Feature-Driven Illustrative Visualization and Graphics

A Dissertation Presented

by

Lujin Wang

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

Stony Brook University

December 2007

Stony Brook University

The Graduate School

Lujin Wang

We, the dissertation committee for the above candidate for the

Doctor of Philosophy degree, hereby recommend

acceptance of this dissertation.

Klaus Mueller – Dissertation Advisor
Associate Professor, Department of Computer Science

Arie Kaufman – Chairperson of Defense
Distinguished Professor and Chairman, Department of Computer Science

Xianfeng Gu – Committee Member
Assistant Professor, Department of Computer Science

Holly Rushmeier – External Committee Member
Professor, Department of Computer Science, Yale University

This dissertation is accepted by the Graduate School

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

Feature-Driven Illustrative Visualization and Graphics

by

Lujin Wang

Doctor of Philosophy

in

Computer Science

Stony Brook University

2007

We present several feature-driven illustrative visualization and graphics techniques to enhance the representation of the features of interest in volume datasets. While the magnitude and resolution of real-life datasets keep increasing dramatically, there is a limit on the screen pixel density the human eye can resolve, and a bound on the information a human brain can visually process at any given time. Therefore, we devise techniques to facilitate the perception of the visual information.

First, we propose a GPU-based focus+context framework that uses various standard and advanced magnification lens rendering techniques to magnify the features of interest, while compressing the remaining volume regions without clipping them away completely. Our technique allows the user to interactively manage the available screen area, dedicating more area to the more resolution-important features. A generalization of this concept is multiperspective rendering, which is also studied in our framework to show the spatial relationships of features.

Second, when features are simply magnified, there will always be a limit on the available level of detail and the resolution of the data. To address these shortcomings, we present a technique to extend regular zooms to semantic zooms. Our technique generates the missing detail from any available and plausible high-resolution datasets, using constrained texture synthesis. We demonstrate our approach by ways of a medical application – the visualization of a human liver – but its principles readily apply to any scenario, as long as data at all resolutions are available.

The third topic is related to surface texture mapping and synthesis, where we present two methods that preserve both scale and angle. By using global conformal parameterization, the 3D surface texture synthesis problem can be converted to a 2D image synthesis problem. Our multi-scale synthesis method maintains a more uniform area scaling factor. By employing a conformal factor-driven mass-spring relaxation on global parameterization, our second method helps preserve orthogonality and size in texture mapping.

This thesis also seeks to break new grounds in embedding concepts from human perception, cognition, and visual processing into visualization design. We present a rule based color design system to provide better control for task-driven or feature-driven visualization tasks. Our system not only assists in the selection of proper colors, it also helps to avoid poor color mixing in semi-transparent rendering and the apparent change in brightness in color harmonization. Then, inspired by our work on volume rendering and color design, we propose a general multi-layer multi-volume rendering framework. Finally, we investigate the influence and settings of various volume rendering parameters by conducting a user study with 750 participants, assessing the results via conjoint analysis, a promising paradigm to conduct user studies in visualization developed by close collaborators.

To My Husband, Yongzhi Chen,

My Father, Mingsen Wang, and My Mother, Yan Li

with My Love!

Contents

List of Tables

List of Figures

Acknowledgments

First of all, I would like to thank my parents, my husband and my brother for their invaluable love and support for me in my life.

I want to express my deep gratitude to my advisor, Professor Klaus Mueller, for his years of guidance, support and encouragement. He helped me set and achieve the higher research goals which makes this thesis possible.

I would like to thank Professors Arie Kaufman, Xianfeng Gu, Michael Ashikhmin, Dimitris Samaras, Hong Qin and Joachim Giesen, for helpful collaborations and valuable discussions over the years. Special thanks to Professor Holly Rushmeier for being my external committee member.

I want to thank all the current and past Visualization Lab members. Thank Bin Zhang for the technical supports, thank Ye Zhao, Feng Qiu and Miao Jin for joint work, thank Neophytou Neophytos, Fang Xu, Xin Guan, Shengying Li, Aili Li, Wei Hong, Jianning Wang, Haitao Zhang, Zhe Fan, Satprem Pamudurthy, Eun_ju Nam and Supriya Garg for their help and friendship. The life at Stony Brook has been a great experience for me.

This work has been supported by NSF career grant ACI-0093157.

Publications

1. Lujin Wang, Klaus Mueller. **Generating Sub-Resolution Detail in Images and Volumes Using Constrained Texture Synthesis**. In *Proceedings of IEEE Visualization ’04*, Austin, Texas, 75-82, 2004.

2. Lujin Wang, Ye Zhao, Klaus Mueller, Arie Kaufman. **The Magic Volume Lens: An Interactive Focus+Context Technique for Volume Rendering**. In *Proceedings of IEEE Visualization ’05*, Minneapolis, Minnesota, 367-374, 2005.

3. Lujin Wang, Xianfeng Gu, Klaus Mueller, Shing-Tung Yau. **Uniform Texture Synthesis and Texture Mapping Using Global Parameterization**. *The Visual Computer (Special issue of Pacific Graphics ’05)*, 21:8-10, 801-810, 2005.

4. Ye Zhao, Lujin Wang, Feng Qiu, Arie Kaufman and Klaus Mueller. **Melting and Flowing in Multiphase Environment**. *Computers & Graphics (the Special Issue on Natural Phenomena)*, 30, 519-528, 2006.

5. Joachim Giesen, Klaus Mueller, Eva Schuberth, Lujin Wang, Peter Zolliker. **Conjoint Analysis to Measure the Perceived Quality in Volume Rendering**. *To appear in IEEE Transaction on Visualization and Computer Graphics (Visualization ’07)*, 2007.

6. Lujin Wang, Klaus Mueller. **Color Design for Visualization**. In submission, 2007.

Chapter 1

Introduction

1.1 Motivation

Recent years have seen a dramatic growth in our ability to compute, acquire, and assemble datasets of increasingly large magnitudes and resolutions. Great advances have also been made in screen technology, bringing high-resolution displays to the desktop at affordable prices, as well as offering sophisticated CAVE environments. The one device that has consistently resisted participation in this spiral of growth is the human eye and the cortical visual processing abilities. In fact, there is a natural limit on the screen pixel density, as a function of distance, which the human eye can resolve, and there is also a natural falloff of retinal receptor density towards the foveal periphery. Finally, there is also a bound on the information the human brain can visually process at any given time, but this is probably an ability that can be trained the most. In view of these natural limitations, which are bound to stay, we must devise ways to make the best use of the available retinal surface and cerebral potential, in light of the growing amount of visual information ready to be presented.

Many illustrative visualization techniques have been proposed to exploit the perception of the human visual system and provide effective visual abstractions to make the visualization clearly understandable [132]. Such techniques are inspired by traditional technical and medical illustrations. Visual emphasis and abstraction have been used for expressive presentation from prehistoric paintings to nowadays scientific and medical illustrations. Many of the expressive techniques used in art are adopted in computer graphics, and are denoted as illustrative or non-photorealistic rendering. Different stroke techniques, or brush properties express a particular level of abstraction. Feature emphasis or feature suppression is achieved by combining different abstraction levels in illustrative rendering.

Most of such visualization techniques focus on improving the traditional volume rendering style based on non-photorealistic rendering or lighting methods, or exploiting smart visibility in visualization. To effectively convey the most important visual information, however, there are many more aspects and approaches in graphics and other related fields worth considering and exploring.

In this thesis, we present several new feature-driven illustrative visualization and graphics techniques (see Figure 1.1), which are derived from different points of view, including

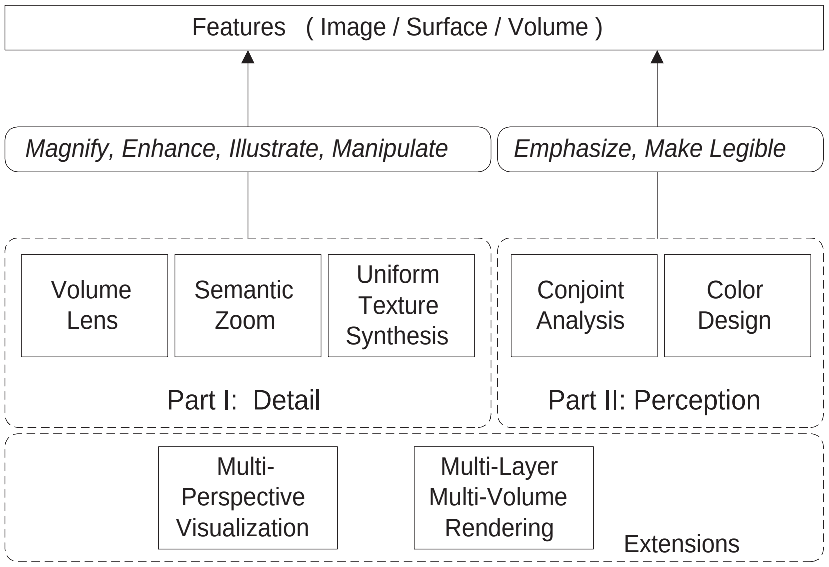

Features (Image / Surface / Volume)
Magnify, Enhance, Illustrate, Manipulate
Emphasize, Make Legible
Volume Lens
Semantic Zoom
Uniform Texture Synthesis
Conjoint Analysis
Color Design
Part I: Detail
Part II: Perception
Multi-Perspective Visualization
Multi-Layer Multi-Volume Rendering
Extensions

Figure 1.1: Our feature-driven illustration techniques.

detail management and perception. All of the techniques proposed here have the same goal, that is, to enhance and better visualize the features of interest in the datasets.

1.2 Contributions

Our major contributions to scientific visualization and graphics research include various feature-driven illustration techniques to enhance the representation of features or details of importance, and new insights and techniques to facilitate better perception of the presented visual information. Our techniques mainly fall into two groups: detail, and perception related techniques. The former includes novel techniques to magnify features of interest in volume datasets by focus+context volume rendering, employing texture synthesis techniques in visualization to generate multi-resolution details for semantic zooming in images and volumes, and creating uniform texture features on the surface based on global conformal parameterization. The latter includes measuring the perceived quality of volume rendering by conjoint analysis, and assigning colors to distinguish or highlight features. Furthermore, we study the feasibility of multiperspective volume rendering for simultaneously showing features, and we propose a multi-layer framework for multi-volume visualization. In summary, our contributions include:

- We have proposed an interactive focus+context volume rendering framework that uses various standard and advanced magnification lens rendering techniques to magnify the features of interest in a volume dataset, while compressing the remaining volume regions without clipping them away completely. Some of these lenses can be interactively configured by the user to specify the desired magnification patterns, while others are feature-adaptive. Our technique allows the user to interactively manage the available screen area, dedicating more area to the more resolution-important features.

- We have presented a method that generates the missing detail from any available and plausible high-resolution datasets, using constrained texture synthesis. The detail generation process is guided by the underlying image or volume data, and is designed to fill in plausible detail in accordance with the coarse structure and properties of the zoomed-in neighborhood. Regular zooms become "semantic zooms", where each level of detail stems from a data source attuned to that resolution. We demonstrate our approach by a medical application, the visualization of a human liver, but its principles readily apply to any scenario, as long as data at all resolutions are available.

- For surface texture synthesis, we have presented two methods for simultaneous scale and angle preservation, based on global conformal parameterization. By using the conformal parameterization, the 3D surface texture synthesis problem can be converted to a 2D image synthesis problem, which is more intuitive, easier, and conceptually simpler. While the conformality of the parameterization naturally preserves the angles of the texture, we provide a multi-scale technique to also maintain a more uniform area scaling factor. Another contribution is to apply a mass-spring method to achieve quasi-isometric parameterization which simultaneously preserves orthogonality and size in texture mapping. Our algorithms are simple, efficient and automatic, and they are theoretically sound and universal to general surfaces as well.

- We have extended our volume lens framework to involve multiperspective volume rendering. Our approach generates a single image combines what can be seen from more than one viewpoints. Although distortions are unavoidable and maybe too much sometime, multiperspective rendering gives a clue of the spatial relationship of features.

- We have been part of a collaboration that demonstrated that conjoint analysis can be a useful and efficient tool to gauge influences of a rich set of rendering parameters on human perception in visualization tasks. Our role in this project was the creation of a large collection of images, suitable for the comparative testing strategy of conjoint analysis and a subsequent statistically valid analysis of the user study results. The framework was demonstrated by a study that measured the perceived quality in volume rendering within the context of large parameter spaces. When generating the pool of 5500 images used in the study, we took great care to reduce the effects of competing adverse parameters, such as image size and occlusion, without reducing the effects of the relevant tested parameters, such as color schemes and rendering mode and precision.

- We have proposed a system which captures the rules explicitly and implicitly formulated in various classic color design books into a color selection framework, providing appropriate colorizations based on user preferences, importance functions, and scene composition. Since our approach incorporates various principles of vision psychology, such as preattention, emotion, and aesthetics, it can provide better control for task-driven or feature-driven visualization tasks. Our rule based system not only assists in the selection of proper colors, it also helps to avoid poor color mixing in semi-transparent rendering and the apparent change in brightness in some color harmonization scenarios.

- We have proposed a preliminary multi-layer multi-volume rendering framework. Employing a Photoshop like multi-layer style, and incorporating GPU-accelerated ray casting rendering, our framework can provide users with a general and efficient tool to explore multiple volume datasets.

1.3 Outline

The overall organization of the thesis is as follows. Chapter 2 gives a brief overview of the related techniques, including volume visualization, illustrative visualization, and example-based texture synthesis. We present our magic volume lens, an interactive focus+context volume rendering technique, in Chapter 3. Chapter 4 follows with our semantic zoom technique, where multi-resolution details in the images and volumes are generated using constrained texture synthesis. In Chapter 5, we describe how to generate uniform textures on the surface by texture synthesis and texture mapping using global conformal parameterization. We study the multiperspective volume visualization in Chapter 6. We demonstrate the conjoint analysis framework by a study that measures the perceived quality in volume rendering in Chapter 7. Chapter 8 presents our work on color design for visualization. Chapter 9 describes our multi-layer visualization framework and multi-volume rendering algorithm. Finally, we draw conclusions in Chapter 10.

Chapter 2

Background

Feature is defined as a prominent or distinctive aspect, quality, or characteristic. A feature can be specified according to a function value, spatial location, local properties, or, in the multi-variate case as a product of function value for each modality. How to show features faithfully and emphasize features of interest efficiently is an important and not trivial task in visualization and graphics.

This chapter starts with the typical volume visualization approaches in Section 2.1, because most of our work deals with the volumetric dataset. Then we give an overview of the state-of-art illustrative visualization techniques in Section 2.2. Since texture synthesis is one of the primary techniques we propose to help generate and enhance features for visualization, we will discuss the most related texture synthesis techniques in Section 2.3.

2.1 Volume Visualization

The central role of visualization is to provide the user with a visual representation of the underlying non-visual data. The goal is to convey properties of the data in an effective and efficient way. A 3D scientific or medical dataset can be represented either by iso-surfaces or by direct volume rendering. Iso-surface extraction algorithms, such as Marching Cubes, extract polygonal surfaces from the dataset with specified density values. These polygonal surfaces, typically triangle meshes, can then be rendered through typical surface rendering techniques or tools in computer graphics. Volume rendering introduced by Drebin et al. [30] is the process of creating a 2D image directly from 3D volumetric data without generating the intermediate geometric primitive representations, hence it is often called direct volume rendering [63]. Our work is mostly related to direct volume rendering.

2.1.1 Volume Rendering

Volume rendering contains image-order, object-order, and domain-based techniques. Image-order volume rendering algorithms use a backward mapping scheme where rays are cast from each pixel in the image plane through the volume data and the grids at discrete locations along their paths are sampled via interpolation to determine the final pixel value. Object-order techniques use a forward mapping scheme where the volume is decomposed into a set of basis elements or basis functions which are individually projected to the screen and assembled into an image. In a domain-based technique the spatial volume data is first transformed into an alternative domain, such as compression, frequency, or wavelet, and then a projection is generated directly from that domain [91]. The major volume rendering algorithms include ray casting (image-order algorithm) [82, 83], splatting (object-order algorithm) [141, 96, 161], shear-warp (a hybrid technique) [75].

X-ray rendering, maximum intensity projection (MIP) and full volume rendering are three basic rendering modes. In ray casting algorithms, these modes differ in how the samples taken along a ray are combined. In X-ray, the interpolated samples are simply summed, giving rise to a typical image obtained in projective diagnostic imaging, while in MIP, only the interpolated sample with the largest value is written to the pixel. In full volume rendering, also called direct volume rendering (DVR), the interpolated samples are further processed to simulate the light transport within a volumetric medium according to one of many possible models. Either back-to-front compositing:

$$C_{dst} = (1-\alpha_{src})C_{dst} + \alpha_{src}C_{src}, \tag{2.1}$$

or front-to-back compositing:

$$C_{dst} = C_{dst} + (1-\alpha_{dst})\alpha_{src}C_{src}, \tag{2.2}$$
$$\alpha_{dst} = \alpha_{dst} + (1-\alpha_{dst})\alpha_{src}, \tag{2.3}$$
$$\tag{2.4}$$

can be used, where C_{dst} and α_{dst} are the composite color and opacity, C_{src} and α_{src} are the color and opacity for the intensity value of the current sampled data point, and usually specified by a transfer function, which is a useful feature classification technique. A transfer function refers to a function that maps the data values directly to optical properties (including R, G, B and Alpha channels). The classical transfer function is a one-dimensional function dealing with scalar data, and has been extended to more dimensions (e.g. the first, second-order derivatives).

The rendering effects are different for these four rendering modes, and a rendering algorithm that merges the different modes into a hybrid image generation model has been proposed [51]. The full volume rendering mode is most widely used since it provides the greatest degree of freedom and better rendering results.

In basic splatting, each object point is first assigned a color and opacity using the shading equation and the transfer functions. Then, each point is splatted into the screen's color and opacity buffers and the result is composited with the present image. There are three types of splatting: composite-only [141], axis-aligned sheet-buffered [141] (to reduce the color bleeding), and image-aligned sheet-buffered splatting [96] (to reduce the popping artifacts).

Volume rendering techniques are very powerful tools to represent real world 3D data.

Beside earlier works mentioned above, there have been so many researches try to improve of the image quality in different ways, or accelerate the rendering both by software or hardware techniques over the years.

2.1.2 GPU-Accelerated Volume Rendering

With the extended programmability of the graphics processing unit (GPU) that has recently become available, combined with the increased performance of PC CPUs, much tasks in computer graphics can be done interactively on personal computers, including volume rendering.

GPU-accelerated volume rendering techniques starts with 2D and 3D texture mapping [15, 109, 50], which properly shifted and composited a set of axis aligned slices. Pre-integrated volume rendering [36] was introduced to cope with under-sampling artifacts. Acceleration techniques, such as early ray termination and empty space skipping [86], and hierarchical acceleration structures [49] were also introduced.

When the render-to-texture capabilities emerged and the intermediate results can be stored onto textures, ray casting on GPU becomes feasible. Krueger and Westermann [72] implemented a ray caster on the GPU. Weiskopf et al. [140] extended this framework to non-linear ray tracing. Both methods explicitly enforced the program flow by rendering control polygons for every major step of the ray casting algorithm, using textures to hold the intermediate computation results. The repeated steps are: advancing the rays, interpolation of samples in the 3D data texture, shading, compositing. Rays that have become opaque could be eliminated (terminated) between steps and empty space could be culled. The addition of loop and branch capabilities into the GPU programming set has enabled the more natural and free-flowing pipeline execution mode used in single-pass ray-casting [118]. Leung et al. [79] further accelerated ray casting by using the PARC (Polygon Assisted Ray Casting) algorithm to ensure that all rays are bounded to the limits of the volume's outermost surface.

Splatting was accelerated on the GPU using early-z culling [100]. A hardware-accelerated adaptive EWA (elliptical weighted average) volume splatting algorithm has also been proposed [20].

2.2 Illustrative Visualization

An Illustration is a visualization such as a drawing, painting, photograph or other work of art that stresses subject more than form. The aim of an illustration is to elucidate or decorate a story, poem or piece of textual information (such as a newspaper article), traditionally by providing a visual representation of something described in the text (Defined by Wikipedia [142]).

Illustration has always been an important visual communication medium among humans. Technical, medical and biological illustrations have accommodated several types of rendering styles, including the line art technique (pen-and-ink technique), the photorealistic drawing style, and a combination of a real photograph and line art. Many approaches, like stippling, hatching, or charcoal shading, allow one to understand the front shape of features by simulating shading. Expressive illustration techniques such as section views, cut-away views, ghosted views, or exploded views and zooming distortions effectively uncover most important information by changing the level of visual abstraction or modifying the spatial arrangement of features. A detailed description of the history of illustration and modern illustration techniques can be found in Viola's thesis [131].

2.2.1 Focus+Context Visualization

Visualization tasks frequently emphasize a particular feature as opposed to the remaining context information. In medical visualization for example features interesting for the diagnosis are shown together with features in their close vicinity. Such visualization strategies are often denoted as focus+context visualization, where focus refers to the most interesting feature and context is the surrounding or less relevant information to provide spatial or other referential relationships. In order to concentrate mostly on the focus information, the context often has to be represented in a sparse way that does not take too much of the image space.

Many techniques have been developed in this area. Zhou et al. [159] devised focus-region based volume rendering for volume feature enhancement. Volume data inside and outside the focus region are rendered in different styles, and the distance to the focal point is further included to control the optical properties of volume features in the context region [158]. Gaze-directed volume rendering [84] takes the observer's viewing focus into account to increase the rendering performance. The volume dataset is rendered at different resolutions, with the focal region represented at full resolution and the other parts at a lower resolution.

Importance-driven volume rendering [133] is a view-dependent model for automatic focus+context volume visualization. The object importance is added as a new dimension to the traditional volume rendering pipeline in order to maximize the visual information. This technique removes or suppresses less important parts of a scene to reveal more important underlying information. The illustrative context-preserving volume rendering model [12, 11] uses a function of shading intensity, gradient magnitude, distance to the eye point, and previously accumulated opacity to selectively reduce the opacity in less important data regions. The method keeps easy and intuitive user control without missing context information.

2.2.2 Cut-Away Views and Deformations

The visibility of prominent features can be achieved by illustrative visualization techniques such as cut-away views or ghosted views. A different smart way to provide information on the data is using exploded views or other types of deformation.

Cut-Away viewing, also known as volume cutting [105], is another way to display volumetric objects. Various cut-away techniques can be achieved automatically [28], and many improvements have been made. Tory et al. [127] provided a framework, called ExoVis, for simultaneously viewing detail and context in volumetric data sets. It allows users to view multiple slices of a volume at arbitrary orientations, along with multiple subvolumes rendered in different styles. All slices and subvolumes are outside or surrounding a 3D overview of the dataset.

Instead of disposing cut-away volume parts, Kurzion [73] presented a method to deform the shape of volume models and reveal the inside features without any modifications to the original model. The method operates in the rendering phase, thus providing the illusion that he modified the model. It equals to changing ray directions, however, the implementation is not straightforward since the texture-based hardware acceleration method is utilized. McGuffin et al. [93] used deformations for browsing volumetric data. Volume splitting technique [44] is intended for displaying multiple enclosed iso-surfaces within the volumetric data. Each iso-surface, except the innermost one, is split into two parts and moved apart.

2.2.3 Lenses and Distortions

Lenses in the real world can be quite complicated [70]. However, simple lenses and magnifications are still very useful and have been thoroughly studied for text, image and information visualizations [80, 64, 65]. Bier et al. [8] introduced Toolglass and Magic Lenses as a see-through interface to modify the visual appearance of application objects, enhance data of interest or suppress distracting information. Viewpoint-dependent distortion of 3D data, see [18, 19] for example, highlights regions of interest by dedicating more space to them.

On the other hand, relatively little work has been done on lenses in the domain of volume visualization. Cignoni et al. [21] provided the Magicsphere metaphor to visualize 3D data with a MultiRes filter. Wei et al. [139] applied fisheye views to magnify particle track volume data using nonlinear magnification functions.

LaMar et al. [77] integrated a 3D magnification lens with a hardware-texture based volume renderer. Zooming is accomplished by modifying texture coordinates, and the 2D perspective correct textures technique is extended to 3D in order to obtain the correct texture coordinates for the lens border. Multiple segments on the border are needed to generate more natural circular lenses. Cohen and Brodlie [22] magnified volume data by generating a new volume using inverse distortion functions, however, this method is slow and is memory-intensive. Further research is clearly needed to design better lenses and find efficient implementations for volume data.

2.2.4 Illustration-Inspired Volume Visualization

Non-Photorealistic and illustration-inspired rendering styles have been applied to effectively visualize the scientific and medical volume datasets. Treavett et al. [128] used pen-and-ink styles in combination with direct volume rendering or surface shaded display. The sparse pen-and-ink representation is applied to outer iso-surfaces while an inner iso-surface is represented using surface shading. Volume rendering is further enhanced by non-photorealistic techniques, such as silhouettes [111]. Lu et al. [90] proposed the stippling rendering of the scientific data. Illustration-inspired techniques also benefit the visualization and understanding of the flow volume [121], and the motion of the flow [61].

2.3 Texture Synthesis

Textures can be random, stochastic, structure or anything in between. Although there has been much sophisticated research on texture analysis, recognition, and synthesis in computer vision, no real satisfactory solution has been achieved so far, due to the great varieties of textures. Here, we only focus on the most important image texture synthesis approaches which have appeared in the graphics area, and which are relatively simple and easy to use.

Texture synthesis algorithms take sample images as input and synthesize new images with similar textures. These algorithms can be roughly classified into three categories: statistical, pixel-based and patch-based texture synthesis. Most approaches, especially statistical and pixel-based approaches, are based on Markov Random Field (MRF) models. Also, most pixel- or patch-based approaches need a distance metric to measure the perceptual difference between two pixels or two image patches. Hence we will first discuss MRFs and the typical distance metric used in texture synthesis. Then we list some important image texture synthesis approaches, and describe several applications and extensions of image texture synthesis technique.

Markov Random Field Markov Random Field (MRF) models (or in a different mathematical form, Gibbs Sampling) have been widely used in texture synthesis, image restoration, and region segmentation. Since MRFs have been proven to be a good approximation for a broad range of textures, the texture synthesis algorithms based on MRFs [35, 137] are general and some of them produce good results. A drawback of basic MRF sampling is that it is computationally expensive.

The property of a MRF is that: a variable X_s at site s on a lattice $S = \{s = (i,j) : 0 \leq i,j < M\}$ may have its value x_s set to any value, but the probability of $X_s = x_s$ depends upon the values x_r at sites neighboring s. The neighboring sites are defined as those sites $r \in N_s \subset S$, where N_s represents the neighborhood of s. A local conditional probability density function (LCPDF) defined over these neighboring sites $r \in N_s$ determines the probability of $X_s = x_s$. Therefore the MRF is defined by the LCPDF [103]:

$$P(X_s = x_s | X_r = x_r, r \neq s) = P(x_s | x_r, r \in N_s) \; s \in S. \tag{2.5}$$

This means the probability distribution of color/intensity values for a pixel given the color/intensity values of its spatial neighborhood is independent of the rest of the image. The neighborhood of a pixel can be modeled as a square window around that pixel. Most MRF synthesis approaches are based on finding similar neighborhoods.

Distance Metric Most pixel- or patch-based approaches use the L_2 norm to measure the distance between two neighborhoods of textures. For the neighborhoods of two pixels, the basic distance metric usually is the sum of squared differences (SSD) as the following equation:

$$d_{SSD} = \sum_{i,j=1}^{n} \sum_{k=1}^{c} (N_{1_k}(i,j) - N_{2_k}(i,j))^2, \tag{2.6}$$

where N_1, N_2 are two neighborhoods, n is the neighborhood size, (i,j) represents the pixel location in the neighborhood, and c is the number of the channel, e.g. 1 channel (intensity) for a gray texture and 3 channels (red, green, blue) for a color texture image. If we say two neighborhoods match, their d_{SSD} should be the minimum or below a user specified threshold.

Measuring the neighborhood distance for each pixel or patch can be time consuming. There are some algorithms that try to speed up the computation [74]. The above equation can be expanded as follows:

$$d_{SSD} = \sum_{i,j=1}^{n} \sum_{k=1}^{c} (N_{1_k}^2(i,j) + N_{2_k}^2(i,j) - 2N_{1_k}(i,j)N_{2_k}(i,j)). \tag{2.7}$$

Using summed area tables (SAT) [25], the computation of the first two terms can be accelerated, while the third correlation term can be calculated as a convolution and be speeded up by Fast Fourier Transform (FFT) techniques [66, 42].

Actually, the L_2 norm is a poor measure for perceptual similarity, hence the minimum d_{SSD} does not yield the most similar appearance. However, most approaches still use it due to its simplicity and the fact that no perfect perceptual metric has been found. Some methods try to improve the basic d_{SSD} by using weighted d_{SSD} [35], or using other color spaces such as YIQ and only the luminance channel Y to reduce the computational cost [53, 3].

2.3.1 Approaches

Statistical Texture Synthesis Statistical texture synthesis approaches first analyze the input texture using certain statistic measures and then synthesize the output according to the analysis results. The main advantages of statistical analysis are that they provide a better understanding of the perceptual process, provide a better model generalization and generate good results for stochastic textures. However, they can not synthesize as large a variety of textures as other techniques and are relatively complicated and computationally slow.

Pixel-Based Texture Synthesis Pixel-based synthesis algorithms synthesize textures pixel by pixel by finding a matching neighborhood, which makes them rather flexible and easy to extend and apply to different areas. The representative algorithms include:

Efros/Leung's non-parametric sampling algorithm [35], Wei/Levoy's multiresolution synthesis algorithm [137], which performs exhaustive search and accelerates based on tree structured vector quantization (TSVQ). Ashikhman's coherent synthesis [2] and Tong's k-coherent synthesis [125] algorithms reduce the search space significantly. Their synthesis process is faster, but only suits particular types of textures well. Hertzmann's Image Analogies algorithm [53] combines [137] and [2], uses PCA, and approximates nearest neighbor search (ANN) to accelerate the search process, offering better results. Zelinka and Garland [157] synthesize textures in real-time using a Jump Map, after a relatively slow analysis process. However, many pixel-based approaches suffer from image blurring and garbage growing.

Patch-Based Texture Synthesis Patch-based synthesis algorithms tile matched patches together, tend to be faster and more stable, and do not suffer from blurring and garbage growing. Although they are less flexible since they generate textures by copying whole patches from the input, and hardly provide any perceptual information of the input texture, they are quite efficient and generate good synthesis results. The two major issues in patch-based approaches are how to choose the appropriate patch and how to eliminate the boundary artifacts. Xu's chaos mosaic [148], Efros/Freeman's image quilting [34], Liang's [87], Kwatra's Graphcut [74] and Cohen's Wang tiles [23] algorithms all belong to this category. Hybrid methods [99] lay out patches and use pixel-based algorithm to hide the seams.

Our work is mostly related to pixel-based and patch-based texture synthesis techniques.

2.3.2 Applications and Extensions

The digital image processing area benefits from image texture synthesis techniques, which have been applied to address many image processing problems, such as texture transfer [34, 53, 2], artistic style simulation [53, 3], super-resolution [53, 39], image restoration, and specific image editing, with sometimes impressive results.

Image texture synthesis techniques also have some exciting extensions, such as synthesizing surface texture [129, 136], temporal textures [114, 137, 74], reflectance texture [125], volumetric or solid textures [136, 57], varied and mixed textures and enhancing vector fields with textures.

Chapter 3

Magic Volume Lens

The size and resolution of volume datasets in science and medicine are increasing at a rate much greater than the resolution of the screens used to view them. This limits the amount of data that can be viewed simultaneously, potentially leading to a loss of overall context of the data when the user views or zooms into a particular area of interest. We propose a focus+context framework that uses various standard and advanced magnification lens rendering techniques to magnify the features of interest, while compressing the remaining volume regions without clipping them away completely.

Interactive operability is the prime key to a successful user experience and his/her exploration and immersion in the data, and the GPU has provided an attractive platform to achieve these goals. Our work embraces this technology to provide a novel focus+context tool that unifies and extends a variety of existing methods in this area. Our techniques are primarily designed for volumetric objects, which have received the least amount of attention so far. Our framework provides a free-form volumetric lens function that can be feature-adaptive or user-configurable for a high-quality, anti-aliased, and interactive display with smooth transitions from high- to low-resolution areas. It is somewhat related to the importance-driven visualization system by Viola et al. [133], but our method allows users not only to highlight and expose an object, but also to non-linearly magnify the object for closer inspection in its spatial and semantic context.

3.1 Volumetric Lenses

In this section we describe several volumetric lenses which are based on geometric optics and conform to sampling theory.

3.1.1 Magnifier

The magnification lens, called magnifier in this thesis, is based on the magnification model in optical physics. It provides users a method for close inspection of regions of interest in volumetric objects. Figure 3.1 illustrates the principle of a magnifier. The blue line segment represents a magnifier lens positioned on the image plane by the user. LC is the center point of the lens and F is the virtual focal point. When orthogonal incident rays hit the image plane, in the region of the magnifier, then the ray directions are modified and go through the focal point F. Therefore, a ray cone is formed between the lens and F. The objects within this cone are rendered in a larger area on the image plane than their original size, while the other objects retain their original size. Consequently, the objects in the region of interest are magnified.

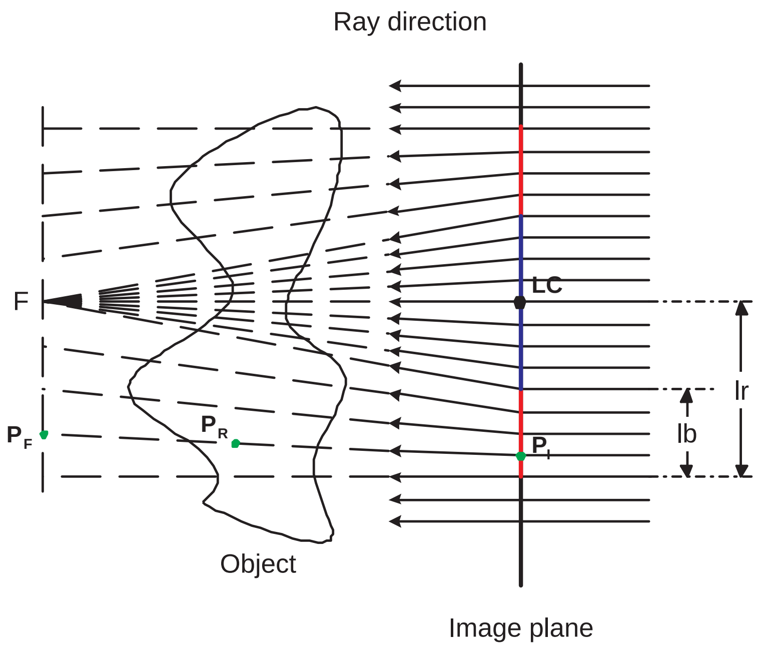


Figure 3.1: Magnifier illustration.

In the basic scenario described above, objects located between the orthogonal rays and the focused rays will not be visible on the image plane. This causes a loss of spatial context for the observed objects and has to be compensated for by special treatments. Our solution is to add a transition region close to the border of the ray cone where the directions of rays are gradually changed from the focused direction to the orthogonal direction. In Figure 3.1, the transition region is represented by the red line segments on the image plane with a width lb, lr is the radius of the lens, and the magnification region of the lens is shown as the blue line segment. For a ray starting from a point P_I in the transition region, the direction is computed according to the distance from P_I to LC as follows:

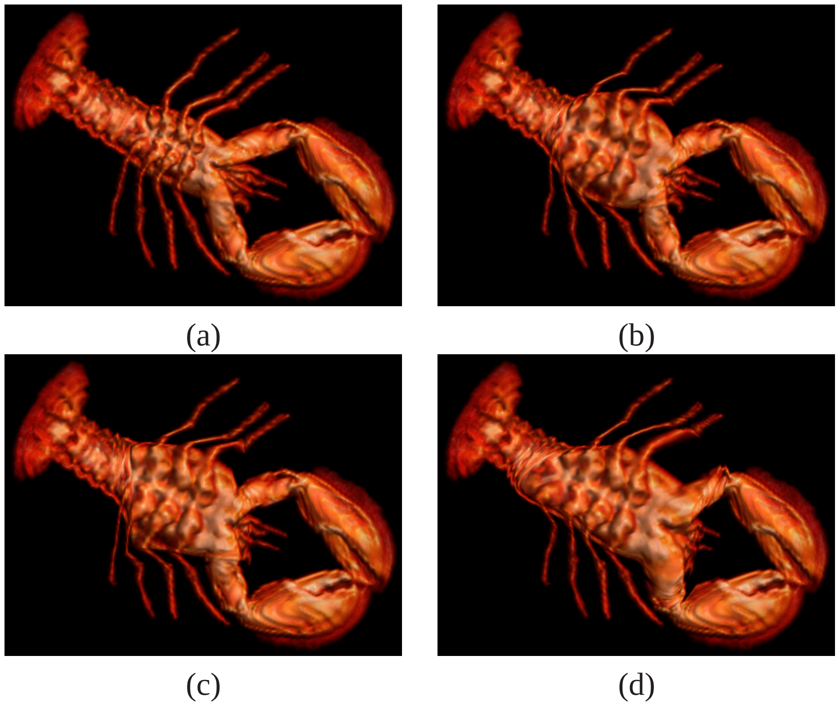


Figure 3.2: Magnifier volume renderings with (a) No lens, (b) Circular lens, (c) Square lens, (d) Arbitrary-shaped lens.

$$\begin{aligned} \frac{|P_F - F|}{lr} &= \frac{|P_I - LC| - (lr - lb)}{lb}, & (3.1) \\ P_F &= F + \frac{P_I - LC}{|P_I - LC|} \cdot |P_F - F|, & (3.2) \\ ray\_dir &= P_F - P_I. & (3.3) \end{aligned}$$

where P_F is the point at which this ray passes through the virtual lens focus plane, which is parallel to the image plane and includes the focal point F.

As a result of the transition region approach, while the objects inside the center region of the lens are magnified, the objects in the transition region are compressed. Therefore, continuous observation of the objects is achieved and no artificial data loss is introduced.

Based on this method, we are able to design magnifiers with any arbitrary shape. Results obtained by using magnifiers in volume rendering are shown in Figure 3.2. Figure 3.2a is the original volume rendering result with no magnifier and Figure 3.2b-d are the results obtained by using circular magnifier, square magnifier and arbitrary-shaped magnifier, respectively. Figure 3.3 shows the transition regions, magnification regions of three magnifiers, and the rendering effects on enlarged portions of Figure 3.2b-d.

The magnification factor can be changed by modifying the focal point position. Moving virtual focal point F towards the image plane achieves a higher magnification factor and

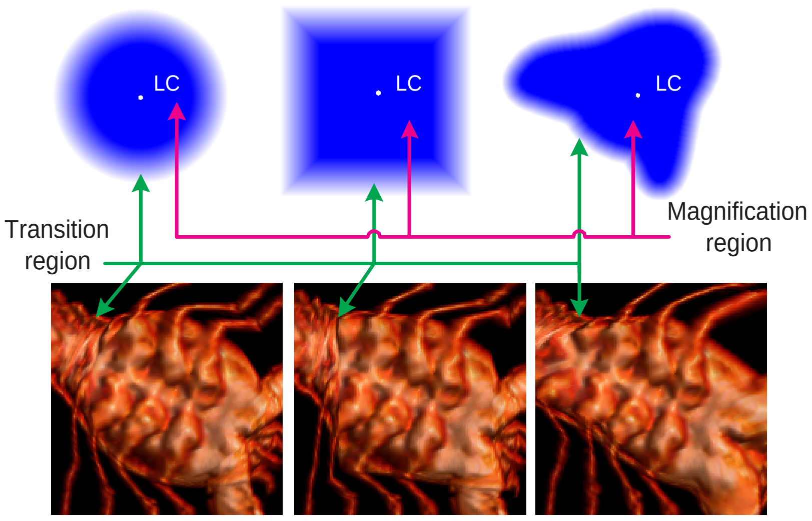


Figure 3.3: Transition region and its rendering effect.

vice versa. The GPU acceleration makes it possible for users to choose this interactively. At the same time, the users can also change the size of the magnifier, for example, the radius of a circular lens, and the size of the transition region to generate the desired results.

