

[2] Maya. In *Alias systems corp. (http://www.aliaswavefront.com/)*.

[3] Softimage xsi. In *Softimage corp. (http://www.softimage.com/)*.

[4] *Zairyo no shitagosirae hyakka (***材料の下ごしらえ百科***)*. Syufu-to-seikatsusya (主婦と生活社).

[5] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 415–421. ACM Press, 1998.

[6] Alexis Angelidis, Pauline Jepp, and Marie-Paule Cani. Implicit modeling with skeleton curves: Controlled blending in contact situations. In *Shape Modeling International*. ACM, IEEE Computer Society Press, 2002. Banff, Alberta, Cananda.

[7] Michael Ashikhmin. Synthesizing natural textures. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 217–226. ACM Press, 2001.

[8] Chandrajit L. Bajaj, Fausto Bernardini, and Guoliang Xu. Automatic reconstruction of surfaces and scalar fields from 3d scans. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 109–118. ACM Press, 1995.

[9] Chandrajit L. Bajaj, Valerio Pascucci, and Daniel Schikore. The contour spectrum. In *IEEE Visualization*, pages 167–174, 1997.

[10] R.A. Banvard. The visible human project®image data set from inception to completion and beyond. In *CODATA 2002: Frontiers*

of Scientific and Technical Data , Track I-D-2: Medical and Health Data, 2002.

[11] Pravin Bhat, Stephen Ingram, and Greg Turk. Geometric texture synthesis by example. In *Eurographics Symposium on Geometry Processing*, pages 43–46.

[12] James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, 1982.

[13] F. Bloch, W.W. Hansen, and M. Packard. Nuclear induction. *Phys. Rev.*, 69:127, 1946.

[14] Jules Bloomenthal and Ken Shoemake. Convolution surfaces. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 251–256. ACM Press, 1991.

[15] Jean-Daniel Boissonnat. Shape reconstruction from planar cross sections. *Comput. Vision Graph. Image Process.*, 44(1):1–29, 1988.

[16] Mark Carlson, Peter J. Mucha, and Greg Turk. Rigid fluid: animating the interplay between rigid bodies and fluid. *ACM Trans. Graph.*, 23(3):377–384, 2004.

[17] Wei Chen, Liu Ren, Matthias Zwicker, and Hanspeter Pfister. Hardware-accelerated adaptive ewa volume splatting. In *VIS '04: Proceedings of the IEEE Visualization 2004 (VIS'04)*, pages 67–74. IEEE Computer Society, 2004.

[18] H. N. Christiansen and T. W. Sederberg. Conversion of complex contour line definitions into polygonal element mosaics. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 187–192. ACM Press, 1978.

[19] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. *ACM Transtions on Graphics (SIGGRAPH 2003 Proceedings)*, 22(3):287–294, 2003.

[20] Ge Cong and Bahram Parvin. A new regularized approach for contour morphing. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR2000)*, pages 1458–1463, 2000.

[21] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* MIT Press, 2001.

[22] G.R. Cross and A.K. Jain. Markov random field texture models. In *Proc. IEEE Trans. Pattern Anal. Mach. Intell.*, volume 18, pages 25–39. IEEE Computer Society Press, 1983.

[23] Barbara Cutler, Julie Dorsey, Leonard McMillan, Matthias Müller, and Robert Jagnow. A procedural approach to authoring solid models. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 302–311. ACM Press, 2002.

[24] Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 361–368. ACM Press/Addison-Wesley Publishing Co., 1997.

[25] Mathieu Desbrun, Nicolas Tsingos, and Marie-Paule Cani. Adaptive sampling of implicit surfaces for interactive modeling and animation. *Computer Graphics Forum*, 15(5), dec 1996. Published under the name Marie-Paule Gascuel.

[26] Jean-Michel Dischler, Djamchid Ghazanfarpour, and R. Freydier. Anisotropic solid texture synthesis using orthogonal 2d views. *Comput. Graph. Forum*, 17(3):87–96, 1998.

[27] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 65–74. ACM Press, 1988.

[28] David Ebert and Penny Rheingans. Volume illustration: nonphotorealistic rendering of volume models. In *VIS '00: Proceedings of the conference on Visualization '00*, pages 195–202. IEEE Computer Society Press, 2000.

[29] H. Edelsbrunner. The union of balls and its dual shape. *Discrete Comput. Geom.*, 13:415–440, 1995.

[30] Herbert Edelsbrunner. *Algorithms in combinatorial geometry.* Springer-Verlag New York, Inc., 1987.

[31] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH 2001*, pages 341–346, August 2001.

[32] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the International Conference on Computer Vision-Volume 2*, page 1033. IEEE Computer Society, 1999.

[33] A. B. Ekoule, F. C. Peyrin, and C. L. Odet. A triangulation algorithm from arbitrary shaped multiple planar contours. *ACM Trans. Graph.*, 10(2):182–199, 1991.

[34] T. Todd Elvins. A survey of algorithms for volume visualization. *SIGGRAPH Comput. Graph.*, 26(3):194–201, 1992.

[35] Jerry Fails and Dan Olsen. A design tool for camera-based interaction. In *Proceedings of the conference on Human factors in computing systems*, pages 449–456. ACM Press, 2003.

[36] P.F. Felzenszwalb and D.P. Huttenlocher. Image segmentation using local variations. *IEEE Computer Vision and Pattern Recognition*, pages 98–104, 1998.

[37] Eric Ferley, Marie-Paule Cani, and Jean-Dominique Gascuel. Practical volumetric sculpting. In *Proceedings of Implicit Surface '99*, Sep 1999.

[38] Randima Fernando, editor. *GPU Gems*. Addison Wesley, Boston, MA, 2004.

[39] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 249–254. ACM Press/Addison-Wesley Publishing Co., 2000.

[40] H. Fuchs, Z. M. Kedem, and S. P. Uselton. Optimal surface reconstruction from planar contours. *Commun. ACM*, 20(10):693–702, 1977.

[41] I. Fujishiro, Y. Maeda, and H. Sato. Interval volume: a solid fitting technique for volumetric data display and analysis. In *VIS '95: Proceedings of the 6th conference on Visualization '95*, page 151. IEEE Computer Society, 1995.

[42] Issei Fujishiro, Yuriko Takeshima, Taeko Azuma, and Shigeo Takahashi. Volume data mining using 3d field topology analysis. *IEEE Comput. Graph. Appl.*, 20(5):46–51, 2000.

[43] Tinsley A. Galyean and John F. Hughes. Sculpting: an interactive volumetric modeling technique. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 267–274. ACM Press, 1991.

[44] Allen Van Gelder and Kwansik Kim. Direct volume rendering with shading via three-dimensional textures. In *VVS '96: Proceedings of the 1996 symposium on Volume visualization*, pages 23–ff. IEEE Press, 1996.

[45] Bruce Gooch and Amy Gooch. *Non-Photorealistic Rendering*. A. K. Peters, 2001.

[46] H. Nishimura H., M. Hirai, T. Kawai, T. Kawata, I. Shirakawa, and K. Omura. Object modelling by distribution function and a method of image generation. *The Transactions of the Institute of Electronics and Communication Engineers of Japan*, J68-D(4):718–725, 1985. in Japanese, translated into English by Takao Fujuwara.

