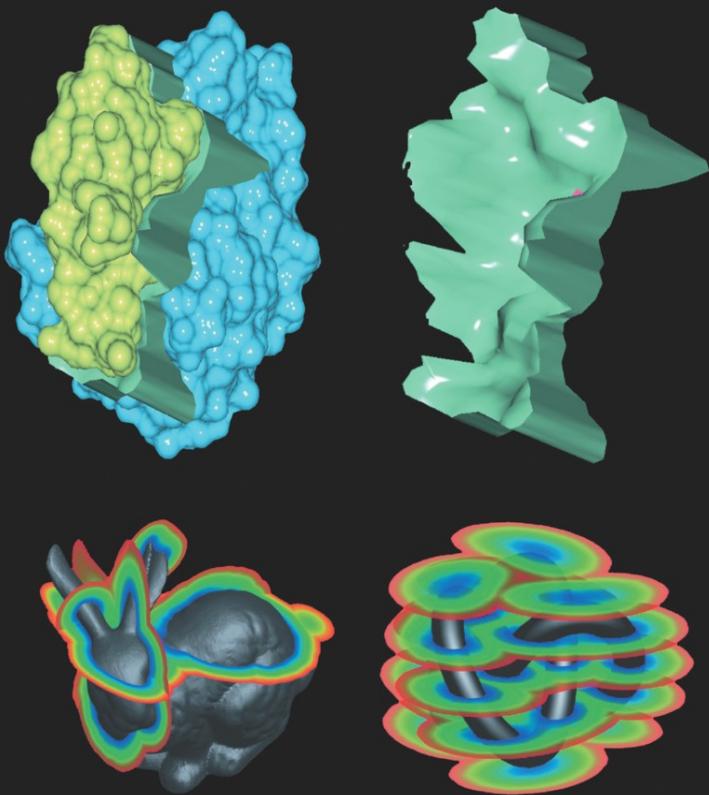


G.-P. Bonneau T. Ertl G. M. Nielson (Editors)

Scientific Visualization: The Visual Extraction of Knowledge from Data



Mathematics and Visualization

Series Editors

Gerald Farin

Hans-Christian Hege

David Hoffman

Christopher R. Johnson

Konrad Polthier

Martin Rumpf

Georges-Pierre Bonneau
Thomas Ertl
Gregory M. Nielson

Editors

Scientific Visualization: The Visual Extraction of Knowledge from Data

With 228 Figures

Georges-Pierre Bonneau
Universite Grenoble I
Lab. LMC-IMAG
BP 53, 38041 Grenoble CX 9
France
E-mail: georges-pierre.bonneau@imag.fr

Gregory M. Nielson
Department of Computer Science and Engineering
Ira A. Fulton School of Engineering
Arizona State University
Tempe, AZ 85287-8809
USA
E-mail: nielson@asu.edu

Thomas Ertl
University of Stuttgart
Visualization and Interactive Systems
Institute (VIS)
Universitätstraße 38
70569 Stuttgart
Germany
E-mail: thomas.ertl@vis.uni-stuttgart.de

Library of Congress Control Number: 2005932239

Mathematics Subject Classification: 68-XX, 68Uxx, 68U05, 65-XX, 65Dxx, 65D18
ISBN-10 3-540-26066-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-26066-0 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable for prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media
springeronline.com
© Springer-Verlag Berlin Heidelberg 2006
Printed in The Netherlands

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typesetting: by the authors and TechBooks using a Springer L^AT_EX macro package
Cover design: *design & production* GmbH, Heidelberg

Printed on acid-free paper SPIN: 11430032 46/TechBooks 5 4 3 2 1 0

Preface

Scientific Visualization is concerned with techniques that allow scientists and engineers to extract knowledge from the results of simulations and computations. Advances in scientific computation are allowing mathematical models and simulations to become increasingly complex and detailed. This results in a closer approximation to reality thus enhancing the possibility of acquiring new knowledge and understanding. Tremendously large collections of numerical values, which contain a great deal of information, are being produced and collected. The problem is to convey all of this information to the scientist so that effective use can be made of the human creative and analytic capabilities. This requires a method of communication with a high bandwidth and an effective interface. Computer generated images and human vision mediated by the principles of perceptual psychology are the means used in scientific visualization to achieve this communication. The foundation material for the techniques of Scientific Visualization are derived from many areas including, for example, computer graphics, image processing, computer vision, perceptual psychology, applied mathematics, computer aided design, signal processing and numerical analysis.