Our volumetric lenses are based on ray casting and it can be easily detected whether a ray pass the feature, therefore the magnifier can be utilized to enlarge only features of interest in the observed volumetric object. The magnification method is straightforwardly applied to the segmented volumetric datasets. The ray modification method does not interfere with the composition of the voxels with different properties because of their segmentation. Figure 3.4 shows the results of applying the magnifier to show the bone features of a segmented frog dataset.

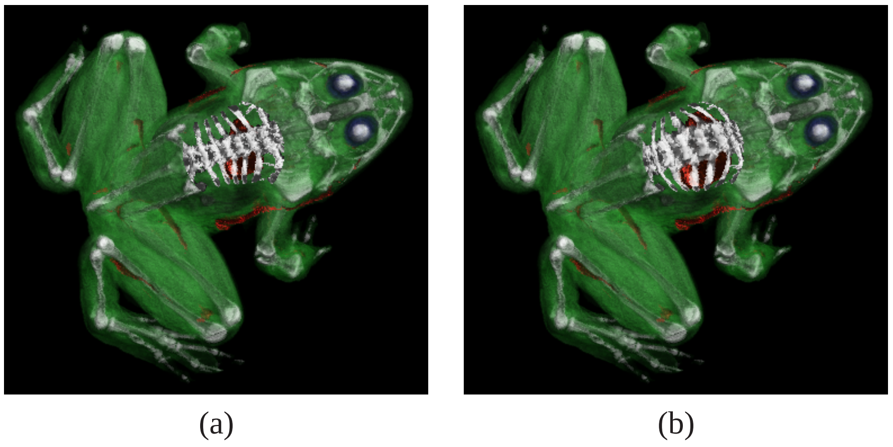


Figure 3.4: Magnifier volume renderings for the bone feature in a segmented frog dataset. (a) and (b) Renderings without and with magnification under circular lens.

Since in the transition region, the ray sampling rate is relatively low, aliasing could occur. Although this is not always noticeable in practice, anti-aliasing techniques can be applied to generate better results. A solution is to use volume texture mip-mapping to adaptively choose the appropriate resolution of the volume data for rendering. A lower resolution volume is chosen for regions sampled at a lower rate, in order to eliminate aliasing. One can determine the required mip-map level by calculating the magnification factor mf for point P_R,

$$mf = \frac{|P_R - P_{RI}|}{|F - LC|}(\frac{lb}{lr} - 1) + 1. \tag{3.4}$$

where P_{RI} is the orthogonal projection of P_R on the image plane. This factor will determine the mip-map level that needs to be used.

3.1.2 Feature-Based Lens

Feature-driven volume visualization provides users a highlighting and exposition of the portions of interest in volume objects. This facilitates an accurate and differentiated understanding of the important features. Besides the traditional fixed-shaped lens used to magnify segmented datasets, our free-form magnifier can be employed to also achieve a feature-sensitive and feature-centric object enlargement. The difference is that the shape of the magnifier is defined dynamically by the shape of the features (represented by the segmentation information) in the dataset, within an arbitrary view port. This is illustrated in Figure 3.5. Whether an incident ray changes direction depends on the distribution of the feature and the current view port. Thus the direction of each ray has to be determined dynamically. Transition regions are also used here to retain the space context of the features.

For each ray orthogonally incident upon the image plane, the new direction is computed as follows. Assuming all rays have changed directions to the focal point F,

- if a ray passes through the feature, then its new direction is pointing to F.
- if the ray does not pass through the feature but is inside the transition region on the image plane, the distance d (see Figure 3.5) from its entry point to the boundary of the feature-projected area is calculated. This distance is used to compute the new direction as in Equations 3.1-3.3.
- otherwise, the ray continues along its original direction.

On the image plane, the distance from a pixel to the boundary of the feature-projected area has to be calculated for some rays. This requires knowledge of the position of such an area on the image plane in each different view port. Therefore, a two pass computation has to be used, where the first pass defines the feature-projected region and the second pass computes the distance from a pixel to this region. Different distance computation methods can be used during the second pass. To facilitate the GPU acceleration for this algorithm, it has to be implemented based on local operations where each pixel only utilizes the knowledge of its neighborhood. Our implementation is to use a searching circle for

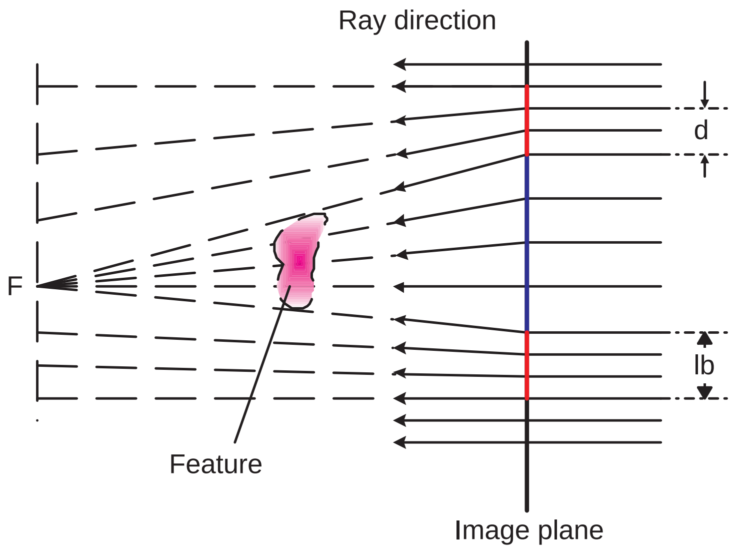
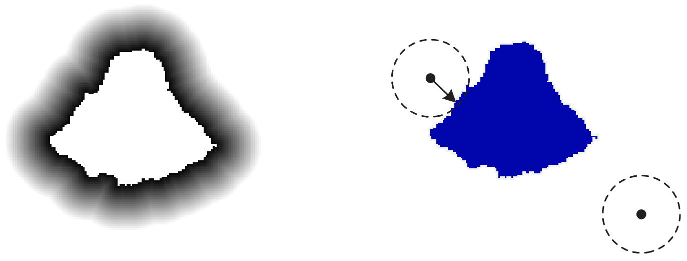


Figure 3.5: Feature-based lens illustration.



(a) (b)

Figure 3.6: Distance computation on the transition region of the feature-based lens. (a) Distance field on the transition region, (b) Searching circle for each pixel outside the feature-projected region is used for local computation.

each pixel with the transition region width lb as its maximal radius (see Figure 3.6 for an illustration). Inside this circle, we compute a neighbor that is a feature projected point and has the smallest distance to the pixel. This smallest distance is used as the distance value for this pixel. This method is implemented directly as a fragment program on GPU (see Section 3.2).

Our lens can be combined with any feature-based ray casting volume rendering method, for example, the two level volume rendering technique [51] for segmented volume data. Figure 3.7 shows some rendering results for a color volume dataset, in which a user selected feature is magnified and the other objects near that feature are compressed. Figure 3.7a shows the skin of the brain. Figure 3.7b shows an interior structure of the brain, without

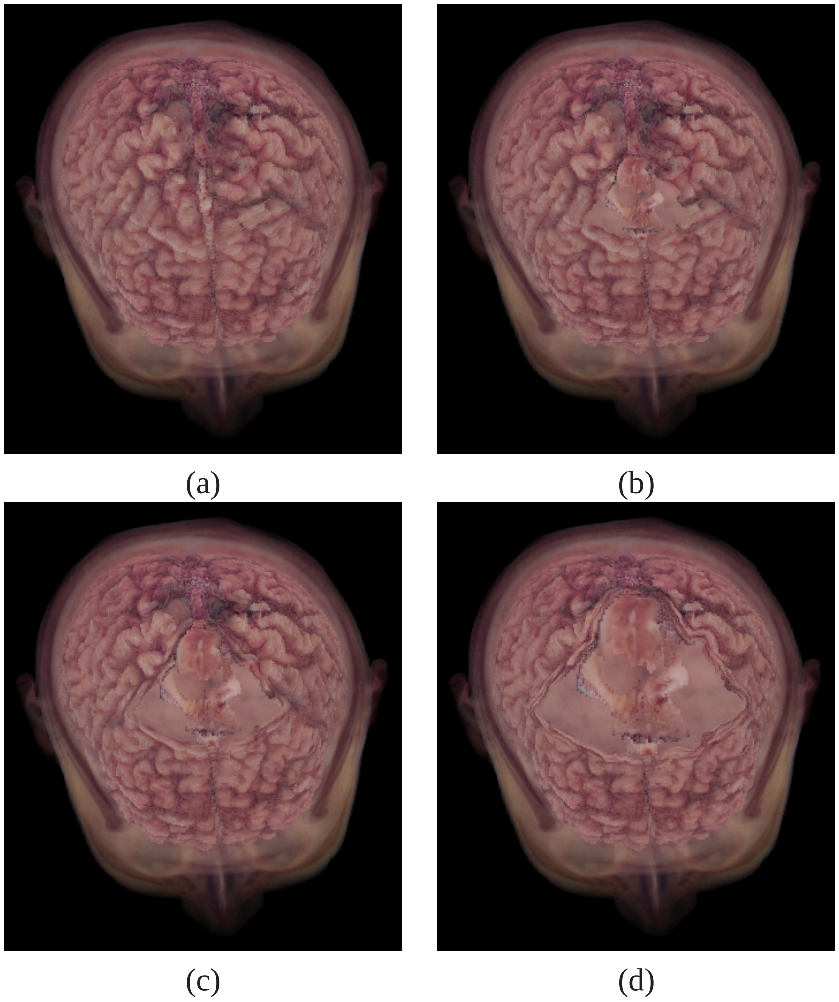


Figure 3.7: Feature-based lens volume renderings for a segmented human brain color volume dataset. (a) Without specifying any feature of interest, (b) With a feature of interest, which is not magnified and appears too small to be seen clearly. From (c) to (d) the magnification factor increases.

rendering other features which occlude this structure, while the magnified structures are shown in Figure 3.7c and 3.7d.

3.1.3 Sampling-Rate-Based Lens

We introduced two magnification lenses that modify the casted rays using geometric optics. They are implemented directly by changing ray directions from different areas of the image plane. The distribution of the areas can be user-defined or feature-based. In this section, we define a lens from another point of view. The rays casted towards the observed object may have varying densities in different portions of the object. This results in a varying sampling rate for the object. Therefore, this special lens is called *sampling-rate-based lens*. Various sampling functions could be adopted to define various volumetric lenses and to achieve different volume rendering results. we can use these lenses in conjunction with the mip-map volumes discussed in Section 3.1.1.

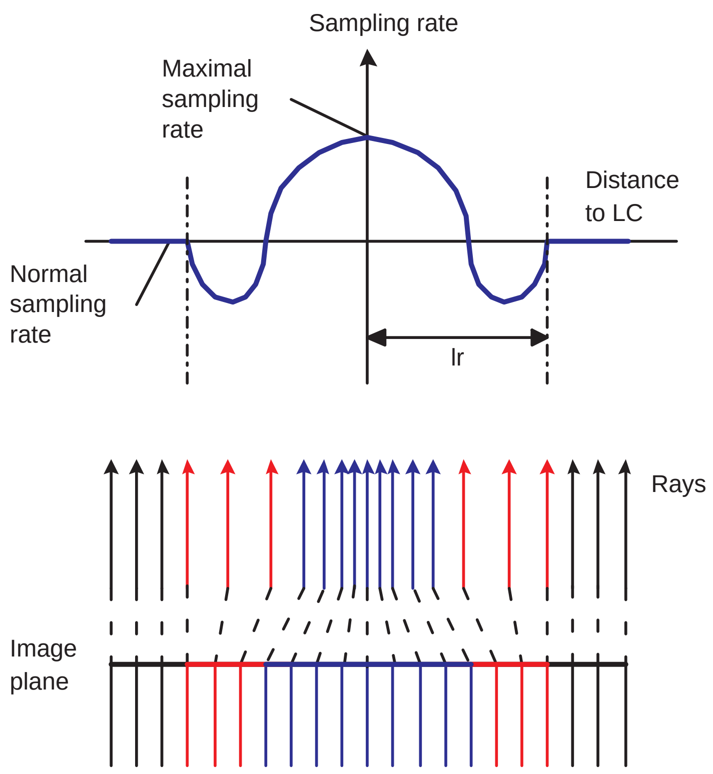


Figure 3.8: Sampling-rate-based lens illustration.

We illustrate the idea of this lens in 1D in Figure 3.8, where a sampling rate function is shown at the top and the corresponding rays are shown at the bottom. In the sampling rate function, lr is the lens radius, the vertical axis is the sampling rate and the horizontal axis represents the distance to the lens center. The sampling rate close to the lens center is the highest. It then decreases and becomes even smaller than the original normal sampling rate towards the boundary of the lens. At the bottom of Figure 3.8, we can see that the rays shot to the object are dense in the center region of the lens and become coarser towards the boundary. Note that the distribution of pixels on the image screen is uniform and that the original orthogonal rays are also distributed uniformly. To distribute the rays according to the sampling rate function, the start point of a ray is not from its original starting pixel but depends on its distance to the lens center and the sampling rate. Thus, we need to compute the correct start point for each ray. As usual, the transition region approach is applied to this lens. Here, the magnification region plus transition region must be exactly equal to the lens region, which means the distance from the cutting point (where sampling rate returns to normal) to the lens center must be equal to the radius of the lens, lr. Define sr as the

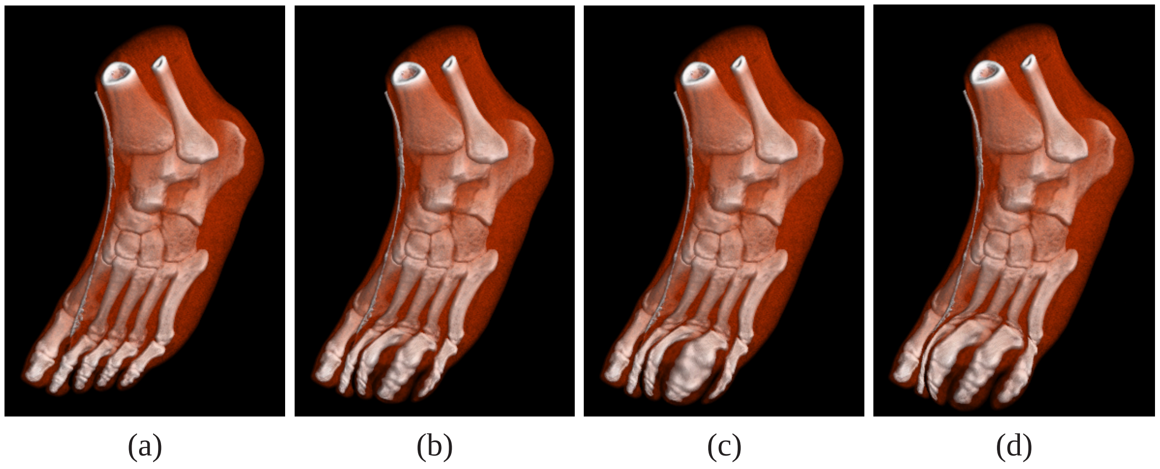


Figure 3.9: Comparing volume renderings with (a) No lens, (b) Magnifier, (c) Sampling-rate-based lenses with cubic sampling function (maximal sampling rate/normal sampling rate = 3), (d) An arbitrary sampling function (shown in Figure 3.10).

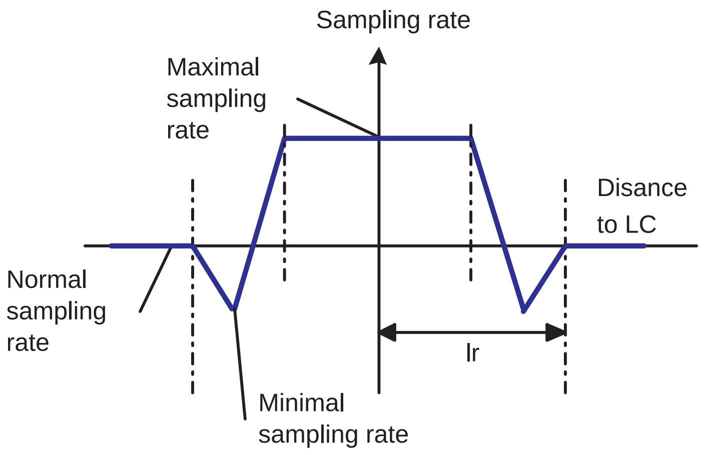


Figure 3.10: Another sampling rate function.

sampling rate and sd as the sampling distance function. Here, sr is inversely proportional to the distance between sampling rays. We first precompute a coefficient C satisfying the integral equations:

$$\int_0^{lr} C \cdot sd(s)ds = lr, \tag{3.5}$$

$$sd(s) = \frac{1}{sr}. \tag{3.6}$$

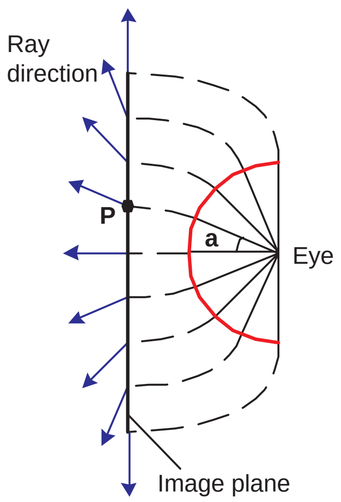
Then for each ray j, the distance between its real start point and the lens center can be calculated using Equation 3.7, which is the discrete form of the distance integral.

$$distance(j) = \sum_{i=0}^{steps} C \cdot sd(i). \tag{3.7}$$

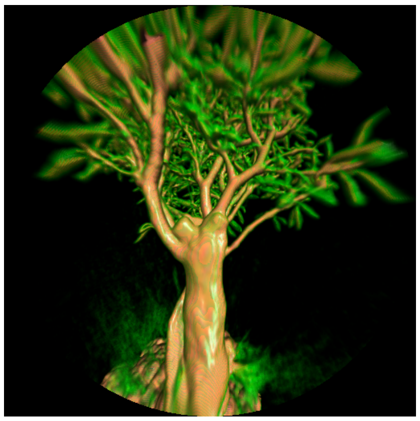
Figure 3.9 shows the results with the sampling-rate-based lenses, comparing it with the results obtained with no lens and with the magnifier. The toes of the foot are rendered with different magnification effects. The difference between Figure 3.9b and 3.9c is mainly caused by the different magnification factor distributions on the lenses. For the magnifier, the factors for points, which project into the magnification region and locate on the same plane parallel to the image plane, are the same. Therefore, objects with the same depth are magnified uniformly. However, for the lens with cubic sampling function, the factor is the highest on the lens center and decreases gradually towards the lens boundary. Objects with projections closer to the lens center are magnified with higher magnification factors. Along any ray, the factor remains the same for different depthes.

3.1.4 Angular Lens

A common widely used lens is the fisheye lens [10], and our GPU accelerated general volumetric lens framework supports this type of lens as well. The fisheye lens is a specially designed lens which achieves wider viewing angles. The original fisheye lenses were photometric lenses designed to take photos of the entire sky. There are two main idealized



(a)



(b)

Figure 3.11: Angular lens. (a) Angular fisheye lens with 180 degrees illustration, (b) 180 degrees view of a bonsai with an angular fisheye lens.

fisheye projections, the hemispherical and the angular fisheye, which are common in computer graphics rendering [10]. The hemispherical fisheye is less used due to the distortion introduced. An angular fisheye projection can be used for angles up to 360 degrees and is defined such that the distance from the pixel P to the center of the image is proportional to the angle α of the viewing direction (see Figure 3.11a). The ray direction corresponding to any pixel on the image can be calculated by a special transform from pixel coordinates to 3D polar coordinates [10]. Figure 3.11b shows an image of a 180 degrees view on a bonsai with an angular lens.

Our framework is based on a ray casting volume rendering scheme. This allows us to walk into the interior of the object to see the augmented volume rendering results. By using an angular lens, larger view port angles can be achieved and more objects can be accommodated in the final image. This is helpful in many interior volume rendering scenarios. A good example is virtual colonoscopy [54]. When navigating inside the colon, more areas can be viewed to achieve a more efficient observation. Figure 3.12 shows the result of viewing a colon from a point on the centerline of the colon. Comparing this with a normal perspective view with 120 degrees, more information can be obtained when using a 180 degrees angular lens.

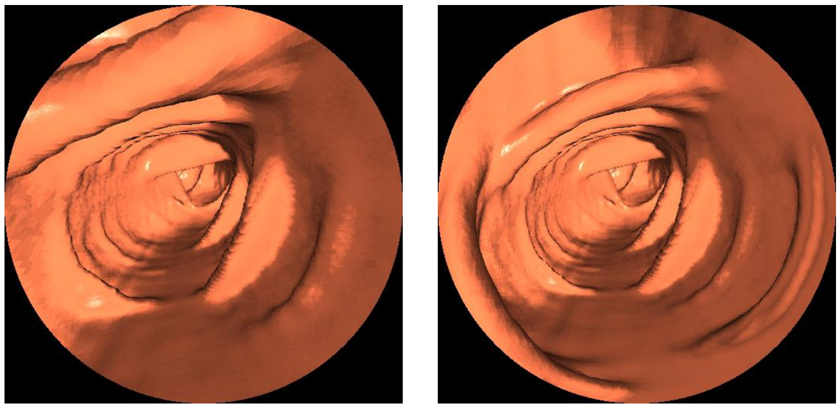


Figure 3.12: Virtual tour of the colon. (a) Perspective view with angle 120 degrees, (b) 180 degrees view with an angular fisheye lens.

3.2 Hardware Acceleration

To achieve interactive focus+context volume rendering, we have implemented all of our volumetric lenses on contemporary graphic hardware. Since our volume lenses are designed based on changes in ray direction or ray sampling rate, it is straightforward to implement, as well as extend, them using a ray casting approach.

In GPU-accelerated ray casting volume rendering [72], front faces and back faces of the volume bounding box are drawn using OpenGL in two fragment passes to get the start and end points for all the rays. However, this approach can not be used for our volumetric lens. Because ray directions are not always orthogonal or perspective, we have to calculate the start and end points of each ray for the various lens algorithms described earlier. Hence, we implemented our own ray casting rendering algorithms with lens effects on the GPU. At first, we calculate the ray directions using the appropriate lens rules. Then, the intersection points of each ray with the bounding box of the volumetric object are computed. Finally, a ray traversal algorithm is implemented for a given step size, with the volume data (density, gradient or color) stored in 3D textures. All these algorithms are translated into Cg fragment programs. The current GPUs (e.g., NVIDIA GeForce 6800) have the required features, such as loop, early termination and branches, making it possible to implement our ray traversal method efficiently.

For our magnifier and angular lenses, we use fragment programs as follows:

Pass 1 *Raycasting* The whole ray casting process includes the following three steps:

- **Step 1** *RayDirection* Calculate the ray direction for each fragment based on the view port and lens parameters. Also the information about whether a ray goes through the lens or hits the feature of interest, or the distance to the lens center can be obtained to achieve different rendering effects.
- **Step 2** *RayTfrontback* Compute the intersections of each ray with the volume bounding box, and store the distances from the front and back intersection points to the ray start point, denoted as *t_front* and *t_back*, which will be used along with the ray direction and view port parameters to define the intersection points in the next step.
- **Step 3** *CastingRay* Cast the ray into the volume and composite the color based on the volume data and transfer function. Different traditional volume rendering modes can be easily added into this step.

Pass 2 *Rendering* Output the rendering results to the frame buffer.

For the feature-based lens, in Pass 1, one more step called Step 1.5: *RayLensBorder*, is added before Step 2, to calculate the distance field for the lens transition region and change the ray directions based on the distance.

For sampling-rate-based lenses, ray directions are never changed, but the real ray start points need to be computed. We also use the above fragment programs, but the first step is changed to Step 1^*: *RayStartPoints*, which computes the ray start points used in later computation.

3.3 Results

We have implemented our methods on a Pentium Xeon 2.4GHz CPU with 2.5GB memory and an NVIDIA GeForce 6800 Ultra GPU with 256MB memory. In Table 3.1, we report the data size and the performance of our method with GPU-accelerated computation.

Table 3.1: GPU performance for different volume datasets.

Data	Data size	Volume lens method		Simple ray casting	
		Rendering speed (ms)	Frames/second	Rendering speed (ms)	Frames/second
lobster	$128 \times 128 \times 128$	70	14.2	61	16.4
engine	$256 \times 256 \times 110$	95	10.5	74	13.6
bonsai	$256 \times 256 \times 128$	110	9	95	10.5
foot	$154 \times 263 \times 222$	97	10.3	90	11.1
aneurism	$256 \times 256 \times 256$	186	5.4	158	6.3
frog	$502 \times 472 \times 138$	308	3.3	258	3.9

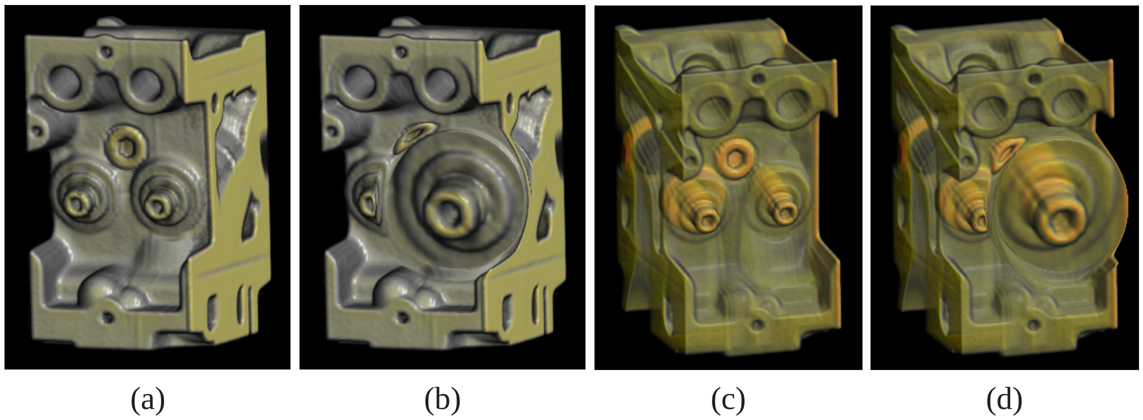


Figure 3.13: Magnification results. (a) and (b) DVR renderings without and with magnifier, (c) and (d) DVR with gradient magnitude modulation renderings without and with magnifier.

For comparison, we also include the performance of a simple ray casting volume renderer (utilizing the front faces and back faces) with the same data sets on the same GPU. All the performances are tested with 512×512 images and with a 1.0 step size. Note that our method has not been optimized for the GPU, therefore, we compare it with the simple ray casting implementation, which is also unoptimized. Our volume lens methods only slightly increase the rendering time comparing to the general ray casting method. In the future, we will implement the standard optimization methods, such as empty space skipping to improve the performance. For example, the speed for aneurism data can be dramatically accelerated with space skipping.

As a ray casting based augmentation for volume rendering, our volumetric lenses can be combined with many volume rendering modes, for example, direct volume rendering (DVR), MIP and DVR with no shading, DVR with gradient magnitude modulation, XRay and the two level volume rendering method for segmented data. We show results with different rendering methods in Figure 3.13. More magnification results are shown in Figure 3.14, Figure 3.15, and Figure 3.16.

Our lenses can be used to interactively choose and magnify regions or features of interest to see small details more clearly while the context region remains. The size and shape of the lenses, and the magnification factor also can be changed interactively, which allows the user to adjust the lenses for desired results. Demo videos that show the interactive volume lens renderings can be obtained at http://www.cs.sunysb.edu/∼lujin/paper/vis05.

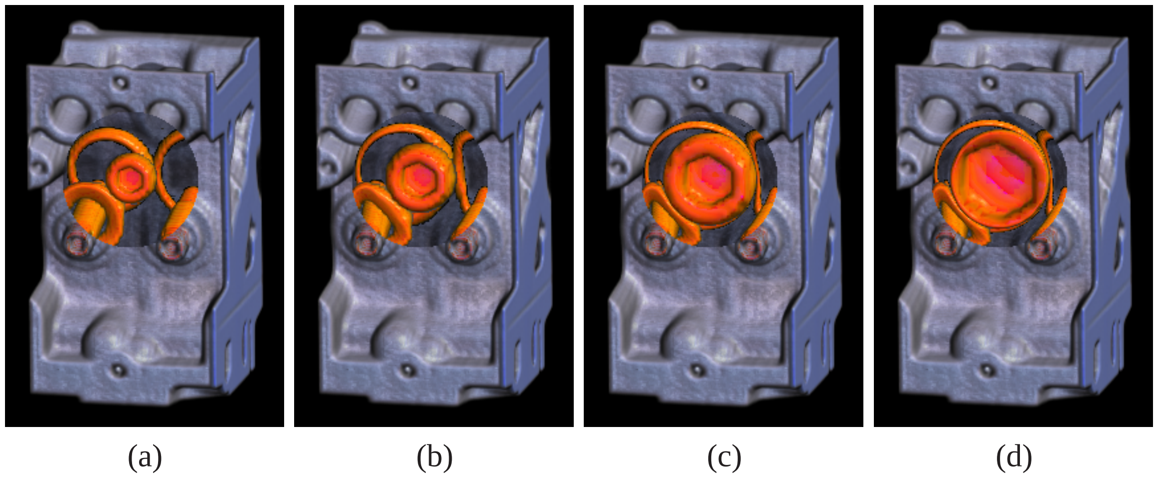


Figure 3.14: Feature magnification results with magnification factor increasing from (a) to (d).

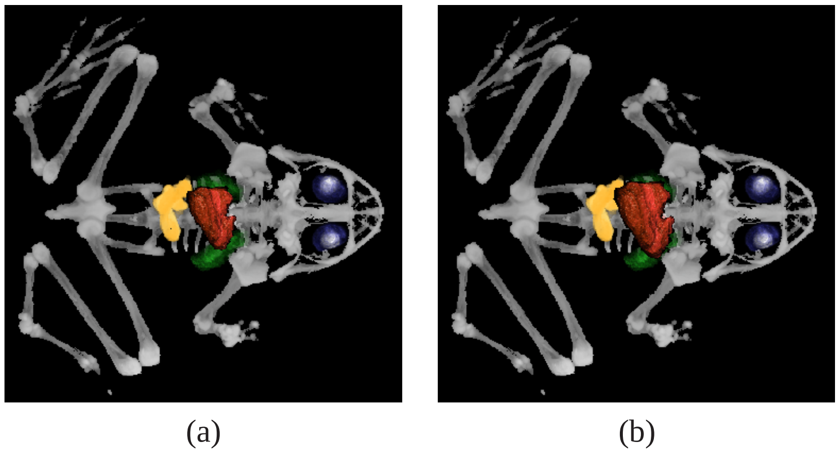


Figure 3.15: Feature-based lens results. From (a) to (b) Frog heart is magnified. Frog is rendered with two level volume rendering method: bone and eye retia are rendered with MIP, all other features are rendered with DVR, with different transfer function for each feature.

3.4 Discussion

We have described a universal and general volumetric lens framework that has applications in many domains. It allows users to apply any well known lenses, such as a fisheye

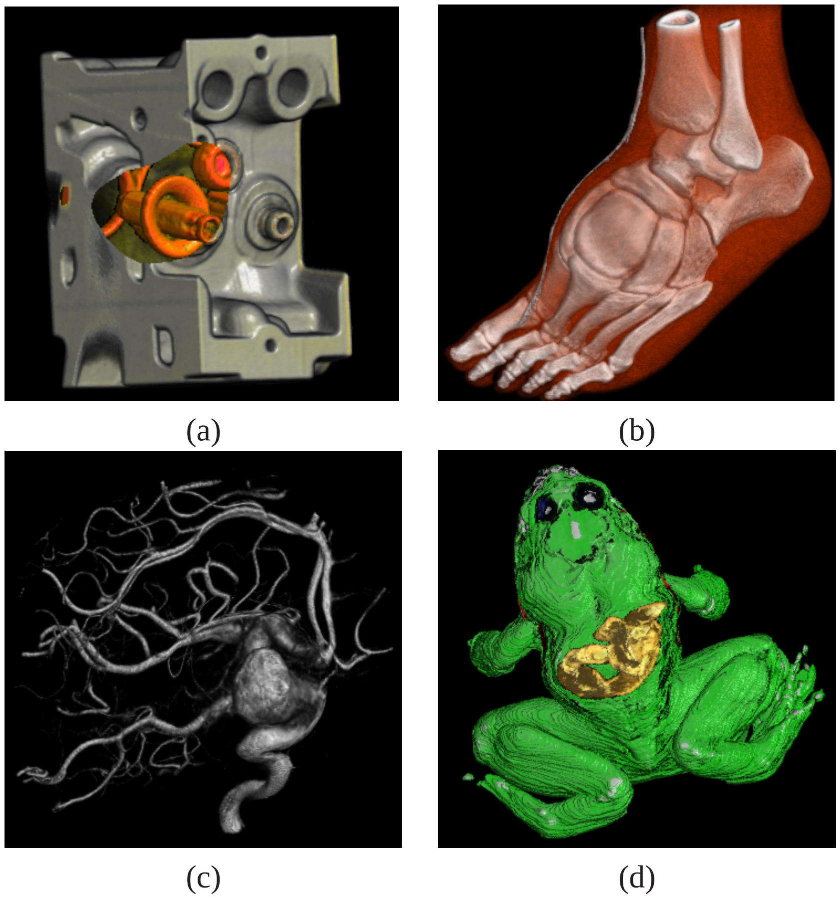


Figure 3.16: Magic volume lens results. (a) Magnifying inside features in an arbitrary-shaped area on an engine, (b) Applying sampling-rate-based lens on a foot, (c) Enlarging area of interest on an aneurism, (d) Magnifying the duodenum of a segmented frog dataset.

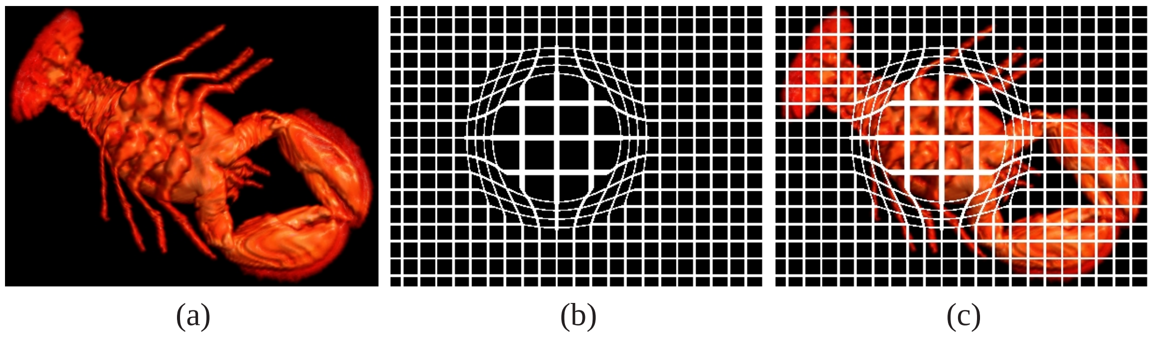


Figure 3.17: Lens-distorted lattice. (a), (b) and (c) Three rendering options.

lens in the context of volumetric distortion, as well as design free-style and feature-adaptive lenses for arbitrary magnified focus+context viewing. For example, coupled with a GPU-based interactive segmentation algorithm it can be used to magnify the segmentation result at great detail and aid in its refinement. The support for free-style lenses, created with our lens design interface, can help illustrators to designed more helpful and informative visualizations of volumetric objects, emphasizing an arbitrary shaped region of interest without losing the context of its surround. Finally, the GPU acceleration of our magic volume lens allows all of these to be done at interactive speeds, fostering both creative design and exploration. It also proofs helpful to users to provide an option for superimposing a lens-distorted lattice on top of the lens area, to aid in the assessment of the non-linear magnification effects (see Figure 3.17).

We would like to extend this free-style zooming capabilities to multi-resolution data and to semantic zooms, where the data appearing under magnification comes from a different data source, or even texture synthesis.

Chapter 4

Semantic Zoom Using Texture Synthesis

4.1 Introduction

When viewing an image (note, a volume can be considered a 3D image for the discussion here) the amount of detail that can be visually explored is fundamentally bounded by the image resolution. Magnification will not extend the amount of visible detail, it will only spread it out in space such that it can be better discerned by the observer. Magnification typically entails some blurring, depending on the quality of the magnification filter used [144]. However, it should be obvious that even with the best filter, pure magnification can not add detail where it has not been sampled before. Therefore, zooming into an image or volume at high magnification factors tends to create a rather boring, non-informative, and non-satisfying viewing experience.

The amount of available detail may be constrained by: (i) economical limits bounding the size and therefore the detail of the image, and/or (ii) technical limits inherent in the image acquisition process. As an example for the latter, optical lenses generally are only able to provide focus within a certain range of scale, while imaging technologies, such as MRI and CT, impose physical limits on the amount of detail they can resolve. Should detail on other scales be desired, alternative lenses or imaging methods, such as optical, confocal, and electron microscopy are required.

In computer graphics, texture mapping has long been a method by which interesting detail can be added. However, texture placement is usually guided by geometry, and not by semantic constraints imposed by the image to be enriched. Texture mapping may also cause repetitive tiling artifacts. Texture synthesis has more promise in this respect. For example, Freeman et al. [39] established a database of coarse-fine resolution mappings that they used to add fine detail to magnified images of natural scenes. This fine detail, however, was on the same order of scale as the base image, and only magnifications at the same semantic level of scale were possible.

In this thesis, we propose to extend the notion of image-guided detail enhancement to multiple levels of scales. However, we would like to avoid traditional image pyramids where multi-scale detail stems from the repeated smoothing of a single high resolution image. This is because requiring such an image would violate one or both of the constraints mentioned above. Instead, we introduce the notion of semantically constrained multi-scale texture synthesis to facilitate zooms at a virtually infinite number of scales, as long as the corresponding texture data are available (see Figure 4.1). Here, the term “semantic zooming” means that the multi-scale detail is not derived from one image to the other via simple filtering, but via different sampling processes tuned to the respective level of scale. An everyday example of semantic zooming [40] is electronic maps, where each level of zoom is an excerpt of a different map, such as country, state, city, neighborhood, etc., bearing a very different style and type of detail.

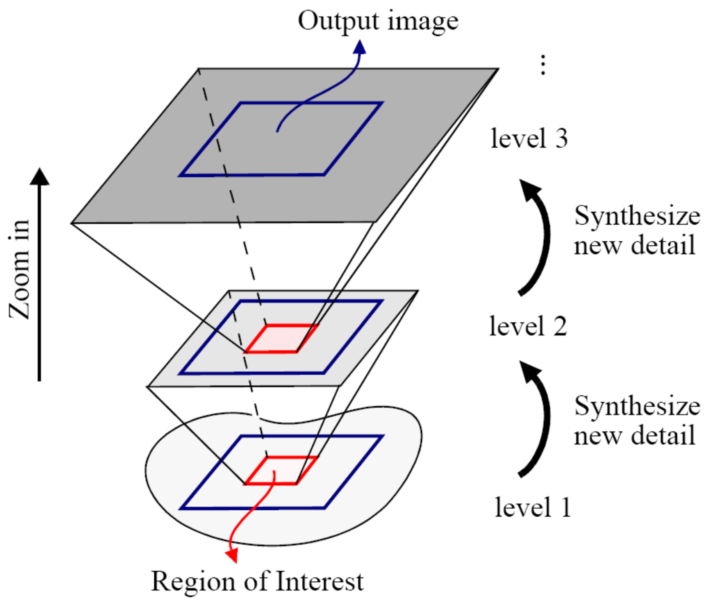


Figure 4.1: Semantic zooming based on texture synthesis.

In contrast to the aforementioned maps, our application does not store complete images at every level. One of our main design goals is to generate the semantic detail at a minimum of memory cost, thus providing a solution that will scale well. Therefore, our system will not yield an accurate multi-scale “map”, rather, it will generate something that looks like an accurate multi-scale map, however, one in which large-scale features and its small-scale detail smoothly blend into one another.