[47] Paul Haeberli. Paint by numbers: abstract image representations. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 207–214. ACM Press, 1990.

[48] T. Haig, Y. Attikiouzel, and M. D. Alder. Border marriage: Matching of contours of serial sections. In *IEE Proceedings I, 138(5)*, pages 371–376, 1991.

[49] Pat Hanrahan and Paul Haeberli. Direct wysiwyg painting and texturing on 3d shapes. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 215–223. ACM Press, 1990.

[50] R. Haralick. Statistical and structural approaches to texture. In *Proc. IEEE*, volume 67, pages 786–804. IEEE Computer Society Press, 1979.

[51] Lowell Harris, R.A. Robb, T.S. Yuen, and E.L. Ritman. Non-invasive numerical dissection and display of anatomic structure using computerized x-ray tomography. In *Proceedings of SPIE 152*, pages 10–18, 1978.

[52] John C. Hart. *Ray Tracing Implicit Surfaces - Modeling, Visualizing, and Animating Implicit Surfaces (course note in SIGGRAPH 93).* 1993.

[53] P. Hastreiter and T. Ertl. Fast and interactive 3D–segmentation of medical volume data. In H. Niemann, H.-P. Seidel, and B. Girod, editors, *Proceedings of Workshop on Image and Multi-dimensional Digital Signal Processing (IMDSP)*, 1998.

[54] Helwig Hauser, Lukas Mroz, Gian-Italo Bischi, and Eduard Gr ler. Two-level volume rendering-fusing mip and dvr. In *VISUALIZATION '00: Proceedings of the 11th IEEE Visualization 2000 Conference (VIS 2000).* IEEE Computer Society, 2000.

[55] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 229–238. ACM Press, 1995.

[56] Henk J.A.M. Heijmans. Connected morphological operators for binary images. *Comput. Vis. Image Underst.*, 73(1):99–120, 1999.

[57] Gabor T. Herman, Jingsheng Zheng, and Carolyn A. Bucholtz. Shape-based interpolation. *IEEE Comput. Graph. Appl.*, 12(3):69–79, 1992.

[58] G.T. Herman and H.K. Liu. Three-dimensional display of human organs from computed tomograms. *Computer Graphics and Image Processing*, 9:1–21, 1979.

[59] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 327–340. ACM Press, 2001.

[60] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L. Kunii. Topology matching for fully automatic similarity estimation of 3d shapes. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 203–212. ACM Press, 2001.

[61] Ken Hinckley, Randy Pausch, John C. Goble, and Neal F. Kassell. Passive real-world interface props for neurosurgical visualization. In *Conference companion on Human factors in computing systems*, page 232. ACM Press, 1994.

[62] K.H. Hoehne and R. Bernstein. Shading 3d-images from ct using gray-level gradients. In *IEEE Transactions on Medical Imaging*, pages 45–47, 1986.

[63] Kenneth E. Hoff III, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. *Computer Graphics*, 33(Annual Conference Series):277–286, 1999.

[64] C.K. Hoffman. *Geometric and Solid Modeling.* Morgan Kaufmann Pub., 1989.

[65] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):71–78, 1992.

[66] G.N. Hounsfield. Computerized transverse axial scanning (tomography). *Br.J. Radiol*, 246(1):1016–1022, 1973.

[67] Godfrey Newbold Hounsfield. Computerized transverse axial scanning (tomography). 1. description of system. In *Br J Radiol*, pages 46: 1016–1022, 1973.

[68] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: a sketching interface for 3d freeform design. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 409–416. ACM Press/Addison-Wesley Publishing Co., 1999.

[69] Robert Jagnow, Julie Dorsey, and Holly Rushmeier. Stereological techniques for solid textures. *ACM Transactions on Graphics (Proc. Siggraph 2004)*, 23(3):329–335, 2004.

[70] James T. Kajiya and Brian P Von Herzen. Ray tracing volume densities. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 165–174. ACM Press, 1984.

[71] Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes, and Adam Finkelstein. Wysiwyg npr: drawing strokes directly on 3d models. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 755–762. ACM Press, 2002.

[72] Armin Kanitsar, Rainer Wegenkittl, Dominik Fleischmann, and Meister Eduard Gröller. Advanced Curved Planar Reformation: Flattening of Vascular Structures. In *IEEE Visualization 2003*, pages 43–50, October 2003. human contact: technical-report@cg.tuwien.ac.at.

[73] Olga Karpenko, John F. Hughes, and Ramesh Raskar. Free-form sketching with variational implicit surfaces. *Computer Graphics Forum*, 21(3):585–594, September 2002.

[74] Arie Kaufman, Rick Avila, Sarah Gibson, Bill Lorensen, Hanspeter Pfister, Milos Sramek, and J. Edward Swan II. *Volume Graphics (course notes for Siggraph 99 conference)*. 1999.

[75] E. Keppel. Approximating complex surfaces by triangulation of contour lines. *IBM Journal of Research and Development*, 19(1):2–11, 1975.

[76] J. Kniss, G. Kindlmann, and C. Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of IEEE Visualization '01*, pages 255–262. IEEE, 2001.

[77] Joe Kniss, Simon Premoze, Charles Hansen, and David Ebert. Interactive translucent volume rendering and procedural modeling. In *Proceedings of the conference on Visualization '02*, pages 109–116. IEEE Computer Society, 2002.

[78] A. König, H. Doleisch, and E. Gröller. Multiple views and magic mirrors—fmri visualization of the human brain. In *Technical Re-*

port TR-186-2-99-08, Inst. of Computer Graphics and Algorithms, Vienna Univ. of Technology, Feb. 1999., 1999.

[79] Jens Krüeger and Rüediger Westermann. Acceleration techniques for gpu-based volume rendering. In *Proceedings IEEE Visualization 2003*, 2003.

[80] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics (Proc. Siggraph 2003)*, 22(3):277–286, 2003.

[81] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 451–458. ACM Press, 1994.

[82] Ares Lagae, Olivier Dumont, and Philip Dutré. Geometry synthesis. In *Siggraph 2004 technical sketch*, 2004.

[83] Adam Lake, Carl Marshall, Mark Harris, and Marc Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *Proceedings of the 1st international symposium on Nonphotorealistic animation and rendering*, pages 13–20. ACM Press, 2000.

[84] Robert S. Laramee, Daniel Weiskopf, Jurgen Schneider, , and Helwig Hauser. Investigating swirl and tumble flow with a comparison of visualization techniques. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 51–58. IEEE Computer Society, 2004.

[85] P.C. Lauterbur. Image formation by induced local interactions: examples employing nuclear magnetic resonance. *Nature*, 242:190–191, 1973.

[86] Marc Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8(3):29–37, 1988.

[87] Yin Li, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. Lazy snapping. *ACM Transactions on Graphics (Proc. Siggraph 2004)*, 23(3):303–308, 2004.

[88] Barthold Lichtenbelt, Randy Crane, and Shaz Naqvi. *Introduction to Volume Rendering (Hewlett-Packard Professional Books)*. Prentice Hall, 1998.

[89] C. Loop. Smooth subdivision surfaces based on triangles. *Master's thesis, University of Utah, Department of Mathematics*, 1987.