This book is based on selected lectures given by leading experts in Scientific Visualization during a workshop held at Schloss Dagstuhl, Germany. Topics include user issues in visualization, large data visualization, unstructured mesh processing for visualization, volumetric visualization, flow visualization, medical visualization and visualization systems. The methods of visualizing data developed by Scientific Visualization researchers presented in this book are having broad impact on the way other scientists, engineers and practitioners are processing and understanding their data from sensors, simulations and mathematics models.

We would like to express our warmest thanks to the authors and referees for their hard work. We would also like to thank Fabien Vivodtzev for his help in administering the reviewing and editing process.

Grenoble,
January 2005

*Georges-Pierre Bonneau
Thomas Ertl
Gregory M. Nielson*

Contents

Part I Meshes for Visualization

Adaptive Contouring with Quadratic Tetrahedra

<i>Benjamin F. Gregorski, David F. Wiley, Henry R. Childs, Bernd Hamann, Kenneth I. Joy</i>	3
---	---

On the Convexification of Unstructured Grids from a Scientific Visualization Perspective

<i>João L.D. Comba, Joseph S.B. Mitchell, Cláudio T. Silva</i>	17
--	----

Brain Mapping Using Topology Graphs Obtained by Surface Segmentation

<i>Fabien Vivodtzev, Lars Linsen, Bernd Hamann, Kenneth I. Joy, Bruno A. Olshausen</i>	35
--	----

Computing and Displaying Intermolecular Negative Volume for Docking

<i>Chang Ha Lee, Amitabh Varshney</i>	49
---	----

Optimized Bounding Polyhedra for GPU-Based Distance Transform

<i>Ronald Peikert, Christian Sigg</i>	65
---	----

Generating, Representing and Querying Level-Of-Detail Tetrahedral Meshes

<i>Leila De Floriani, Emanuele Danovaro</i>	79
---	----

Split 'N Fit: Adaptive Fitting of Scattered Point Cloud Data

<i>Gregory M. Nielson, Hans Hagen, Kun Lee, Adam Huang</i>	97
--	----

Part II Volume Visualization and Medical Visualization

Ray Casting with Programmable Graphics Hardware

Manfred Weiler, Martin Kraus, Stefan Guthe, Thomas Ertl, Wolfgang Straßer 115

Volume Exploration Made Easy Using Feature Maps

Klaus Mueller, Sarang Lakare, Arie Kaufman 131

Fantastic Voyage of the Virtual Colon

Arie Kaufman, Sarang Lakare 149

Volume Denoising for Visualizing Refraction

David Rodgman, Min Chen 163

Emphasizing Isosurface Embeddings

in Direct Volume Rendering

Shigeo Takahashi, Yuriko Takeshima, Issei Fujishiro, Gregory M. Nielson 185

Diagnostic Relevant Visualization

of Vascular Structures

Armin Kanitsar, Dominik Fleischmann, Rainer Wegenkittl, Meister Eduard Gröller 207

Part III Vector Field Visualization

Clifford Convolution and Pattern Matching

on Irregular Grids

Julia Ebliing, Gerik Scheuermann 231

Fast and Robust Extraction

of Separation Line Features

Xavier Tricoche, Christoph Garth, Gerik Scheuermann 249

Fast Vortex Axis Calculation Using Vortex Features

and Identification Algorithms

Markus Rütten, Hans-Georg Pagendarm 265

Topological Features in Vector Fields

Thomas Wischgoll, Joerg Meyer 287

Part IV Visualization Systems

Generalizing Focus+Context Visualization

Helwig Hauser 305

Rule-based Morphing Techniques for Interactive Clothing Catalogs <i>Achim Ebert, Ingo Ginkel, Hans Hagen</i>	329
A Practical System for Constrained Interactive Walkthroughs of Arbitrarily Complex Scenes <i>Lining Yang, Roger Crawfis</i>	345
Component Based Visualisation of DIET Applications <i>Rolf Hendrik van Lengen, Paul Marrow, Thies Bähr, Hans Hagen, Erwin Bonsma, Cefn Hoile</i>	367
Facilitating the Visual Analysis of Large-Scale Unsteady Computational Fluid Dynamics Simulations <i>Kelly Gaither, David S. Ebert</i>	385
Evolving Dataflow Visualization Environments to Grid Computing <i>Ken Brodlie, Sally Mason, Martin Thompson, Mark Walkley and Jason Wood</i>	395
Earthquake Visualization Using Large-scale Ground Motion and Structural Response Simulations <i>Joerg Meyer, Thomas Wischgoll</i>	409
Author Index	433