For example, one of the possible domain applications of our system is the “virtual microscope”, where users start at a low-resolution MRI or CT image of some biological tissue and then slowly zoom in anywhere they desire to reveal the underlying cell structure, and finally the interior of the individual cells themselves. This process is illustrated in Figure 4.2, for a human liver. Other possible applications include multi-resolution viewers for terrains, the universe, a sheet of metal, or any other domain that offers multiple levels of semantically constrained data, under the assumption that these data can be obtained. The fact that the different levels are obtained via synthesis and not via filtering of a common source imposes certain restrictions on the use of our technique. For example, our medical viewer would not be suitable for diagnosis of a diseased liver. However, it could be employed in a surgical simulation trainer, an electronic atlas for medical students, or a scientific illustration tool. Note that in these application scenarios the data at the different levels of scale do not have to be acquired from the same specific object, or in this case, person. This is especially advantageous since some of the higher resolution acquisition methods may be destructive. Similar restrictions and applications can also be envisioned for other application domains.

For our 2D application, we combine pixel-based synthesis [137], image quilting [34], and our pattern-based synthesis. For 3D applications, instead of image quilting, we use Graphcuts [74], since it is efficient and easy to implement in 3D. Our approach is fundamentally different from that of Nealen and Alexa [99] who use pixel-based re-synthesis to eliminate remaining errors in the overlap regions of patch-based synthesis. In contrast, we apply different types of synthesis methods to synthesize different regions and features in an image. Further, our pattern-based synthesis is location constrained and differs from the algorithms based on pattern placement in the surface texture synthesis domain, such as pattern-based texturing revisited [101], and texture particles [29].

In our application, we make frequent use of *constrained texture synthesis*, where the patch selection and texture generation is made dependent on some underlying constraints. This technique has been utilized in image processing, such as image restoration [149] and texture transfer [2, 3, 34, 53]. Another example is the texture-by-numbers technique [53], which is able to perform synthesis from images in which the texture distribution is not stationary but is based on the labelling of the component textures of images. These label images, representing the segmentation information of images, are created beforehand, possibly by the user. Some automatic color or texture segmentation methods are used for guiding the texture synthesis process in [59, 31]. Our constrained texture synthesis follows a similar idea, but here only the segmentation of the sample images can be performed in advance. The features or patterns in the synthesized images have to be detected and labelled automatically when they are needed during zooms (see Section 4.2.2 for further detail). To enable proper semantic relationships across zoom levels, component textures should be placed carefully, following certain constraints including color, intensity, distance fields, location, and features/patterns of the image.

In contrast to Freeman's super-resolution algorithm [39] which generates enlarged images on the same semantic level than the base image, our application performs enlargement/zooming [104] that spans several semantic levels. Our main contributions are:

- *Semantic zooming* uses texture synthesis to extend image-guided detail enhancement to multiple levels of scales.

- *Constrained texture synthesis* facilitates smooth semantic evolution and detailing of features across zoom levels.

- *Feature-guided texture synthesis* considers the properties of features or patterns in the image at a certain semantic level and chooses image quilting, pixel-based, or pattern-based texture synthesis methods in accordance with the region's synthesis requirements.

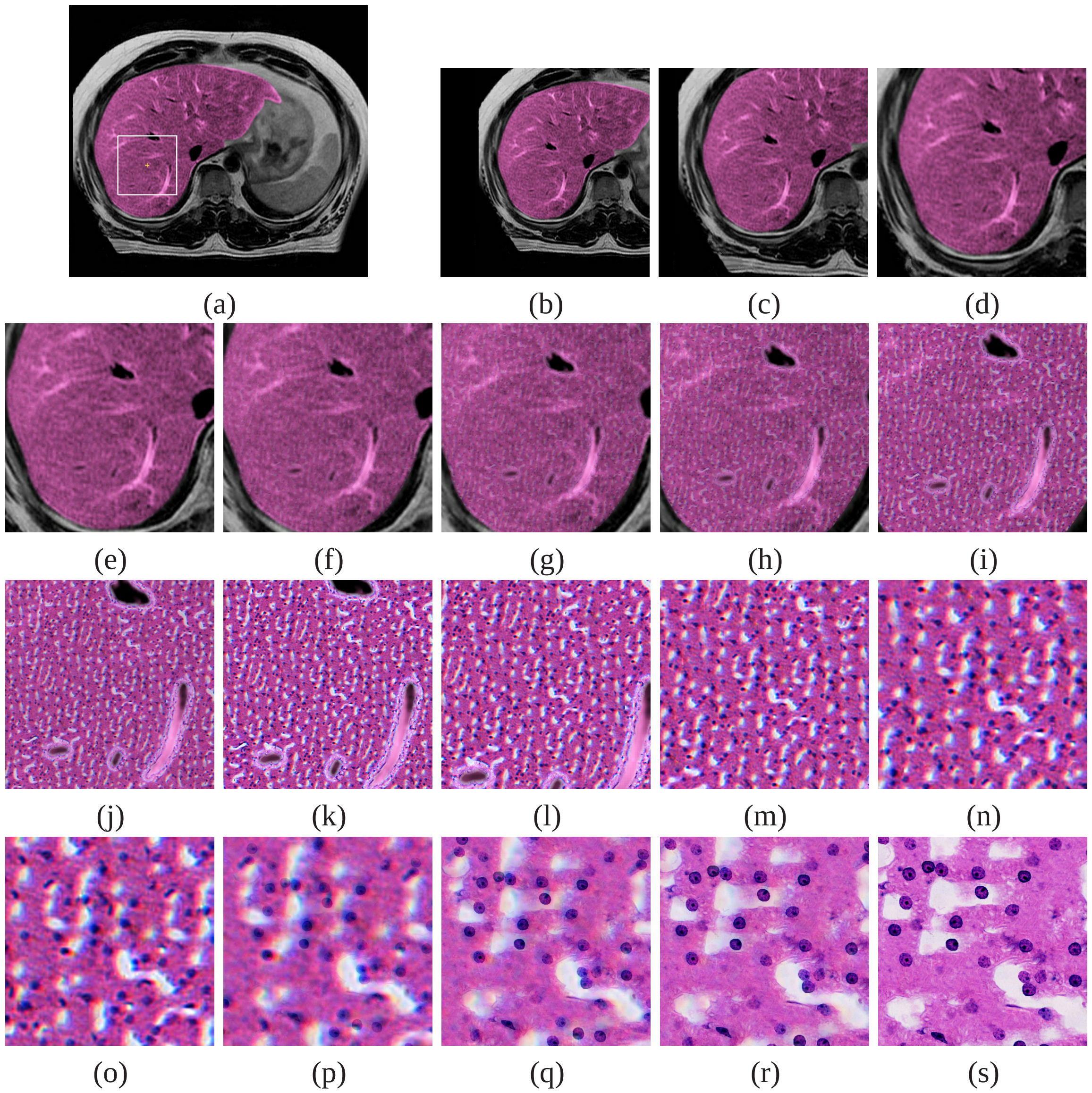


Figure 4.2: Illustration of the semantic zooming capabilities facilitated by the virtual microscope, using a human liver as an example: (a) MRI image of a liver, where the white square is the user-specified region of interest, (b)-(s) A typical image sequence during a semantic zoom, in which (k) is the synthesized histology level image, and (s) is the synthesized cell level image, (c)-(e) Magnified MRI level images, (l)-(p) Magnified histology level images, (f)-(k) Images obtained by blending magnified MRI and minified histology level images, (o)-(s) Images obtained by blending magnified histology and minified cell level images.

4.2 The Virtual Microscope – A 2D Viewer

We first discuss the 2D application, which acts like a microscope with a wide range of magnification. Then, in the next Section, we will discuss its extension to 3D. A system overview is shown in Figure 4.3. First the underlying multi-resolution image data are collected and preprocessed to build a set of sample images. Then the sample images are analyzed to choose the appropriate texture synthesis approaches and constrained rules for each pair of adjacent levels. All these are stored in a small database, which will be used during the semantic zoom operation.

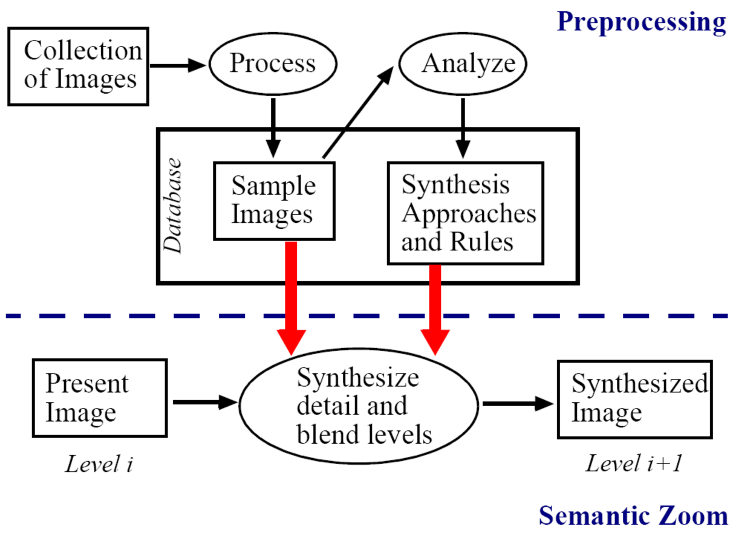


Figure 4.3: System Overview.

At the beginning, the user views the image at the coarsest resolution (Figure 4.2a). Once the user specifies a region of interest in this image and zooms in, this part of the image is gradually magnified. When the image magnification reaches a certain scale, the image detail of the next level is generated through semantically constrained texture synthesis based on the currently magnified image region. For instance, when the user zooms into the image from the MRI level to the histology level, the system needs to synthesize the corresponding histology level image. The same is the case for the cell level. Blending of two consecutive levels enable the system to go smoothly from small-scale features to high-scale features. Thus, there are three main tasks in our system: data preprocessing, constrained texture synthesis, and level blending. We will now describe each of them in detail.

4.2.1 Preprocessing

We first need to collect data corresponding to the various levels and perform some amount of preprocessing on them. Figure 4.4 shows the sample images used in the liver

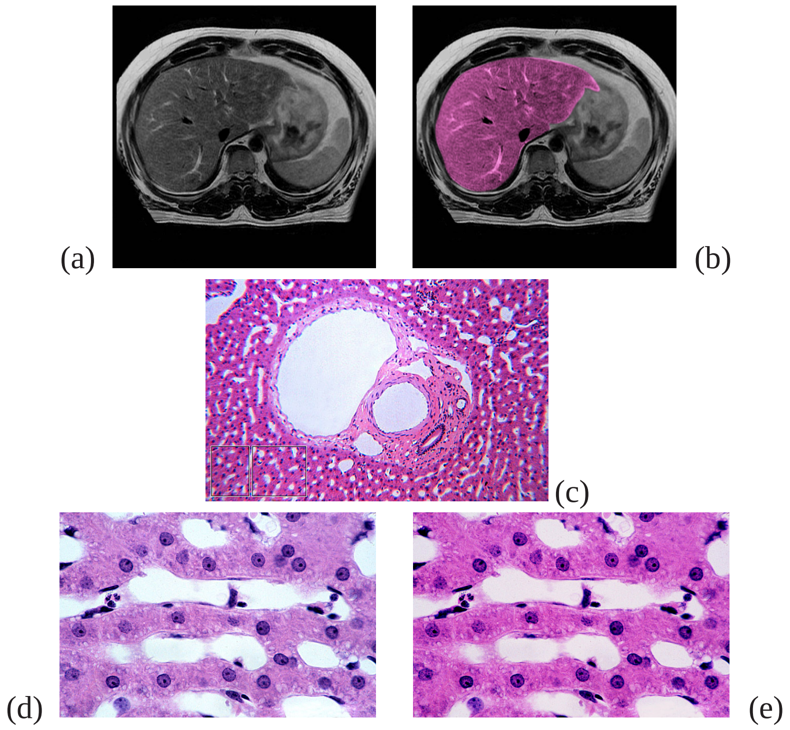


Figure 4.4: Image data and pieces of colorized sample images. (a) MRI liver image, (b) Colorized image, (c) Low-scale histology image, (d) High-scale histology image, (e) Colorized image. (Images (c) and (d) courtesy of http://www.bu.edu/histology).

example: an MRI image, a low-scale histology image, and a large-scale histology image. These three levels will be referred to as MRI level, histology level, and cell level, respectively. However, it is easy to increase the number of levels as long as the corresponding texture data are available. Once the images have been collected the following pre-processing steps have to be performed.

Colorization Typically, the images that are collected have different colors. In order to reduce the distinct discontinuities arising from mismatched colors during zooms, we need to match the colors across levels. The color correction can be easily done by image processing methods or tools, such as Adobe Photoshop. The colorized images shown in Figure 4.4 are the sample images that will be used to guide the synthesis later on. Since we use the color of the low-scale histology image for transfer, this image requires no change.

Segmentation The sample images need to be segmented into prominent features or patterns, based on color, shape, or pre-knowledge. In our particular example, for the MRI image, we segment out the liver region as well as the portal vein and the artery elements. The segmentation can mostly be done via image processing methods [42] or tools. The segmentation results, which will later help us to match texture synthesis methods with different features or patterns, are stored in tag images (see Figure 4.5).

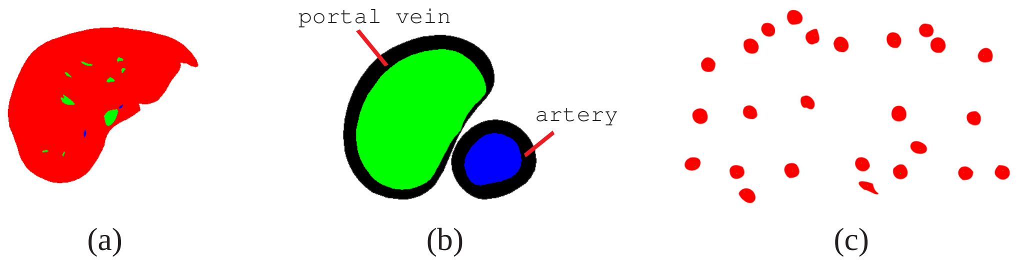


Figure 4.5: Some tag images for the liver example. (a) MRI image, (b) Histology-level image, (c) Cell level image.

The data preprocessing is the only part in our system which may require some manual work to refine the image processing results, but it needs only to be done once. After that, no manual work is required. The colorized sample images and the corresponding tag images are then stored in a database.

4.2.2 Constrained Texture Synthesis

Synthesis Approaches

As mentioned before, a variety of texture synthesis approaches could be applied to generate the image detail for semantically different levels. For each pair of adjacent levels, which texture synthesis approaches should be used depends on the texture features, and the region in which the texture will grow.

- If the texture is isotropic, semi-structured, or structured, and grows in a large region, image quilting or other patch-based algorithms produce better quality results than pixel-based methods. The primary parameters in image quilting include patch size and overlapping region size. Both mainly depend on the prominent structures of the texture and should be decided before synthesis.

- If the texture has layers and/or grows within a small irregularly shaped region, then a modified pixel-based approach forms a convenient way to add fine detail in the magnified images. We give the details of our algorithm later on. The parameters in a pixel-based synthesis algorithm [137] include the shape and size of the pixel neighborhood, as well as the number of levels if a multi-resolution algorithm is applied.

- If the texture is composed of atomic patterns which should be preserved during synthesis, our pattern-based synthesis is employed to synthesize the patterns, while other pixel-based or patch-based approaches can be applied to synthesize the background color.

Why do we need *constrained* texture synthesis? We need it to ensure that the generated textures on one level are semantically consistent with the level before. Since we use level blending to facilitate intermediate zooms, this is obviously very important. Standard texture synthesis algorithms only use the present layer information in the generation process, and Figure 4.6 demonstrates the poor blending that will occur if we perform texture synthesis on the histology level without constraining it to the lower-scale MRI level. Similar problems arise for the cell level and the lower-scale histology level. Thus, textures of the high-scale image should always be synthesized to match the features of the low-scale image under specific constraints. For this reason, the system always computes a tag image of the current result image to facilitate the matching process. This is somewhat similar to the label-constraints used in [53] and [2], but in our application the constraint tags are not specified by the user but generated automatically, using image processing techniques.

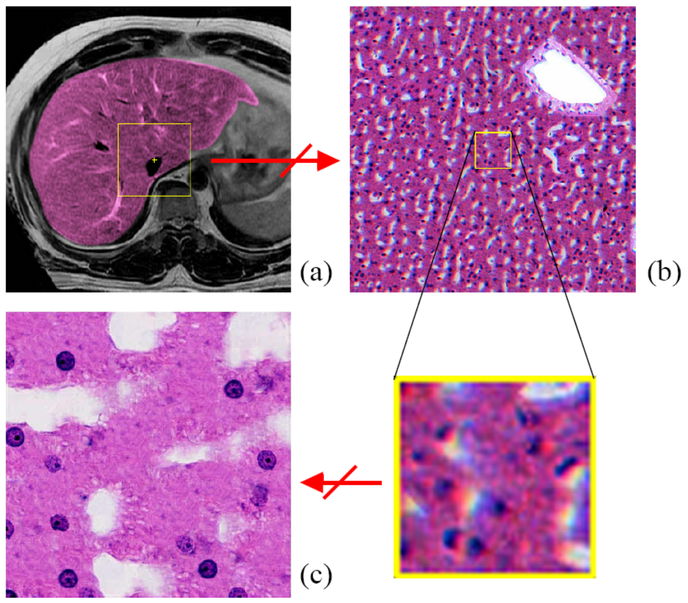


Figure 4.6: Mismatched levels. The histology level image (b) does not match the specified region of the MRI level image (a), and the cell level image (c) does not match the specified region of (b) either.

In our system, three texture synthesis methods are combined to synthesize the image. We mainly discuss the algorithm modifications which need to perform constrained synthesis.

Constrained Image Quilting

Image quilting is used to generate the background texture for the histology level and the cell level image, but other patch-based synthesis methods, such as Graphcuts, may also work. In the histology level, background is defined as everything except the vessels and their surrounding layer. In the cell level, background is defined as everything except the cells, the vessels and their surrounding layer. We also tried pixel-based synthesis methods to generate the background as well, but neither the single resolution nor the multiresolution (with TSVQ acceleration or PCA and ANN acceleration) algorithm seemed to work well for the textures used here, mainly because the features in the texture tended to come out blurred.

Our constrained quilting algorithm differs from typical quilting in the following two ways. First, not all patches in the segmented sample image can be used for synthesis. For example, at the histology level, the textures around the portal vein and the artery are different from the background texture (see Figure 4.4c). Hence, the patches falling into those regions should not be used to generate the background texture. Second, both patch placement and selection are constrained to satisfy the match requirement. Especially at the cell level, in order to match the histology level features, the quilting process is constrained by the color/intensity of the magnified histology level image. An example for this are the white areas, called *sinusoids*, which appear on both synthesized levels and should be matched. Thus, when selecting a candidate patch for the third level, the location, shape, and distribution of its sinusoids must match that of the corresponding second-level area. This is not a limitation since our sample database is diverse enough, and we have never encountered a case where no fit could be found. Considering the texture structure size, the quilt patch size is chosen to be 40×40 pixels, and the overlapping width is 6-8 pixels.

A further constraint for background texture synthesis are object boundaries, both interior and exterior. The tag images play an important role in complying to these boundary constraints, and this will be discussed at the end of this section.

Constrained Pixel-Based Synthesis

Smaller structures constrained to tight and curved boundaries are better generated using pixel-based synthesis methods, since patch-based methods work on a scale too large to adhere well to the object's geometry. In our application, we use this type of approach to generate the small textures in the surrounding layer around the portal veins. However, at the same time it is desirable to transfer the global characteristics of the sample texture to the output image as well. For example, texture features, such as smooth muscle cells in our application, which are closer to the object boundary in the sample should also be placed closer to the boundary in the output image. We can achieve this by constraining the texture generation process by a measure imposed by the object geometry – distance fields, which we use here to (i) constrain the texture generation and (ii) help to find the outside boundaries for magnified veins to guide the synthesis process. We will first illustrate our pixel-based algorithm for the general case (see Figure 4.7) and then discuss how it is applied within a specific example (see Figure 4.8).

We calculate the distance field using a distance transform and normalize it to a range of [0,1]. The distance field is shown in Figure 4.7a as a grey image, in which pixel value maps to distance. If the given sample texture has a layered appearance (Figure 4.7a), then the synthesis process must depend on these distance values. After calculating the distance fields for both sample and result image, we use the standard scan-line order to synthesize

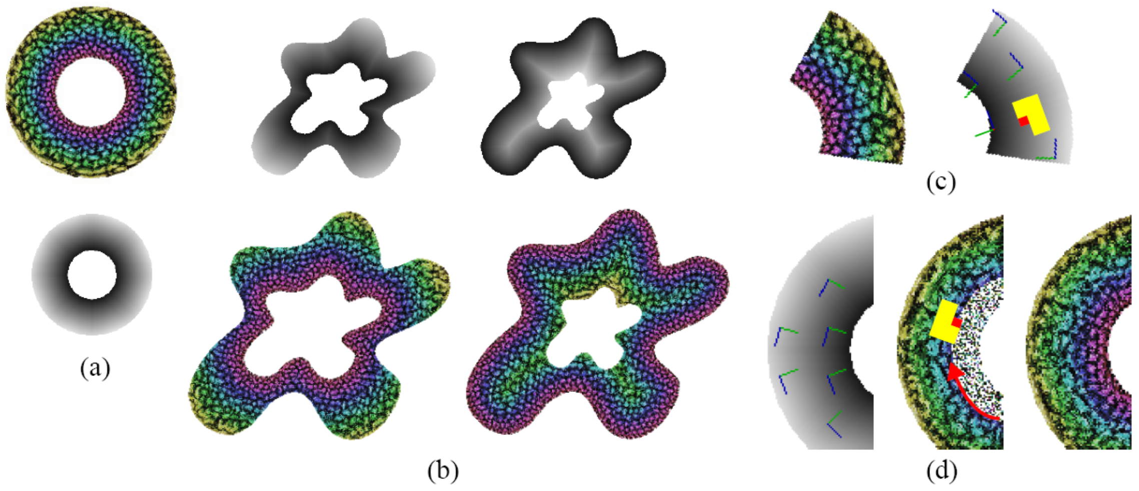


Figure 4.7: Our pixel-based synthesis methods. Pixel synthesis based on distance field: (a) Sample image and its distance field, (b) Reference distance fields and corresponding synthesis results. Pixel synthesis based on distance field and gradient field: (c) Sample image and its distance and gradient fields, (d) Synthesis process and result.

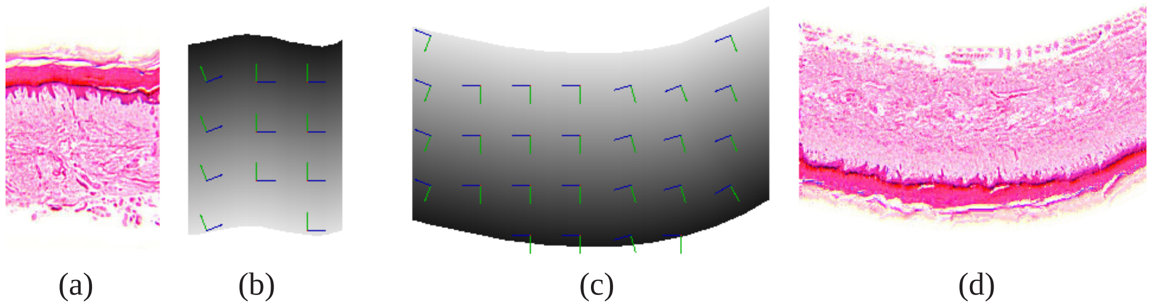


Figure 4.8: Our pixel-based synthesis results. Sample thick skin histology image (a) and its distance field (b), reference distance fields (c) and the result of synthesizing a thick skin histology image (d).

the pixels. There, for each pixel in the result image, the matched pixel must be chosen from the set of pixels that (i) observe the usual texture synthesis metrics [137] and (ii) have a similar distance field value.

If the input image is part of a layered texture, or if we want to reduce the sample image size to speed-up the synthesis, our pixel-based synthesis method will not only depend on the distance value, but also on the texture direction, which is calculated from the distance field and represented by a gradient field (see Figure 4.7c). The pixel synthesis order depends on the distance values, and, based on the gradient, rotated L-neighborhoods are compared to find the best match.

In our bio-tissue example, we pre-compute the normalized distance field around the portal vein based on the tag image of the sample histology level image (see Figure 4.9a).

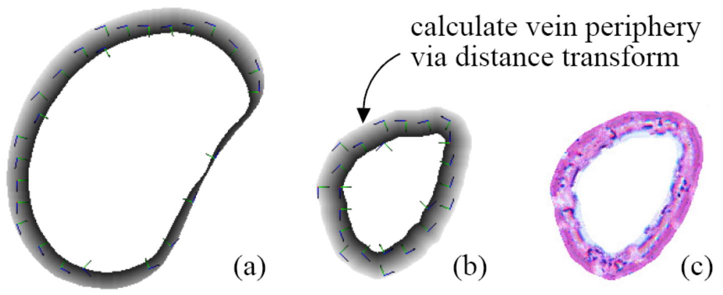


Figure 4.9: Vein periphery synthesis based on distance fields. (a) Generated from the segmented sample image, (b) Generated from magnified MRI image, (c) Texture detail.

When synthesizing the histology level image, we compute a similar distance field around the vein of the magnified MRI image to find the boundary of the vein structures (Figure 4.9b). The detail in the vein periphery is then synthesized based on the distance and gradient values.

Pattern-Based Synthesis

Our pattern-based algorithm is designed to preserve potential atomic structures, i.e., structures that cannot be cut, such as cells. Pixel-based or patch-based synthesis methods cannot generally guarantee that features remain uncut or undistorted, since they have no knowledge about which part of the texture constitutes a whole atomic pattern. We require an algorithm that will ensure that atomic structures remain intact and, at the same time, satisfy the match requirements.

We can achieve this by identifying the location of the atomic structures on the low-resolution level and replace them by high-resolution versions in the magnified level. If these structures have fuzzy boundaries that blend with the background, it is useful to keep these as well. They can then later help to integrate the features into the background in a coherent way.

The first step involves identifying the atomic features. In our liver tissue example, these atomic features are represented by the cells in the cell level sample image (Figure 4.4e) and are segmented as patterns (Figure 4.5c). When synthesizing the cell level image, the algorithm first detects all possible cells (dark points) of the magnified histology level image based on the image intensity, and records this location information. We detect the dark points using two thresholds. Then location-constrained pattern placement proceeds, and the cell patterns are chosen randomly to increase the variation of the result. A similar method can also be used for magnifying the cells in the layer around the portal veins.

As we have mentioned above, the tag image, which corresponds to the current zoomed image, is important to comply with the match requirement. For example, the vessels (portal veins and arteries) represent interior objects which should be preserved as they are and properly scaled under zoom. However, scaling the tag image presents a problem. When the image is magnified, the corresponding tag image should also be enlarged at the same

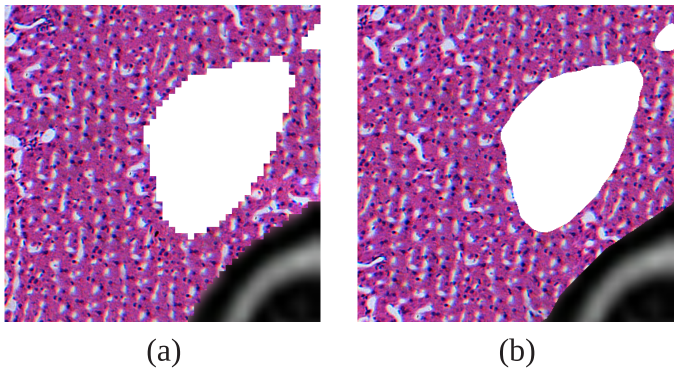


Figure 4.10: Smooth boundary problem caused by tag image magnification. (a) Dentate boundary, (b) Smooth boundary.

rate. Without any specific process, the boundary of the enlarged tag image will have a binarized effect (Figure 4.10a). To prevent this, we use a smooth interpolator for the tag values, and then choose an intermediate value as the threshold to decide the boundary. Using this procedure, the magnified image will still have a smooth boundary (Figure 4.10b). Another possible solution is to represent the boundary as a spline curve. If the segmentation information is stored using a spline curve, the enlarged spline curve can be calculated based on several control points while the image is magnified. In this way, the boundary can be very accurate.

4.2.3 Smooth Semantic Zooms

When zooming into a specific region of the image, our system combines two processes: (i) magnification of the current level image, and (ii) minification of the synthesized next high-scale level image. This achieves any level of magnification from only a few images with different semantic detail.

The system has a number of parameters, some are set by the user and some are decided by the available data. The first such parameter is the size of the output image, $M \times M$, which specifies the screen size of the microscope. A second parameter is the maximum zoom scale Z_{max} for each level, which is determined by the resolution of the subsequent, more fine-scale level. This factor determines the amount of standard magnification that needs to be performed using the current level data before new semantic detail can be filled in by synthesizing from next-level data. Obviously, the more levels are available, the less blur will be encountered when zooming in. Since for real optical, confocal, or electron microscopes the maximal zoom scale can be from thousands to millions, our application accelerates the zooming activity by dramatically reducing Z_{max}. When the present level data is magnified at Z_{max}, the resolution has been reached at which the next higher level data can be synthesized to provide the missing detail.

Also, at the beginning, the user specifies a zoom focal point F, which determines the center of the region of interest R. This region R has a size $N \times N$ and is calculated by the

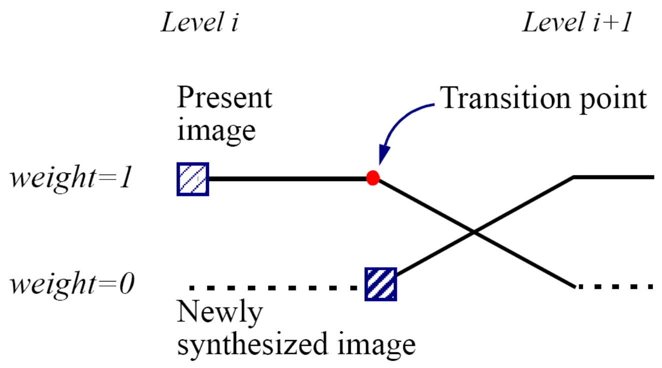


Figure 4.11: Image transition process.

system, such that $N = M/Z_{max}$. R marks the image region that will be replaced by the next higher level detail when the zoom scale Z reaches Z_{max} (in our example, this region is shown as the white square in Figure 4.2a).

The last parameter that our system maintains is the view port VP which is centered at F and has a size $V \times V$. It varies with Z, such that $V = M/Z$. At any given Z, the system will capture the image inside the VP, and then magnify and fit it into the output image. At startup, the image is not magnified, i.e., $V = M$ and $Z = 1$, and is shown as the output image directly (Figure 4.2b). When the image is gradually magnified by the user, Z increases, while the VP decreases. Once the VP has reached R, synthesized image data due to the next higher-level detail should be made available.

It is desirable to avoid a sudden change of the display, where the image generated from the next higher level of resolution suddenly pops in. We accomplish a graceful transition by blending the images of two consecutive levels over some range of zooms, properly weighted by a zoom-related weighting function. In addition, we prefer to do this without having to view blurred features of the present level. We can achieve both of these requirements by specifying a transition point t with a zoom scale Z_t, where $Z_t < Z_{max}$, at which we compute the image for the next level, minify it, and blend it with the magnified present level. This early computation of the high-resolution image, however, requires the computation of extra data at boundaries, later culled with further zooming until the $Z = Z_{max}$. More specifically, suppose that the synthesized image has size $S \times S$, then $S = M \times Z_{max}/Z_t$. The advantage of having a larger image available is that it allows more panning activity within the next semantic level.

The smooth image transition process over a range of consecutive zooms is illustrated in Figure 4.11. After the transition point, the magnified present image and minified synthesized image are smoothly blended by gradually changing their weights inversely, i.e., the magnified image will fade out while the synthesized image will fade in.

4.2.4 Results

In this section we report on our specific application – the virtual microscope viewing a liver datasets at three levels of semantic scales. The sample images and corresponding tag images stored in our database have been shown in Figure 4.4 (b,c,e) and Figure 4.5. A few frames of the resulting image sequence during a semantic zoom are shown in Figure 4.2. When the user specifies a region-of-interest in the MRI image of a liver and zooms in, then this part of the MRI level image is gradually magnified and blended with the synthesized histology level image. If the user further zooms in from the histology level, the histology level image is magnified and eventually blends with the synthesized cell level image. This resembles the functionality obtained with a real microscope, when slowly examining an interesting part of a liver. Besides zooming, the user can also pan to inspect nearby regions.

In our algorithm, once the sample images are chosen, the time to synthesize a certain level image mainly depends on the output image size M and the magnification scale Z_t of the transition point. When M is fixed, the time spent on synthesis and the blending process can be adjusted by Z_t. For example, suppose the output image size is fixed on 400×400 and the maximal scale $Z_{max} = 4$. If the specified scale Z_t of the transition point is 2, then the synthesized image has a size of 800×800. With the current implementation, it will take several minutes to generate the result image. If Z_t is increased to 3, the corresponding synthesized image becomes 533×533, which reduces the time spent on synthesis. However, the blending effect is also reduced, which means the synthesized next-level image will pop in more abruptly.

4.3 Extension to 3D

The idea extends well to volumetric data. In order to generate sub-resolution detail for volume data, we extend image quilting to volume quilting, and also apply a 3D pixel-based synthesis algorithm. In volume quilting, we apply the graph cuts algorithm [154, 155] to find the best seam surface between two neighboring blocks, instead of using the shortest path algorithm, which is applied in image quilting but not easy to be extended to 3D [74].

From the Visible Man's cryosection data, we reconstructed the volume and segmented out the liver. Similar to the 2D case, the volume data is also colorized to match the histology data. The sample histology volume is built based on the features in the 2D image and certain 3D growth rules. We could also apply Wei's solid texture synthesis method [136] to generate a sample volume, however, it is difficult to get a high quality solid texture. Figure 4.12 shows the volume data required by the synthesis procedure.

In the 3D extension of our viewer, the user specifies a volume-region-of-interest (Figure 4.13a), and this volume region is cut out from the original volume and rendered. During 3D zooms, the volume region is magnified and smoothly blended with the minified synthesized higher level volume. The observed volume size changes during zooms, in contrast to the fixed-size output images in the 2D system. Some volumetric semantic zooms are shown in Figure 4.13. For the histology level, as in 2D, the textures around the vein are synthesized

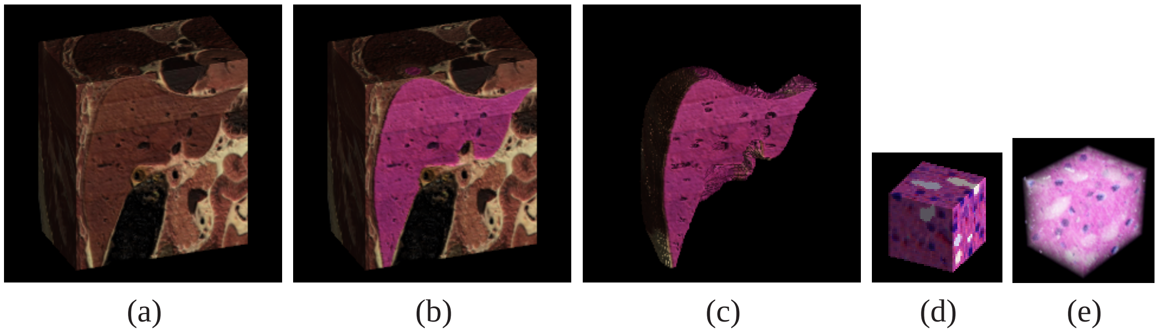


Figure 4.12: Volume data and colorized volume. (a) Visible man's volume, (b) Colorized volume, (c) Segmented liver, (d) An example of the sample histology level volume and its translucent result (e).

by a pixel-based algorithm, while other textures are created by volume quilting. Figure 4.14 shows volume with cut and translucent rendering results. The translucent volumes are rendered using the OpenQVis software (http://openqvis.sourceforge.net/). An advantage of volume synthesis over traditional surface synthesis is that only the former can illustrate the translucent effect of internal structures.

4.4 Discussion

We have described a new constrained multi-scale texture synthesis method to facilitate semantic zooms. Pixel-based, image quilting, and pattern-based synthesis methods were unified to generate high-detail images under certain constraints. Our demo application, a virtual microscope, demonstrated that quite interesting and useful image sequences can be generated using our framework.

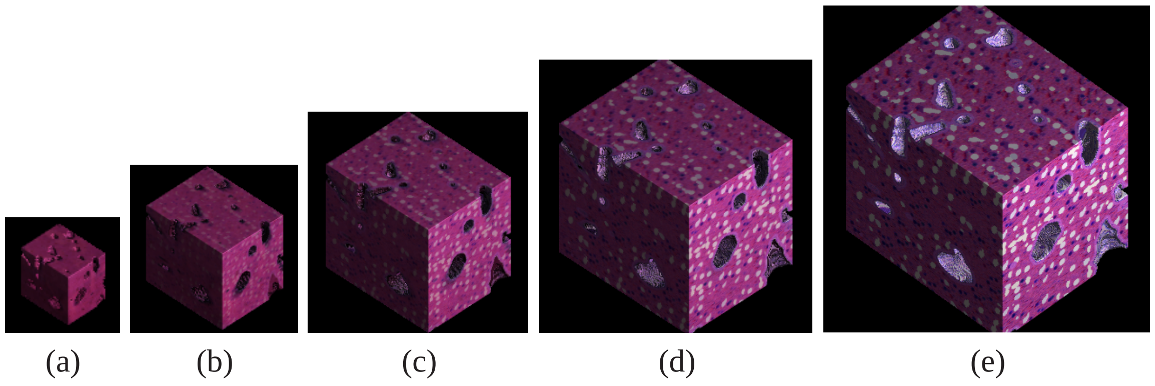


Figure 4.13: Illustration of semantic zooming into volume data. (a) First level for part of the liver, (e) Histology level of (a), (b)-(d) Volumes obtained by blending the magnified first level volume and minified histology level volume.

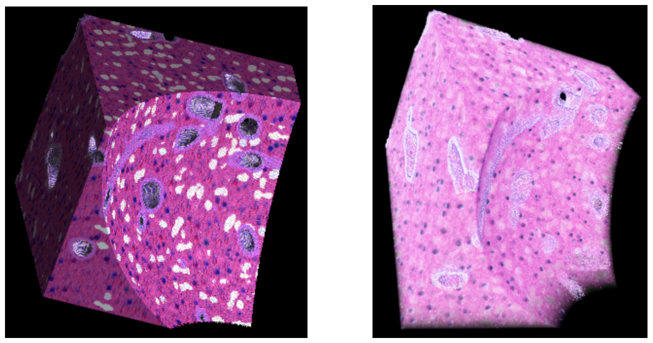


Figure 4.14: Synthesized volume with sub-details.

Comparing ours with the Volumetric Illustration work presented by Owada et al. [102], both employ texture synthesis techniques to illustrate the volumetric details. They synthesize textures on the cross-section surface mesh to illustrate the internal texture of 3D models. We, however, directly generate volumetric textures with high-resolution details. Inspired by our work, Lu and Ebert proposed example-based volume illustration technique also with impressive results [89].

In future work, our algorithm could be improved in terms of accuracy and speed. For the former, more sophisticated segmentation and constraints may yield more refined small detail. We would also like to explore better interpolation methods for the oriented texture synthesis to overcome some of the remaining visual artifacts. Finally, optimization and GPU acceleration of our algorithm will provide more interactive capabilities, i.e., for generating the detail on demand when zooming into an image or volume.

Chapter 5

Uniform Texture Synthesis

5.1 Introduction

Texture mapping is a technique that is used to map 2D images to both planar and curved surfaces in order to enhance their visual effect. Texture synthesis has similar goals, but instead of using the explicit tiling approach of texture mapping, it aims to generate the surface decor from a relatively small texture sample. Both approaches are useful in their own right. Texture mapping is the technique to use when the goal is an exact preservation of the detail in the provided texture sample, but it tends to suffer from repetition and tiling artifacts, as well as seams, and it also requires overhead for texture storage. The detail generated in texture synthesis, on the other hand, is not an exact match, but only similar to the provided texture sample. It, however, is seamless and non-repetitive. Texture synthesis replaces the memory overhead of texture mapping with computational overhead incurred by the synthesis process. Both methods have similar demands in terms of avoiding local and global distortion, which, as we will show, can be achieved by preserving the texture sample local angles and global scale during the mapping or the synthesis process, using a conformal parametrization approach.