[90] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169. ACM Press, 1987.

[91] Aidong Lu, Christopher J. Morris, David Ebert, Penny Rheingans, and Charles Hansen. Non-photorealistic volume rendering using stippling techniques. In *VIS '02: Proceedings of the conference on Visualization '02*. IEEE Computer Society, 2002.

[92] Eric B. Lum and Kwan-Liu Ma. Lighting transfer functions using gradient aligned sampling. In *VIS '04: Proceedings of the IEEE Visualization 2004 (VIS'04)*, pages 289–296. IEEE Computer Society, 2004.

[93] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design galleries: a general approach to setting parameters for computer graphics and animation. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 389–400. ACM Press/Addison-Wesley Publishing Co., 1997.

[94] Kevin T. McDonnell and Hong Qin. Dynamic sculpting and animation of free-form subdivision solids. In *Proceedings of the Computer Animation*, page 126. IEEE Computer Society, 2000.

[95] M.J. McGuffin, L. Tancau, and R. Balakrishnan. Using deformations for browsing volumetric data. In *Proceedings of the conference on Visualization '03*, pages 401–408. IEEE Computer Society, 2003.

[96] Shinji Mizuno, Minoru Okada, and Jun ichiro Toriwaki. Virtual sculpting and virtual woodcut printing. *The Visual Computer*, 14(2):39–51, 1998.

[97] Lukas Mroz. *Real-Time Volume Visualization on Low-End Hardware*. Vienna University of Technology, 2001.

[98] James C. Mullikin. The vector distance transform in two and three dimensions. *CVGIP: Graph. Models Image Process.*, 54(6):526–535, 1992.

[99] Andrew Nealen and Marc Alexa. Hybrid texture synthesis. In *Proceedings of the 14th Eurographics workshop on Rendering*, pages 97–105. Eurographics Association, 2003.

[100] Michael Meißner, Ulrich Hoffmann, and Wolfgang Straßer. Enabling classification and shading for 3d texture mapping based volume rendering using opengl and extensions. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 207–214. IEEE Computer Society Press, 1999.

[101] Michael Meißner, Jian Huang, Dirk Bartz, Klaus Mueller, and Roger Crawfis. A practical evaluation of popular volume rendering algorithms. In *VVS '00: Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 81–90. ACM Press, 2000.

[102] Fabrice Neyret and Marie-Paule Cani. Pattern-based texturing revisited. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques (Proc. Siggraph 99)*, pages 235–242. ACM Press/Addison-Wesley Publishing Co., 1999.

[103] R. Nock and F. Nielsen. Grouping with bias revisited. In A. Bobick L.-S. Davis, R. Chellapa, editor, *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 460–465. IEEE CS Press, 2004.

[104] Richard Nock and Frank Nielsen. Statistical region merging. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(11):1452–1458, 2004.

[105] Y. Ogawa, T. Ohtani, J. Sugiyama, S. Hagiwara, M. Kokubo, K. Kudoh, and T. Higuchi. Three dimensional visualization of internal constituents in a rice grain. *ASAE/CSAE-SCGR Annual International Meeting*, (993059), 1999.

[106] A. Opalach and S. C. Maddock. An overview of implicit surfaces. In *Introduction to Modelling and Animation Using Implicit Surfaces*, pages 1.1–1.13, 1995.

[107] Shigeru Owada, Frank Nielsen, Kazuo Nakazawa, and Takeo Igarashi. A sketching interface for modeling the internal structures of 3d shapes. In *Proceedings of the 4th International Symposium on Smart Graphics*, pages 49–57. Springer-Verlag LNCS 2733, July 2003.

[108] 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.

[109] Bradley A. Payne and Arthur W. Toga. Distance field manipulation of surface models. *IEEE Comput. Graph. Appl.*, 12(1):65–71, 1992.

[110] Hans Køhling Pedersen. A framework for interactive texturing on curved surfaces. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 295–302. ACM Press, 1996.

[111] Ken Perlin. An image synthesizer. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 287–296. ACM Press, 1985.

[112] Hanspeter Pfister, Jan Hardenbergh, Jim Knittel, Hugh Lauer, and Larry Seiler. The volumepro real-time ray-casting system. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 251–260. ACM Press/Addison-Wesley Publishing Co., 1999.

[113] Hanspeter Pfister and Arie Kaufman. Cube-4-a scalable architecture for real-time volume rendering. In *VVS '96: Proceedings of the 1996 symposium on Volume visualization*, pages 47–55. IEEE Press, 1996.

[114] Hanspeter Pfister, Bill Lorensen, Chandrajit Bajaj, Gordon Kindlmann, Will Schroeder, Lisa Sobierajski Avila, Ken Martin, Raghu Machiraju, and Jinho Lee. The transfer function bake-off. *IEEE Comput. Graph. Appl.*, 21(3):16–22, 2001.

[115] K. Popat and R. Picard. Novel cluster-based probability model for texture synthesis, classification, and compression. In *Proceedings SPIE visual Communications and Image Processing '93, Boston, 1993.*, 1993.

[116] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 465–470. ACM Press/Addison-Wesley Publishing Co., 2000.

[117] Proceedings of Pacific Graphics 2002. *Feature-Enhanced Visualization of Multidimensional, Multivariate Volume Data Using Non-photorealistic Rendering Techniques*. IEEE, 2002.

[118] David Pugh. Designing solid objects using interactive sketch interpretation. In *Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 117–126. ACM Press, 1992.

[119] E.M. Purcell, H.C. Torrey, and R.V. Pound. Resonance absorption by nuclear magnetic moments in a solid. *Phys. Rev.*, 69:37–38, 1946.

[120] J. Radon. On the determination of functions from their integrals along certain manifolds. In *Berichte Seachsische Acad. Wiss. 69*, pages 262–271, 1917.

[121] S. P. Raya and J. K. Udupa. Shape-based interpolation of multidimensional objects. *IEEE Transactions on Medical Imaging*, 9(1):32–42, 1992.

[122] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume on standard pc graphics hardware using multitextures and multi-stage rasterization. In *HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 109–118. ACM Press, 2000.

[123] John C. Russ. *The Image Processing Handbook Fourth Edition*. CRC Press, 2002.

[124] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 197–206. ACM Press, 1990.

[125] Michael Shantz. Surface definition for branching, contour-defined objects. *SIGGRAPH Computer Graphics*, 15(2):242–270, 1981.

[126] Anthony Sherbondy, Mike Houston, and Sandy Napel. Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In *Proceedings of IEEE Visualization 2003*, pages 171–176. IEEE, 2003.

[127] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.

[128] Yoshihisa Shinagawa and Tosiyasu L. Kunii. Constructing a reeb graph automatically from cross sections. *IEEE Computer Graphics and Applications*, 11(6):44–51, 1991.

[129] Yoshihisa Shinagawa and Tosiyasu L. Kunii. The homotopy model: a generalized model for smooth surface generation from cross sectional data. *The Visual Computer*, 7(2):72–86, 1991.

[130] Yoshihisa Shinagawa, Tosiyasu L. Kunii, and Yannick L. Kergosien. Surface coding based on morse theory. *IEEE Computer Graphics and Applications*, 11(5):66–78, 1991.