Part I

Meshes for Visualization

Adaptive Contouring with Quadratic Tetrahedra

Benjamin F. Gregorski¹, David F. Wiley¹, Henry R. Childs², Bernd Hamann¹, and Kenneth I. Joy¹

¹ Institute For Data Analysis and Visualization

University of California, Davis

`bfgregorski,dfwiley,bhamann,kijoy@ucdavis.edu`

² B Division Lawrence Livermore National Laboratory

`childs3@llnl.gov`

Summary. We present an algorithm for adaptively extracting and rendering isosurfaces of scalar-valued volume datasets represented by quadratic tetrahedra. Hierarchical tetrahedral meshes created by longest-edge bisection are used to construct a multiresolution C^0 -continuous representation using quadratic basis functions. A new algorithm allows us to contour higher-order volume elements efficiently.

1 Introduction

Isosurface extraction is a fundamental algorithm for visualizing volume datasets. Most research concerning isosurface extraction has focused on improving the performance and quality of the extracted isosurface. Hierarchical data structures, such as those presented in [2, 10, 22], can quickly determine which regions of the dataset contain the isosurface, minimizing the number of cells examined. These algorithms extract the isosurface from the highest resolution mesh. Adaptive refinement algorithms [4, 5, 7] progressively extract isosurfaces from lower resolution volumes, and control the quality of the isosurface using user specified parameters.

An isosurface is typically represented as a piecewise linear surface. For datasets that contain smooth, steep ramps, a large number of linear elements is often needed to accurately reconstruct the dataset unless extra information is known about the data. Recent research has addressed these problems with linear elements by using *higher-order* methods that incorporate additional information into the isosurface extraction algorithm. In [9], an extended marching cubes algorithm, based on gradient information, is used to extract contours from distance volumes that contain sharp features. Cells that contain features are contoured by inserting new vertices that minimize an error function. Higher-order distance fields are also described in [12]. This approach constructs a distance field representation where each voxel has a complete description of all surface regions that contribute to the local distance field. Using this representation, sharp features and discontinuities are accurately represented as their exact locations are recorded. Ju et al. [11] describe a dual contouring scheme for

adaptively refined volumes represented with Hermite data that does not have to test for sharp features. Their algorithm uses a new representation for quadric error functions to quickly and accurately position vertices within cells according to gradient information. Wiley et al. [19, 20] use quadratic elements for hierarchical approximation and visualization of image and volume data. They show that quadratic elements, instead of linear elements, can be effectively used to approximate two and three dimensional functions.

Higher-order elements, such as quadratic tetrahedra and quadratic hexahedra, are used in finite element solutions to reduce the number of elements and improve the quality of numerical solutions [18]. Since few algorithms directly visualize higher-order elements, they are usually tessellated by several linear elements. Conventional visualization methods, such as contouring, ray casting, and slicing, are applied to these linear elements. Using linear elements increases the number of primitives, i.e. triangles or voxels, that need to be processed. Methods for visualizing higher-order elements directly are desirable.

We use a tetrahedral mesh, constructed by longest-edge bisection as presented in [5], to create a multiresolution data representation. The linear tetrahedral elements used in previous methods are replaced with quadratic tetrahedra. The resulting mesh defines a C^0 -continuous, piecewise quadratic approximation of the original dataset. This quadratic representation is computed in a preprocessing step by approximating the data values along each edge of a tetrahedron with a quadratic function that interpolates the endpoint values. A quadratic tetrahedron is constructed from the curves along its six edges. At runtime, the hierarchical approximation is traversed to approximate the original dataset to within a user defined error tolerance. The isosurface is extracted directly from the quadratic tetrahedra.