Global conformal parameterization was introduced in [46]. It guarantees that the shapes embodied in the textures are preserved on the surface, and it is global, which means there are no seams or cracks. Furthermore, the parameterization can segment the surface into patches, where each patch is mapped to a planar rectangle. This is valuable for real applications. The existence of the global conformal parameterization is equivalent to the fact that all oriented surfaces are Riemann surfaces [62]. Therefore, global conformal parameterization discovers more profound geometric structures on surfaces. For example, it induces the so-called affine structure, which is the foundation for generalizing splines with planar domains to be defined on surfaces [45]. It applies the concept of a differential form from Riemann surface theory [62], which can be interpreted as a pair of smooth vector fields orthogonal to each other.

The intrinsic difficulties for texture synthesis are due to two main aspects. The first originates from the local geometric properties of the surface. The texture image is defined on a flat planar region, and once it is mapped to the curved surface region, there must be distortion, which is challenging to mediate. The second is caused by topology. Since the surface and plane are topologically different, there is no global one-to-one smooth mapping between them. Thus the existence of seams and singularities is unavoidable, as well as it is challenging to remove these seams and minimize the number of singular points. While texture synthesis applied directly on surfaces (see Section 5.2) can do a reasonably good job, it requires a surface flow analysis which can be complicated. It is more intuitive to do texture synthesis on a 2D plane, where the sample itself is also defined. Conformal parameterization offers a convenient way to do this.

For texture mapping, while the angle is already preserved using the conformal mapping, a related application is to also seek the preservation of scale for texture mapping. Although it is mathematically impossible to make the parameterization preserve both the angle and the area, we can try to describe a convenient paradigm with which the two can be traded off.

Compared to traditional methods, our texture synthesis method based on global conformal parameterization has the following advantages:

- *Global structure.* Traditional texture synthesis methods are unable to produce textures with strong global structures, because they generate the textures locally and extend to cover the surface without coherent global consideration. In practice, it is highly desirable to generate textures with global structures. Since our method is based on global parameterization, it is easy to synthesize globally structured textures.

- *Purely 2D operation.* Traditional methods need to march on the 3D surfaces, and the data structure and the operations for this are complicated. In our method, all operations are entirely performed in the 2D parameter domain, which is much simpler and more efficient.

Further advantages of our conformal parameterization method, both for texture mapping and synthesis, are:

- *Angle preservation.* Conformal parameterization preserves the angles from the surface to the parameter plane. Therefore, the local mapping from the texture to the surface is just a scaling without angular distortion. The synthesis method then needs to only focus on the scaling factor, without having to pay attention to angle changes.

- *Regularity.* Global conformal parameterization induces a canonical way to segment the surface, such that each segment is conformally parameterized by a rectangle (Figure 5.1). The regular pattern of this global parameterization is helpful to simplify the algorithms.

- *Rigor.* Global conformal parameterization is based on solid geometric theories, and based on the parameterization, the distortion of textures on surfaces can be quantitatively measured. This makes it convenient for quality control.

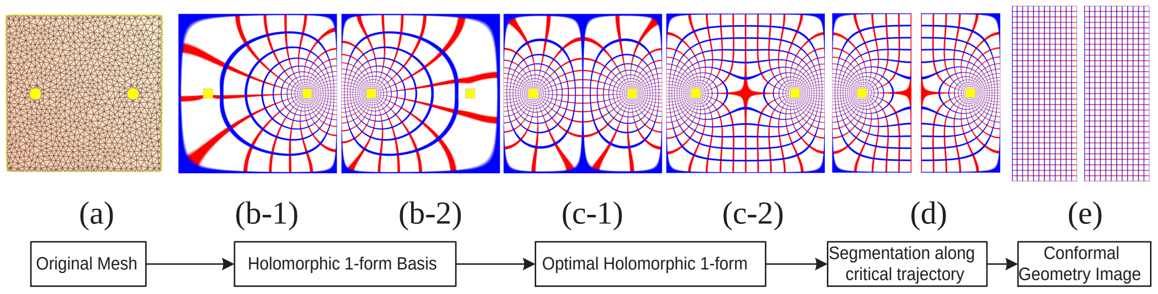


Figure 5.1: Process of Global Conformal Parameterization. The input surface is represented as a triangle mesh (a). The holomorphic 1-form basis is computed, (b-1) and (b-2) are the 2 base 1-forms [46]. By linear combining the basis, different holomorphic 1-forms can be constructed as shown in (c-1) and (c-2), then the optimal 1-form with most uniform 1-form is selected (c-2) [60]. The red and blue curves are the horizontal and vertical trajectories. Horizontal trajectories intersect at the zero point, the center of (c-2). The mesh is then segmented along the horizontal trajectory through the zero point as shown in (d), each segment is conformally parameterized to a planar rectangle illustrated in (e). The trajectories are mapped to the iso-parametric curves.

5.2 Related Work

Before presenting our new method, we shall discuss related work in two areas: surface texture synthesis and conformal parameterization.

Surface texture synthesis Surface texture synthesis extends the 2D texture synthesis methods to synthesize texture directly on the surface, including pixel-based methods [138, 129, 150, 125] and pattern mapping methods [101, 107, 117, 29]. With these methods, the discontinuities and cracks almost disappear, while the distortion problem is greatly reduced.

Another way to put textures on the surface is a texture mapping based on parameterization. But texture periodicity is obvious and often spoils the visual quality of the results.

Traditional surface texture synthesis methods are generally performed on the 3D surfaces to be decorated. In contrast, we propose a novel method which synthesizes the texture in the 2D parameter space, which is easier, more intuitive, and conceptually simpler. By using global conformal parameterization, the synthesized textures will not suffer from angular distortion. But we also have to deal with area stretching problems during the mapping. We will achieve this by using variable-size textures with local control. While most texture synthesis methods generate textures with uniform size features, textures with variant feature size are generated in [126, 74]. Their methods synthesize textures variant with respect to certain directions, and local control changes slightly compared to our technique.

Conformal parameterization Several recent advances in surface parameterization [38] have been based on solving a discrete Laplace system [106, 32]. Lévy et al. [85] describe a technique for finding conformal mappings by least squares minimization of *conformal energy*, and Desbrun et al. [27] formulate a theoretically equivalent method of discrete conformal parameterization. Sheffer et al. [116] give an angle-based flattening method for conformal parameterization. Gu and Yau [46] considered construction of a global conformal structure for a manifold of arbitrary topology by finding a basis for holomorphic differential forms. Gortler and Gotsman proposed one forms on meshes in [43] and applied for surface parameterization. Degerner et al. [26] provided user control to trade off between angle and area preservation.

Lai et al. [76] synthesized geometric textures based on geometry images. Their synthesized geometric textures will have more distortions than ours, because we use an improved global conformal parameterization with segmentation.

5.3 Global Conformal Parameterization

We assume that the reader is familiar with the basics of complex analysis and differential geometry. A detailed explanation for these concepts can be found in [62].

Suppose M is a surface with handles, either open or closed. A *global conformal parameterization* is a map $\phi : M \to R^2$, such that each point p on M is mapped to a point on the planar parameter domain $\phi(p) = (u(p), v(p))$. Furthermore, ϕ is *angle preserving*, this is equivalent to the following fact: suppose we arbitrarily draw two intersecting curves γ_1, γ_2 on M, the intersection angle is α, then their images $\phi(\gamma_1)$ and $\phi(\gamma_2)$ are planar curves, the intersection angle is also α. Mathematically, the conformality of the parameterization is formulated in the following way: the first fundamental form of M under conformal parameterization (u, v) is represented as

$$ds^2 = \lambda^2(u, v)(du^2 + dv^2), \tag{5.1}$$

where λ is called the *conformal factor*, it indicates the area ratio between the area on M and that on the plane.

In practice, it is more convenient to compute the gradient fields of ϕ, namely $(\nabla u, \nabla v)$. If ϕ is conformal, then they satisfy the following criteria:

$$\nabla v(p) = \mathbf{n}(p) \times \nabla u(p),$$

where $\mathbf{n}(p)$ is the normal at the point p, also

$$\nabla \times \nabla u = \nabla \times \nabla v = 0,$$

because the gradient fields are curl-free. Formally, a pair of vector fields satisfying the above conditions is a *holomorphic 1-form*. There is an infinite number of this kind of vector fields, they form a $2g$ dimensional real linear space, where g is the number of handles (genus) of M. The method of computing holomorphic 1-form basis has been introduced in [46].

The concept of holomorphic 1-form and the computational procedure are demonstrated in Figure 5.1. For simplicity, we only illustrate a naive example: a planar 2-hole square. In practice, the pipeline works for all 3D surfaces with arbitrary topologies. The red curves are the integration curves ∇u and called horizontal trajectories, the blue curves are the integration curves of ∇v and are called vertical trajectories. These trajectories are the preimages of the iso-u and iso-v curves. Figure 5.1(c-1) and (c-2) show different holomorphic 1-forms, (b-1) and (b-2) are the bases for the linear space of all holomorphic 1-forms.

From the infinite set of holomorphic 1-forms, we need to pick the best one for our texture synthesis. We choose the one with the most uniform conformal factor using the method introduced in [60], as shown in Figure 5.1(c-2).

The global behavior of the trajectories are very complicated. From Figure 5.1, it is obvious that the vertical and horizontal trajectories are orthogonal everywhere and two horizontal trajectories do not intersect each other in general. There are special points on M, where two horizontal trajectories intersect (two vertical trajectories also intersect). It can be proven that, at those points, the conformal factors are zero, therefore, such kind of points are called *zero points* of the holomorphic 1-form. In general, for a genus g closed surface, there are $2g - 2$ zero points. In Figure 5.1(c-2), the intersection points of the red curves is the zero point. The trajectories through zero points are called *critical trajectories*.

A trajectory can be a finite circle, a finite curve segment terminating at the boundaries, or an infinite spiral dense on the surface. If the horizontal critical trajectories are finite, then the whole family of horizontal trajectories are finite due to [120]. In practice, for simplicity, we choose a holomorphic 1-form with finite horizontal trajectories.

The critical horizontal trajectories segment the surface M into several connected components, each component is either a topological disk or a topological cylinder and can be parameterized by ϕ to a planar rectangle. Figure 5.1(d) illustrates this fact, the critical horizontal trajectory segments the surface into 2 patches, and each is conformally mapped to a rectangle. The horizontal trajectories are mapped to the iso-v curves (red), while the vertical trajectories are mapped to the iso-u curves (blue).

In practice, it is convenient to synthesize the textures on these rectangular parameter domains. Therefore, in our algorithm, we locate the zero point first by finding a vertex with minimal conformal factor, then trace the horizontal trajectory to segment the surface.

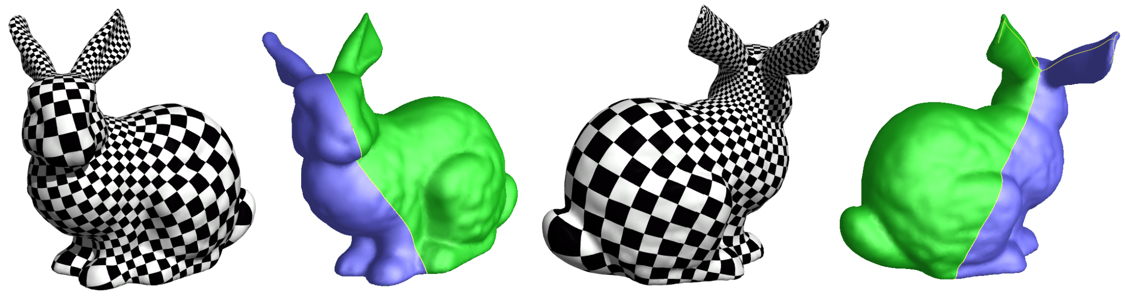


Figure 5.2: Global conformal parameterization.

Figure 5.2 illustrates a global conformal parameterization of the Stanford bunny surface. The bunny surface has 3 boundaries, two are at the tips of ears, one is at the bottom, therefore, it is topologically equivalent to the 2-hole disk in Figure 5.1. The double covered surface is of genus 2. A zero point is between the roots of the two ears. The horizontal trajectories through it are illustrated as yellow curves. The whole surface is partitioned into 2 topological disks, each segment is color-encoded. Figure 5.4(a) and (b) demonstrates that each segment can be conformally mapped to a rectangle on the plane.

Textures can be easily synthesized on those rectangles directly. For convenience, in the following discussion, we call each surface component with its conformal parameters a *conformal geometry image*.

5.4 Uniform Texture Synthesis

Global conformal parameterization on a 3D surface (see Figure 5.2) induces conformal geometry images (see Figure 5.4(a)(b)), which allow textures to be easily mapped to the surface without angular distortion. Unfortunately, the area stretching of textures is unavoidable, as is shown in Figure 5.3. Ideally, we want to preserve both the angle and the area of the texture on the surface, that is, we want to find an *isometric* parameterization. Although in theory this is definitely impossible, in practice, we are able to improve the texture synthesis method to make it as isometric as possible.

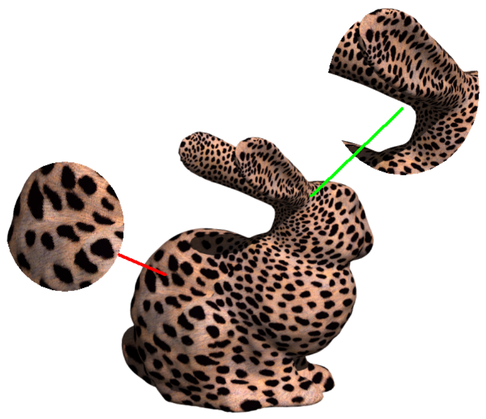


Figure 5.3: Nonuniform texture on a surface. It is generated by global conformal parameterization, uniform texture synthesis on 2D geometry images and texture mapping.

We propose a multi-scale texture synthesis method to generate uniform textures on the surface. This method synthesizes nonuniform textures on a 2D geometry image by considering the area stretching factor (the inverse of the conformal factor in Equation 5.1) in order to obtain the uniform 3D textures. The estimation of the area stretching factor on the conformal geometry images will be introduced first, and then the details of our multi-scale synthesis algorithm will be described.

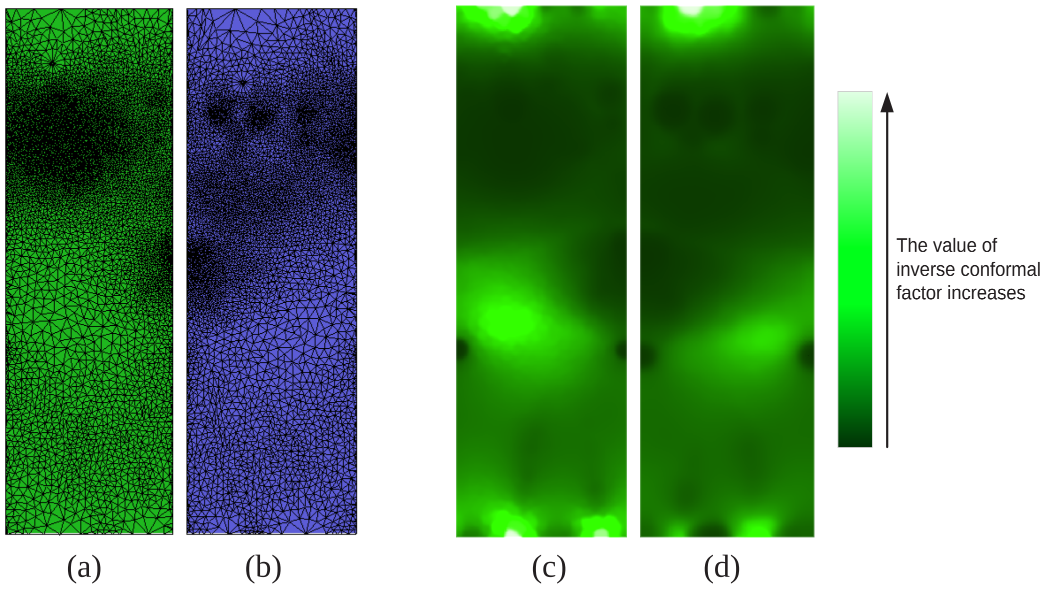


Figure 5.4: Conformal geometry images (a) and (b), and corresponding inverse conformal factor fields (c) and (d).

5.4.1 Estimation of Conformal Factor

The conformal factor indicates the amount of area stretching from the 3D surface to the 2D parameter domain. Our goal is to calculate the inverse conformal factor field on the geometry image. The inverse conformal factor is τ, and $\tau = \frac{1}{\lambda}$, where λ is the conformal factor in Equation 5.1. If the area shrinks from the 3D mesh to the 2D plane, τ is smaller than 1, otherwise, τ is larger than 1. This field will be used to choose the appropriate scale level of the sample texture when we synthesize textures on certain regions of the geometry image.

First, we normalize the parameters of each conformal geometry image. Then we choose the maximal size C for each dimension of the synthesized textures. The size of the output texture is simply the product of C and the normalized parameter for each geometry image. The size of the output texture affects the speed and the quality of the synthesis, and also the texture feature size on the surface. For all results shown here, we set C to be 1024.

By using Equation 5.1, the values of τ on the vertices are easily calculated directly from the geometry image with the original mesh connectivity. The τ values of the other texels are then interpolated using a Gaussian radial basis function (RBF). The calculated inverse conformal factor fields of two geometry images are illustrated in Figure 5.4(c)(d). Here, whenever the color changes from dark green to bright green and finally to greenish white, the inverse conformal factor value increases gradually.

5.4.2 Multi-Scale Synthesis Algorithm

Most texture synthesis methods synthesize textures without or with quite simple size variations of texture features [126, 74]. In contrast, we use the conformal factor to control the local scale of the texture. Therefore, the output texture is still similar to the sample texture, but with different feature sizes in different regions.

Our multi-scale synthesis algorithm is based on the patch-based texture synthesis method. Although pixel-based synthesis methods or hybrid methods should also work, better quality can be obtained using patch-based method according to our experiments. We put equal sized texture patches in the order of image quilting [34], and use the graph cut algorithm [74] to hide the seams of neighboring patches. The patch size is chosen according to texture features depending on the input texture, we use 30 to 50 pixels as patch width for the results in this chapter. In the synthesis process, we choose a patch, not just from a single sample texture, but from multiple scale levels of the sample texture. First, we calculate the average value of the inverse conformal factor in the region covered by a patch; Then we decide an appropriate scale level based on this average value. From the sample texture of that scale level, we find the best matched patch to fit the neighboring patches and put it on the output texture.

Multi-scale Sample Textures

In order to preserve memory and improve speed, we store a certain number of discrete scale levels of sample textures (see Figure 5.5). We call the enlarged sample texture the *high-scale sample texture*, and the minified texture the *low-scale sample texture*. For better quality, the scale between neighboring levels is not a power of 2. The parameters in our algorithm include the highest scale, the lowest scale, and the desired levels, which can be specified by the user and affect the size and the quality of the synthesized textures. Different scale level textures are then generated by cubic interpolation.

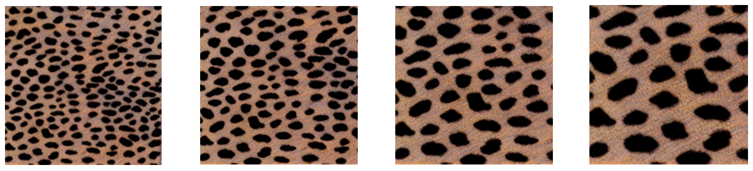

(a) (b) (c) (d)

Figure 5.5: Multi-scale sample textures. From (a) to (d), scales of sample textures increase gradually.

For regions with higher inverse conformal factors, higher level sample textures should be chosen, because the texture mapping will shrink the texture. Similarly, for regions with lower inverse conformal factors, lower level textures should be selected, because texture mapping will enlarge the texture.

The lowest level needs to be determined with caution, because the sampling rate is reduced when the texture is minified. Depending on the feature size of the texture, important features may be lost irrecoverably if the sampling rate is set below a certain level. We place our original sample texture close to the lowest level to lower the risk associated with downsampling. In contrast, high scale level texture does not suffer from this problem, and can be used safely.

Preserving Boundary Consistency

Since the surface is segmented and mapped to more than one geometry image, the boundary consistency problem needs to be addressed carefully. When mapping a segment (see Figure 5.1(d)) to its conformal geometry image, boundaries on the segment are mapped to boundaries on the geometry image, respectively. Figure 5.6(a) shows the mapping of a segment (segment 1) to its geometry image (geometry image 1). Figure 5.6(b) shows the boundary correspondences of this geometry image to another geometry image (geometry image 2), due to an adjoining segment. Corresponding boundary parts are neighbors in 3D space, and therefore must have consistent textures.

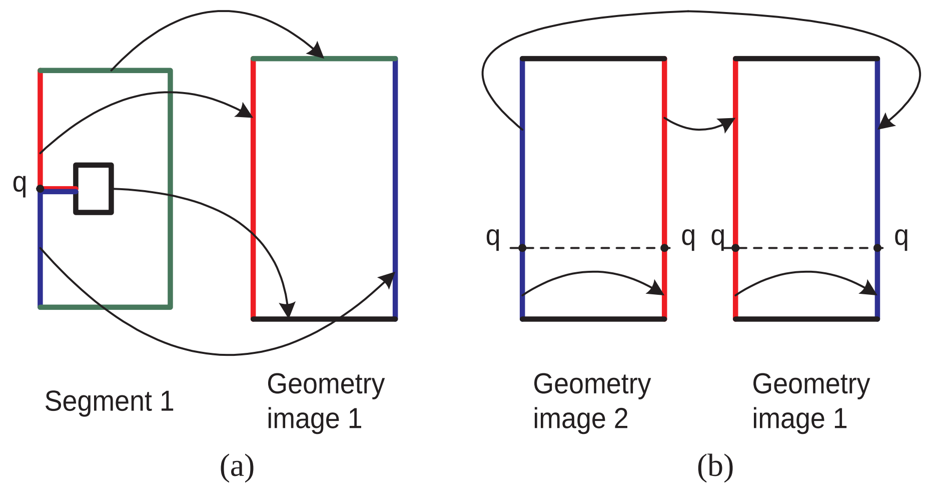


Figure 5.6: Boundary problem. (a) Corresponding boundaries on a segment and its conformal geometry image; (b) Corresponding boundaries on two geometry images. q is the zero point.

Our solution to synthesize textures consistently across the corresponding boundaries is as follows. First, we add margins to geometry images which have boundary parts corresponding to other boundary parts, as shown in Figure 5.7. Here, $P1$ to $P4$ are margins we added. During synthesis, after textures on $P1$ of geometry image 1 is synthesized, the textures are copied to fill $P1$ of geometry image 2. Then, when synthesizing textures on geometry image 2, for patches overlapped with margins, the matched patch will be chosen with additional constraints, treating the overlapped parts as already synthesized pixels.

Therefore, the patch-based synthesis algorithm is slightly modified to cope with different patch-overlapping situations, which solves the boundary problem. This way, our patch-based synthesis algorithm can easily generate periodical textures, which is quite useful in texture mapping as well.

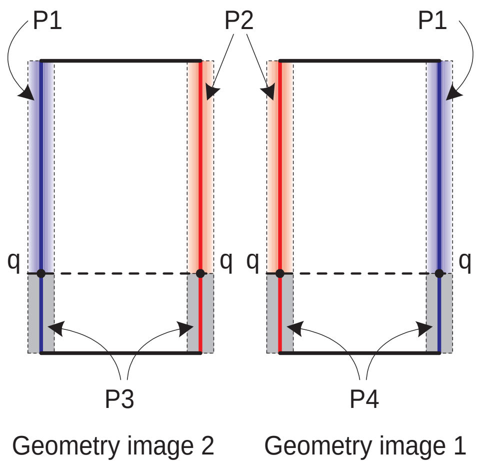


Figure 5.7: Consistent boundary synthesis, by adding margins and copying boundary texture patches. q is the zero point.

5.4.3 Results

Our texture synthesis results are demonstrated in Figure 5.8, 5.9. When we synthesize uniform textures on geometry images without considering area stretching, the texture feature sizes on different regions on the surface are highly non-uniform. In contrast, by using our multi-scale synthesis method, textures on the surface are quite uniform.

5.5 Quasi-Isometric Texture Mapping

The multi-scale texture synthesis method just presented modifies the textures directly to improve the uniformity of the synthesized texture on the surface. In contrast, the method for texture mapping, introduced in this section, revises the parameters instead.

In theory, it is impossible to make the parameterization preserve both the angle and the area. In that case, the parameterization would be an isometry, with a surface of zero Gaussian curvature (that is, a flat surface). But what we can accomplish is to make the parameterization on the interior of one component as isometric as possible and in return sacrifice some of the angle structure along the boundaries. We apply a mass-spring method to achieve this *quasi-isometric* parameterization, which is close to the desired isometric one.

Figure 5.10 illustrates the basic idea. The original conformal geometry image has a highly non-uniform density, whereas preserving the angle. After the process, the mesh (b) with quasi-isometric parameters has more uniform density, but the boundaries are distorted. Hence, the boundary consistency is sacrificed. On the other hand, the stretch-minimizing method of Yoshizawa et al. [151] fixes the boundary vertices and therefore can keep the boundary rectangular, but the anisotropic texture stretching is considerably higher.

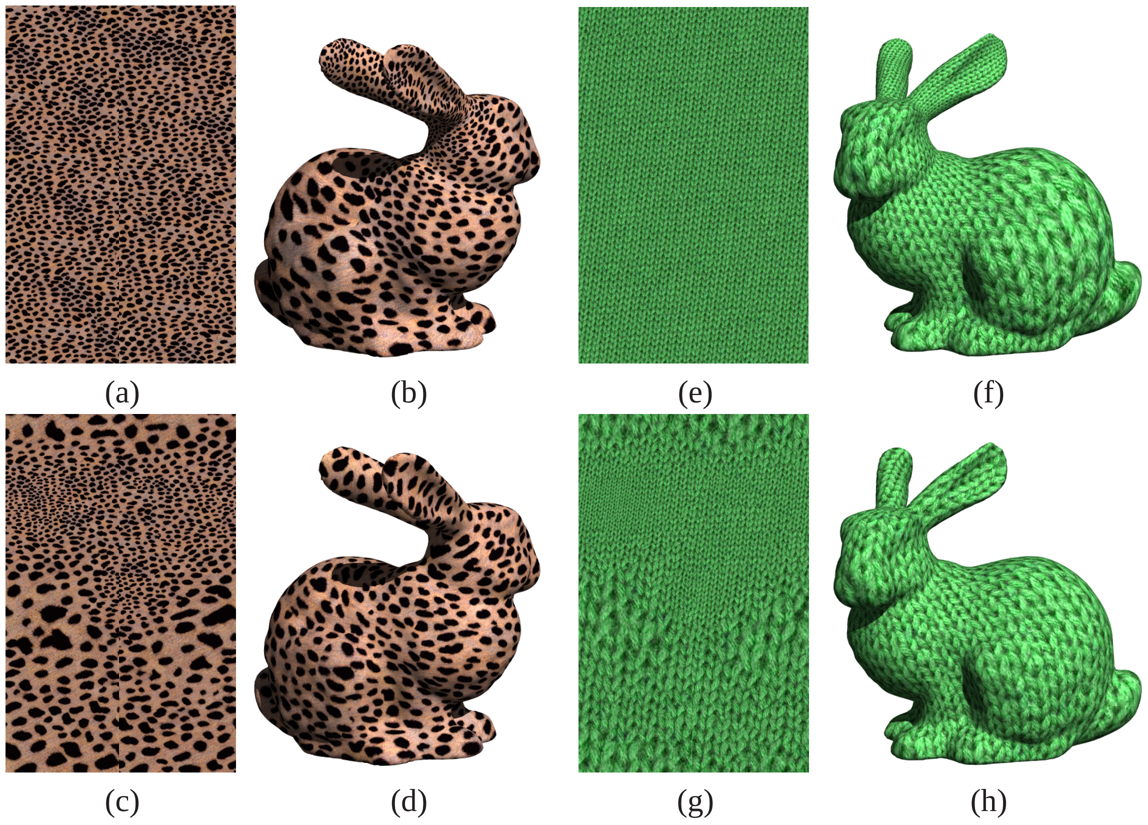


Figure 5.8: Multi-scale texture synthesis results. (a) and (e) Uniform texture synthesized on geometry images without considering area stretching factor, (b) and (f) Nonuniform texture by mapping (a) and (e) onto 3D surfaces, (c) and (g) Nonuniform texture synthesis considering area stretching factor, (d) and (h) Uniform texture by mapping (c) and (g) onto 3D surfaces.

5.5.1 Mass-Spring Model

The mass-spring model is carried out on the conformal geometry images using the original mesh connectivity. The mass-spring system is modelled as follows: each vertex is treated like a node and each edge as a spring. The motion of all nodes is confined to the 2D parameter plane.

We denote the parameterization of the conformal geometry image as $\phi : U \to R^2$, where U is the conformal geometry image. Then, the mass-spring evolution can be formulated as $\delta\phi(v) = \epsilon\mathbf{F}(v)$, where ϵ is a constant carefully chosen to ensure no flipping of triangles. In practice, ϵ is inversely proportional to the maximum magnitude of the force field. Here, $\mathbf{F}$ is the *external force*, and calculated as

$$\mathbf{F}(v) = \sum_u \eta(u)\eta(v)(\phi(v) - \phi(u)), \eta(v) = \frac{1}{n}\frac{|\mathbf{r}(u) - \mathbf{r}(v)|}{|\phi(u) - \phi(v)|} \tag{5.2}$$

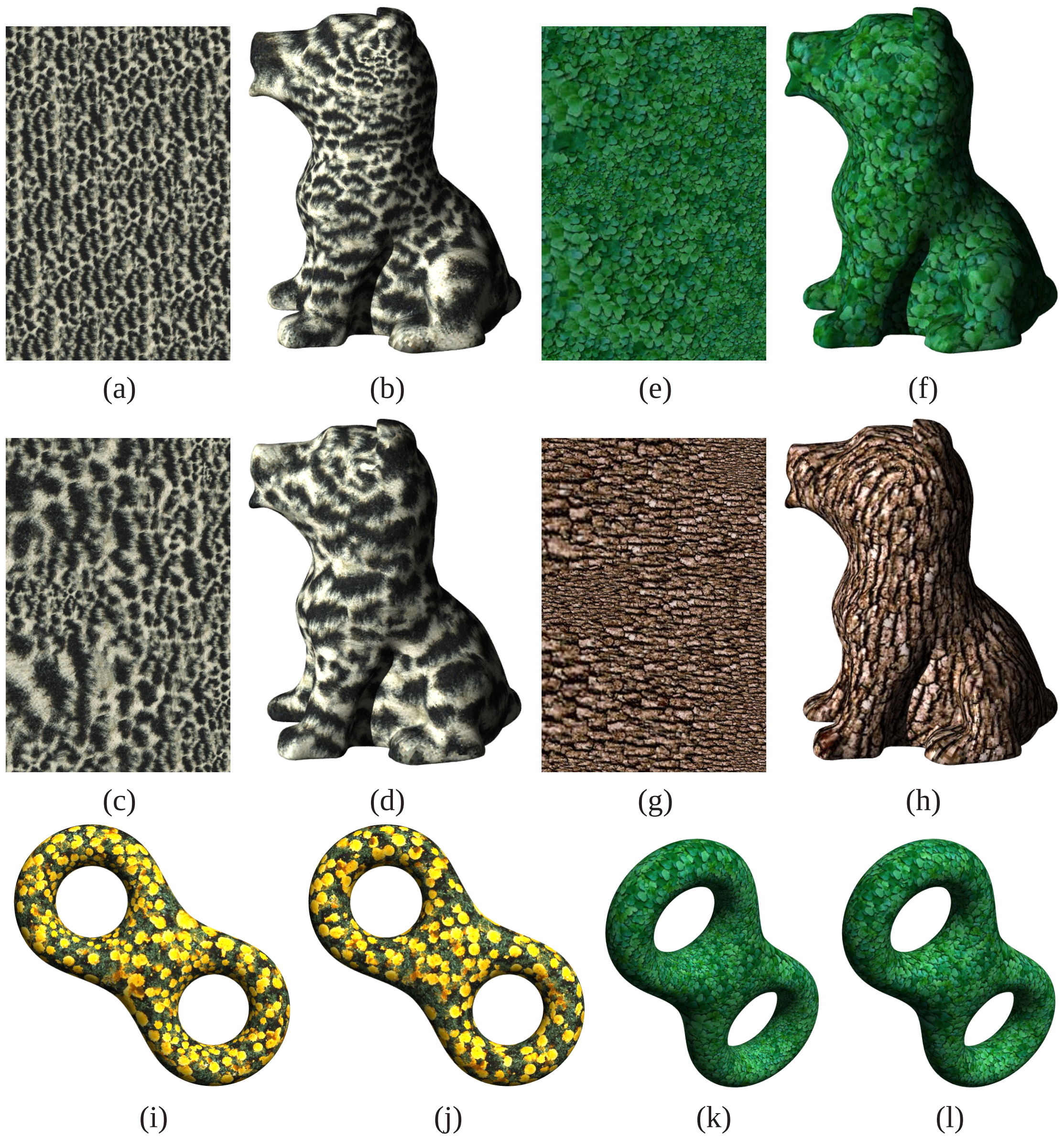


Figure 5.9: More texture synthesis results. (a) and (e) Uniform texture synthesized on geometry images without considering area stretching factor, (b) and (f) Nonuniform texture by mapping (a) and (e) onto 3D surfaces, (c) and (g) Nonuniform texture synthesis considering area stretching factor, (d) and (h) Uniform texture by mapping (c) and (g) onto 3D surfaces, (i) and (k) Nonuniform textures on 3D surfaces, texture features inside the handles are smaller than those outside, (j) and (l) Uniform textures on 3D surfaces. High resolution images as well as videos can be obtained at http://www.cs.sunysb.edu/∼lujin/paper/pg05/.

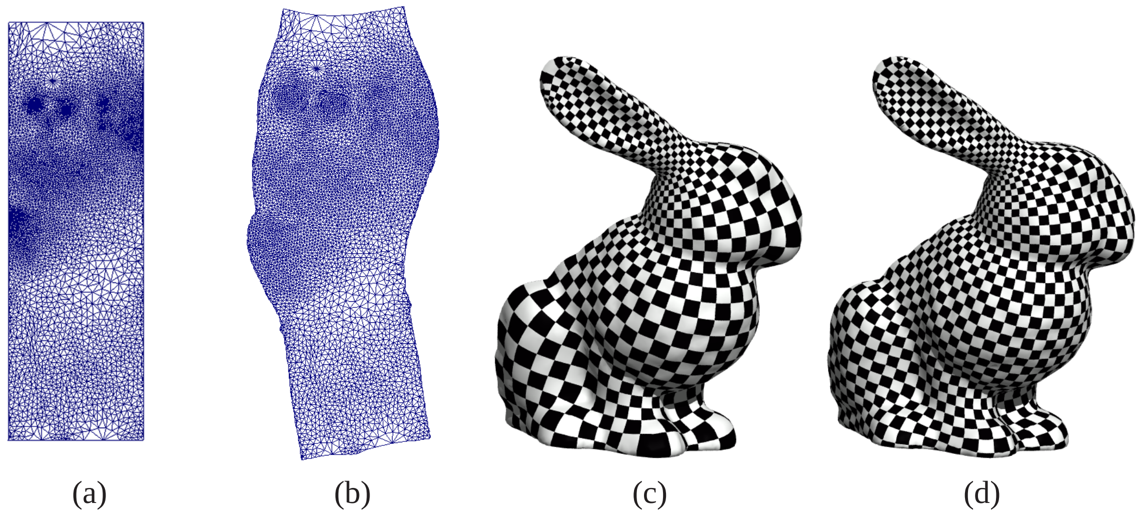


Figure 5.10: Mesh changed on mass-spring model. (a) Mesh on one geometry image, (b) Modified mesh with mass-spring relaxation, (c) Conformal texture mapping based on global parameterization, (d) More uniform texture mapping using our quasi-isometric parameterization.

where u runs through all neighboring vertices of vertex v, n is the valence of vertex v, and $\mathbf{r}(v)$ is the 3D position of vertex v.

In Equation 5.2, $\eta^2(v)$ is a discrete approximation of the conformal factor at v. Intuitively, the external force is proportional to the conformal factor, and expands the regions with high conformal factors. The nodes on the parameter domain with higher density will be expanded gradually and make the distribution more uniform, that is, the process will improve the parameterization to be closer to an isometry.

In our implementation, we use the mass-spring model code for arbitrary nodes in [33]. Figure 5.10 demonstrates the improvement of the parameterization using our mass-spring algorithm. The improved parameters are used for texture mapping. Figure 5.10(c) is the texture mapping result based on global conformal parameterization, while Figure 5.10(d) is the result after conformal parameterization and mass-spring relaxation, upon which the squares on the checkerboard become more isometric.

5.5.2 Results

Figure 5.11 compares the results obtained with and without our quasi-isometric parameterization method, for the task of mapping 2D textures onto 3D models. Figure 5.11(a) shows the outcome of an image-to-surface mapping via standard global conformal parameterization, while Figure 5.11(b) shows the result obtained when applying the mass-spring model to the conformal map first. We observe that the uniformity of the parameterization is greatly improved. And video can also be mapped to or synthesized on the surface with considerably better quality. One frame of our video (Matrix) on the surface is shown in

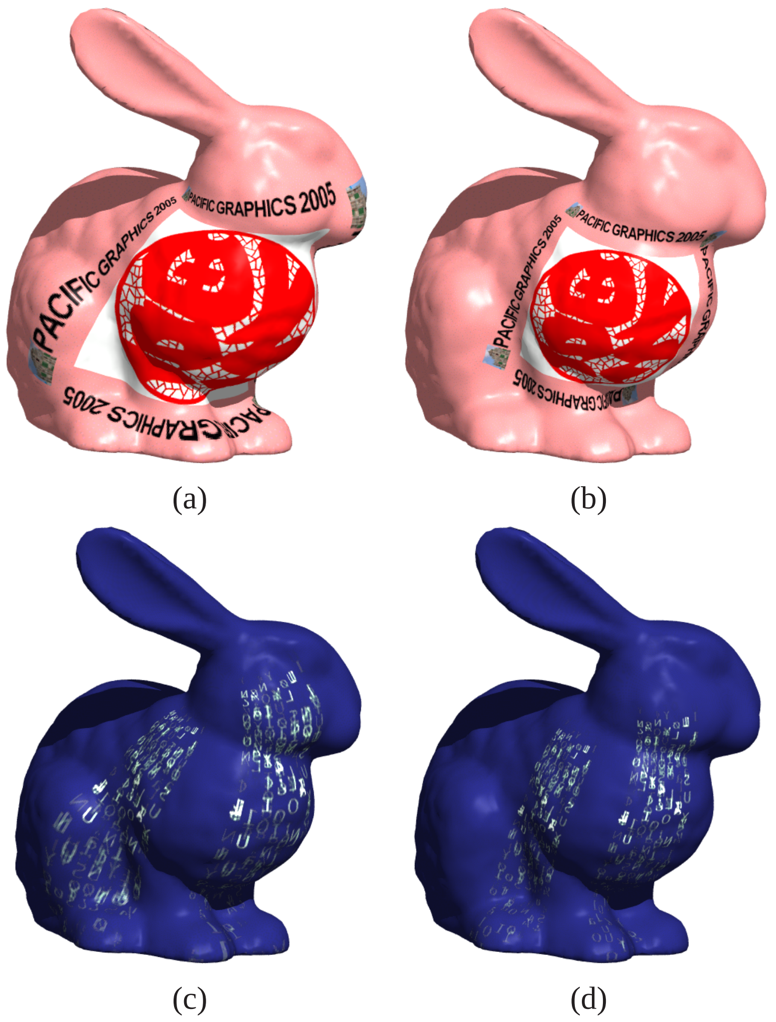


Figure 5.11: Texture mapping results. (a) and (b) Image on the surface, (c) and (d) Video on the surface, in which (a) and (c) are based on conformal global parameterization, (b) and (d) are based on improved parameterization using mass-spring method.