[131] Peter G. Sibley, Philip Montgomery, and G. Elisabeta Marai. Wang cubes for geometry placement and video synthesis. In *Siggraph 2004 Poster*, 2004.

[132] Jos Stam. Aperiodic texture mapping. In *Technical Report R046, European Research Consortium for Informatics and Mathematics (ERCIM)*, 1997.

[133] Matus Straka, Michal Cervenansky, Alexandra La Cruz, Arnold Kochl, Milos Sramek, Eduard Groller, and Dominik Fleischmann. The vesselglyph: Focus & context visualization in ct-angiography. In *VIS '04: Proceedings of the IEEE Visualization 2004 (VIS'04)*, pages 385–392. IEEE Computer Society, 2004.

[134] Steve Strassmann. Hairy brushes. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 225–232. ACM Press, 1986.

[135] Ivan E. Sutherland. *Sketchpad: A Man-Machine Graphical Communication System. Ph.D. Thesis.* MIT, 1963.

[136] Eric Tabellion and Arnauld Lamorlette. An approximate global illumination system for computer generated films. *ACM Trans. Graph.*, 23(3):469–476, 2004.

[137] Shigeo Takahashi, Yuriko Takeshima, and Issei Fujishiro. Topological volume skeletonization and its application to transfer function design. *Graphical Models*, 66(1):24–49, 2004.

[138] Ikuko Takanashi, Eric Lum, Shigeru Murakin, and Kwan-Liu Ma. Ispace: Interactive volume data classification techniques using independent component analysis. IEEE, 2002.

[139] Gabriel Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358. ACM Press, 1995.

[140] S. M. F. Treavett and M. Chen. Pen-and-ink rendering in volume visualisation. In *VIS '00: Proceedings of the conference on Visualization '00*, pages 203–210. IEEE Computer Society Press, 2000.

[141] Graham M. Treece, Richard W. Prager, Andrew H. Gee, and Laurence H. Berman. Surface interpolation from sparse cross-sections using region correspondence. *IEEE Transactions on Medical Imaging*, 19(11):1106–1114, 2000.

[142] G. Turk and J.F. O'Brien. Variational implicit surfaces. In *Technical Report GIT-GVU-99-15, Graphics, Visualization, and Usability Center. Georgia Institute of Technology.*, 1999.

[143] Greg Turk. Texture synthesis on surfaces. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 347–354. ACM Press, 2001.

[144] Fan-Yin Tzeng, Eric B. Lum, and Kwan-Liu Ma. A novel interface for higher-dimensional classification of volume data. In *Proceedings of IEEE Visualization 2003*, pages 505–512. IEEE, 2003.

[145] John Viega, Matthew J. Conway, George Williams, and Randy Pausch. 3d magic lenses. In *Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 51–58. ACM Press, 1996.

[146] Ivan Viola, Armin Kanitsar, and Meister Eduard Gröller. Importance-driven volume rendering. In *Proceedings of IEEE Visualization'04*, pages 139–145, 2004.

[147] Sidney W. Wang and Arie E. Kaufman. Volume sculpting. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 151–157. ACM Press, 1995.

[148] Li-Yi Wei. *Texture Synthesis by Fixed Neighborhood Searching. Ph.D. Thesis.* Stanford University, 2001.

[149] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488. ACM Press/Addison-Wesley Publishing Co., 2000.

[150] Daniel Weiskopf, Klaus Engel, and Thomas Ertl. Volume clipping via per-fragment operations in texture-based volume visualization. In *Proceedings of the conference on Visualization '02*, pages 93–100. IEEE Computer Society, 2002.

[151] William Welch and Andrew Witkin. Free-form shape design using triangulated surfaces. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 247–256. ACM Press, 1994.

[152] Lee Westover. Footprint evaluation for volume rendering. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 367–376. ACM Press, 1990.

[153] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 269–277. ACM Press, 1994.

[154] Qing Wu and Yizhou Yu. Feature matching and deformation for texture synthesis. *ACM Transactions on Graphics (Proc. Siggraph 2004)*, 23(3):364–367, 2004.

[155] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1986.

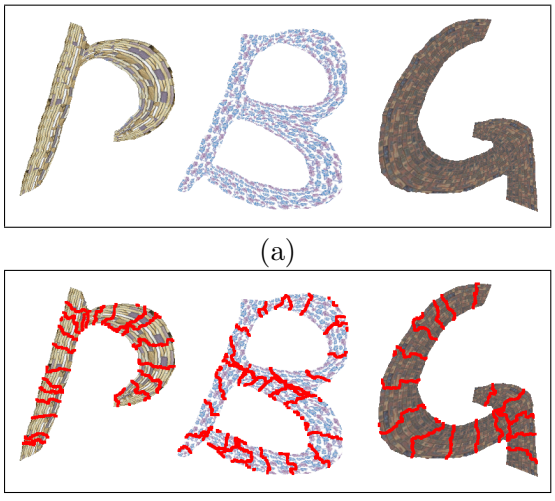
[156] Stella X. Yu and Jianbo Shi. Segmentation given partial grouping constraints. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(2):173–183, 2004.

[157] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. Sketch: an interface for sketching 3d scenes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 163–170. ACM Press, 1996.

[158] Jingdan Zhang, Kun Zhou, Luiz Velho, Baining Guo, and Heung-Yeung Shum. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Transactions on Graphics (Proc. Siggraph 2003)*, 22(3):295–302, 2003.

Appendix A

Improving quality of 2D distorted texture synthesis



(a)

(b)

Figure A.1: Font design using our system. The images are rendered without (a) and with (b) patch boundaries.

One problem of the cross-sectional image synthesis technique shown in the Chapter 4 is the quality of the generated image. Therefore, we focus on this quality problem in this Appendix. We propose a new 2D texture synthesis algorithm with its local orientation and scaling factor controlled by a distortion field. This technique can be applicable to not only volumetric illustration system but also to general distorted 2D texture synthesis. Our new algorithm produces much higher quality images than previous methods, mainly due to a patch-based algorithm and lack

of resampling process. Most existing methods for distorted texture synthesis use a pixel-based approach and output an image that has regular grid alignment. Pixel-based approaches, however, are usually less successful for visual quality than patch-based approaches. In addition, fixed pixel layout requires resampling of an input image, which produces undesirable degraded output. Our approach generates an irregularly sampled image using a patch-based technique for distorted texture synthesis. According to a predefined distortion field, we first distort an input image to form patches that have irregular pixel locations, and then merge them by a graphcut technique extended to support irregularly sampled images. Our method produces higher quality result than existing techniques, due to patch-based large scale coherency and its omission of resampling process. In our framework, an input image is treated as a set of independent sample points, producing a set of irregularly distributed original sample points. We also propose a hardware-accelerated method to effectively compute an approximate alpha shape of a point set, which is used to define a region of a point set and to construct a graph network used for our graphcut algorithm.