The remainder of our paper is structured as follows: Section 2 reviews related work. Section 3 describes what quadratic tetrahedra are, and Sect. 4 describes how they are used to build a multiresolution representation of a volume dataset. Sections 5 describes how a quadratic tet is contoured. Our results are shown in Sect. 6.

2 Previous Work

Tetrahedral meshes constructed by longest-edge bisection have been used in many visualization applications due to their simple, elegant, and crack-preventing adaptive refinement properties. In [5], fine-to-coarse and coarse-to-fine mesh refinement is used to adaptively extract isosurfaces from volume datasets. Gerstner and Pajarola [7] present an algorithm for preserving the topology of an extracted isosurface using a coarse-to-fine refinement scheme assuming linear interpolation within a tetrahedron. Their algorithm can be used to extract topology-preserving isosurfaces or to perform controlled topology simplification. In [6], Gerstner shows how to render multiple transparent isosurfaces using these tetrahedral meshes, and in [8], Gerstner and Rumpf parallelize the isosurface extraction by assigning portions of the binary tree created by the tetrahedral refinement to different processors. Roxborough and Nielson [16] describe a method for adaptively modeling 3D ultrasound data. They

create a model of the volume that conforms to the local complexity of the underlying data. A least-squares fitting algorithm is used to construct a best piecewise linear approximation of the data.

Contouring quadratic functions defined over triangular domains is discussed in [1, 14, 17]. Worsey and Farin [14] use Bernstein-Bézier polynomials which provide a higher degree of numerical stability compared to the monomial basis used by Marlow and Powell [17]. Bloomquist [1] provides a foundation for finding contours in quadratic elements.

In [19] and [20], quadratic functions are used for hierarchical approximation over triangular and tetrahedral domains. The approximation scheme uses the normal-equations approach described in [3] and computes the best least-squares approximation. A dataset is approximated with an initial set of quadratic triangles or tetrahedra. The initial mesh is repeatedly subdivided in regions of high error to improve the approximation. The quadratic elements are visualized by subdividing them into linear elements.

Our technique for constructing a quadratic approximation differs from [19] and [20] as we use univariate approximations along a tetrahedron's edges to define the coefficients for an approximating tetrahedron. We extract an isosurface directly from a quadratic tetrahedron by creating a set of rational-quadratic patches that approximates the isosurface. The technique we use for isosurfacing quadratic tetrahedra is described in [21].

3 Quadratic Tetrahedra

A linear tetrahedron $T_L(u, v, w)$ having four coefficients f_i at its vertices \mathbf{V}_i is defined as

$$\begin{aligned} T_L(u, v, w) = & f_0 u + f_1 v + f_2 w \\ & + f_3 (1 - u - v - w). \end{aligned} \quad (1)$$

The quadratic tetrahedron $T_Q(u, v, w)$ (called T_Q) that we use as our decomposition element has linearly defined edges such that its domain is completely described by four vertices (the same as a conventional linear tetrahedron). The function over T_Q is defined by a quadratic polynomial. We call this element a *linear-edge quadratic tetrahedron* or *quadratic tetrahedron*. The quadratic polynomial is defined, in Bernstein-Bézier form, by ten coefficients c_m , $0 \leq m \leq 9$, as

$$T_Q(u, v, w) = \sum_{k=0}^1 \sum_{j=0}^{2-k} \sum_{i=0}^{2-k-j} c_{ijk} B_{ijk}^2(u, v, w) \quad (2)$$

The Bernstein-Bézier basis functions $B_{ijk}^2(u, v, w)$ are

$$\begin{aligned} B_{ijk}^2 = & \frac{2!}{(2-i-j-k)! i! j! k!} \\ & (1 - u - v - w)^{2-i-j-k} u^i v^j w^k \end{aligned} \quad (3)$$