Figure 5.11(c) and (d). We should note that while the mass-spring relaxation process is relatively slow (about 1 hour for the bunny model), it only needs to be done once for each model, and after that the improved parameterization results can be reused for various image and video mappings. The extra cost for storage is minimal.

5.6 Discussion

In this chapter, we have presented novel methods to accomplish distortion-minimized texture synthesis and texture mapping on 3D surfaces. For this, we have augmented the conformal mapping approach, which preserves angular fidelity, with a process that controls the distortion of scale. For texture synthesis on 3D surfaces, it allows the synthesis process to be done intuitively in 2D space and, afforded by the conformal mapping, achieves global control over the mapping result. The synthesis result is locally angle-torsion free, while globally it is continuous. Further, we also devised a method based on a mass-spring model which offers a good tradeoff for angular distortion and size preservation in texture mapping. Both methods are conveniently implemented using conformal mapping, are simple and efficient, and are universal for arbitrary surfaces.

While we currently do not provide explicit controls to balance angular and size distortions, we plan to incorporate those in future work, using the updated conformal factor fields.

Chapter 6

Multiperspective Visualization

6.1 Introduction

A perspective rendering represents the spatial relationships of objects in a scene as they would appear from a single viewpoint. Recently perception principles have been applied to help find optimal viewpoints for volume datasets [9, 123]. Each rendering result corresponds to one viewpoint. Finding the minimal set of optimal viewpoints, and putting the result images side by side, the user can see all the important features and get a good overview of the data.

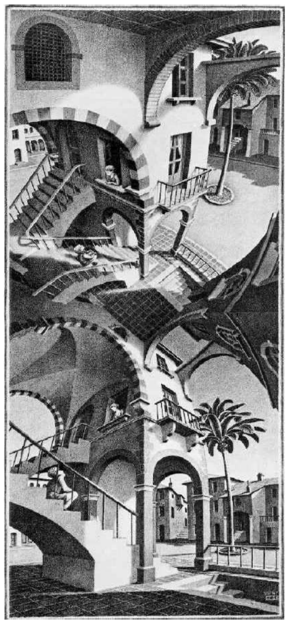


Figure 6.1: “High and Low” by M. C. Escher.

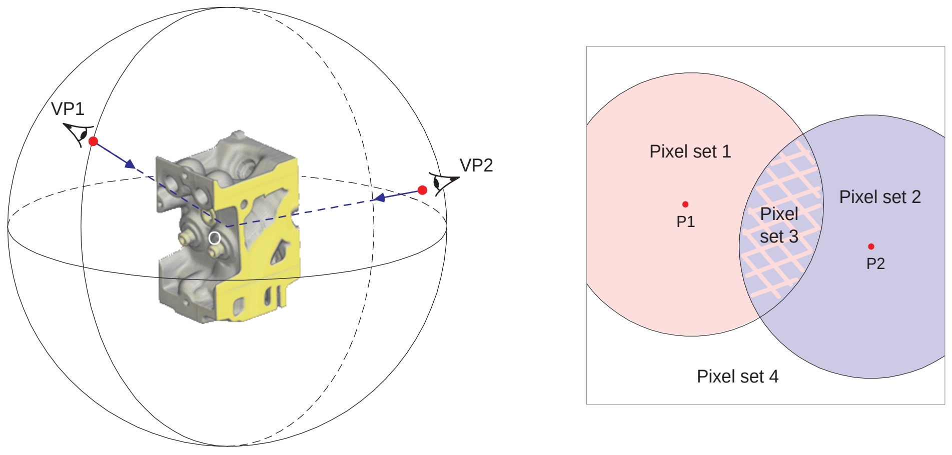
Multiperspective rendering is a powerful mechanism to unify different views into one image, while keeping the context between them alive. Multiperspective rendering has been employed both by artists, such as MC Escher [37], as well as in mainstream computer graphics [108]. M. C. Escher's work, shown in Figure 6.1, is a stunning multiperspective example. Multiperspective camera models have been introduced and employed in computer vision [160, 47], 3D surface and 2D image graphics [88, 41, 1, 130], and in Cel animation [147]. Yu and McMillan [152] showed that multiperspective images can be characterized as continuous manifolds in ray space under an appropriate parameterization. They used general linear cameras (GLC) [153], which describe all 2D linear subspaces of rays, as primitives for constructing multiperspective images. In order to generate good results, users have to select and lay out image fragments from different GLCs on the image plane, and perform transformations, including translation, scaling, and rotation.

Despite the incongruity of view, effective multiperspective images are still able to preserve spatial coherence. Inspired by the previous work, especially Escher's work, we propose to extend our warping lens to a multiperspective lens to widen the view of the data and provide the spatial relationship of features for the user. Here we present our preliminary multiperspective visualization approach and show some results. Our long term goal is to incorporate it into an entropy-maximizing view selection framework, to construct even more comprehensive views on the data in the gallery. Initially, we will couple this capability with the feature or view specification to have users take on a more designing role. Later on, however, we aim to derive a set of rules for view optimization to construct more comprehensive views automatically. Lastly, in all of these warp-based endeavors, Tufte's rule of "scaling with honesty" will be enforced by superimposing a grid, which will indicate the amount of local distortion of the volume, upon request.

6.2 Sphere-Based Multi-View Approach

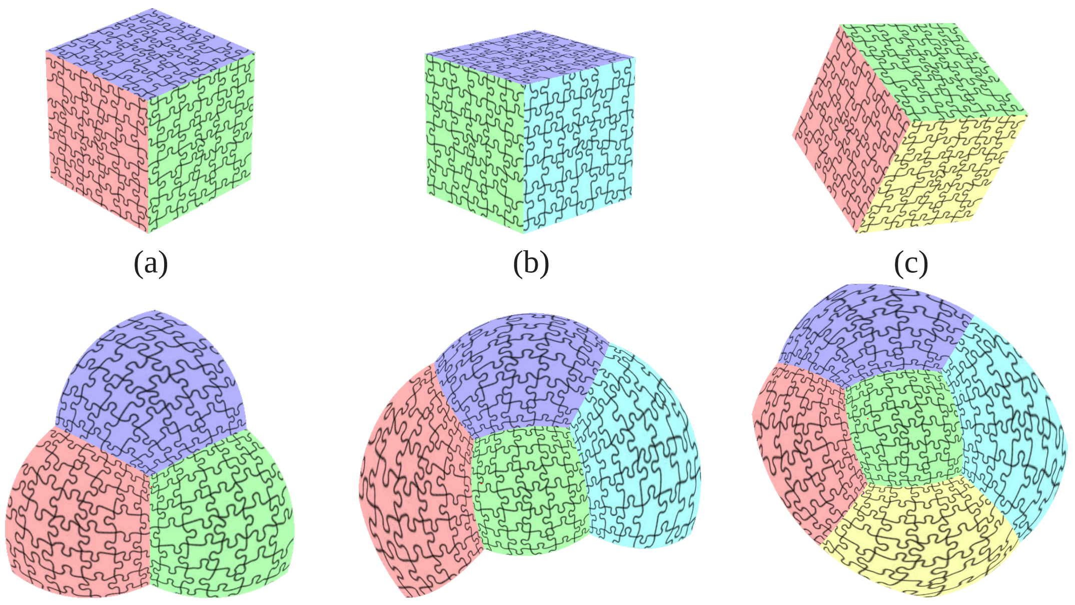
In our volume rendering framework, the user can interactively rotate and observe the volume around the center of the volume data. Our first try is to let the user chose two or three viewpoints of interest during his/her exploring of the data, then our approach automatically generates the image combining these views, without the user's further interactions, such as a manipulations on the image fragments [152].

Figure 6.2 illustrates the basic idea of our sphere-based multi-view approach. When viewpoints are only allowed to locate on the surface of a sphere S, which is concentric to the volume bounding sphere, the user can specify two viewpoints, such as $VP1$ and $VP2$, shown in Figure 6.2a. Our method warps the volume by computing the ray direction for each pixel on the image plane based on sphere S. First, the pixels ($P1$ and $P2$) corresponding to the centers of viewpoints ($VP1$ and $VP2$) are located on the image plane based on the relative positions of the two viewpoints, and their up and right vectors. For any pixel P within the range of the first viewpoint $VP1$, our approach maps the pixel P to a point SP on sphere S based on the coordinates of P and $P1$ on the image plane. Then the ray direction simply goes through P and the data center O. Therefore, as shown in Figure 6.2b, ray directions for all pixels located in Pixel set 1 or Pixel set 2 can be computed easily. No rays pass through the pixels in Pixel set 4. For pixels in Pixel set 3, which is the overlapping



(a) (b)

Figure 6.2: Illustration of our sphere-based multi-view approach. (a) All viewpoints located on the surface of sphere, which is concentric to the bounding sphere of the volume data, (b) Image plane.



(a) (b) (c)

(d) (e) (f)

Figure 6.3: Our multiperspective renderings considering visibility. (a), (b) and (c) Renderings with three viewpoints, (d), (e) and (f) Multiperspective renderings.

region affected by both viewpoints, interpolation is applied to determine the corresponding points on the sphere S. Finally, for each pixel, there is one ray passing through, therefore, the rendering can be done efficiently on the GPU.

6.3 Results

Based on the sphere-based multi-view approach we discussed above, we can also consider the visibility of each viewpoint to avoid rendering regions with too much distortions. Figure 6.3 shows our multiperspective renderings of a box with textures considering the visibility. Figure 6.3a-c are three renderings from different viewpoints. The user has to study three views to find the correspondences of features in order to get the spatial relationships of features. Figure 6.3d just shows how we warp the box from one viewpoint. Figure 6.3e,f are the multiperspective renderings for two and there viewpoints. Although there is much distortion, the spatial coherence is maintained.

We can also keep the feature or focus of interest undistorted, and only warp the surrounding data. Context is then shown in a distorted way. Figure 6.4 shows results generated by this method.

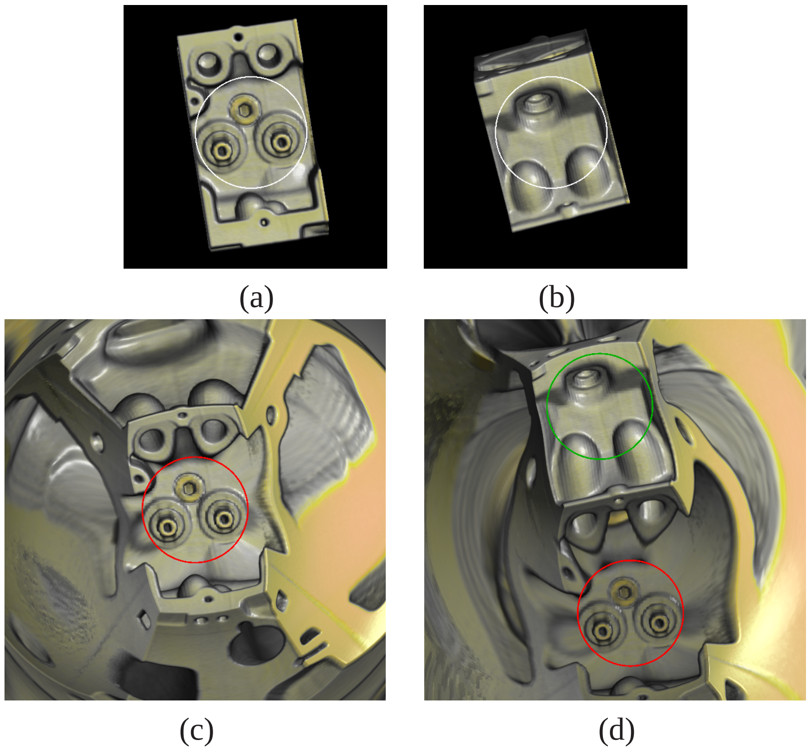


Figure 6.4: Our multiperspective renderings with focus feature protected. (a) and (b) Direct volume rendering from two viewpoints, (c) and (d) Multiperspective renderings.

6.4 Discussion

The primary concern with our multiperspective renderings is whether the approach causes too much distortion to be useful for visualization. Comparing our task with Escher's work, we find one fundamental difference. His work and many other successful multiperspective researches in graphics try to see objects outward, for example, the user looks at some buildings from the street on which the user walks. However, we observe the data from outside towards the inside, and move around the data, which introduces more distortions. Furthermore, Escher dealt with buildings, which have straight line features, such as walls and ceilings. These special features help to separate multiple views easily and efficiently. But in volume datasets, such features are hard to find and do not always exist. Another problem that appears in volume visualization is that due to occlusion the same feature can be shown differently for multiple views, especially in semi-transparent renderings, which cause difficulties to represent them correctly in the final image. All these make it a quite difficult task to design a good multiperspective technique for volume visualization.

Chapter 7

Conjoint Analysis to Measure the Perceived Quality

7.1 Introduction

The main purpose of visualization is to produce images that allow users to gain more insight into the illustrated data. This is a complex issue, depending on many factors of the visualization system, starting from human-computer interaction, to rendering speed, to rendering style and algorithm, and finally human perception and cognition. With the exception of the last component all of these factors have been designed by humans and many diverse technologies have emerged, and are still emerging, over the years. But in the end, human perception is the ultimate judge that determines which of these are the most effective. A popular focus of the field of visualization is the modeling and optimization via engineering and mathematics tools and frameworks, and often the designer/engineer him/herself judges the success of the method. Here, the easiest parameters to measure are rendering speed and memory consumption and others, which are all engineering quantities. However, in light of the importance of the last element in the chain, the human observer, a more recent focus has become to also conduct adequate user studies to measure the success of a proposed method. This practice is already common place in the field of human-computer interaction, and to a more limited extent also in information visualization, but less so in scientific and medical visualization. In essence, user studies are always considered burdensome since in many cases there are a large number of parameters and algorithmic alternatives, requiring many trials, that is, human subjects and experiments, to produce statistically significant results. This has been a major obstacle in assessing a method's success in terms of the human perceptive and cognitive system.

The pressing question is: can we make this task easier by introducing a more methodical and organized approach. For this it pays to look at other fields, especially those driven by heavy monetary investments. One then finds that user studies play a major and dominant role in product marketing, where it is important to tune the various parameters of a product before it is being launched to market or to determine its launch at all. Clearly, these studies must be conducted as thoroughly as possible, in order to maximize the outcome, but they also must be conducted as efficiently as possible in order to minimize the time, burden put on the participants in the study and samples needed to explore the vast parameter space in a statistically significant manner. A technique called conjoint analysis [48] is the answer to all of these design goals, and our goal is to make this technique accessible to visualization researchers and their specific domain setting. Visualization researchers are faced with the task that a large number of algorithms need to be compared. However, the number of algorithms is too large for a single user to compare/rank all of them in reasonable time and with reasonable accuracy. Fortunately, in many visualization areas, such as volume visualization, the algorithms are not strictly arbitrary but to some extent related; that is, they are all different incarnations of one parameterized algorithm and are obtained by fixing the parameter values. A comparison of the algorithms then leads to a ranking of the algorithms/parameter settings. This is essentially the same problem that market researchers face when eliciting consumers' preferences on substitute goods that can be described in terms of attributes and attribute levels. Conjoint analysis, as introduced above, is a well established family of questionnaire based techniques to elicit consumer's preferences. It frees the evaluator from the daunting burden of presenting the effects of all attribute levels to all users for evaluation but nevertheless allows statistically significant results.

This is a joint work with Joachim Giesen (Max Plank Instutitute, Saarbruecken, Germany). His research concentrates on developing a conjoint analysis technique as an extension of Thurstone's method of comparative judgment [124]. Like most techniques it is based on some assumptions (model), but it has the advantage that all assumptions can be tested. The model assumptions allow us to derive robust preference estimates from sparse data, i.e., every user needs to 'explore' only a small fraction of the large parameter space. We use this conjoint analysis technique to measure the perceived quality in volume rendering.

We demonstrate our conjoint analysis technique in four related studies that fit two important visualization purposes: visual aesthetics and the conveyance of detail. In this pursuit, we can gain further insights. For example, we determine the relative importance of the algorithm's parameters and their levels. This is important information if one has to tradeoff perceived quality against other objectives like time or file size. Conjoint analysis allows us to quantify these tradeoffs. We can also study the effects of age, gender, culture, or color deficiencies on users' preferences.

Our analysis framework is timely in light of the various recent efforts to optimize viewpoints [9, 58, 123], transfer functions [69], sampling intervals [145, 6], high-level appearance descriptors [115], illustrative rendering parameters [11], perceived salience [67], and others. All of these use mostly mathematical, engineering, but also sometimes aesthetics and perception-motivated arguments to devise their methods. Controlled user studies need to eventually decide which strategy is most effective and relevant for the human observer, especially in conjunctive terms. Furthermore, these user studies can also be helpful to fine tune the parameters of these methods, which may also be task and domain dependent.

In the following sections we first describe a typical multi-parameter volume rendering scenario, in which we generate all the volume renderings. The choice based analysis approach is then introduced, and the overview of our framework is presented. Finally we show some insightful results, which can be valued as indicators for the analytical power of our framework that provides a guideline on how to conduct and analyze a conjoint study in the context of visualization algorithms evaluation/comparison.

7.2 Volume Rendering Scenario

As mentioned our goal was to measure the perceived quality of a visualization algorithm for different parameter settings. We have chosen a relatively standard volume visualization scenario to demonstrate our user study framework. For our study we used two data sets. The first data set FOOT is meant to cover the medical application domain, whereas the second data set ENGINE covers the engineering applications area. The ENGINE data size is $256 \times 256 \times 256$, and the FOOT data size is $154 \times 263 \times 222$. Using GPU-accelerated ray casting rendering, the visualization of the volume data set can be described in terms of the following parameters:

COLORMAP This parameter has three levels which correspond to different color maps that are applied for transfer function design. For all transfer functions, the alpha channel has been set to always reveal most of the object's structures, in order to suppress 'occlusion' to act as an independent variable.

RENDERING This parameter describes the applied rendering mode and has five levels: DVR (Direct Volume Rendering), DVRNS (Direct Volume Rendering with No Shading, just compositing), DVRGM (Direct Volume Rendering with Gradient Modulation to highlight surfaces), XRAY (Colored X-Ray) and MIP (Colored Maximum Intensity Projection).

VIEWPOINT This parameter has six levels for the ENGINE and five levels for the FOOT data set. It describes the viewpoint under which the observer sees the object. Different viewpoints are chosen in such a way that most structures are always kept visible, again to prevent 'occlusion' from playing a significant role in the study.

RESOLUTION This parameter describes the screen resolution used for rendering. We render at the resolution of the data set and twice that. Note that in the end the image size was always 512×512 (the image rendered at reduced resolution, that is, at volume resolution, was scaled up with bilinear filtering).

STEP SIZE This parameter is the ray traversal increment (measured in voxel size), which has three levels, 0.2, 0.5 and 1.0.

BACKGROUND This parameter describes the color of the background and has five levels: BLACK, WHITE, DARK GREEN, DARK BLUE and YELLOW.

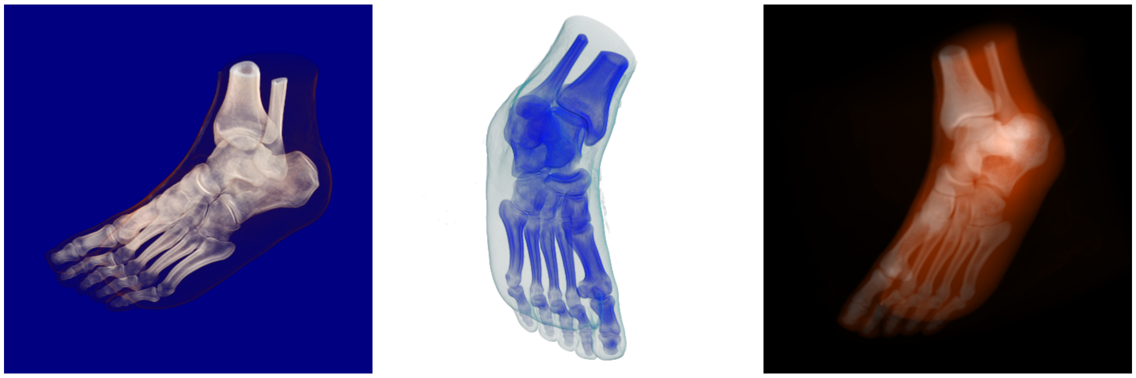


Figure 7.1: Renderings with different parameter settings.

Finally combining these parameters results in the 2700 ENGINE images (renderings) and in the 2250 FOOT images. Figure 7.1 shows some FOOT images.

7.3 Choice Based Conjoint Analysis

A class of items has a conjoint structure if it can be described by the Cartesian product $A_1 \times \ldots \times A_n$ of attribute sets A_i. The elements of the attribute sets are called attribute levels. An item a is then represented by a vector $(a_1, \ldots, a_n)$ with $a_i \in A_i$, i.e., by fixing the attribute levels. Conjoint analysis is a family of techniques for eliciting from a population of people their ranking (on some scale) of the elements in $A_1 \times \ldots \times A_n$, i.e., on the items. Conjoint analysis techniques can be distinguished by two (not independent) parameters: firstly the elicitation procedure, i.e., the way preference data are obtained from respondents, and secondly the way the elicited data are processed in order to derive a representation of individual or aggregated preference information (typically in form of a value or utility function).

In recent years choice based conjoint analysis has become the most popular conjoint analysis technique. It got its name from the method employed for elicitation, namely, preferences are elicited in a sequence of choice tasks. In a choice task a small number of items (typically between two and four) is presented to a respondent who has to state which one out of these she/he prefers most. Choice tasks are popular in market research since they resemble real buying situations and thus tend to provide the most reliable information.

There are many different ways to analyze the data obtained from several respondents and several choice tasks each, but any analysis method defines a *scale* on which the items are compared. A scale assigns to each item a number. In conjoint analysis there are essentially two types of scales used: on *ordinal scales* the numbers assigned to the items are their ranks in a linear order. Note that the nominal difference between ranks has no meaning. On *interval scales* an item is preferred over another if it gets assigned a larger number. Differences of the assigned numbers have a meaning on interval scales, but these scales have no *natural zero*. Note that translating all scale values on an interval scale has no effect.

Another difference in analysis methods is whether they define a scale for each respondent, or just a scale for a population of respondents (aggregated scale). Our analysis method defines an interval scale for a population of respondents.

7.4 Overview of the Framework

We use choice based conjoint analysis as our elicitation procedure, where each choice task was a paired comparison between two renderings, i.e., between two parameter settings. Note that the cognitive burden increases with the number of items from which to choose. Higher cognitive burden should result in poorer data quality. We decided to use choice tasks with the least cognitive burden, namely paired comparisons.

Perceived quality itself can be measured along different directions. We made this more explicit by asking two different questions: Which image do you like best? and Which image shows more detail? We will later refer to the first question as AESTHETICS and the second as DETAIL. Note that the second question is more specific than the first, which is fairly general. Each combination of data set and question is considered as a different study, i.e., we conducted the four different conjoint studies [ENGINE, AESTHETICS], [ENGINE, DETAIL], [FOOT, AESTHETICS] and [FOOT, DETAIL].

Joachim Giesen et al. conducted the user study at an exhibition and elicited data from 786 visitors. Then the data analysis method is applied to define an interval scale for a population of respondents from their choices in paired comparisons. First scale values for all levels of a single attribute are estimated. To this end any paired comparison is interpreted as a comparison of just the two levels of the given attribute that are present in this comparison, ignoring differences in the levels of all other attributes. This method is then applied to all attributes to obtain scale values for all their levels. And a rescaling method is proposed to make the scale values for levels of different attributes comparable. Finally, the scale value of an algorithm, i.e., complete parameter set, is just the sum of the scale values of the parameter values. Please refer to our conjoint analysis paper for details on user studies and the data analysis method.

7.5 Results

From our four visualization case studies, we obtain some insightful results, including relative importance of parameters, most preferred levels, dependency on the respondent, dependency on the data set, dependency on the question, and parameter interdependence. Here we will discuss the first two in details. Readers are recommended to read our conjoint analysis paper for more results.

Relative importance of parameters The standard deviation for an attribute can be interpreted as the relative importance of this attribute. In our setting the attributes are the

Table 7.1: Rank order of the parameters used in our four studies. The rank order is derived from estimated variances (shown in brackets).

		AESTHETICS		DETAIL	
FOOT	1.	RENDERING-STEPSIZE	(0.31)	RENDERING-STEPSIZE	(0.52)
	2.	COLORMAP-BACKGROUND	(0.3)	COLORMAP-BACKGROUND	(0.35)
	3.	VIEWPOINT	(0.14)	VIEWPOINT	(0.12)
	4.	RESOLUTION	(0.05)	RESOLUTION	(0.08)
ENGINE	1.	RENDERING-STEPSIZE	(0.56)	RENDERING-STEPSIZE	(0.77)
	2.	BACKGROUND	(0.19)	RESOLUTION	(0.09)
	3.	RESOLUTION	(0.12)	VIEWPOINT	(0.08)
	4.	VIEWPOINT	(0.09)	BACKGROUND	(0.05)
	5.	COLORMAP	(0.05)	COLORMAP	(0.01)

parameters of the visualization algorithm. Using the estimated standard deviation we get the rank ordering of the parameters as shown in Table 7.1. From these results it is safe to conclude that overall the rendering mode (combined parameter RENDERING-STEPSIZE) is the most important parameter. The importance of this parameter is relatively higher for the DETAIL than for the AESTHETICS question. A second important parameter is the color scheme used (or the background), although this finding is not as pronounced. The viewpoint is somewhat important (mostly for the FOOT), while the resolution is somewhat important for the ENGINE. The other parameters are relatively unimportant, at least at the levels we have measured.

Most preferred levels The results of Tables 7.1 as well as Figure 7.2 reveal a good deal of useful information. We observe that the algorithms XRAY and MIP are not considered useful by our respondents (but note that these were non-expert viewers C doctors can see a lot more in those renderings). The DVRGM algorithm performs (slightly) better than DVR, which performs better than DVRNS. This ranking shows that the more structure enhancement, the better.

There is also a clear preference for achromatic backgrounds. Only blue is also found to be somewhat useful, possibly because blue is a monocular depth cue in that colors very far away shift to the blue spectrum, or because of the background shade of blue and the object. Highly saturated backgrounds are generally disliked. Interestingly, there are also differences between the two achromatic backgrounds: a black background is considered more aesthetic, whereas white seems to show detail better. This is particularly true for the ENGINE which is overall a more complex data set. It is most likely also an object that is less familiar to the respondents. Therefore they require more detail; higher resolution is also more important (than for the less complex FOOT).

For the ENGINE, the color map applied does not seem to matter as much, but for the DETAIL question, the FOOT (bone) is strongly preferred to be seen in a color resembling that of bright bone (skin grey). This indicates that for object inspection, viewers like to see objects in colors that are most natural and at the same time bright (when such a color is generally agreed on), but for objects less defined in that respect the color choice is a matter

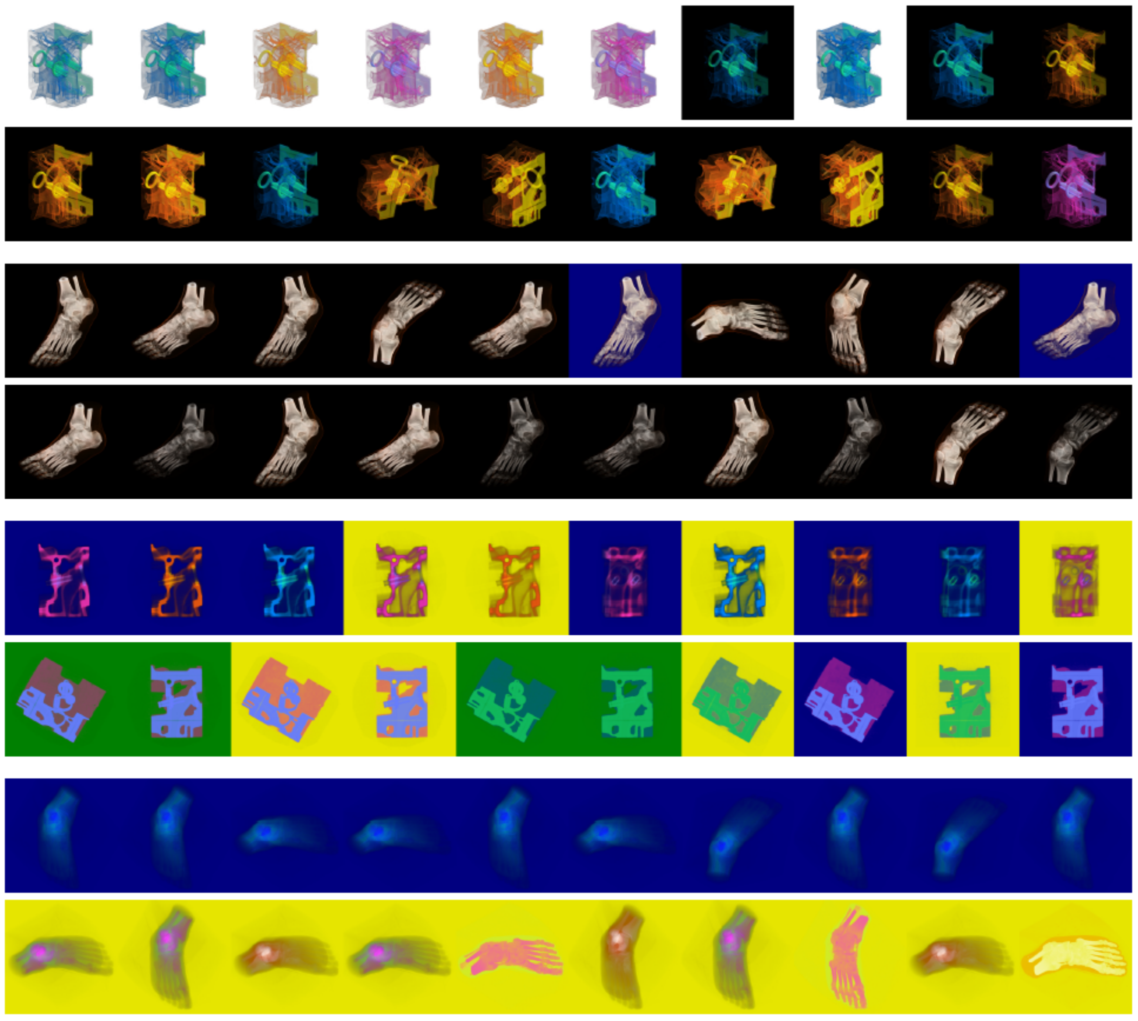


Figure 7.2: Best and worst renderings for our four conjoint studies. (a) and (b) Best ten renderings (ranking decreasing from left to right) for ENGINE and FOOT respectively, (c) and (d) Worst ten renderings (ranking increasing from left to right). Top row of (a)-(d) is for DETAIL, bottom row of (a)-(d) is for AESTHETICS.

of taste (as is the case for the ENGINE), as long as they are bright and define contrast well. In the AESTHETICS category viewers still preferred a natural color (for the FOOT), but the brightness condition was no longer so important (by definition of the task criterion).

An interesting observation can also be made with respect to the viewpoint. A common feature is that viewers prefer to see objects at oblique angles, which generally gives objects a more three dimensional appearance and also reveals more features (such views are also used for product advertisements). But the engine was in general preferred to be situated as standing on a surface – the views where the engine was rotated at an arbitrary angle (and appeared as it were flying towards the viewer) were rated low. On the other hand, the foot was acceptable at most orientations. We believe that the ‘flying’ engine was deemed unrealistic, and perhaps even dangerous and therefore unappealing, while a foot is seen commonly at general orientation in real life (just not as a bone).

7.6 Discussion

We took first steps to demonstrate that conjoint analysis can be a useful and efficient tool to gauge influences of a rich set of rendering parameters on human perception in visualization tasks. We believe that the data analysis technique that we have developed here can even be used to analyze data gathered in the first phase of the 'human-in-the-loop' method of House, Bair and Ware [55]. Note that our analysis method only needs paired comparisons between renderings that even can be obtained from measurement of how well a test person performs a task on different renderings.

We have tested the framework within a familiar visualization environment, a parameterized volume renderer, where we have taken great care to reduce the effects of competing adverse parameters, such as image size and occlusion, without reducing the effects of the relevant tested parameters, such as color schemes and rendering precision and algorithm. In this process we verified a few known results, such as the effect of rendering fidelity, but we also teased out some lesser known but important results, such as preferred object orientations, color schemes, and the relationship of step size and rendering modality. Another interesting finding is that our conjoint analysis method can help to resolve tradeoff decisions. In particular for the DVRGM algorithm it is not necessary to go down to step size 0.2, step size 0.5 even gives perceptually better results. That is, it is often not worthwhile to spend the extra computing time required by smaller step size (time-quality tradeoff). A second tradeoff concerns perceived quality and file size, which is to a large extent determined by the resolution. Our methods allow us to quantify this tradeoff, i.e., to answer the question of how much quality gets sacrificed when the file size (resolution) decreases.

In the future we will conduct more user studies and apply our framework on other visualization tasks. In next chapter, We will discuss another perception related work: color design for visualization. Conjoint analysis technique may help to test the efficiency of color designs, or find user preferences on color schemas for certain visualization tasks.

Our vision is to create a (web based) user study analysis suite that can be used by researchers to conduct and analyze multi-parameter user studies. Conjoint analysis should be an integral component of such a suite.

Chapter 8

Color Design for Visualization

8.1 Introduction

Recent years have seen multifarious efforts to better integrate and exploit properties of human visual perception and cognition into visualization design. Illustrative rendering techniques have been developed that render the scene at different levels of abstractions or in different rendering styles, ranging from sparse stroke-based depictions to full-scale volume rendering [11, 13, 14, 122]. In these approaches, the levels of abstractions are most often controlled by a task- or object-dependent importance parameter [134]. Another perception-motivated strategy is to guide viewer attention to salient features [67]. However, it is interesting to note that color has never played a major role in these efforts. There is no system so far that incorporates rules from color design directly into the visualization engine. One system that exists, the PRAVDA system by Rogowitz and co-workers [113, 5] was more purposed for the display of continuous scalar data with transfer functions, and not for the segmented data commonly used in illustrative visualization.

In our work, a transfer function is considered a general mapping of a numerical parameter value into a visual parameter value. The bulk of work in transfer function design in volume visualization has mainly concentrated on the specification of the A (opacity) portion of the transfer function, in order to capture shapes and contours of iso-surfaces at great fidelity. On the other hand, the RGB portion of the transfer function has in most cases been guided by personal preferences of the system's user or even just random assignments.

While color design has received less attention in the visualization community, despite the existence of two books on the topic [135, 119], professional designers and artists are quite cognizant of rules that guide the design of color palettes, not only from an aesthetic point of view but also from an attention-guiding, salient one. Here, the notion of color harmony is only one of these fundamental design rules, which has been more motivated by aesthetic arguments and has found recent application in the computer graphics literature for those considerations. On the other hand, visualization is not only concerned with providing a pleasing image – visualization also has a mission, that is, to help the user to gain quick and accurate insight into the visualized data. Good visual aesthetics makes this task an enjoyable one and therefore reduces stress, while exploiting the perceptional rules can often aid the salient cognition process.

Our work is motivated by rules established in the classic color design literature. The framework described captures these rules into a knowledge-based system which then provides appropriate colorizations based on user preferences, importance functions, and scene composition. The scope of the system is both volume and information visualization. It is important to note that we only consider the effects of color, and not those of illustrative style and the combination of these. We believe that a decoupling of these visualization parameters is necessary to develop a rudimentary framework, which can then later be applied in the context of stylistically more advanced systems.

Section 8.2 briefly presents the highlights of previous work relevant to our, and Section 8.3 gives a system overview. In Section 8.4, we summarize our design in form of expert system rules and describe design details. Section 8.5 shows all applications and results of our system.

8.2 Related Work

In visualization, image and volume datasets typically come in form of 2D and 3D arrays of scalar densities, which are mostly obtained via simulations or scanning (CT, MRI, etc). Due to the human visual system's excellent sensitivity to variations in brightness, grayscale displays are already quite adequate to perceive the inherent variations of densities, at least in a local sense. At the same time, the illumination contrasts generated in shading can provide excellent shape cues in 3D displays. However, the range of grey levels distinguishable by humans is limited to only about 100 [110], and distinguishing different objects or features can be quite difficult with grayscale alone. In addition, such grayscale displays often also lack aesthetic appeal, which may lead to a reduction of interest as well as recall in the human observer – after all, the world around us is in color. Mapping the densities to color can help overcome these problems, and it also can be used to highlight density contrasts, guiding the viewer to these areas.

In scanned and simulated datasets the scalar densities usually map to certain material properties, and it is desirable to preserve these variations as intensity modulations and, apart from highlighting, only use color for better object differentiation and labeling. Here, the number of colors that can be used for this purpose has been studied by Healey [52], who optimized the separation of label colors in CIE LUV space and then conducted user studies on target identification capabilities. He found that users did quite well when the number of colors was 5 or less, while greater numbers (7 to 9 were studied) posed difficulties. These deficiencies are also partially rooted in the limitations of human working memory.

Also limiting the maximal numbers of colors is the phenomenon of simultaneous contrast which changes the appearance of a color based on its surround [135]. It widens the required "safety margin" of a given color and limits the number of these. This problem is in some way related to the color mixing effects that can occur when two or more semi-transparent surfaces overlap, but these compositing effects can be much more severe as they can result in radically different colors, which can be both disturbing and distracting at the same time, especially if this color has already been reserved for another object. Our system provides a solution for this.

In essence, even across cultures, colors have been classified into 8 classes (in addition to white, black, and grey): the basic opponent color pairs red and green, blue and yellow, as well as purple, pink, orange, and brown [7]. But revealing quantitative properties by color is difficult, because no learned ranking of these colors exists, and for this reason, the popular rainbow colormap is considered a poor choice [119, 113, 5]. The work by [5] has shown that the human visual system can only differentiate hue for low-frequency variations, while high frequency variations, such as fine detail, are best resolved by luminance. This in some sense generalizes the dedicated use of color as a label. In other work, [112] also demonstrated that colormaps should preserve a monotonic mapping in luminance. In this vein, some researchers suggested [81] that the best mapping results from a straight line through a perceptional color space such as CIE LAB. Note that all of these approaches involve all three perceptional components of color: hue, saturation, and brightness. Our system also makes use of all three color components, but in a more intuitive yet free-form fashion, where users can only pick the most intuitive component, the hue, and the system optimizes the other two.

Finally, color can also be used as a means to focus attention [4], which is known as *pop-out*. This can be in addition to other cues, such as shape, size, motion, and blur [71]. Pop-out exploits pre-attentive cognition, which translates to involuntary awareness of a feature within a ms-time interval. In this context, an important finding is that pre-attentiveness is strongly related to the vividness of a color patch, as well as its size and the degree at which it differs from the vividness of the surrounding colors [17]. This visual pop-out parameter is attractive since it does not require extra colors (that is, hues) to be chosen to generate attention effects, avoiding an overloading of human target tracking capabilities.

8.3 System Design Goals and Overview

We have already mentioned the role of color as a means to increase aesthetics in a display, and in fact, this topic has been studied for a long time in the arts and design literature. There, a number of landmark texts on color design have been published [56, 146], and these texts provide a wealth of information with great insight on human perception of color and the aesthetic aspects of it. Much information is also available in the books by Stone [119] and Ware [135]. One popular design aspect is color harmony. Color harmony is a fairly old concept, already expressed by Goethe and other greats of that epoch, and a quantitative representation was described by Moon and Spencer [95]. This representation was based on the Munsell color-order system [97], which consists of three perceptional coordinates: hue, value (lightness/brightness), and chroma (colorfulness). In search of an intuitive 2D representation for visual designers, Itten then arranged the harmonic colors into a color wheel, which reduced (flattened) this color space to mostly variations in hue. Later, Matsuda [92] employed Itten's wheel, in conjunction with extensive psycho-physical studies, to introduce a set of 80 harmonic colors schemes. These were the basis of the recent automated image color harmonization system by Cohen et al. [24].