A.1 Background

Recently, texture synthesis techniques have gained much attention in computer graphics community and great number of applications are proposed such as texture mixing, hole-filling, and non-photorealistic rendering. Some application areas such as texture synthesis on arbitrary surface [158] or volumetric illustration [108] require distorted texture synthesis. However, in existing systems, pixel layout is predetermined (usually a regular grid) before synthesis is performed, which causes mismatch of pixel locations in original texture example and target image thus requiring resampling of pixel colors. It is a common knowledge that resampling causes significant degradation of image quality, unless the target sampling rate is higher than the original one and also resampling filter is carefully designed to avoid losing information. However, it is also known that resampling based on purely sampling theory has various side effects such as infinite support of the filter kernel. Therefore, virtually any practical resampling method loses information. Another problem is that most existing distorted texture synthesis algorithms use pixel-based synthesis because of its simplicity of creating a neighborhood structure (the distortion field is assumed to be locally linear). However,

the quality of images synthesized using pixel-based method is usually inferior to patch-based approach because of less coherency.

In this work, we propose a new patch-based distorted texture synthesis algorithm that uses 2D irregularly sampled images to avoid resampling (for short, we call an irregularly sampled image as an *ISI* in our context.) The notion of an ISI is not new nor original especially in the context of experimental science since sensor devices do not necessary produce regular data, regardless of its dimension. Even commercial software is available that handles irregular dataset [1]. We define an ISI as a 2D image in which the pixel alignment is arbitrary. In other words, it is a set of irregularly distributed colored 2D points. An ISI is rotated, distorted, and non-integer translated without being resampled. In our framework, an input (regular) image is first distorted to fit the predefined distortion field. The result forms an ISI. It is then connected to the partly completed target image (which is also an ISI) by using a graph-cut technique [80] with our extension to find an optimal seam between two ISIs. The graph structure between ISIs is temporarily constructed using alpha shapes [29]. Intuitively, alpha shape is a polygonal region that represents the shape of a point set which takes 'alpha' as a parameter. We also propose a method to effectively compute an approximate alpha shape of a point set using graphics hardware. The resulting image is provided again as an ISI that maintains the original sample points. Our method achieves higher quality than existing techniques because (1) the intermediate and final images are represented as ISIs and (2) the algorithm adopts a patch-based synthesis that maintains large-scale coherency.

A.2 Our algorithm

Our system inputs an example image and a distortion field function. The distortion field function should be smooth for later optimization process to work correctly. The function consists of two subcomponents: a scaling function and an orientation function. The scaling function inputs a 2D coordinate and outputs a scalar value that represents a magnitude of scaling at the position. The orientation function also inputs a 2D coordinate and outputs an oriented unit vector. Although they can be specified independently in our implementation, they can contradict (imagine the orientation function always returns (1,0) and the scaling function differs point by point.) However, the optimization process tries

to minimize the matching error. Like other existing patch-based algorithms, our algorithm consists of two phases: registration and stitching. Registration phase translates and distorts an input image to find a good location to paste a patch. This process uses an optimization of a cost function to evaluate the quality of matching to a given distortion field (Section A.2.2). Figure A.2 describes the entire procedure of our algorithm.
Point-based stitching phase is further divided into three subphases: identifying pixels in the overlapping region, graph construction, and graphcut operation. We explain each process in turn (section 6.3.3-5), but we first briefly mention our general representation for images.

A.2.1 Image as points

Images are usually provided as a set of pixels that are located on gridpoints of a regular grid. Each pixel represents a discrete sampling point of a continuous scene and stores the point's attribute values, typically RGB color components. We rather use images that have irregular alignment of pixels, which we call an ISI. We do not assume any predefined connectivity nor neighboring topological structure of points. Instead, neighborhood structure is dynamically constructed from the alpha shape of points. Therefore, each image can be locally or globally distorted, depending on the specific application. Although we mainly focus on synthesizing distorted texture, we will show other examples in a later section.

A.2.2 Registration

Typical image registration algorithms optimize an objective function that evaluates the quality of matching. There are a variety of objective functions that are closely related to specific optimization purposes. The original graphcut paper proposes three methods for registration [80].
For initial placement of an example image, we closely follow [80]. While the target region is not completely covered, a new patch is located so that the patch contains both an already synthesized region and an incompleted region. After the target region is fully covered, the new patch is located so as to cover a "*seam node*" that has the largest error. A seam node is an extra node that is added on a patch boundary for the purpose of storing quality of cut for future patch placement [80]. A difference from [80] is that our method rotates and uniformly scales the

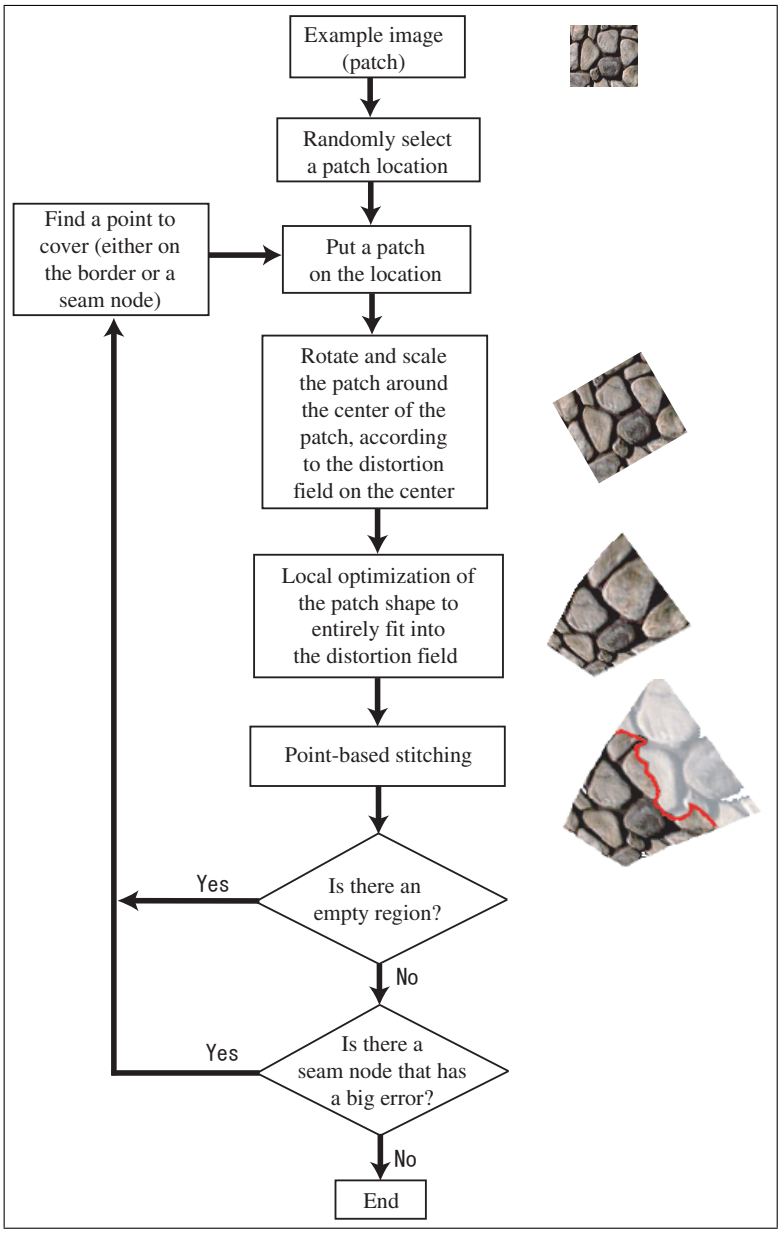


Figure A.2: Flow chart of our algorithm.