In fact, it was this automated system that inspired our work. But color harmonization is only one aspect of aesthetic design. There are many more rules that govern good visual color design and these are very relevant to visualization, which includes information visualization as well. Many of the parameters determining good visual design are directly measurable, both in the underlying data and in a generated viewing configuration, and can therefore be captured into an automated color design system. Here it should be noted that such a rule-based system needs to have a much stronger analytical and computational component than interactive color palette tools [94] which require much more user interaction. In fact, for these tools the support for creative interactions is desirable since they are meant for graphics designers who demand such freedom in producing only a few sheets of artwork a day. On the other hand, while our optimizing system can also benefit that community, it is predominantly targeted for interactive image generation or rendering engines, where many images are produced and only little time is available to tweak the color composition of each. It also captures at least some of the knowledge of these experienced graphics designers, for everyone to use.

Another design goal addresses the need of visualization to guide the observer to the most important features of the data. A recent paper in that regard is that of [67] who employed an emphasis function based on the center-surround mechanism of the human visual system to enhance the visual saliency of features important within the visualization task. Thus, an automated color design framework must not only incorporate color design knowledge, but it must also be parameterizable by feature importance. While color is not the only way to encode the visual field, show similarity and difference relationships, and direct viewer attention, it is generally the best and fastest.

Finally, as mentioned before, such a system will embrace a strong interplay of the three perceptional color parameters, that is, hue, chroma, and brightness, and therefore an associated color harmonization method must also support all of these. However, in order to achieve this, a suitable extension to Itten's hue-based color wheel needs to be devised, which in turn also requires an extension to the automated color harmonization algorithm by Cohen et al. Since the concept of a color wheel is convenient and intuitive – which was most likely the reason it was invented for – this extension should be formulated as a post-process to rectify any imbalances of chroma and brightness during the harmonization process. This is most suitably executed in a perceptional color space, such as CIE LAB or Munsell.

Our system supports all of these goals, and similar to the recently emerging illustrative rendering engines it takes a feature-based approach. That is, a dataset is first decomposed into its semantic constituents, using available segmentation or grouping information. Then the user applies a color palette to assign hues, and the system optimizes chroma and brightness, taking into account the programmed visual design rules in conjunction with importance parameters and computed scene parameters, such as feature size, density ranges, and interactions. Thus, our system differs greatly from the ground-breaking rule-based system of [5], which was more focused on scalar data with a continuous field flavor.

8.4 The Computational Design Expert System

Color vision can be studied with two rather different goals in mind: *aperture color* and *surface color* [156]. Aperture color takes a more physics-based, wavelength-oriented approach to color vision, conducting experiments in very controlled laboratory settings. Test subjects compare small patches of color, embedded on black backgrounds and under exclusion of all other effects, such as lighting and surrounding scene. These types of experiments can explain the fundamental color matching properties of the human visual system very well. However, they are less suitable to explain the effects and interaction of colors within a more general, less controlled scope, as embodied by real-world viewing conditions. Studies that operate in these settings explore the aspects of surface color, which is more complex, varied, and medium constrained than aperture color. As a distinguishing example may serve the situation where one visits a paint store, armed with a carpet swatch, seeking to select a matching wall color by ways of a set of similarly-sized store-provided paint swatches. In many cases the anticipated interplay is vastly different from the actual one, once the wall has been painted. This can be due to varied lighting conditions, but also be due to the different actual proportionate sizes of the two matched color surfaces, and the effects of other colored items in the environment – the living space in this example.

A number of papers have appeared in the field of visualization that have studied the effects of color, but they did so more from an aperture color perspective [112, 68]. These efforts have produced valuable insights for transfer function design in scientific visualization and the rules postulated there are in frequent use today. The focus of these efforts was to ensure good delimitation (contrast) of fine features in continuous data fields. We share the goal of these earlier works, but we add the more holistic aspects of surface color science to this rule set. This allows the incorporation of various principles of vision psychology, such as pre-attention, emotion, and even aesthetics, in order to better control the specific visualization task at hand.

In cognitive science, color is a psychological experience (that is, we can also imagine it) of three orthogonal components: quality (the hue), quantity (the lightness, which is perceived brightness), and purity (the chroma or saturation). To conceive our automated color design system, we can take advantage of a well established body of knowledge in color design and perceptional science. This alleviates us from the task of having to conduct extensive user studies of our own. In the following, we attempt to give a snapshot of the relevant (to us) portions of this knowledge base, which we subsequently fashion into a set of specific rules used in our system, and then exploit for more advanced and novel manipulations. These rules are in addition to other concepts such as simultaneous contrast and color harmony.

8.4.1 Encoded Principles and Guidelines

In the following we enumerate a set of guidelines following the design goals defined above.

Mood and emotions:

G1: Warm colors (red, orange, yellow) excite our emotions and grab our attention. Cold colors (green to violet) produce the feeling of openness and distance. They have exactly the opposite effect of warm colors. Warm hues will tend to overpower the cool ones. Yellow-green and red-violet are borderline warm colors.
G2: Vivid colors (bright, saturated colors) stand out, bringing attention to a particular feature – the pop-out effect. However, combining two or more vivid colors is perceived as unpleasant and overwhelming. Vivid colors should be used sparingly or between dull background tones. Finally, large areas should not be made highly saturated. People generally find large saturated areas tiring and annoying.

Sensation of depth and separation of foreground/background:

G3: Color can affect perception of 3D space. Due to chromatic aberration, cold-colored areas are perceived as being more distant than warm-colored ones. This helps foreground-background separation, which works best when the foreground color is bright and highly saturated, while the background is desaturated. Blue is least suitable for representing high frequency detail since it has the fewest number of cones in the fovea. Warm colors on a cold background are effective to enhance foreground-background separation.

Acuity and detail perception:

G4: The achromatic visual subsystem has about 5-times better visual acuity than the chromatic subsystem. Therefore, fine detail, high frequencies and shape is better conveyed with brightness contrast.

Discrimination:

G5: Color discrimination is much poorer when the samples are separated without sharing a border – the greater the separation the worse the discrimination.
G6: Colors will be more discriminable if they differ simultaneously in hue, saturation and brightness.
G7: The low end brightness steps should be very small while the high end needs larger steps (Weber's Law). On the other hand, the number of just noticeable difference steps (JND's) in the hue spectrum is about 150, but discrimination varies across the spectrum.
G8: Discrimination is poorer for small objects. Hue, saturation and brightness discrimination all decrease.

Hue contrast:

G9: Complementary hues lie on the opposite side of the color wheel. They have the highest chromatic contrast and when mixed they may cancel each other out, generating a neutral grey. Examples are red-cyan and blue-yellow, which are both opponent colors.

Relationship of hue with saturation and brightness:

G10: Some hues appear inherently more saturated than others. Studies indicate that there are only 10 saturation steps around yellow with the number gradually rising as wavelength increases or decreases. Therefore, viewers will find small differences in saturation for blue, violet and red highly discriminable, while small differences in yellow saturation will be hard to detect. Green and orange are in the middle. The brightest lights fall in the yellow range, while blues, violets and reds are least bright.

Labeling and semantics:

G11: There are 11 basic color distinctions that fall into three classes: i) achromatic (3): black, gray, white, ii) primary colors (4): red, green, blue, yellow, and iii) secondary colors (4): brown, orange, purple, pink. These distinctions and labels are valid across cultures. One should choose the maximal number of color labels under 6-7.
G12: Increasing the brightness of an item for highlighting draws attention to it without changing its hue, and therefore, without losing perceptual information about its semantic class. Brightness and saturation variation can help in distinguishing objects of the same semantic class.

Linking and guiding:

G13: Objects of similar hue form a common group, while objects of different hue belong to different groupings. If the different hues are complementary colors, then the viewer will infer opposition. Color can be used to organize the display into perceptual chunks, viewed pre-attentively. The color layout indicates that parts of the image form distinct areas, and if the same color appears in different parts of the image, these areas appear linked together, suggesting that they have something in common. On the other hand, variations in a basic color can convey variations in the class data. Similar colors suggest a similarity relationship and different colors suggest a difference relationship between objects and areas of the screen. Similar colors are adjacent on the color wheel, while complementary colors are on opposite sides of the color wheel.

Recall and interest:

G14: A final motivation is that an aesthetic visualization will be remembered more and looked at more carefully.

We can assign these guidelines into the specific rules implemented by our system. All steps will be discussed in detail later in Section 8.4.3.

R1: Hue Selection (G1, G5, G6, G11, G12, G13, G14), and Background Selection (G3) – presented to the user in Step 1;

R2: Vividness Selection (G2, G6, G14) – decided by the system in Step 2;

R3: Lightness Selection (G4, G6, G7, G10, G14) – optimized by the system in Step 3;

R4: Mixing Rule (G9) – chosen by the system in Step 1.

8.4.2 Framework Theory

Here we show how to capture these rules into an actual color design framework for visualization tasks. As mentioned above, our system is designed to work for the visualization of 2D entities as well as for 3D visualizations with opaque or semi-transparent object surfaces (where a prior segmentation is assumed). For the 3D applications, the visual color design can be either optimized for a specific view or globally for all views. We begin by first clarifying some important basic concepts that form the foundations of our system, assisted by Figure 8.1. We then describe our extensions of these basic concepts to enable our framework.

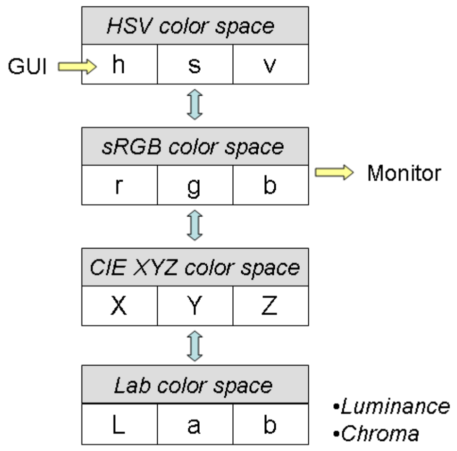


Figure 8.1: Color spaces utilized in our system.

Basic Concepts

Color: Color has three components: Hue, Brightness, and Saturation. A color in RGB color space has its own r, g, and b values. On the other hand, a color in HSV space also has three components, h, s, and v. All three values together determine a color, that is, any value by itself is not a color. For any color space, a color corresponds to a point.

Color spaces: The color spaces utilized in our system are shown in Figure 8.1. Hue is chosen from the hue wheel of the HSV color space. The CIE LAB (Lab) color space embedded in our system is essential for computing the luminance and chroma due to its perceptual uniform characteristic (more on this below). The color transfer between HSV and Lab color spaces will be assisted by the standard RGB (sRGB) color space and the CIE XYZ color space. The r, g, b values in sRGB color space are finally transmitted to the monitor for display.

Hue categories: As mentioned before, the hue wheel of the HSV color space is popular and widely used in computer graphics as well as in color design for its convenience. We also employ it in our system to assist users in choosing the hue. Besides the neutral color, we roughly separate hue into 8 color categories, which include Red, Orange, Yellow, Green, Cyan, Blue, Purple, and Magenta. We define similar hues as either in the same category or adjacent on the hue wheel.

Lightness: Luminance, or lightness, is the perceived brightness. In our system, in order to calculate the lightness of a HSV color, the color is first transferred to sRGB space, then to CIE XYZ space, and finally to Lab space, where the component L is the lightness. The value range of L is $[0, 100]$, with 0 for black, and 100 for white.

Formally,

$$(h,s,v) \Rightarrow (L,a,b), \tag{8.1}$$
$$lightness_{(h,s,v)} = L, \tag{8.2}$$

where "⇒" denotes transformation.

Vividness: We define the vividness of a color as its *relative* purity (or chroma). The absolute chroma of a color can be computed in Lab color space using Equation 8.3.

$$chroma_{(h,s,v)} = \sqrt{a*a+b*b}. \tag{8.3}$$

For a given hue, the higher the chroma value, the more vivid the color appears. In HSV color space, the color $(h,1,1)$ is the most vivid color for a given hue h. It has the maximum chroma for this hue. Then the vividness of a color, that is, its relative chroma, is defined as follows:

$$vividness_{(h,s,v)} = chroma_{(h,s,v)}/chroma_{(h,1,1)}. \tag{8.4}$$

Although the maximum chroma values for different hues are quite different, the value range for vividness is always [0,1].

Enabling Extensions

As mentioned before, similar to popular color design utilities our system also employs the HSV space as a convenient and intuitive design medium for color specification. However, in our framework the user only picks the hue of a scene component, while the expert design system determines vividness V and lightness L, taking into account the present visualization goals and constraints and the overall scene composition. From the resulting (h, V, L) triple, the system then computes the remaining HSV components s and v needed for actual display. This (h,s,v) triple is then converted to sRGB space and fed to the monitor.

In our framework, L (and also V) plays a crucial role and requires accurate control. In fact, there are a number of linear equations which are in frequent and popular use to determine the lightness of a given (r,g,b) triple. Equation 8.5 below gives the lightness derived from the YIQ color space, and Equations 8.6-8.7 immediately following provide the lightness derived from CIE XYZ color space.

$$Y_{YIQ} = 0.299r + 0.587g + 0.114b, \tag{8.5}$$

and,

$$Y = r*Y_R + g*Y_G + b*Y_B, \tag{8.6}$$
$$Y = 0.2126r + 0.7152g + 0.0722b, \tag{8.7}$$

where constants Y_R, Y_G, and Y_B are the lightness values for the R,G,B primary colors. Note that for a different white point, the lightness equation will change (the lightness equation of a sRGB monitor with a white point of D65 is used in our examples).

To assess the results obtained with these equations we have employed a pair-wise comparison test in which we have sought to match the lightness of two swatches of different h using the above conversion equations. The left and center panel of Figure 8.2 show two examples each for the above equations. In each such pair we have attempted to match the lightness of the right swatch image to that of the left via a search procedure. We observe that while the popular linear lightness equations work well for light colors (top row) they provide poor results for darker colors (bottom row) where the match always comes out too dark (that is, the two swatches have the same lightness according to the conversion formulas, but this is not perceived as such – on our D65 monitor). In contrast, matching the lightness using the (non-linear) Lab space relationships gives consistently better matches for both bright and dark colors. These experiments indicate that these widely used equations are in fact only sub-accurate for darker colors, while the Lab space is quite accurate throughout. Thus we find that the Lab space is most appropriate for our application, although the transform is non-linear.

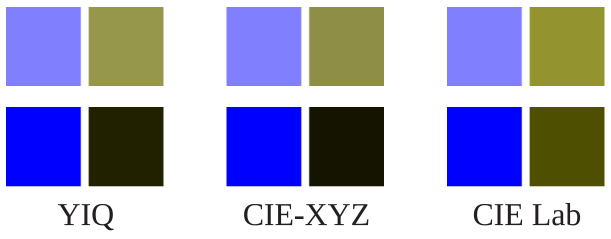


Figure 8.2: Color pairs with lightness matches performed using different conversion methods.

Table 8.1: Relationship between s, v, lightness, and vividness for given hue in HSV color space, where C is a constant except 0, ↑ means value increase, and ↓ means value decrease.

s	v	Lightness	Vividness
0	↑	↑	0
0	↓	↓	0
C	↑	↑	↑
C	↓	↓	↓
↓ / ↑	0	0	0
↑	C	↓	↑
↓	C	↑	↓

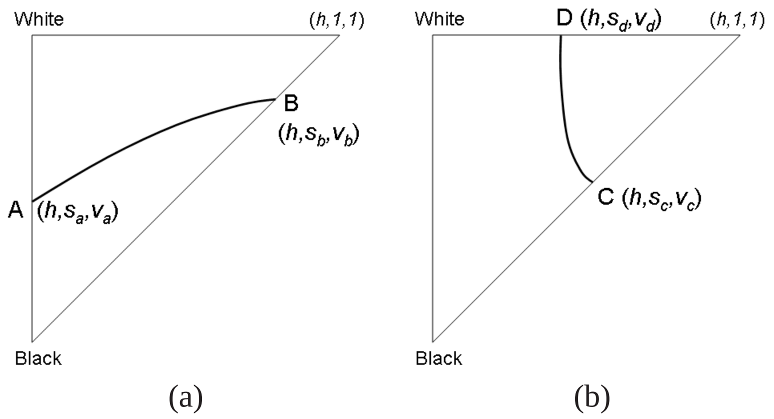


Figure 8.3: Illustration of equi-lightness and equi-vividness curves in HSV color space. (a) An equi-lightness curve, (b) An equi-vividness curve.

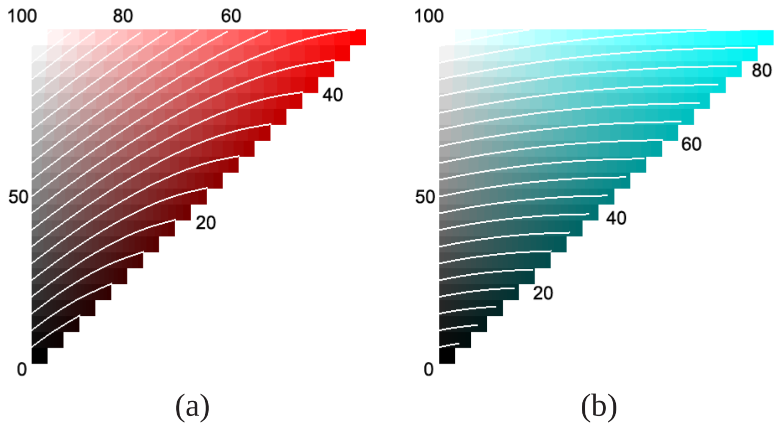


Figure 8.4: Examples of equi-lightness curves in HSV color space. (a) Hue slice with $h = 0$, (b) Hue slice with $h = 180$.

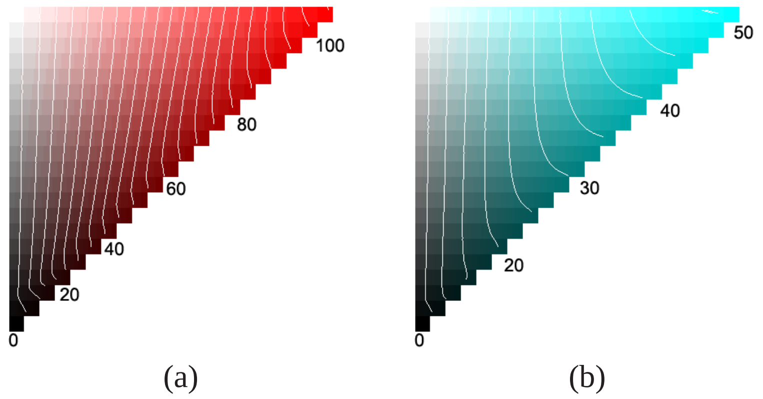


Figure 8.5: Examples of equi-vividness curves in HSV color space. (a) Hue slice with $h = 0$, (b) Hue slice with $h = 180$.

There are various non-linear relationships when transforming an (h,s,v) triple to Lab-space L and V, and back. The conversion to (r,g,b) is only piecewise linear, and the conversion to Lab space is strictly non-linear. Thus, the colors of equi-lightness (and equi-vividness) in HSV space will likely reside on non-linear trajectories. Table 8.1 shows the relationships between s, v and L and V. For a given hue h in HSV space, if s is constant, the L monotonically increases or decreases with v, and if v is constant, the L reversely monotonically changes with s. Based on these monotonic relationships, fixing s, we can compute v from h and L, and vice versa. By moving s from 0 to 1 we can then find all colors with the given h and L. Figure 8.3a shows the equi-lightness curve, which is comprised of all the colors from A to B that have the same L on this hue slice.

These monotonic relationships enable efficient binary search procedures to be devised for the mapping, alleviating the need for more complex optimization methods, such as gradient descent. We have designed two binary search-based (BSB) algorithms to efficiently compute the color from a given hue h and lightness L, when either s or v is known. Here we show the BSB algorithm that computes v from a specified h, L, and s. The algorithm for computing s from a specified h, L, and v is similar. Using our BSB algorithms, for any hue, we can easily find the colors with the desired L.

```
// BSB algorithm of computing $v$ from $h$, $L$, and $s$
S0:   initialize $v_{min}$, $v_{max}$;
S1:   $(h, s, v_{min}) \Rightarrow (L_{min}, a, b)$;
      $(h, s, v_{max}) \Rightarrow (L_{max}, a, b)$;
      if ($L$ equals $L_{min}$)     return $v_{min}$;
      if ($L$ equals $L_{max}$)     return $v_{max}$;
      if ($L > L_{max}$){
          either decrease $s$, go to S1;   // sacrifice $s$ to get $L$
          or return $v_{max}$;   // $L_{max}$ is the closest lightness we
                                 // can get for given parameters
      }
S2:   while ($L > L_{min}$ and $L < L_{max}$){
          $v_{mid} = (v_{min} + v_{max})/2$;
          $(h, s, v_{mid}) \Rightarrow (L_{mid}, a, b)$;
          if ($L$ equals $L_{mid}$)     return $v_{mid}$;
          if ($L_{mid} > L$){
              $v_{max} = v_{mid}$;
              $L_{max} = L_{mid}$;
          }else{
              $v_{min} = v_{mid}$;
              $L_{min} = L_{mid}$;
          }
      }
```

Likewise for V, for a given hue h, there are many color points which have the same V. Again, due to the monotonic relationships between s, v, and V shown in Table 8.1, we can design efficient BSB algorithms to compute the color from a given h and V, when either s or v is known. In Figure 8.3b, all colors with the same V on this hue slice make up the equi-vividness curve, which is also the equi-chroma curve.

Figure 8.4 shows examples of equi-lightness curves on two different hue slices. Some curves have been labeled with their lightness value. From the bottom curve to the top curve, the lightness increases gradually. For different hues, the lightness values of the most vivid colors (the top-most outside points on the hue slices) are quite different. Vivid cyan has a much higher lightness than vivid red. Figure 8.5 gives examples of equi-vividness curves on the given hue slices. Some curves are labeled with their chroma values. Note that the chroma interval between adjacent curves is the same. From the right-most curve to the left-most curve, the vividness decreases from 1 to 0. We can see that the absolute chroma varies substantially for the different hues. Cyan has considerably fewer distinguishable saturation steps than red. In fact, this hue-dependent resolution in perceived saturation was the subject of guideline G10.

We now describe our algorithm that computes s and v values from a given h, L, and V. We see that on the equi-lightness curve in Figure 8.3a, the vividness increases monotonically from A to B. Also on the equi-vividness curve in Figure 8.3b, the lightness increases from C to D. The intersection point of this two curves then yields a color with the desired L and V. Due to the monotonic relationships, we have designed a bi-BSB algorithm to compute s, v from a desired L and V. In this procedure, we first use our BSB algorithm to locate points A with $s = 0$, B with $s = 1$, and middle point M, whose s is the average of those of A and B, and then compare the vividness V_A, V_B, and V_M with V to determine which half curve need to be searched further, finally finding the intersection point. Due to the efficiency of binary search, our algorithm is quite fast and enables interactive manipulations.

However, not any two equi-lightness curves and equi-vividness curves will intersect. Therefore in some cases it is impossible to get a color with both the desired lightness and the desired vividness. To compensate, we have to either sacrifice lightness or vividness. Figure 8.6 gives a series of swatch-pair examples where the goal is to reach a matched lightness, such that no swatch is over-emphasized. In Figure 8.6 (center), two colors C_{cyan} and C_{red} have the same vividness V=1, but the lightness $L_{C_{cyan}}$ is much higher than $L_{C_{red}}$, leading to an over-emphasis of the cyan swatch. To cope, we have three choices to achieve equal lightness for the two swatches, relaxing the goal of matching vividness: setting both to the (i) minimum, (ii) the maximum, or (iii) the average lightness of the two. The results are shown in the left, right, and bottom swatches in Figure 8.6, respectively. We observe that either one (min, max) or both swatches (avg) need to reduce their vividness, which may be problematic in generating the desired pop-out effect. While this cannot be avoided since it is caused by an inherent property of the color spectrum, we can produce a reasonable trade-off, where we move the lightness values of both of the two colors halfway towards their average, that is, the middle point between the old value and the average value. The visual effect of this compromise is shown in the top-most swatch pair of Figure 8.6. This

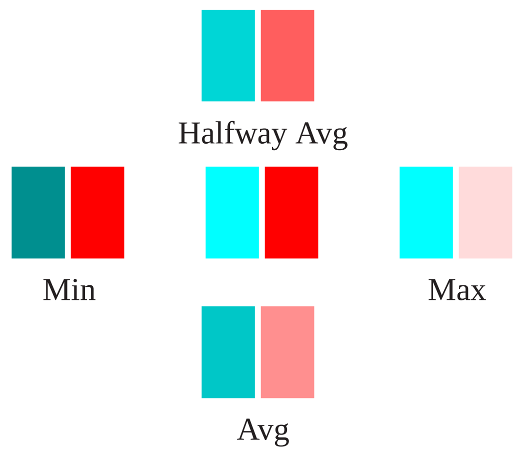


Figure 8.6: Examples of controlling the lightness. Two vivid colors with the same vividness but different lightness values are shown in the middle. Colors are recomputed with the desired lightness, which can be minimal, maximal, average, or halfway to the average, of the lightness values of the two original colors.

solution in fact seems to equalize the vividness and the lightness of the two swatches the best, although neither L nor V are really equal.

From these examples, we can see that in order to obtain the desired lightness the vividness of the color may have to be sacrificed, or vice versa. Since the lightness is more comparable, and the lightness contrast is crucial to help distinguish features and provide harmonic colorings, in our work, we prefer to keep the lightness in most cases.

8.4.3 Implementation Details: 2D System

In this section we discuss the system details relevant for 2D visualizations, while the following section will then extend these to 3D visualization applications. All steps outlined below make use of the design principles and guidelines enumerated in 8.4.1.

Step 0: Preprocessing In order to assign colors to 2D data with discontinuous intensities or separated features, the data should be available in segmented form. In our system, the smallest unit is defined as *Object*. Several objects with the same properties can belong to one *Class*. Each object has attributes, such as its area, and a class has an importance value. A class computes its total area from its member objects. The system also collects information between objects, including distance and adjacency. Next the user, or the system's host application, specifies importance factors for all classes, which may be interactively changed at any time. Currently, our system supports three importance levels: most (3), less (2), and least (1) important, but this can be extended easily.

Step 1: Hue selection Starting from the most important class, for each class, the hue wheel is activated with the proper hue categories from which users may select the classes hue. The system suggests hues of warm colors for the most important classes and hues of cool or neutral colors for the least important classes, but users are free to make other choices. The exact hue will then be assigned for each class based on the associated class information. Here, all objects in one class will be assigned the same main color (h, s, v), but the color finally used can vary to some extent based on the intensity or other properties of the objects (see later Figure 8.10). In order to generate color-harmonic visualizations, our system will not allow users to choose hues falling into all hue categories. To facilitate this choice, harmonic color templates [24] can be applied to provide guidelines on harmonic hue selections.

Step 2: Vividness selection Next, the vividness of each class is computed based on a classes importance and its area. Our system provides suggested vividness values for different importance levels, for example, a vividness of 1 for the highest importance, 0.7 for less, and 0.3 for the least. The relative area (the ratio of area vs. whole area) is then used to adjust the vividness in order to avoid a large area to receive a very high vividness, and a global weighting parameter can be tuned to account for the real image size. Therefore, important classes will be colored with a higher vividness, and for the same importance level, a class with smaller area will be colored with a higher vividness, while a class with larger area will receive a color with lower vividness.

Step 3: Lightness selection Appropriate lightness contrast is very important to help discriminate different features and provide harmonic colorings. Figure 8.7 illustrates how our system determines the feature lightness levels, given the assigned vividness values. First, for each most important class, our system computes the lightness according to its hue and vividness. This yields a lightness range $[L_{mmin}, L_{mmax}]$ for all most important classes. Design rules G6 and G12 in Section 8.4.1 imply that lightness difference (contrast) will help

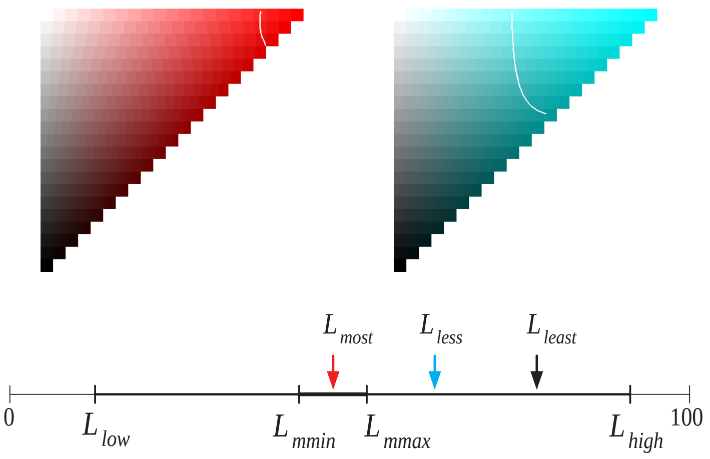


Figure 8.7: Illustration of the lightness selection. The class in red with the highest importance has higher vividness, and the class in cyan with less importance has lower vividness. L_{most} is the lightness selected for the class in red. With the *Lighter* option, L_{less} is the lightness for the class in cyan, and L_{least} is the lightness for classes with the least importance.

in discriminating features. Thus, in order to highlight the classes of interest, other classes will be assigned a lightness either all smaller (darker) than this range, or all higher (lighter). Although this can be decided automatically by an optimization algorithm, our system provides two options to the user: *Darker* or *Lighter*, which allows more control over the overall lightness of the visualization. The system also provides the following user-adjustable parameters: the lowest lightness L_{low}, the highest lightness L_{high}, and the highest v for the most important objects, v_{high}. When the user chooses *Darker*, the lightness of the lesser or the least important classes will fall into the range $[L_{low}, L_{mmin}]$. In contrast, for the *Lighter* option, the range will be $[L_{mmax}, L_{high}]$. We have chosen to provide equal lightness intervals between adjacent importance levels to provide good lightness contrast. Here v_{high} is a parameter, with the default value 1, which us used to bound L_{mmax}. The system provides a more vivid color when v_{high} equals 1, and a more darker vivid color when v_{high} decreases. This parameter is most useful in the *Lighter* option. A similar parameter, v_{low}, can be used to bound L_{mmin} in the *Darker* option.

Step 4: Color computation Once the hue, lightness, and vividness have been selected, our binary search-based algorithms are then used to compute the associated s and v. Since the lightness and vividness may not be preserved at the same time (see Section 8.4.2), our strategy is to rather preserve the lightness while adjusting to the nearest possible vividness, which guarantees the desired lightness contrast.

Applications of this color design system just described are presented in Section 8.5.

8.4.4 Implementation Details: 3D System

Basic Extensions

In order to extend our system to volume visualization, we can either use the 2D projection area for a specific view optimization or use the 3D region of the object for a global optimization. For a specific view, we first draw the objects of a segmented volume dataset into the texture buffer to obtain their label and depth information. Then for each object, we apply our 2D algorithm to calculate the suggested color, which represents their main color for the transfer function setting. After rendering the volume we evaluate the color histogram of the image to gauge whether there is enough contrast between objects. If not we change the color setting accordingly, to arrive at better result gradually.

The Layer-based 3D Scene Decomposition Framework

For volume visualization applications in which the objects have been or can be segmented, we designed a layer-based framework. This allows users to freely turn objects on or off and enhance them, in ways similar to the layer style in Adobe Photoshop. The color assignment will change when the object selections are updated. The importance of each object can be interactively changed as well, to enhance features of interest.

The color mixing rule for semitransparent rendering can also be applied in our system. In the following section, we will discuss this aspect in further detail, and also show examples and more applications.

8.5 Applications

All of the results in this chapter have been generated for a CRT monitor, with the white point set to D65. In this respect we would like to caution the reader when examining the images presented in the next sections. The perceived effects may be slightly or moderately different than on our monitor.

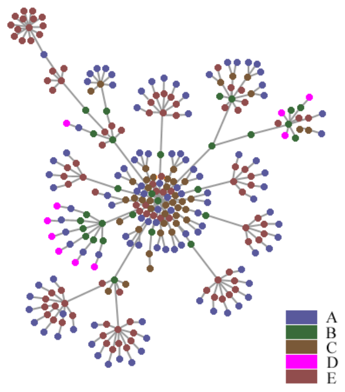
8.5.1 2D Visualization

We first illustrate our color design results by ways of a StarTree visualization, shown in Figure 8.8. The corresponding parameters are listed in Table 8.2. For the following, the reader may examine an image first before reading on. Else the solution follows now. In Figure 8.8a, D is emphasized; in Figure 8.8b, A is emphasized; in Figure 8.8c, A has the highest importance, and E has higher importance than others; in Figure 8.8d, both A and E are the most important; in Figure 8.8e, B is emphasized; in Figure 8.8f, C is emphasized; in Figure 8.8g, E is emphasized; in Figure 8.8h, E is the most important, and B is slightly less important; in Figure 8.8i, C and D have the highest importance. The visualizations demonstrate our system's ability to successfully balance the overall scene vividness and lightness. For example, in Figures 8.8b, c, d the lightness of the unimportant nodes (for example, the green class B nodes) increases as the importance of the red nodes (class E) rises (this widens the L_{mmax}, pushing the remaining intervals up towards L_{max}). Figures 8.8g, h demonstrate the use of v_{high} to control the lightness for the most important nodes (here class E), which is set higher in Figure 8h.

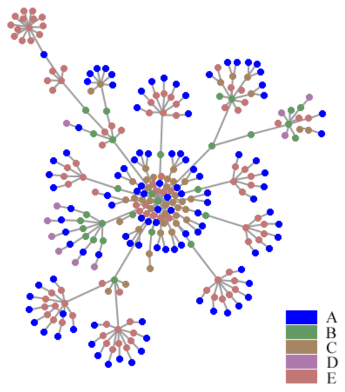
Finally, we also present, in Figure 8.9, two colorings which do not use these rules and fail to successfully emphasize any feature, either by using too many vivid colors, or by not using a sufficient lightness contrast.

Varying the color of an object according to its intensity or other property variations is also a frequently encountered visualization need and our color optimization algorithm supports this as well. To demonstrate, we applied our color design system to pseudo-color a Transmission Electron Microscopy image, and the results are presented in Figure 8.10. Note that the original grey-level intensity variation is preserved in the colorizations, and different features are highlighted in each image. Our system produces visualizations with high color harmony, and avoids the generation of inharmonic results with too much or too little contrast. This sequence of images also demonstrates the use of the ($Lighter$ option) and ($Darker$ option), which allows the user to select the overall lightness of the image.

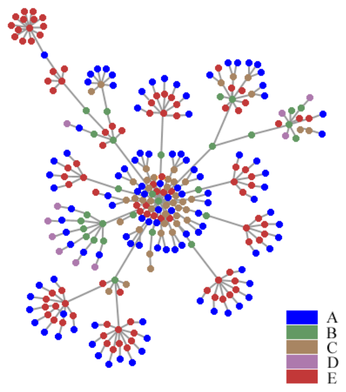
We applied our color design system to pseudo-color a Transmission Electron Microscopy images, and the result is presented in Figure 8.10. Note that the original grey-level intensity variation is preserved in the colorized image, and different features are highlighted in each image. Our system yields color visualization with high harmony, and avoids a generation of inharmonic results with too much or too less contrast.



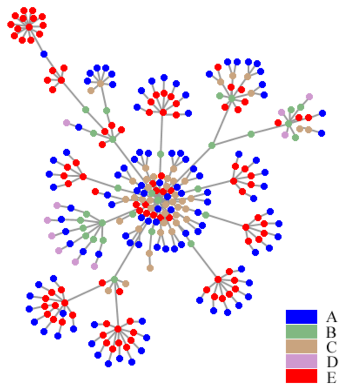
(a)



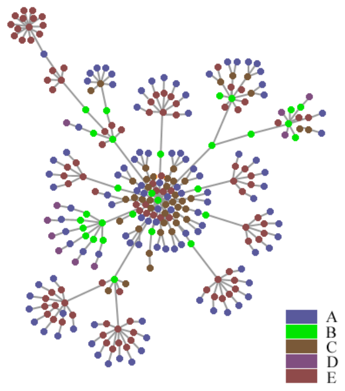
(b)



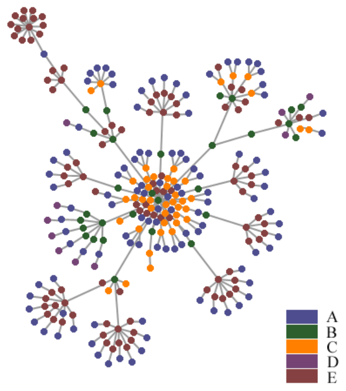
(c)



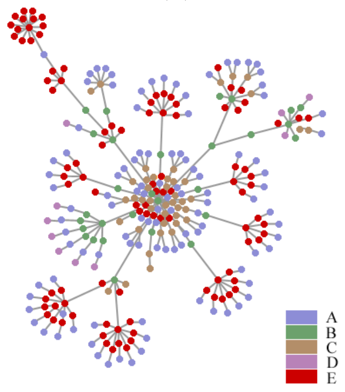
(d)



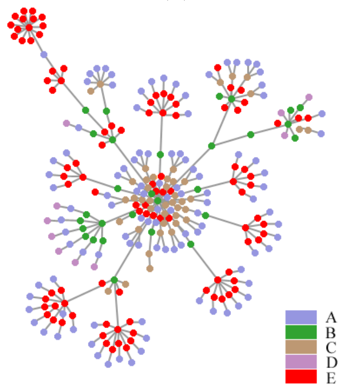
(e)



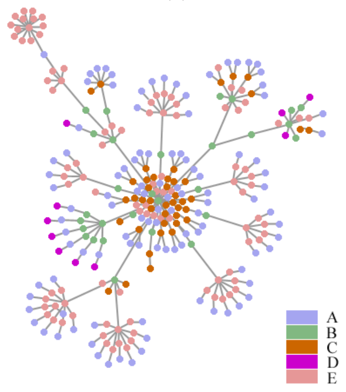
(f)



(g)



(h)

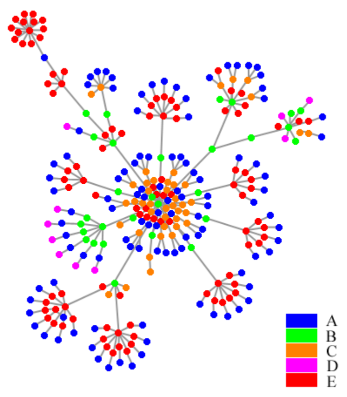


(i)

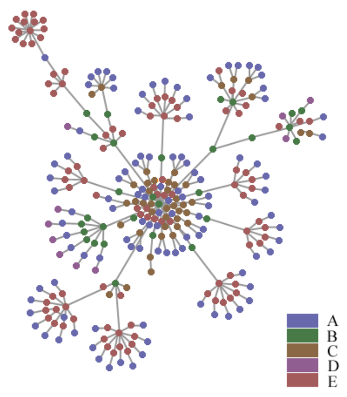
Figure 8.8: Color designs for startree visualization. The parameters of tasks are listed in Table 8.2, including the number of classes, and for each class, the importance factor specified by the user, while the hue is generated by the system based on the user's choice. The tree nodes with higher importance are highlighted in the coloring results.

Table 8.2: Parameters of startree visualization tasks

Class	Hue	Importance								
A	240	1	3	3	3	1	1	1	1	1
B	120	1	1	1	1	3	1	1	2	1
C	30	1	1	1	1	1	3	1	1	3
D	300	3	1	1	1	1	1	1	1	3
E	0	1	1	2	3	1	1	3	3	1
Option		*D*	*L*	*L*	*L*	*D*	*D*	*L*	*L*	*L*
Figure No.		8.8a	8.8b	8.8c	8.8d	8.8e	8.8f	8.8g	8.8h	8.8i



(a)



(b)

Figure 8.9: Startree examples. (a) All vivid colors compete to catch the eye, (b) All colors with similar lightness, no feature is emphasized well in both (a) and (b).

8.5.2 Lightness-preserving Color Harmonization and Color Shifting

Color harmonization is an easy and efficient way to improve the harmony of an image by shifting colors to an appropriate harmonic color scheme [24]. However, since only the hue is changed, if the shifting angle is not small, the lightness will change. This is undesirable as it may reduce contrast. We have therefore extended the original color harmony algorithm to preserve the lightness and contrast as much as possible. To achieve this, we first calculate the lightness and vividness of the color before shifting. After the hue shifts, we assign the same lightness and vividness to the new hue, and then calculate from these the new s, v values. As mentioned at the end of Section 8.4.2, it may not be possible to preserve the lightness and vividness at the same time. In these cases our algorithm will seek to preserve the lightness while using the nearest vividness. Although the hue contrast may change slightly, the lightness contrast important for the visual resolve of fine object detail can be maintained.