patch after the patch placement according to the distortion field on the center, and then locally distort the patch so as to completely fit it into the distortion field (See Figure 6.2).
This local patch distortion is done on a point-by-point basis. We assume that the alpha shape of the patch is precomputed and it consists of one connected component (section 6.3.3). A seed point is selected (typically the point which is the closest to the center of the bounding box of the patch) and marked as "visited". Then an unvisited point that is closest to the seed point is selected one by one using a priority queue and marked as "visited" after iterative optimization of its location. The pseudocode is as follows:

```
function DistortPatch( Patch patch,DistortionField distortField )
  currentPoint = CenterOfPatch( patch )
  currentPoint.appliedDistortion
    = distortField.Eval( currentPoint.position )
  priorityQueue.push( currentPoint )

  while( priorityQueue is not empty )
    currentPoint = priorityQueue.GetTopAndErase()
    if( currentPoint.visited ) continue

    initialPosition = PredictPosition( VisitedNeighbors(currentPoint) )
    currentPoint.position
      = LocalOptimization( initialPosition
                         , VisitedNeighbors(currentPoint)
                         , distortField )
    currentPoint.visited = true
    priorityQueue.push( UnvisitedNeighbors(currentPoint) )
```

The function $PredictPosition()$ computes an initial guess of the current point location based on already applied distortions on neighboring points. The applied distortion on a point is the result of the optimization process of $LocalOptimization()$, which is not necessary equal to the given distortion field's value.
Function $LocalOptimization()$ seeks the good location of the $currentPoint$ by minimizing the difference between approximate distortion (computed from location of $currentPoint$ and its neighbors) and given distortion field. An energy function that takes x and y values of the $currentPoint$ as variables is defined. The gradient of the energy function is estimated by sampling the energy function with small offsets in both x and y direction. Then the location of $currentPoint$ is iteratively improved by

modifying the location of *currentPoint* towards the steepest decent of the energy function (typically, 3 to 6 iterations per pixel are sufficient.) The energy function E has the following form:

$$E = (ScalingError) + \gamma(RotationError)$$

where γ is a user-defined weight. The two elements on the right side are computed as follows:

Scaling error

A scaling factor is estimated by the current length of edges in the alpha shape that are emanating from *currentPoint* to already visited points, divided by the length in the original patch. The square difference between this estimated scaling factor and ideal scaling factor of the given distortion field measures the fitting error of the scaling factor.

Rotation error

The orientation of *currentPoint* is estimated by edges that are emanating from *currentPoint* to already visited points. That is, the average difference between each emanating edge's orientation in an input image (without distortion) and in the current distortion field (with distortion) is defined to be the point's current orientation. The square of angle difference between this estimated orientation and the desired orientation from the distortion field defines the rotation error of the point.

Although this per-pixel distortion works, this process significantly increases the computational cost in practice. In addition, local errors cause an non-uniform patch shape, even if the given distortion field is uniform. To avoid these problems, we may constrain the distortion (such as an affine transform) or introduce multi-resolution distortion. We leave this possibility as a future work.

A.2.3 Defining overlapping region

In our framework, the problem of seamlessly stitching two pre-registered ISIs amounts to finding an optimal seam in the overlapping region and removing a portion of points in the region bounded by the seam. We first need to identify which points are in the overlapping region. Only those points in the overlapping region are of interest and subject to possible removal. We use an alpha shape to define a (continuous) shape of a point set and the overlapping region [29]. Intuitively, alpha shape is a polygonal region that represents a shape of a point set. Computing an alpha shape requires a predefined alpha value that de-

termines a maximum region of effect of each point. Although there are several definitions of what an alpha value describes [30], we define an alpha value as a radius of a region of effect. If two neighboring points are close enough, say, the distance is equal to or below $2 \cdot alpha$, an edge is spanned between the points that forms a part of the alpha shape. Therefore, in 2D, the boundary of an alpha shape is a subset of the Delaunay triangulation. A point in one patch is defined to be in the overlapping region if and only if it is in the alpha shape of the other patch. In our application, an alpha value is fixed. This may cause possible breaks in the highly-distorted region. Setting a larger alpha value weakens this effect but too large alpha value tends to close concave boundaries. If this is not desirable, nonuniform or adaptive alpha value should be introduced. We leave this possibility as a future work.

A.2.4 Fast approximate alpha shape computation with graphics hardware

Exact alpha shapes can be computed from Delaunay triangulation of the input points. However, this can easily form the bottleneck of the entire process. Therefore, we propose a method to quickly approximate an alpha shape using graphics hardware. The time complexity of hardware accelerated method is $O(N)$ where N is the number of points, while an exact analytical method requires $O(N \log(N))$.

We first compute a Voronoï diagram of an input point set by rendering cones into a frame buffer [63]. A slight difference from [63] is that the radius of each cone's base is set as the corresponding alpha value (Figure 6.3 (a)(b)). Therefore, there are possibly some empty regions.

We then scan the rendered image in the frame buffer to find pixel boundaries where three regions meet (Figure 6.3(c)). Those points represent triangles that belong to the alpha shape in the dual space (Figure 6.3(d)). If there are points where four regions meet simultaneously, the points are degenerate. To obtain a triangulated alpha shape, the quadrangle formed by four points that correspond to four regions should be split into two triangles. One of the two diagonals is selected so as for minimum angle among 6 corners to be maximized. Note that points where three or four regions meet can exist outside of the boundary box of the point set. Therefore, the size of an offscreen buffer should be bigger than the boundary box by the extent of alpha.

The resolution of the frame buffer determines the error tolerance. If the

size of each pixel of a frame buffer is larger than the smallest distance between points in the set, there may be some points that do not appear in the buffer. However, too small pixel sizing requires large frame buffer. In our implementation, the user defines a unique pixel size as a tolerance value and those points that do not appear in the frame buffer are simply removed.
Finally, the triangles in the alpha shape are rendered into the frame buffer. The resulting image is a rasterized version of the alpha-shape. It is used to quickly determine if an arbitrary location is in the alpha shape or not.

A.2.5 Graph construction

To apply a graphcut technique to ISIs, we first need to construct a graph structure representing the cost of cut between points.
Firstly, the system selects points that are used as nodes of the graph. There are two options; using all points in the overlapping region or using a portion of points. We use points that are from only one of the two ISIs for several reasons: (1) Using all points causes higher density of points, which requires a large offscreen buffer. Since the amount of graphics memory is limited, smaller offscreen buffer is desired. (2) The number of points significantly affects the computational cost of a graph-cut operation. (3) We already have an alpha shape of each ISI that was computed in the previous section. This alpha shape can directly be used as the graph.
We use an ISI that has higher average point density than the other one. The crude density estimate is computed from the Voronoï diagram and a depth buffer computed in the previous section, by averaging depth values of pixels where three regions meet.
Then the system assigns color difference cost on each node. Color difference cost of a node is computed by taking a squared difference between the color of the node itself and the interpolated color on the other ISI at the same location. Although this is simply done by just taking a squared difference of color vector, it requires interpolation of colors because location of pixels from different ISI does not match. Note that in our context, interpolation is used to guide our graph-cut operation and not for synthesizing new output points at prescribed coordinates. Therefore, interpolation is not used for resampling inputs as conventional methods do but rather for selecting appropriate points of ISIs. We just take 0-