Figure 8.11 and Figure 8.12 compare our lightness and contrast-preserving color harmonization algorithm with the original color harmonization algorithm. In both figures, the (b)-panels hold the original images, subjected to harmonization / hue-shifting. The corresponding hue wheels are also provided. We observe that our method keeps the contrast

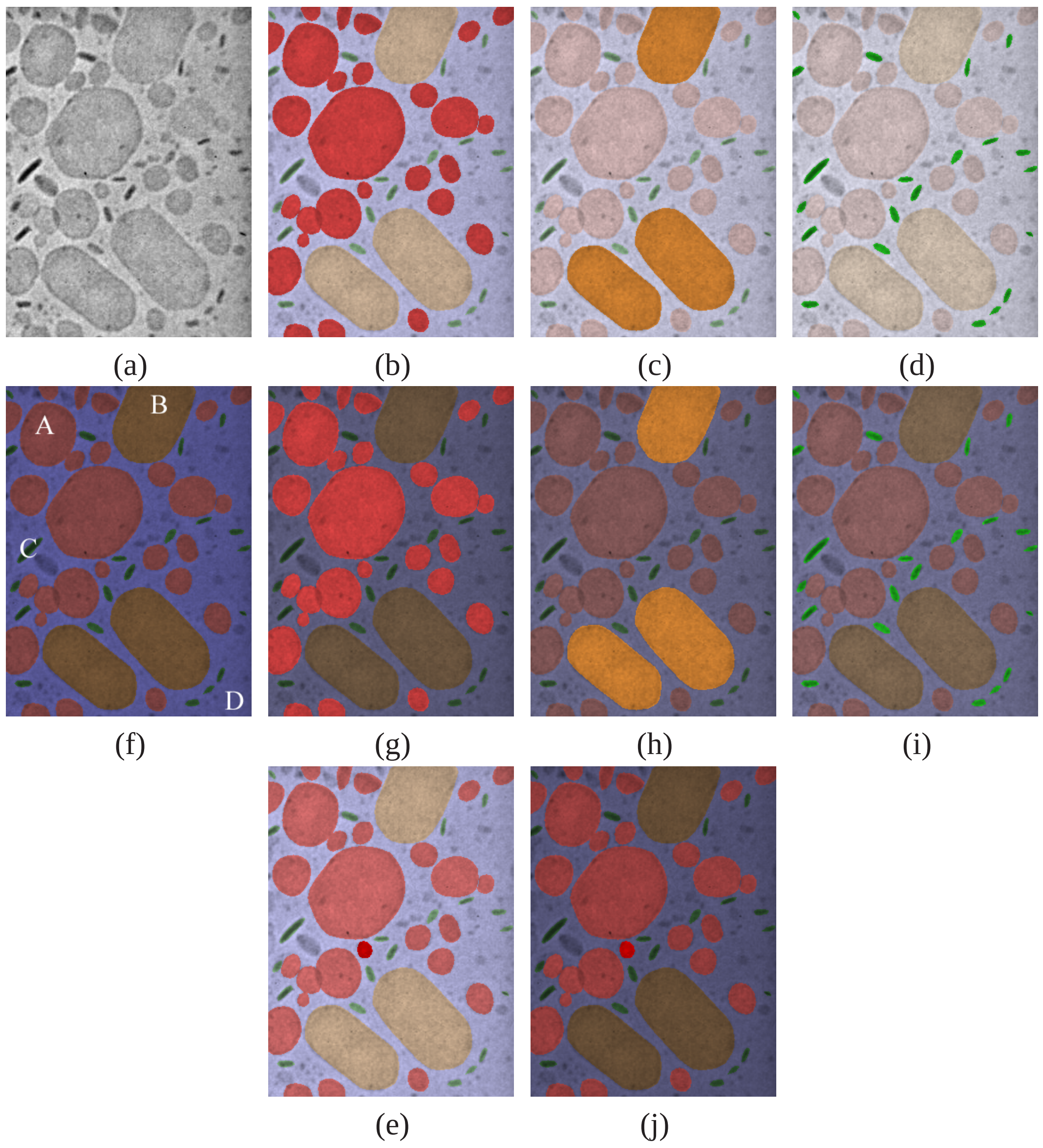


Figure 8.10: Color designs for Transmission Electron Microscopy image. Different features are highlighted in images. (a) Original grey-level image, (f) Colors with equal lightness, no feature is emphasized, features are labelled as A,B,C,D for reference, (b)-(e) Features highlighted have the lowest lightness comparing to other features ($Lighter$ option), (g)-(j) Features highlighted have the highest lightness ($Darker$ option), (b) and (g) A is emphasized, (c) and (h) B is emphasized, (d) and (i) C is emphasized, (e) and (j) A is emphasized, a rather small region colored with high vividness to produce the pop-out effect despite its small size.

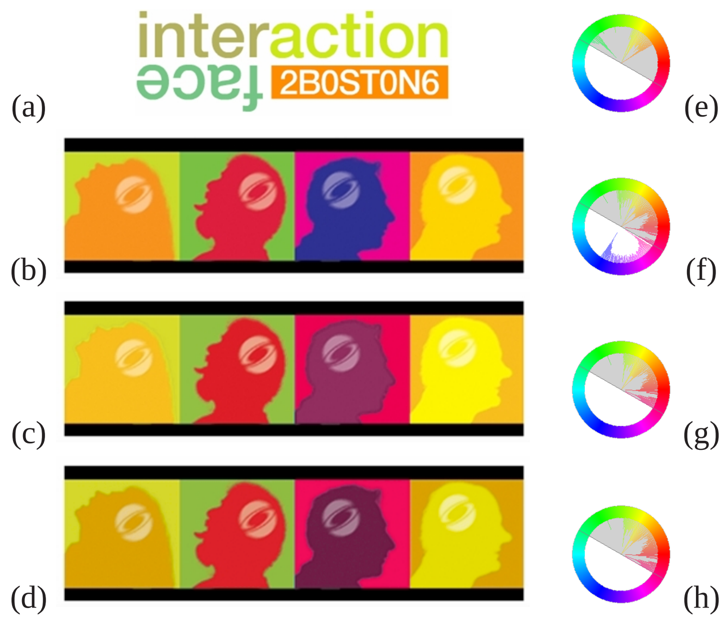


Figure 8.11: Lightness preserved color harmonization. (a) Cut and paste part1, (b) Cut and paste part2, which will be color harmonized to part1, (c) Color harmonization by Cohen's method[24], (d) Lightness preserved color harmonization, (e) and (f) Color distribution on the hue wheel of the original image (a) and (b), (g) and (h) Color distribution after harmonization.

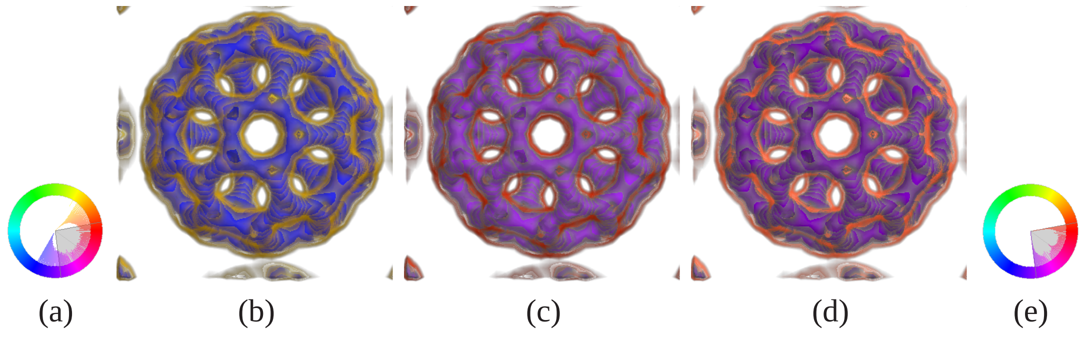


Figure 8.12: Lightness preserved color harmonization. (a) Color distribution on the hue wheel of the original image (b), (c) Color harmonization by Cohen's method[24], (d) Lightness preserved color harmonization, the brightness contrast and feature details of the original image are kept, (e) Color distribution after harmonization.

and details better. For example, in Figure 8.11c, resulting from application of the original method, the left-most person's silhouette now almost blends into the background. On the other hand, in Figure 8.11d, generated with our method, it stands out like in the original image. Further, the contrast between the third person and the background is also better preserved. Figure 8.13 shows the lightness-preserving color shifting result. In Figure 8.13b, generated with the original method, some colors around the neck become too dark, which

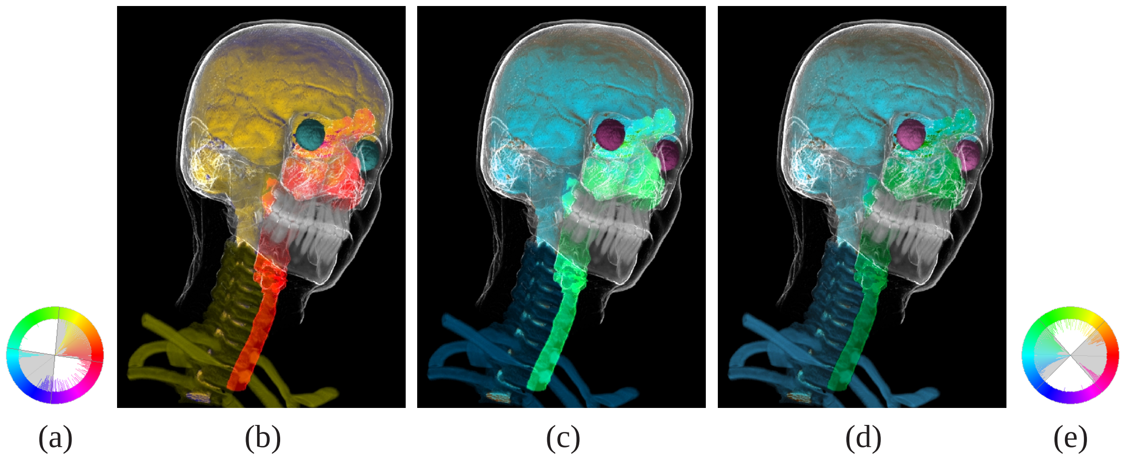


Figure 8.13: Lightness preserved color shifting. (a) Color distribution on the hue wheel of the original image (b) [50], (c) Color shifting with only hue shifted, (d) Color shifting while preserving the lightness, (e) Color distribution after shifting.

renders the details unclear, while colors around the nose become too bright, which draws more attention to the user. Also, some details that had been bright before the shifting are now hidden. On the other hand, Figure 8.13c presents the result obtained with our method. We observe that our method preserves the lightness and contrast better in most regions. However, we also note that in some regions the overall contrast is not shown as well as in the original visualization, which is unavoidably due to the hue contrast change.

8.5.3 Volume Rendering

Color design examples for volume visualization are shown in Figure 8.14. Features of interest in the volume data are highlighted by more vivid colors.

8.5.4 Color Mixing in Semitransparent Rendering

In semitransparent rendering, good color designs aim to avoid false color, i.e., the introduction of new hues due to color mixing, and they also seek to preserve the ordering of the objects. In the following we study a variety of scenarios involving color mixing and propose appropriate measures using our color design framework to achieve these objectives.

Two Overlapping Objects

If there are two translucent objects overlapping, our visualization system will suggest users to choose opposite hues, which will not introduce new hues.

Suppose the colors of the two overlapping objects are C_1 in front and C_2 in the back, and their opacities are α_1 and α_2. Then, using the front-to-back color composition, the mixed color C is:

$$C = C_1\alpha_1 + C_2\alpha_2(1 - \alpha_1). \tag{8.8}$$

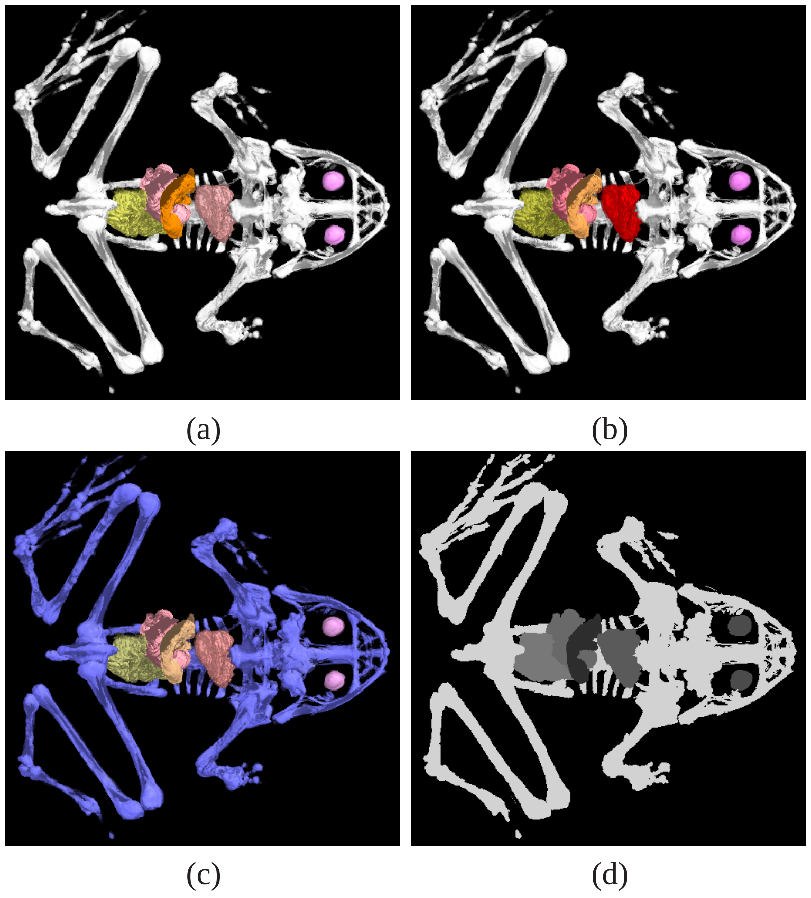


Figure 8.14: Color designs for volume visualization. (a) Kidney of the frog is highlighted with vivid orange, (b) Heart is highlighted with vivid red, (c) Bone is assigned blue hue and emphasized, (d) Object labels traced by ray casting.

We define the weights for two colors as $W_1 = \alpha_1$, and $W_2 = \alpha_2(1 - \alpha_1)$. The mixed color C will have higher lightness than either $C_1 W_1$, or $C_2 W_2$. If the opacity values for two objects are the same, the front color contributes more than the back color. Therefore, if two opposite hues are chosen, the mixed color will most likely have a hint of the one in front.

Figure 8.15 shows two color mixing examples. The opacity is 0.4 for all objects. Aided by the hue wheels, we observe that new hues between red and green are generated in Figure 8.15a and 8.15c (an orange tone in the former and a yellowish tone in the latter). On the other hand, as shown in Figure 8.15b and 8.15d, when two opposite hues are mixed, there are no false colors generated, and the mixed color can be either neutral or with a hint of the original hues, which will help prevent the color mislabeling problem.

From Figure 8.15, we note that the ordering of objects can not be visually preserved all the time. In all of the renderings, the top right square is physically always in front of the bottom left square. Figure 8.15c and 8.15d show the correct ordering without any doubt,

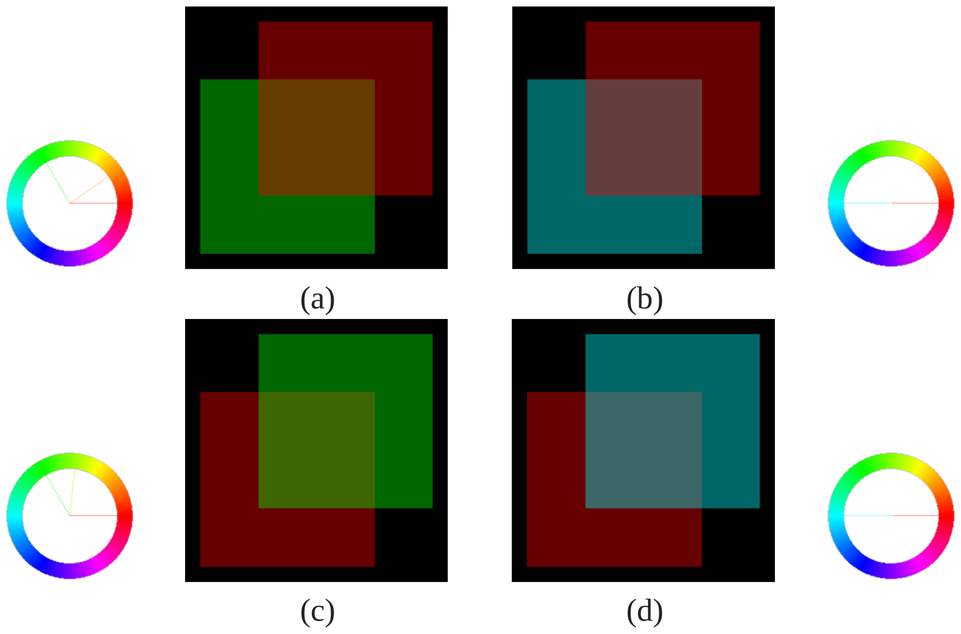


Figure 8.15: Two color mixing examples. (a) and (c) Red and green are mixed with red in front or green in front, (b) and (d) Red and cyan are mixed with red in front or cyan in front.

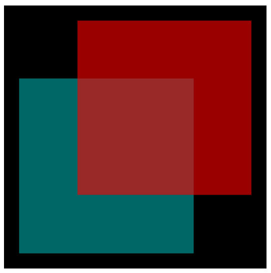


Figure 8.16: Ordering preserved color mixing example. Red and cyan are mixed with red in front.

however, 8.15a and 8.15b are less convincing. Based on our experience, we find that unsaturated colors will not preserve the ordering very well. We also find that darker colors give a feeling of distance, while lighter colors appear closer to observers. Therefore, a vivid color with high lightness is suggested for the front object. However, the relationship between colors and perceived orderings bring complications. For example, if the user already assigned red in front of cyan, either increasing the lightness of red, which will reduce its saturation, or decreasing the lightness of cyan, which makes it look more transparent, will not help provide the correct ordering. Only increasing the opacity (weight) of red will yield the right ordering, as shown in Figure 8.16, where the opacities of red and cyan are 0.6 and 0.4 respectively.

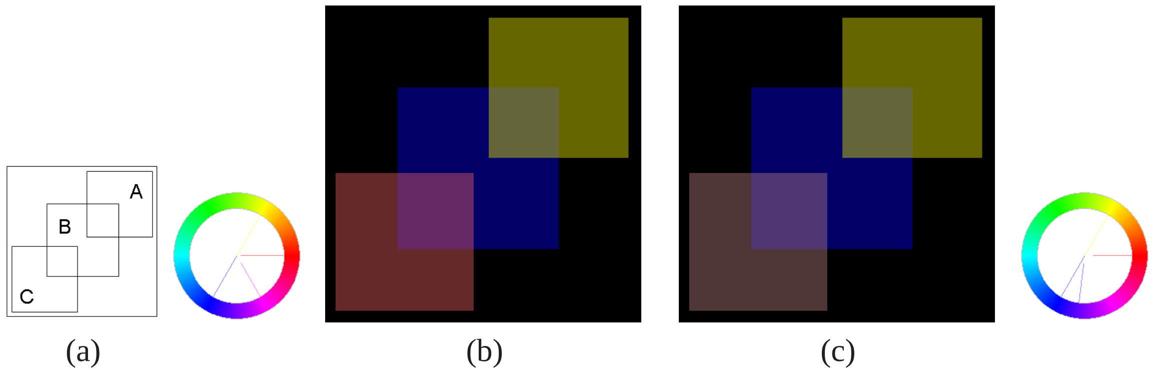


Figure 8.17: Three color mixing examples. (a) Object labels, (b) A and B are assigned opposite hues: yellow and blue, c is red, the overlapping region of B and C gives false color, which can be observed from hue wheel as well, (c) The saturation of C is reduced, but its lightness is kept, and the false color is reduced.

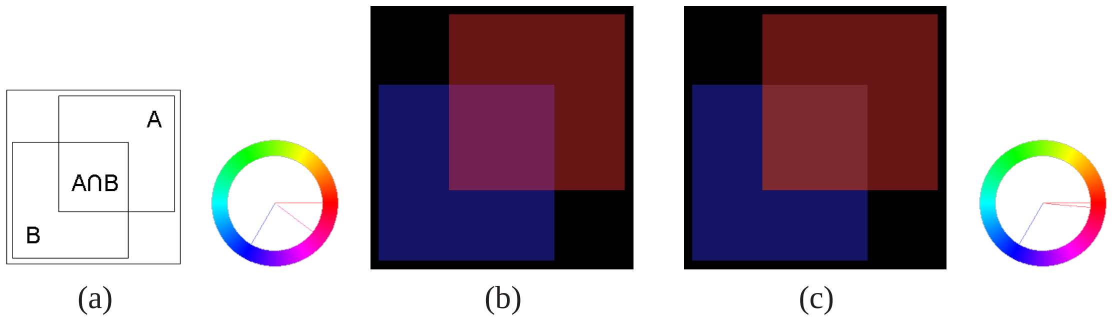


Figure 8.18: Local solution to reduce false color. (a) Object labels, A is in front of B, (b) False color generates when mixing red and blue, (c) False color is reduced by our local solution.

More than Two Overlapping Objects

Whenever more than two translucent objects overlap, false colors will always be generated after color mixing. To minimize these adverse effects, we suggest assigning two opposite hues for the two important objects, and more neutral colors for the lesser important object. But to be safe, we should avoid using any hue which can be generated by mixing two hues already chosen. Figure 8.17 shows a three color mixing example. By reducing the vividness (saturation) of C while keeping its lightness, we can reduce the false color.

Local Solution for Partial Overlap

Sometimes other constraints dictate the choice of certain colors, preventing the selection of opposite hues for partially overlapping objects. To provide a solution even in these cases, we devised a local solution to reduce the false color in the overlap regions. Our scheme is demonstrated in Figure 8.18. It works by reducing the saturation of the color in

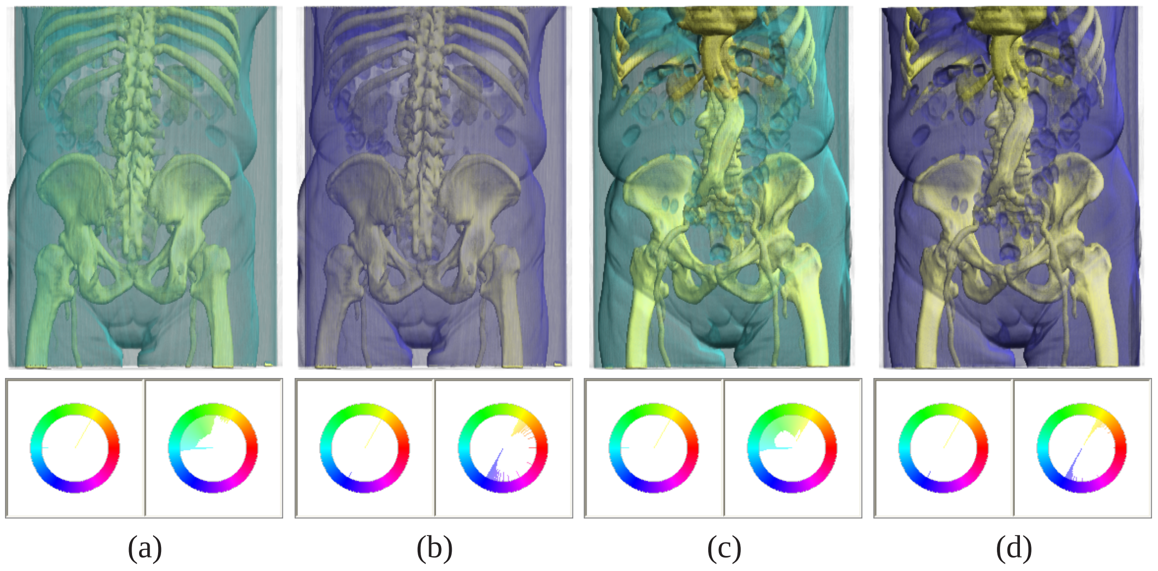


Figure 8.19: Color mixing on Body data for different view points. (a) and (c) Cyan and yellow are mixed, (b) and (d) Blue and yellow are mixed, (a) and (b) show back view of the body, (c) and (d) show front view of the body. Corresponding hue wheels are below each rendering.

the rear object only in the overlap region $A \cap B$, while keeping its lightness. After the color mixing, this region clearly has more hint of the front-object color, and the correct ordering is also visually preserved. Our local solution is very general – it can also be used to reduce the false-color effect in areas in which more than two objects overlap.

Semi-transparent Volume Renderings

Figure 8.19 shows the color mixing that occurs for semi-transparent volume renderings from different view points. When cyan and yellow are mixed, some green hues will be generated. In contrast, a blue and yellow opposite-color combination will keep the original hues. Furthermore, Figure 8.20 indicates that the features in overlapping regions are also clearer if two opposite hues are chosen.

We also devised a method to reveal interior colors. With two or even more features embedded, the color of an inside feature may be occluded or changed. As we have seen if two hues with at least one hue category between the two on the hue wheel are chosen, new hues will be generated. However, if two opposite hues are applied, the color of the mixed region will be neutral. If the interior object is the most important object, our system decreases the saturation of the outside features' color to reveal the interior object's real color, without having to change the alpha channel of the transfer function. As shown in Figure 8.21, the color of the inside feature is disclosed increasingly more as the color of the outside feature becomes less saturated. In all three cases, the center setting represents a good compromise, still keeping some of the outside object's hue, while allowing the interior object's hue to show through with minimal color distortion.

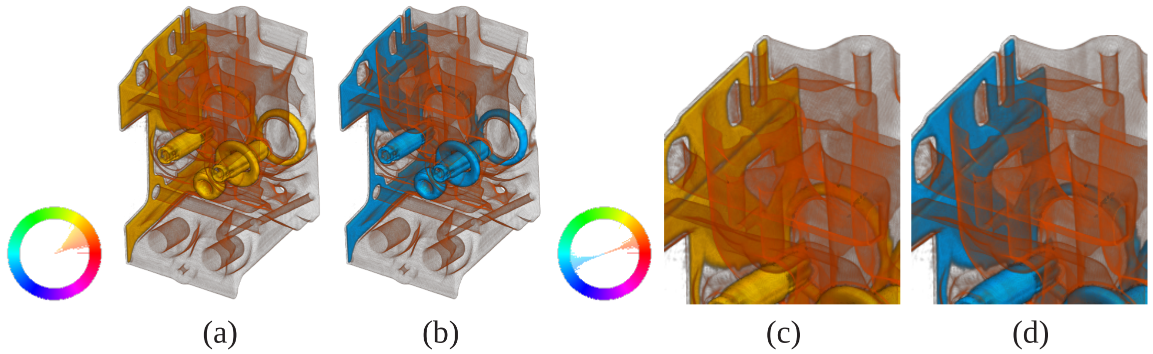


Figure 8.20: Color mixing on Engine Data. (a) Similar hues are chosen, (b) Opposite hues are chosen, (c) and (d) Zoom-in views of (a) and (b).

8.5.5 Feature Highlighting

From the equi-lightness curves A and the equi-vividness curves B on the hue slices, shown in Figure 8.22a and 8.22b, we can design color scales. Figure 8.22c and 8.22d show the equi-lightness color scales derived from curves A, with only the vividness changing, while Figure 8.22e and 8.22f show the equi-vividness color scales derived from curves b, with only the lightness changing gradually.

Based on these equi-lightness color scales, we design a scheme to highlight the features in the volume data one by one, shown in Figure 8.23. All features are always visible since the lightness is not changed, but one feature is highlighted by a more vivid color each time, drawing the observer's attention.

8.6 Discussion

In this research we have attempted to create an expert system that first captures a set of prominent guidelines from visual color design, then joins these with insights from human visual perception, and finally encodes this body of knowledge into a set of rules that can optimize the assignment of colors in 2D and 3D visualization tasks. Our system is meant to help researchers and practitioners to achieve more aesthetic color designs with ease. It seeks to eliminate the trial and error process that comes with picking the "right" colors from a set of millions. We strived to create an interface where users can select the mood of a visualization by picking from a set of suggested color palettes, with the system then performing the more tedious task of ensuring that it "looks good", assigning the lightness and saturation appropriate for the given task and goals. It turned out that these rules were applicable to also resolve a variety of existing problems in graphics and visualization, such as the color mixing artifacts when compositing semi-transparent surfaces and the brightness deviations in color harmonization. We also altered non-hue attributes to reveal the interior features in volume rendering, while preserving the visual appearance in compositing orderings.

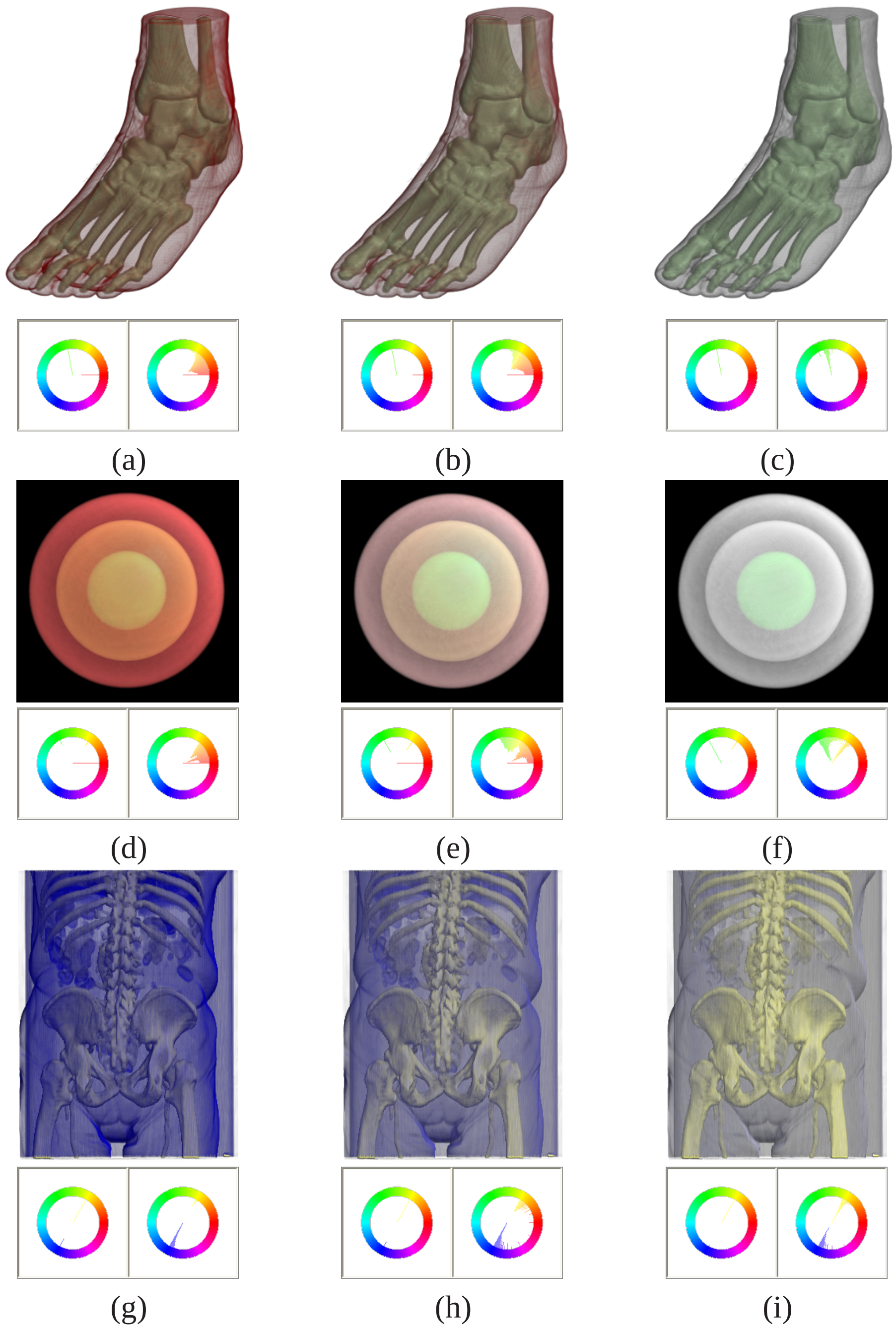


Figure 8.21: Reveal the color of inside object by decreasing the saturation of outside objects. (a)-(c) The green hue of the bone shows gradually, (d)-(f) The yellow hue of the middle sphere shows, and the green hue of most inside sphere shows, (g)-(i) The yellow hue shows when blue becomes less saturated.

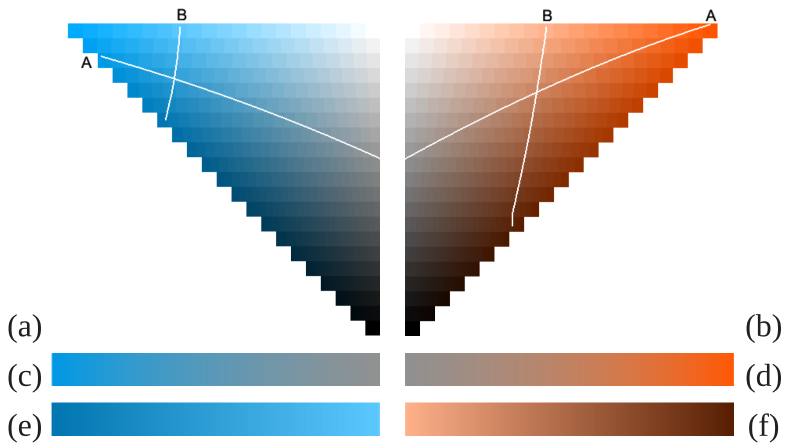


Figure 8.22: Color scales. (a) Hue slice with $h = 20$, (b) Hue slice with $h = 200$, A is equi-lightness curve and B is equi-vividness curve, (c) and (d) Equi-lightness color scales, (e) and (f) Equi-vividness color scales.

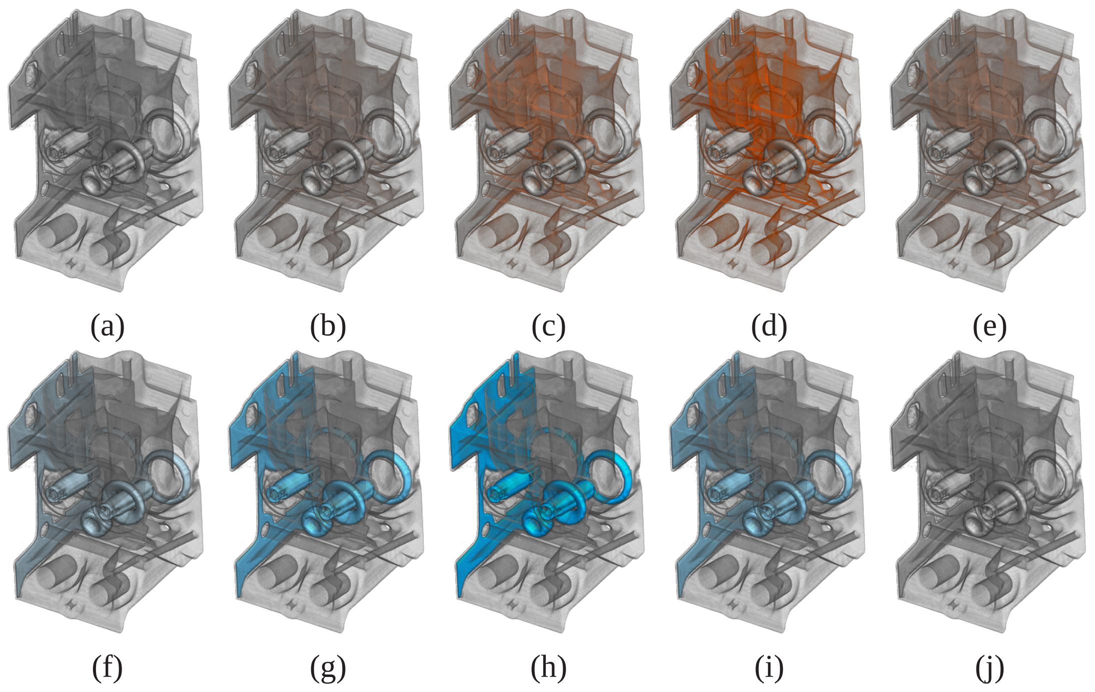


Figure 8.23: Features are highlighted one by one. (a) All features are rendered in neutral colors, no feature is highlighted, (b)-(d) The outside feature is highlighted by increasing the vividness of its color gradually, while preserving the lightness, (e) The vividness of the outside feature decreases, (f)-(h) The inside feature is highlighted gradually, (i)-(j) The vividness of the inside feature decreases.

In future work, we would like to also incorporate and test these rules for color blindness, time-varying effects, and to provide better support for different display platforms by a prior calibration step. An interesting variation would also be to use disharmonious colors for highlighting and pop-out. Finally, more formal user studies are also on our research agenda. While we have tested most of our system components with members of our lab and other affiliates, and received affirmative feedback, we plan to launch a larger study with a broader population group. Using these results we would then like to devise mechanisms that can personalize the color design for a specific user or application, using techniques from machine learning.

Chapter 9

Multi-Layer Multi-Volume Rendering

9.1 Introduction

A volume dataset can be interactively visualized with our GPU-accelerated ray casting volume rendering framework. It would be great to give users the flexibility to explore multiple volumes at the same time, which can show the intersections, help the comparison between different datasets, and provide insightful and interesting visualization results. We propose a multi-layer multi-volume rendering framework, inspired by the successful multi-layer user interface of the image editing tool Adobe Photoshop.

Methods for combining multiple volume data sets have been investigated in the context of multi-modal data. Cai and Sakas [16] discussed different methods for data intermixing in volume rendering, assuming volumes have the same size and position. Wilson et al. [143] proposed a parallel algorithm for multi-volume visualization on a PC cluster. Leu and Chen [78] presented a two-level hierarchy for modeling scenes of multiple non-intersecting volumetric objects. However, the display of intersecting semi-transparent objects can be a powerful visualization technique. The approach by Nadeau [98] can deal with intersecting volumes, but the complete scene description has to be re-sampled by voxelization whenever a volume is transformed. Grimm et al. [44] presented a method to efficiently visualize multiple intersecting volumetric objects, which uses multi-volume processing only for intersections, but efficient brick-wise volume traversal scheme for non-intersection regions. Bruckner and Gröller [13] proposed VolumeShop, an interactive hardware-accelerated application for direct volume illustration, which combines artistic visual styles and expressive visualization techniques, and allows multiple intersecting volumetric objects to be rendered directly, without requiring costly resampling. Similar to Bruckner's work, our method is based on GPU acceleration and does not require resampling, however, we bring more general concepts and interface design ideas, which are our main contributions.

9.2 Overview of Our Framework

Our multi-volume rendering framework is designed to have the following attributes:

- Multi-layer: Multiple volumes can be loaded into the system on multiple layers. Each layer has its corresponding volume, and also has its own attributes, including transfer function setting, rendering style, and transformation, which includes 3D translation, rotation, and scaling. Deformation are supported now, but can be incorporated into the framework.
- Layer On/Off: Each layer can be turn on or off, its volume will be considered or not during renderings.
- Layer Operations: Users choose an active layer, operations, such as transformation, changing color/alpha, changing the rendering style, and volume filtering, only work on the volume shown on this layer. The volume on the active layer will be separately shown in a small preview window.
- Layer Copy/Delete: Each layer can be duplicated, or deleted. Duplicated layer will refer to the source volume, and only new attributes for this layer will be created and copied from the source layer. When a layer is deleted, if its volume is not referred by other layers, this volume will unloaded (deleted) from the system.

Whenever the system setting changed, multiple volumes will be rendered by our multi-volume ray casting algorithm, discussed in the next section.

9.3 Multi-Volume Ray Casting Algorithm

There are two ways to render multiple volumes. When multiple volume datasets are transformed in object space, we can either transform each volume, resample the volumes to get a whole new dataset and then render this dataset, or we can reversely transform the image plane for each volume, and combine the rendering process for image planes. The first alternative is time consuming due to the voxelization process whenever any volume is transformed, added, deleted, or edited. We chose the latter approach of rendering multiple volumes, since it only changes viewport related attributes, volume datasets remain intact, and the rendering process is still simple and straightforward.