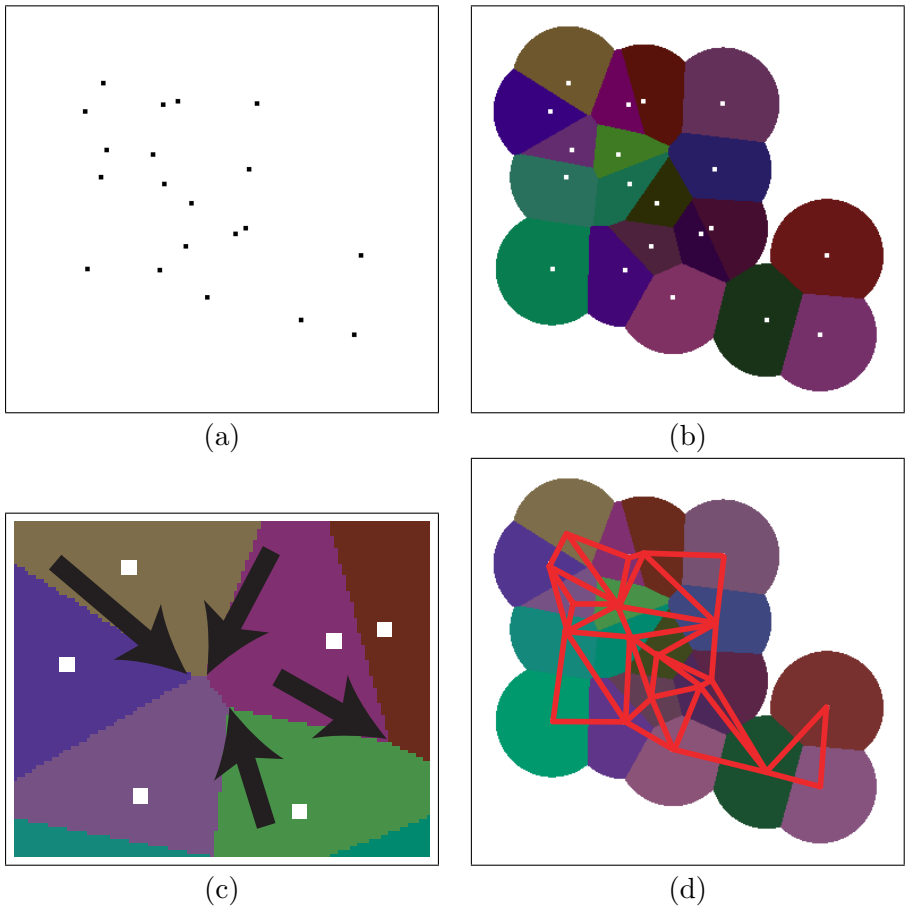


Figure A.3: Hardware assisted alpha shape construction. (a) Input points. (b) A cone is drawn for each point. The radius of each cone's base is set as the corresponding alpha value (c) Points where three regions meet are searched (closeup). Those points correspond to a triangle that forms the alpha shape. (d) Computed alpha shape.

order interpolation (nearest neighbor) for two reasons; the nearest point is efficiently obtained by observing a Voronoï diagram precomputed in the previous section and, higher order interpolation usually has a blurry effect.
The remaining process shows much similarity to an existing method [80]. Edge cost is computed as the sum of two nodes on both ends, source and sink nodes representing each ISI are added to the graph and connected to the outmost nodes by edges that have infinite cost.

A.2.6 Merging point sets

After constructing a graph structure, an optimal boundary between ISIs is computed as the minimum cut of the graph. The system then merges points from two ISIs by using the boundary information. The cut strictly splits nodes into two sets; source set and sink set. However, in our algorithm, every node is from only one ISI. Therefore, one of the source/sink sets should be discarded and replaced by the points from the other ISI. To select appropriate points from the ISI, we define source region and sink region by, again, the alpha shape of points. The overlapping region is strictly divided into two regions and points to be removed are determined by checking which region it belongs to.
Finally, seam nodes between new and old ISIs are added to the resulting ISI. The seam nodes are generated on edges that form optimal cut. However, the density of the seam nodes are usually much higher than original ISIs, because graphcut is done on triangular mesh while ISIs are originated from regular images, that is, quadrangular meshes. We therefore remove seam nodes alternately.

A.3 Results

We applied our algorithm to various examples. The point sets are rendered by their triangulated alpha shapes using polygon rendering hardware instead of splatting. An example in Figure 6.4 shows a texture synthesis result using our system without a distortion field. To create this example, we randomly locate and rotate a new patch.

We developed a tool to interactively define a distortion field. “Curvy field” tool enables the user to draw two curves that defines orientation and scaling factor at the same time (Figure 6.5). The value ‘0’ is assigned to the first curve and ‘1’ is assigned to the second curve. Then the two

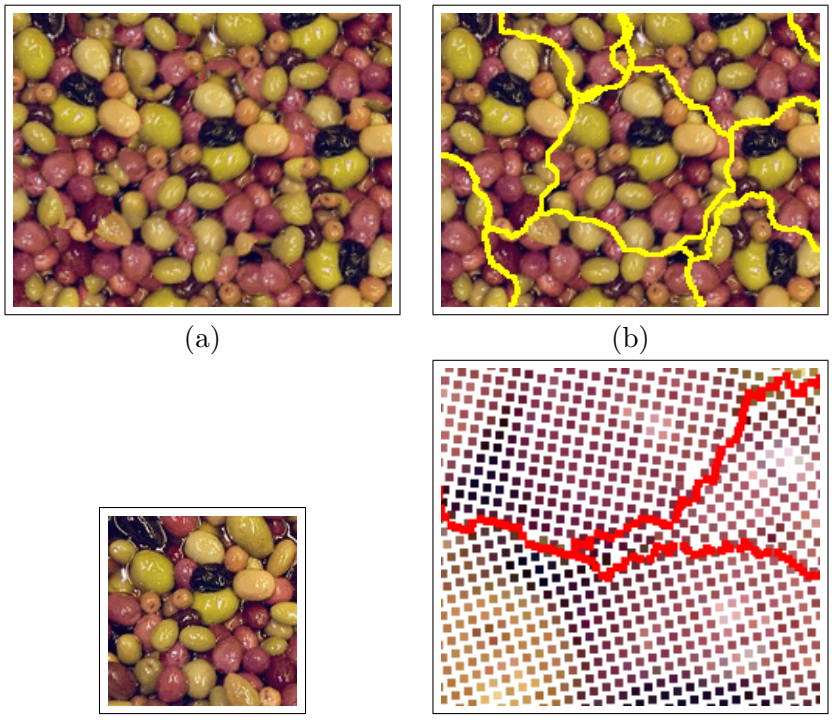


Figure A.4: Our texture synthesis result without a distortion field. (a) and (b) are the output and (c) is the source image. (d) is the closeup of (b) (the seam color is changed to be clearly seen.