Our multi-volume ray casting algorithm has the following steps:

Step 1: Transform the image plane for each volume;

Step 2: Determine the ordering of volumes, and the volume intersection information;

Step 3: Raycast the intersected volumes and non-intersecting volumes, and composite the final result.

Aided by Figure 9.1, we now discuss these steps in detail. Figure 9.1a shows the 2D illustration of the bounding boxes of volume A and volume B, and the image plane S. After A is rotated and translated to A', and B is rotated, scaled, and translated to B', the transformed volumes can be rendered on the image plane S (see Figure 9.1b). Reversely,

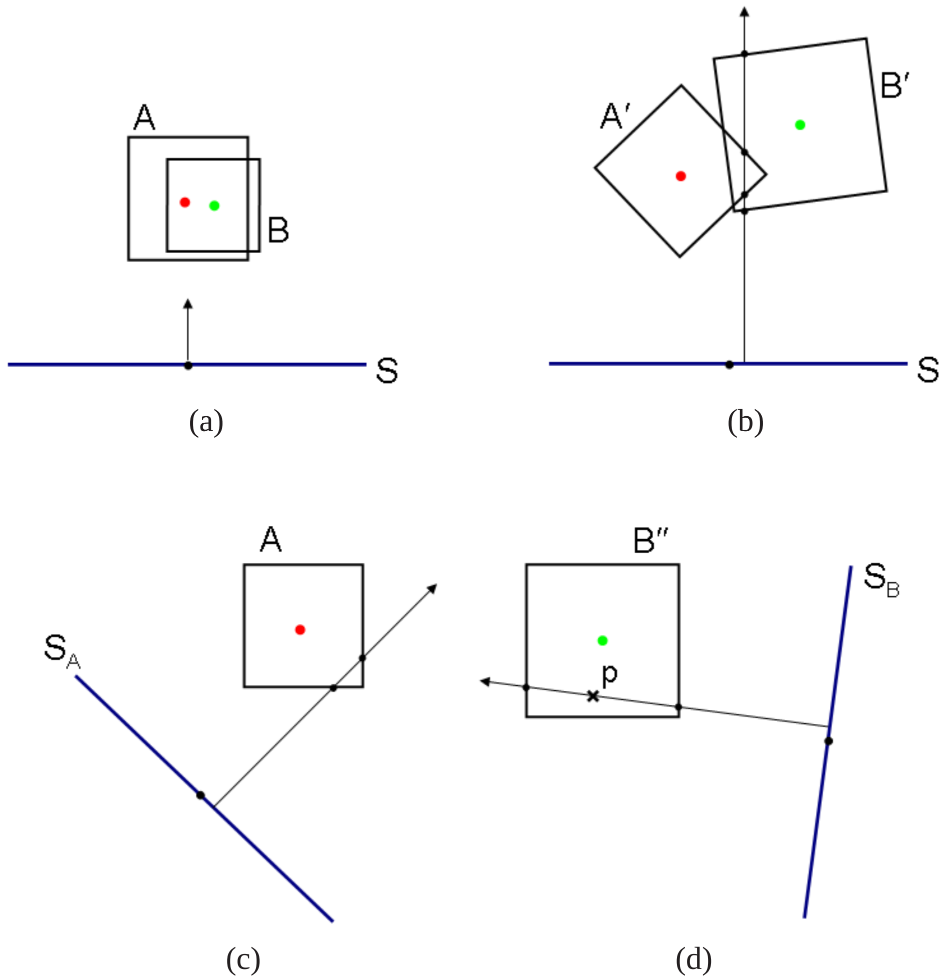


Figure 9.1: 2D illustrations of volume transformations. (a) Volumes without any transformation, (b) Volumes transform in object space, (c) and (d) Transformations taken in image space for volume A and b respectively.

our algorithm transforms the image plane. Imagining there is one image plane for each volume, our algorithm first rotates the view point (the center of the image plane), and then translate the image plane oppositely. Therefore, as shown in Figure 9.1c and 9.1d, the image planes for A and scaled B (B'') are transformed to S_A and S_B respectively. This first step of our algorithm computes all view port parameters which will be referred in later steps.

The second step is to determine the volume intersection information and the ordering of volumes. As shown in Figure 9.1b, for a pixel on the final image, one ray is shot from the image plane, and this ray intersects with both A and B. In our algorithm, a ray is shot from a pixel on the image plane S_A (or S_B), see Figure 9.1c and 9.1d, we compute the intersections of this ray and the bounding box of volume A (or B''), and store the distances from the front and back intersection points to the ray start point as *t_front* and *t_back*. For all pixels, we obtain the distance maps for each volume. From all distance maps, the ordering and volume intersection information can be detected. Note, in order to get the correct distances, in our algorithm, the bounding box of the scaled volume instead of the original volume is used to calculate the intersections. This is the reason why we have not considered the scale transformation in the first step. The bounding box of the scaled volume can be computed as follows:

$$V_{BoundingBox} = (V_{BoundingBox} - V_{Center}) * V_{Scale} + V_{Center}, \tag{9.1}$$

where $V_{BoundingBox}$ has the volume's minimal and maximal values of x, y, and z coordinates. V_{Center} is the central point of the bounding box of unscaled volume. V_{Scale} is the volume's scale value.

For non-intersecting volume, the Cg fragment program for single volume rendering can be applied, and the final color can be correctly composed for several volumes based on their ordering. For intersected volumes, we design fragment programs for different cases. For example, for two different volumes intersection, two volume datasets along with their view port parameters, scale values, and distance maps are fed into the fragment program. During ray casting, for each sample point on the ray, we know whether it is in the intersection and non-intersection region from distance values t_front and t_back of two volumes. Therefore, in non-intersection regions, only the color getting from one volume is composited. In intersection regions, different methods can be applied to composite colors getting from two volumes. The distance is calculated based on the scaled bounding box, but the volume dataset is unscaled. Therefore, for any sampling point, such as p shown in Figure 9.1d, its coordinates should be scaled back to the original ones:

$$p = (p - V_{Center})/V_{Scale} + V_{Center}., \tag{9.2}$$

For the duplicated volumes, we only need to feed one volume dataset into the fragment program as well as the parameters for all views. Furthermore, the fragment program for more than two volume intersection is similar. Our algorithm works through these steps to provide correct volume intersection and composition results. For our GPU-accelerated rendering, the memory on the graphics card limits the amount of volume datasets which can be fed into the fragment program at one time. We have not do any special optimizations to improve the performance, which can be our future work.

9.4 Results

From the above mentioned three steps of our algorithm, multi-volume visualization results can be generated. Figure 9.2 shows multi-volume renderings with different settings. Volumes can be transformed to get interesting postures. For the color composition of sampling points in intersection regions, we use averaged color (each color is assigned 0.5 as the weight). The intersections between Foot and Engine can be seen in 9.2a and 9.2b, with semi-transparent transfer function settings. With our framework, users can also explore

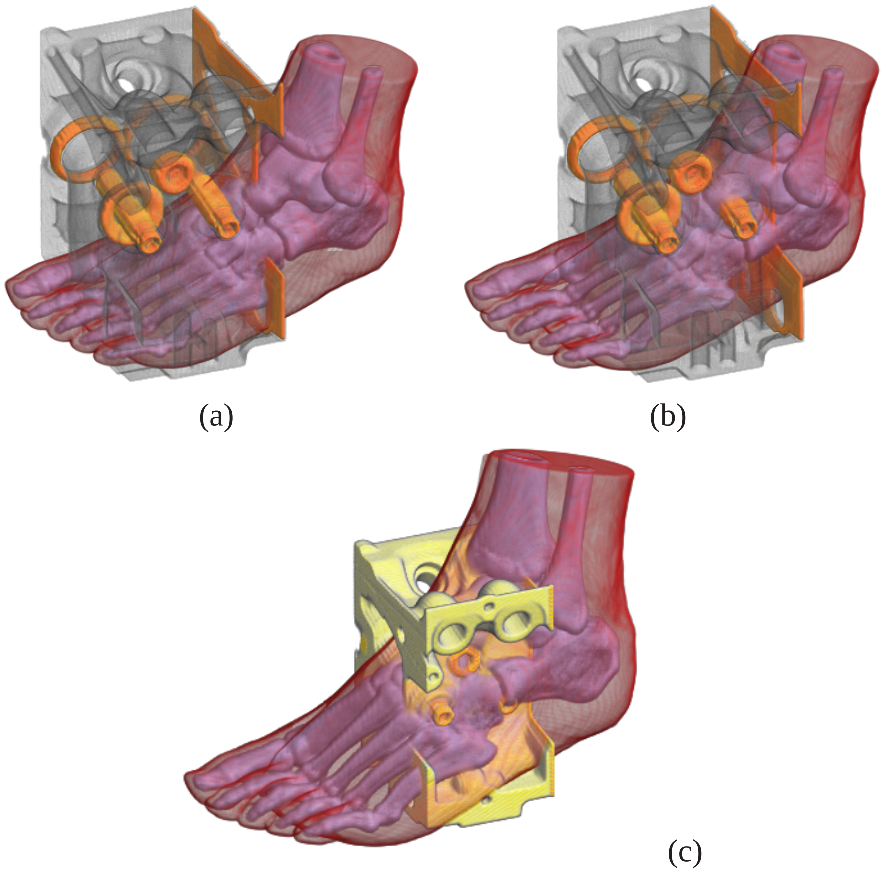


Figure 9.2: Multi-volume renderings. (a) and (b) Foot moves into Engine, (c) More opaque transfer function setting is applied for Engine.

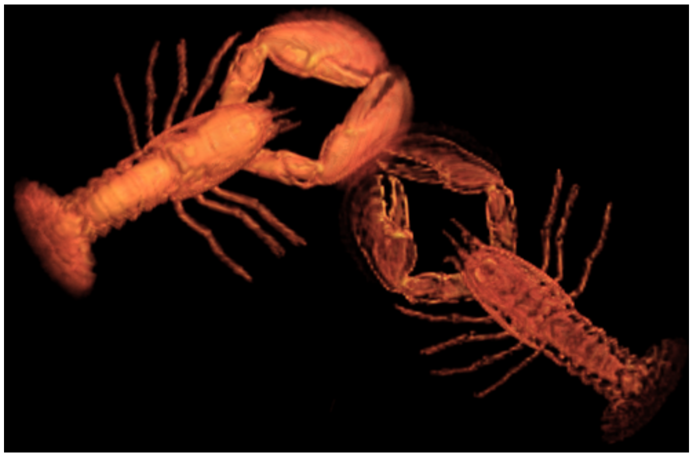


Figure 9.3: Volume exploration example. The original volume is visualized along with the edited volume showing its highest frequency details.

volumes in different ways. Figure 9.3 shows an example that users can visualize the original volume and filtered volume together, which is a easy way to compare the differences. Here multi-resolution details of the volume, i.e. a pyramid of high frequency volumes and a low frequency volume at the lowest level, are generated using low-passing filtering. The volume with the highest frequency details are rendered along with the original volume.

9.5 Discussion

We propose a multi-layer style multi-volume rendering framework, and present our multi-volume rendering algorithm. Our framework brings more general concepts than Bruckner's VolumeShop, and provides useful ideas in terms of the user interface. It is a general 3D rendering and editing tool for volumes, but it is still in the preliminary stage. There are many opportunities for improving the performance, and incorporating more helpful concepts and ideas into the framework. We propose to also provide more choices on composition methods to explore more insightful visualizations.

Chapter 10

Conclusion

In this thesis, we present several novel feature-driven illustrative visualization and graphics techniques to enhance the features of interest in the dataset. Instead of focusing on studying one illustrative technique, we try to explore multiple good concepts or techniques that could be helpful to the illustrative visualization and graphics.

Our techniques aim to improve the perceived quality of features of interest either from detail or perception points of view. Detail related techniques include magnifying features of interest in a smart way, generating multi-resolution details with different semantic meanings from available examples, and uniforming the size of features. Perception based techniques include measuring the perceived quality of volume rendering using the conjoint analysis based framework, and assigning features colors for visualization tasks through our rule based color design system. We also work on illustrating different views in one image, and propose a multi-volume rendering framework. We believe our work will benefit the visualization and graphics research, and draw more attentions to some aspects which are helpful for illustrative visualization, but have not been touched or thoroughly studied before. The primary highlights of this thesis are:

- An universal and general volumetric lens framework that has applications in many domains has been presented. It allows users to apply any well known lenses, such as a fisheye lens in the context of volumetric distortion, as well as design free-style and feature-adaptive lenses for arbitrary magnified focus+context viewing. Incorporating the GPU-accelerated ray casting, we can interactively exploit features of interest in a more efficient way. We give a solution to the limited screen problem. A multiperspective rendering problem is also studied in our framework.

- A new constrained multi-scale texture synthesis method is proposed to facilitate semantic zooms. Our 2D viewing application, a virtual microscope, demonstrated that quite interesting and useful image sequences can be generated using our framework. And our technique is extend to 3D volumetric viewing. We present a possible solution to the limited zoom or limited data resolution problem.

- A simple texture synthesis algorithm for surfaces with arbitrary topology using global conformal parameterization is proposed. We also present an algorithm based on the global parameterization to simultaneously preserve angle and scale in texture mapping. We show that a conformal-factor driven mass-spring method offers a convenient way to trade off these two metrics. Our algorithms are simple, efficient and theoretically sound.

- A rule based color design system is proposed. The rules, which guide the color design from both aesthetic and attention-guiding, salient point of views, and have been established in various classic color design books, are captured into a color selection framework, providing appropriate colorizations based on user preferences, importance functions, and scene composition of the visualization tasks.

In the future, we could continue improving and extending our techniques. The multilayer visualization framework is promising for efficiently exploring multiple volumes. To make it feasible, we will improve the algorithm and performance, and coupling more useful operations and concepts into it. It would be great to embed the semantic zoom work into our magic volume lens framework. Although now it is hard to make an interactive constrained detail synthesis in our semantic zoom, it may be accomplished with the improvements of texture synthesis techniques. Illustrating feature correspondences between renderings with different viewpoints will benefit the visualization. Our multiperspective rendering does not work as we expected for the volume dataset. Using illustration based method to show the correspondences between features maybe a feasible solution. Applying conjoint analysis method to evaluate the color design results will help us testify our color selection method, and even discover new rules.

In illustrative visualization and graphics field, there are still a lot of problems deserve further study, and many potential improvements could be achieved. Incorporating more principles of human perception in the research is a promising way.

Generally, our work on feature-driven illustrative visualization and graphics achieves some encouraging results. In the future, there will always be new concepts or techniques emerging or embed in the visualization to provides even better results.

Bibliography

[1] M. Agrawala, D. Zorin, and T. Munzner. Artistic multiprojection rendering. In *Proc. of Eurographics Rendering Workshop '00*, 2000.

[2] M. Ashikhmin. Synthesizing natural textures. In *Proc. of ACM Symposium on Interactive 3D Graphics '01*, pages 217–226, 2001.

[3] M. Ashikhmin. Synthesizing natural textures. *IEEE Computer Graphics and Applications*, 23(4):38–43, 2003.

[4] B. Bauer, P. Jolicoeur, and W. Cowan. Distractor heterogeneity versus linear separability in visual search. *Perception*, 25:1281–1294, 1996.

[5] L. D. Bergman, B. E. Rogowitz, and L. A. Treinish. A rule-based tool for assisting colormap selection. In *Proc. of IEEE Visualization '95*, pages 118–125, 1995.

[6] S. Bergner, T. Möller, D. Weiskopf, and D. J. Muraki. A spectral analysis of function composition and its implications for sampling in direct volume visualization. *IEEE Trans. on Visualization and Computer Graphics*, 12(5):2006, 1353–1360.

[7] B. Berlin and P. Kay. *Basic Color Terms: Their Universality and Evolution*. University of California Press, Berkeley, 1969.

[8] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses: The see-through interface. *Computer Graphics*, 27:73–80, 1993.

[9] U. D. Bordoloi and H.-W. Shen. View selection for volume rendering. In *Proc. of IEEE Visualization '05*, pages 487–494, 2005.

[10] P. Bourke. Computer generated angular fisheye projections. URL://astronomy.swin.edu.au/∼pbourke/projection/fisheye/, 2001.

[11] S. Bruckner, S. Grimm, A. Kanitsar, and M. Gröller. Illustrative context-preserving exploration of volume data. *IEEE Trans. on Visualization and Computer Graphics*, 12(6):2006, 1559–1569.

[12] S. Bruckner, S. Grimm, A. Kanitsar, and M. E. Gröller. Illustrative context-preserving volume rendering. In *Proc. of EuroVis 2005*, pages 69–76, 2005.

[13] S. Bruckner and M. E. Gröller. Volumeshop: An interactive system for direct volume illustration. In *Proc. of IEEE Visualization '05*, pages 671–678, 2005.

[14] S. Bruckner and M. E. Gröller. Style transfer functions for illustrative volume rendering. *Computer Graphics Forum (accepted for publication) (Eurographics '07)*, 26(3), 2007.

[15] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Symposium on Volume Visualization '94*, pages 91–98, 1994.

[16] W. Cai and G. Sakas. Data intermixing and multi-volume rendering. *Computer Graphics Forum*, 18(3):359–368, 1999.

[17] T. C. Callaghan. Interference and domination in texture segregation: Hue, geometric form, and line orientation. *Perception and Psychophysics*, 46(4):299–311, 1989.

[18] M. S. T. Carpendale, D. J. Cowperthwaite, and F. D. Fracchia. Distortion viewing techniques for 3-dimensional data. In *Proc. of IEEE Symposium on Information Visualization '96*, pages 46–53, 1996.

[19] M. S. T. Carpendale, D. J. Cowperthwaite, and F. D. Fracchia. Extending distortion viewing from 2D to 3D. *IEEE Computer Graphics and Applications: Special Issue on Information Visualization*, 17(4):42–51, 1997.

[20] W. Chen, L. Ren, M. Zwicker, and H. Pfister. Hardware-accelerated adaptive EWA volume splatting. In *Proc. of IEEE Visualization '04*, 2004.

[21] P. Cignoni, C. Montani, and R. Scopigno. Magicsphere: an insight tool for 3D data visualization. *Computer Graphics Forum (Proc. of Eurographics '94)*, 13(3):317–328, 1994.

[22] M. Cohen and K. Brodlie. Focus and context for volume visualization. In *Proc. of Theory and Practice of Computer Graphics '04*, pages 32–39, 2004.

[23] M. F. Cohen, J. Shade, S. Hiller, and O. Deussen. Wang tiles for image and texture generation. In *Proc. of SIGGRAPH '03*, volume 22, pages 287–294, 2003.

[24] D. Cohen-Or, O. Sorkine, R. Gal, T. Leyvand, and Y.-Q. Xu. Color harmonization. *ACM Trans. on Graphics (SIGGRAPH '06)*, 25(3):624–630, 2006.

[25] F. C. Crow. Summed-area tables for texture mapping. *Computer Graphics (Proc. of SIGGRAPH '84)*, 18(3):207–212, 1984.

[26] P. Degener, M. J., and R. Klein. An adaptable surface parameterization method. In *Proc. of 12th International Meshing Roundtable*, pages 201–213, 2003.

[27] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. In *Proc. of Eurographics '02*, volume 12, pages 209–218, 2002.

[28] J. Diepstraten, D. Weiskopf, and T. Ertl. Interactive cutaway illustrations. In *Proc. of Eurographics '03*, pages 523–532, 2003.

[29] J. M. Dischler, K. Maritaud, B. Lévy, and D. Ghazanfarpour. Texture particles. *Computer Graphics Forum*, 21(3):401–410, 2002.

[30] R. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. In *Proc. of SIGGRAPH '88*, volume 22, pages 65–74, 1988.

[31] I. Drori, D. Cohen-or, and H. Yeshurun. Fragment-based image completion. In *Proc. of SIGGRAPH '03*, pages 303–312, 2003.

[32] T. Duchamp, A. Certain, A. DeRose, and W. Stuetzle. Hierarchical computation of PL harmonic embeddings. Technical report, University of Washington, July 1997.

[33] D. H. Eberly. *Game Physics*. Morgan Kaufmann, 2004.

[34] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *Proc. of SIGGRAPH '01*, pages 341–346, 2001.

[35] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *Proc. of International Conference on Computer Vision '99*, volume 2, pages 1033–1038, 1999.

[36] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proc. of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware '01*, pages 9–16, 2001.

[37] M. C. Escher. *The Graphic Work*. Evergreen, Germany, 1992.

[38] M. S. Floater and K. Horman. Surface parameterization: a tutorial and survey. In *Advances in Multiresolution in Geometric Modelling*. Springer, 2004.

[39] W. T. Freeman, T. R. Jones, and E. Pasztor. Example-based super-resolution. *IEEE Computer Graphics and Applications*, pages 56–65, 2002.

[40] G. W. Furnas and B. B. Bederson. Space-scale diagrams: understanding multiscale interfaces. In *Proc. of CHI'95 Human Factors in Computing Systems*, pages 234–241, 1995.

[41] A. Glassner. Cubism and cameras: Freeform optics for computer graphics. *Microsoft Research MSR-TR-2000-05*, 2000.

[42] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley Pub Co., Boston, MA, 1992.

[43] S. J. Gortler, C. Gotsman, and D. Thurston. One-forms on meshes and applications to 3d mesh parameterization. Technical Report CS TR-12-04, Harvard University, 2004.

[44] S. Grimm, S. Bruckner, A. Kanitsar, and E. Gröller. Flexible direct multi-volume rendering in interactive scenes. In *Proc. of Vision, Modeling, and Visualization '04*, pages 379–386, 2004.

[45] X. Gu, Y. He, and H. Qin. Manifold splines. In *Proc. of ACM Symposium on Solid and Physical Modeling '05*, 2005.

[46] X. Gu and S.-T. Yau. Global conformal surface parameterization. In *Proc. Eurographics/SIGGRAPH Symposium on Geometry Processing '03*, pages 127–137, 2003.

[47] R. Gupta and R. Hartley. Linear pushbroom cameras. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(9):963–975, 1997.

[48] A. Gustafsson, A. Herrmann, and F. Huber. Conjoint analysis as an instrument of market research practice. *Conjoint Measurement, Methods and Applications.*, pages 5–45, 2000.

[49] S. Guthe, M. Wand, J. Gonser, and W. Straβr. Interactive rendering of large volume data sets. In *Proc. of IEEE Visualization '02*, pages 53–60, 2002.

[50] M. Hadwiger, C. Berger, and H. Hauser. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *Proc. of IEEE Visualization '03*, pages 40–47, 2003.

[51] H. Hauser, L. Mroz, G.-I. Bischi, and E. Gröller. Two-level volume rendering-fusing MIP and DVR. In *Proc. of IEEE Visualization '00*, pages 211–218, 2000.

[52] C. G. Healey. Choosing effective colours for data visualization. In *Proc. of IEEE Visualization '96*, pages 263–270, 1996.

[53] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *Proc. of SIGGRAPH '01*, pages 327–340, 2001.

[54] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He. Virtual voyage: Interactive navigation in the human colon. In *Proc. of SIGGRAPH '97*, pages 27–34, 1997.

[55] D. H. House, A. Bair, and C. Ware. An approach to the perceptual optimization of complex visualizations. *IEEE Trans. on Visualization and Computer Graphics*, 12(4):2006, 509–521.

[56] J. Itten. *The Art of Color*. Van Nostrand Reinhold Company, New York, 1961.

[57] R. Jagnow, J. Dorsey, and H. Rushmeier. Stereological techniques for solid textures. *ACM Trans. on Graphics (SIGGRAPH '04)*, 23(3):329–335, 2004.

[58] G. Ji and H. Shen. Dynamic view selection for time-varying volumes. *IEEE Trans. on Visualization and Computer Graphics*, 12(5):2006, 1109–1116.

[59] J. Jia and C.-K. Tang. Inference of segmented color and texture description by tensor voting. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(6):771–786, 2004.

[60] M. Jin, Y. Wang, S.-T. Yau, and X. Gu. Optimal global conformal surface parameterization. In *Proc. of IEEE Visualization '04*, pages 267–274, 2004.

[61] A. Joshi and P. Rheigans. Illustration-inspired techniques for visualizing time-varying data. In *Proc. of IEEE Visualization '05*, pages 679–686, 2005.

[62] J. Jost. *Compact Riemann Surfaces*. Springer, 2000.

[63] A. Kaufman and K. Mueller. Volume visualization and volume graphics. URL://www.cs.sunysb.edu/ mueller/teaching/cse332/volvisChapter.pdf, 2003.

[64] T. Keahey and E. Robertson. Techniques for non-linear magnification transformations. In *Proc. of IEEE Symposium on Information Visualization '96*, pages 38–45, 1996.

[65] T. Keahey and E. Robertson. Nonlinear magnification fields. In *Proc. of IEEE Symposium on Information Visualization '97*, pages 41–49, 1997.

[66] S. L. Kilthau, M. S. Drew, and T. Möller. Full search content independent block matching based on the fast fourier transform. In *proc. of IEEE ICIP '02*, pages 669–672, 2002.

[67] Y. Kim and A. Varshney. Saliency-guided enhancement for volume visualization. *IEEE Trans. on Visualization and Computer Graphics*, 12(5):925–932, 2006.

[68] G. L. Kindlmann, E. Reinhard, and S. Creem. Face-based luminance matching for perceptual colormap generation. In *Proc. of IEEE Visualization '02*, pages 309–406, 2002.

[69] J. Kniss, S. Premoze, M. Ikits, A. Lefohn, C. Hansen, and E. Praun. Gaussian transfer functions for multi-field volume visualization. In *Proc. of IEEE Visualization '03*, pages 497–504, 2003.

[70] C. Kolb, D. Mitchell, and P. Hanrahan. A realistic camera model for computer graphics. In *Proc. of the 22nd annual conference on Computer graphics and interactive techniques*, pages 317–324, 1995.

[71] R. Kosara, S. Miksch, and H. Hauser. Semantic depth of field. In *Proc. of InfoVis '01*, pages 97–104, 2001.

[72] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proc. of IEEE Visualization '03*, pages 287–292, 2003.

[73] Y. Kurzion. *Visualization Enhancement by Embedding Local and Global Modeling Operations in the Rendering Process*. PhD thesis, The Ohio State University, 1998.

[74] V. Kwatra, A. Schöl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. In *Proc. of SIGGRAPH '03*, volume 22, pages 277–286, 2003.

[75] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proc. of SIGGRAPH '94*, pages 451–458, 1994.

[76] Y.-K. Lai, S.-M. Hu, X. Gu, and R. R. Martin. Geometric texture synthesis and transfer via geometry images. In *Proc. of ACM Symposium on Solid and Physical Modeling*, 2005.

[77] E. LaMar, B. Hamann, and K. I. Joy. A magnification lens for interactive volume visualization. In *Proc. of the Ninth Pacific Conference on Computer Graphics and Applications*, pages 223–232, 2001.

[78] A. Leu and M. Chen. Modelling and rendering graphics scenes composed of multiple volumetric datasets. *Computer Graphics Forum*, 18(2):159–171, 1999.

[79] W. Leung, N. Neophytou, and K. Mueller. SIMD-aware ray casting. In *Volume Graphics Workshop '06*, pages 59–62, 2006.

[80] Y. K. Leung and M. D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Trans. on Computer-Human Interaction*, 1(2):126–160, 1994.

[81] H. Levkowitz and G. T. Herman. GLHS: A generalized lightness, hue, and saturation color model. *CVGIP: Graphical Model and Image Processing*, 55(4):271–285, 1993.

[82] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Application*, 8(5):29–37, 1988.

[83] M. Levoy. Efficient ray tracing of volume data. *ACM Trans. on Computer Graphics*, 9(3):245–261, 1990.

[84] M. Levoy and R. Whitaker. Gaze-directed volume rendering. *Computer Graphics (Proc. of Symposium on Interactive 3D Graphics '90)*, 24(2):217–223, 1990.

[85] B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. In *Proc. of SIGGRAPH ’02*, volume 21, pages 362–371, 2002.

[86] W. Li, K. Mueller, and A. Kaufman. Empty space skipping and occlusion clipping for texture-based volume rendering. In *Proc. of IEEE Visualization ’03*, pages 317–324, 2003.

[87] L. Liang, C. Liu, Y. Xu, B. Guo, and H.-Y. Shum. Real-time texture synthesis by patch-based sampling. *ACM Trans. on Graphics*, 20(3):127–150, 2001.

[88] H. Loffelmann and E. Groller. Ray tracing with extended cameras. *The Journal of Visualization and Computer Animation*, 7(4):211–227, 1996.

[89] A. Lu and D. Ebert. Example-based volume illustrations. In *Proc. of IEEE Visualization ’05*, pages 655–662, 2005.

[90] A. Lu, C. Morris, J. Taylor, D. Ebert, C. Hansen, P. Rheingans, and M. Hartner. Illustrative interactive stipple rendering. *IEEE Trans. on Visualization and Computer Graphics*, 9(2):127–138, 2003.

[91] T. Malzbender and F. Kitson. A fourier technique for volume rendering. *Focus on Scientific Visualization*, pages 305–316, 1991.

[92] Y. Matsuda. *Color design (in Japanese)*. Asakura Shoten, 1995.

[93] M. J. McGuffin, L. Tancau, and R. Balakrishnan. Using deformations for browsing volumetric data. In *Proc. of IEEE Visualization ’03*, pages 401–408, 2003.

[94] B. J. Meier, A. M. Spalter, and D. B. Karelitz. Interactive color palette tools. *IEEE Computer Graphics and Applications*, 24(3):64–72, 2004.

[95] P. Moon and D. E. Spencer. Geometrical formulation of classical color harmony. *Journal of the Optical Society of America*, 34(1):46–60, 1944.

[96] K. Mueller and R. Crawfis. Eliminating popping artifacts in sheet buffer-based splatting. In *Proc. of IEEE Visualization ’98*, pages 239–245, 1998.

[97] A. H. Munsell. *A Grammar of Colors*. New York: Van Nostrand Reinhold Company, 1969.

[98] D. R. Nadeau. Volume scene graphs. In *Proc. of the IEEE symposium on Volume visualization ’00*, pages 49–56, 2000.

[99] A. Nealen and M. Alexa. Hybrid texture synthesis. In *Proc. of the 14th Eurographics workshop on Rendering*, pages 97–105, 2003.

[100] N. Neophytou and K. Mueller. GPU accelerated image aligned splatting. In *Volume Graphics Workshop '05*, pages 197–205, 2005.

[101] F. Neyret and M.-P. Cani. Pattern-based texturing revisited. In *Proc. of SIGGRAPH '99*, pages 235–242, 1999.

[102] S. Owada, F. Nielsen, M. Okabe, and T. Igarashi. Volumetric illustration: Designing 3d models with internal textures. pages 322–328, 2004.

[103] R. Paget and I. D. Longstaff. Texture synthesis via a noncausal nonparametric multiscale markov random field. *IEEE Trans. on Image Processing*, 7(6):925–931, 1998.

[104] K. Perlin and D. Fox. Pad: an alternative approach to the computer interface. In *Proc. of SIGGRAPH '93*, pages 57–64, 1993.

[105] B. Pflesser, U. Tiede, and K. H. Höhne. Towards realistic visualization for surgery rehearsal. In *Proc. of Computer Vision, Virtual Reality and Robotics in Medicine*, pages 487–491, 1995.

[106] U. Pinkall and K. Polthier. Computing discrete minimal surfaces and their conjugate. *Experimental Mathematics*, 2(1):15–36, 1993.

[107] E. Praun, A. Finkelstein, and H. Hoppe. Lapped textures. In *Proc. of SIGGRAPH '00*, pages 465–470, 2000.

[108] P. Rademacher and G. Bishop. Multiple-center-of-projection images. In *Proc. of SIGGRAPH '98*, pages 199–206, 1998.

[109] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume rendering on standard pc graphics hardware using multi-textures and multi-stage rasterization. In *Proc. of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware '00*, pages 109–118, 2000.

[110] P. Rheingans. Task-based color scale design. In *Proc. of Applied Image and Pattern Recognition '99*, pages 35–43, 1999.

[111] P. Rheingans and D. Ebert. Volume illustration: Nonphotorealistic rendering of volume models. *IEEE Trans. on Visualization and Computer Graphics*, 7(3):253–264, 2001.

[112] B. E. Rogowitz and A. D. Kalvin. The "Which Blair Project": a quick visual method for evaluating perceptual color maps. In *Proc. of IEEE Visualization '01*, pages 183–190, 2001.

[113] B. E. Rogowitz and L. Treinish. An architecute for rule-based visualization. In *Proc. of IEEE Visualization '93*, pages 236–244, 1993.

[114] A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa. Video textures. In *Proc. of SIGGRAPH '00*, pages 489–498, 2000.

[115] P. Shanbhag, P. Rheingans, and M. desJardins. Temporal visualization of planning polygons for efficient partitioning of geo-spatial data. In *Proc. of IEEE Symposium on Information Visualization '05*, page 28, 2005.

[116] A. Sheffer and E. Sturler. Parameterization of faceted surfaces for meshing using angle-based flattening. *Engineering with Computers*, 17(3):326–337, 2001.

[117] C. Soler, M. P. Cani, and A. Angelidis. Hierarchical pattern mapping. In *Proc. of SIGGRAPH '02*, volume 21, pages 673–680, 2002.

[118] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl. Simple and flexible volume rendering framework for graphics-hardware-based raycasting. In *Volume Graphics Workshop '05*, pages 187–195, 2005.

[119] M. Stone. *A Field Guide to Digital Color*. A.K. Peters, Natick, MA, 2003.

[120] K. Strebel. *Quadratic Differentials*. Springer-Verlag, 1984.

[121] N. Svakhine, Y. Jang, D. Ebert, and K. Gaither. Illustration and photography inspired visualization of flows and volumes. In *Proc. of IEEE Visualization '05*, pages 687–694, 2005.

[122] N. A. Svakhine, D. S. Ebert, and D. Stredney. Illustration motifs for effective medical volume illustration. *IEEE Computer Graphics and Applications*, 25(3):31–39, 2005.

[123] S. Takahashi, I. Fujishiro, Y. Takeshima, and T. Nishita. A feature-driven approach to locating optimal viewpoints for volume visualization. In *Proc. of IEEE Visualization '05*, pages 495–502, 2005.

[124] L. L. Thurstone. A law of comparative judgement. *Psychological Review*, 34:1927, 273–286.

[125] X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo, and H.-Y. Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. In *Proc. of SIGGRAPH '02*, volume 21, pages 665–672, 2002.

[126] L. Tonietto and M. Walter. Towards local control for image-based texture synthesis. In *Proc. of XV Brazilian Symposium on Computer Graphics and Image Processing*, pages 252–258, 2002.

[127] M. Tory and C. Swindells. Comparing ExoVis, orientation icon, and in-place 3D visualization techniques. In *Proc. of Graphics Interface '03*, pages 57–64, 2003.

[128] S. Treavett and M. Chen. Pen-and-ink rendering in volume visualization. In *Proc. of IEEE Visualization '00*, pages 203–210, 2000.

[129] G. Turk. Texture synthesis on surfaces. In *Proc. of SIGGRAPH '01*, pages 347–354, 2001.

[130] S. Vallance and P. R. Calder. Multi-perspective images for visualisation. In *VIP '01: Pan-Sydney Area Workshop on Visual Information Processing*, pages 69–76, 2001.

[131] I. Viola. *Importance-Driven Expressive Visualization*. PhD thesis, Vienna University of Technology, Austria, 2005.

[132] I. Viola, E. Gröler, K. Bühler, M. Hadwiger, B. Preim, D. Ebert, M. C. Sousa, and D. Stredney. Illustrative visualization. IEEE Visualization '05 tutorial, Minneapolis, MN, 2005.

[133] I. Viola, A. Kanitsar, and M. E. Gröller. Importance-driven volume rendering. In *Proc. of IEEE Visualization '04*, pages 139–145, 2004.

[134] I. Viola, A. Kanitsar, and M. E. Gröller. Importance-driven feature enhancement in volume visualization. *IEEE Trans. on Visualization and Computer Graphics*, 11(4):408–418, 2005.

[135] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann, San Francisco, second edition, 2004.

[136] L.-Y. Wei. *Texture synthesis by fixed neighborhood searching*. PhD thesis, Stanford University, 2001.

[137] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proc. of SIGGRAPH '00*, pages 479–488, 2000.

[138] L.-Y. Wei and M. Levoy. Texture synthesis over arbitrary manifold surfaces. In *Proc. of SIGGRAPH '01*, pages 355–360, 2001.

[139] X. Wei, A. E. Kaufman, and T. J. Hallman. Case study: visualization of particle track data. In *Proc. of IEEE Visualization '01*, pages 465–468, 2001.

[140] D. Weiskopf, T. Schafhitzel, and T. Ertl. GPU-based nonlinear ray tracing. *Computer Graphics Forum*, 23(3):625–634, 2004.

[141] L. Westover. Footprint evaluation for volume rendering. In *Proc. of SIGGRAPH '90*, pages 367–376, 1990.

[142] Wikipedia. the free encyclopedia. http://wikipedia.org/, 2006.

[143] B. Wilson, E. B. Lum, and K.-L. Ma. Interactive multi-volume visualization. In *Proc. of the International Conference on Computational Science '02*, pages 102–110, 2002.

[144] G. Wolberg. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[145] H.-C. Wong, H. Qu, U.-H. Wong, Z. Tang, and K. Mueller. A perceptual framework for comparisons of direct volume rendered images. In *Proc. of IEEE Pacific-Rim Symposium on Image and Video Technology*, 2006.

[146] W. Wong. *Principles of Color Design*. Wiley, 1996.

[147] D. Wood, A. Finkelstein, J. Hughes, C. Thayer, and D. Salesin. Multiperspective panoramas for cel animation. In *Proc. of SIGGRAPH '97*, pages 243–250. ACM, 1997.

[148] Y. Q. Xu, B. Guo, and H.-Y. Shum. Chaos mosaic: fast and memory efficient texture synthesis. Technical Report MSR-TR-2000-32, Microsoft Research, 2000.

[149] H. Yamauchi, J. Haber, and H.-P. Seidel. Image restoration using mutli-resolution image synthesis and image inpainting. In *Proc. Computer Graphics International '03*, pages 120–125, 2003.

[150] L. Ying, A. Hertzmann, H. Biermann, and D. Zorin. Texture and shape synthesis on surfaces. In *Proc. of 12th Eurographics Workshop on Rendering '01*, pages 301–312, 2001.

[151] S. Yoshizawa, A. Belyaev, and H.-P. Seidel. A fast and simple stretch-minimizing mesh parameterization. In *Proc. of Shape Modeling and Applications '04*, pages 200–208, 2004.

[152] J. Yu and L. McMillan. A framework for multiperspective rendering. *Rendering Techniques '04, Eurographics Symposium on Rendering (EGSR)*, pages 61–68, 2004.

[153] J. Yu and L. McMillan. General linear cameras. In *Proc. of the 8th European Conference on Computer Vision (ECCV) '04*, pages 14–27, 2004.

[154] B. Yuri and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. In *Third International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 359–374, 2001.

[155] B. Yuri, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. In *International Conference on Computer Vision*, pages 377–384, 1999.

[156] R. Zakia. *Perception and Imaging*. Focal Press, second edition, 2001.

[157] S. Zelinka and M. Garland. Towards real-time texture synthesis with the jump map. In *Proc. of the 13th Eurographics Workshop on Rendering Techniques '02*, pages 99–104, 2002.

[158] J. Zhou, A. Döring, and K. D. Tönnies. Distance based enhancement for focal region based volume rendering. In *Proc. of Bildverarbeitung für die Medizin '04*, pages 199–203, 2004.

[159] J. Zhou, M. Hinz, and K. D. Tönnies. Focal region-guided feature-based volume rendering. In *Proc. of* $1st$ *International Symposium on 3D Data Processing Visualization and Transmission*, pages 87–90, 2002.

[160] A. Zomet, D. Feldman, S. Peleg, and D. Weinshall. Mosaicing new views: The crossed-slits projection. *IEEE Trans. on PAMI*, pages 741–754, 2003.

[161] M. Zwicker, H. Pfister, J. V. Baar, and M. Gross. EWA volume splatting. In *Proc. of IEEE Visualization '01*, pages 29–36, 2001.