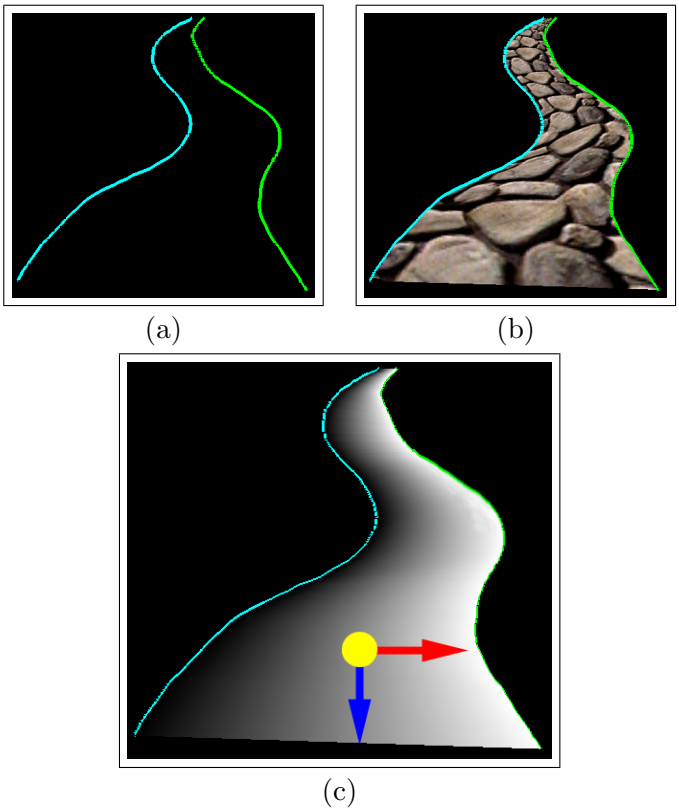
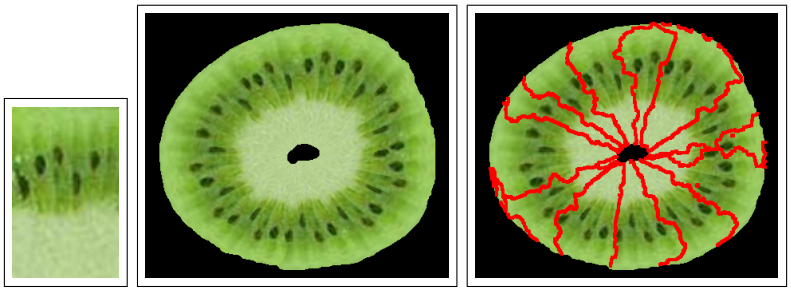


Figure A.5: Curvy field tool. From the user-specified strokes (a), distorted texture synthesis is performed (b) using a distortion field (c).

curves are linearly interpolated to form a scalar field as in Figure 6.5(c). The scaling factor of each point is defined as an inverse of a magnitude of the gradient vector while the orientation of the field is either the gradient direction (rightward arrow in Figure 6.5(c)) or perpendicular to the gradient vector (downward arrow in Figure 6.5(c)). Patch location can be constrained to synthesize a texture as in Figure 6.6. Note that the images can self-intersect. Therefore, one cannot synthesize an image in a regular coordinate system and distort it later in this case.

Our point-based image stitching algorithm is also suitable to stitch images in panoramic image mosaicing. Typical image mosaicing method requires distorting multiple source images, based on homographies. Those registered images are then stitched to form a single environment map by either blurring the edge (which is sometimes called *feathering*) or finding an optimal seam in the target pixel location. We rather propose to find



(a) (b) (c)

Figure A.6: Kiwi example. The patch placement is constrained so as for top and bottom rows in the input image (a) match the external and internal boundaries of the resulting image.

an optimal seam directly from input sample points. The stitched result is provided as a set of original sample points, thus can be rendered without any resampling artifact. Figure 6.7 shows an application of our new point-based graphcut algorithm to an image stitching problem.

Note that we do not claim that our method does not have any presence of resampling artifact such as a Moiré pattern when rendered on the screen. A Moiré pattern can appear because the source images themselves may already contain Moiré pattern and also because a typical computer display has fixed resolution. We would rather emphasize that our method does not have any intrinsic process that loses original information. Therefore, the quality of the output image purely depends on the original image quality and the final rendering environment. This is especially beneficial for professional graphic designers. They currently allocate extremely high resolution work space partly because repeatedly applying distortions on their material causes significant degradation of the quality. Therefore, designers usually let the resolution of their work space even higher than high-end printers. However, simply introducing irregularly sampled images and conduct the operations in those space, unnecessary growth of resolution can be avoided.

A.4 Discussions

In this Appendix, we introduced a patch-based distorted texture synthesis algorithm using irregularly sampled images (ISIs). We extended a graphcut technique to support ISIs and showed some applica-

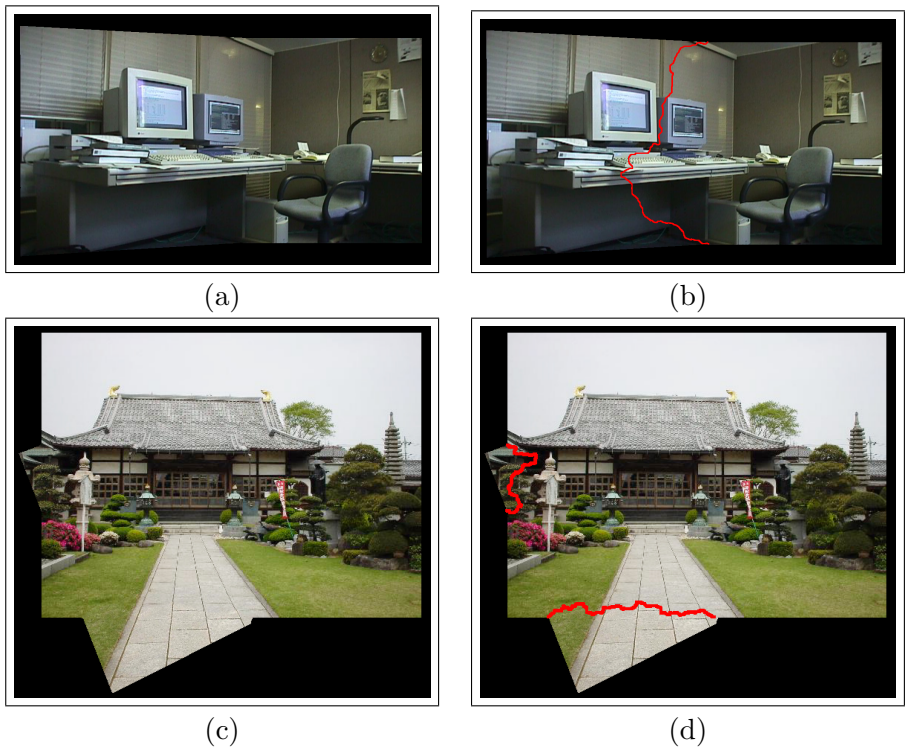


Figure A.7: Image stitching example

tions. We also introduced a hardware accelerated method to compute an approximate alpha shape of a point set.

Our possible extension is to handle variable alpha values. In our implementation, too much distortion causes breaks of an original image since our alpha value is fixed. We need to dynamically control alpha values according to the distortion field.