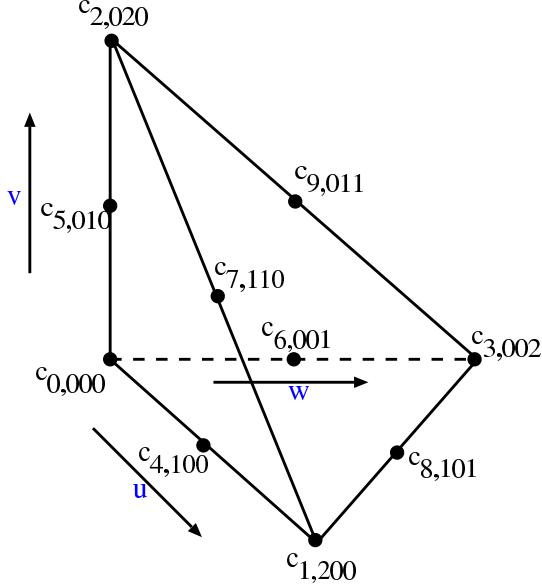


Fig. 1. Indexing of vertices and parameter space configuration for the ten control points of a quadratic tetrahedron

The indexing of the coefficients is shown in Fig. 1.

4 Constructing a Quadratic Representation

A quadratic tetrahedron T_Q is constructed from a linear tetrahedron T_L with corner vertices V_0, V_1, V_2 , and V_3 , by fitting quadratic functions along the six edges of T_L . Since a quadratic function requires three coefficients, there is an additional value associated with each edge.

4.1 Fitting Quadratic Curves

Given a set of function values $f_0, f_1 \dots f_n$ at positions $x_0, x_1 \dots x_n$, we create a quadratic function that passes through the end points and approximates the remaining data values.

The quadratic function $C(t)$ we use to approximate the function values along an edge is defined as

$$C(t) = \sum_{i=0}^2 c_i B_i^2(t) \quad (4)$$

The quadratic Bernstein polynomial $B_i^2(t)$ is defined as

$$B_i^2(t) = \frac{2!}{(2-i)!i!} (1-u)^{2-i} u^i \quad (5)$$

First we parameterize the data by assigning parameter values $t_0, t_1 \dots t_n$ in the interval $[0, 1]$ to the positions $x_0, x_1 \dots x_n$. Parameter values are defined with a chord-length parameterization as

$$t_i = \frac{x_i - x_0}{x_n - x_0} \quad (6)$$

Next, we solve a least-squares approximation problem to determine the coefficients c_i of $C(t)$. The resulting overdetermined system of linear equations is

$$\begin{bmatrix} (1-t_0)^2 & 2(1-t_0)t_0 & t_0^2 \\ (1-t_1)^2 & 2(1-t_1)t_1 & t_1^2 \\ \vdots & \vdots & \vdots \\ (1-t_n)^2 & 2(1-t_n)t_n & t_n^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix}. \quad (7)$$

Constraining $C(t)$, so that it interpolates the endpoint values, i.e. $C(0) = f_0$ and $C(1) = f_n$, leads to the system

$$\begin{bmatrix} 2(1-t_1)t_1 \\ 2(1-t_2)t_2 \\ \vdots \\ 2(1-t_{n-1})t_{n-1} \end{bmatrix} [c_1] = \begin{bmatrix} f_1 - f_0(1-t_1)^2 - f_n t_1^2 \\ f_2 - f_0(1-t_2)^2 - f_n t_2^2 \\ \vdots \\ f_{n-1} - f_0(1-t_{n-1})^2 - f_n t_{n-1}^2 \end{bmatrix} \quad (8)$$

for the one degree of freedom c_1 .

4.2 Approximating a Dataset

A quadratic approximation of a dataset is created by approximating the data values along each edge in the tetrahedral mesh with a quadratic function as described in Sect. 4.1. Each linear tetrahedron becomes a quadratic tetrahedron. The resulting approximation is C^1 -continuous within a tetrahedron and C^0 -continuous on shared faces and edges. The approximation error e_a for a tetrahedron T is the maximum difference between the quadratic approximation over T and all original data values associated with points inside and on T 's boundary.

In tetrahedral meshes created by longest-edge bisection, each edge E in the mesh, except for the edges at the finest level of the mesh, is the split edge of a *diamond* D , see [5], and is associated with a split vertex SV . The computed coefficient c_1 for the edge E is stored with the *split vertex* SV . The edges used for computing the quadratic representation can be enumerated by recursively traversing the tetrahedral mesh and examining the refinement edges. This process is illustrated for the 2D case in Fig. 2. Since quadratic tetrahedra have three coefficients along each edge, the leaf level of a

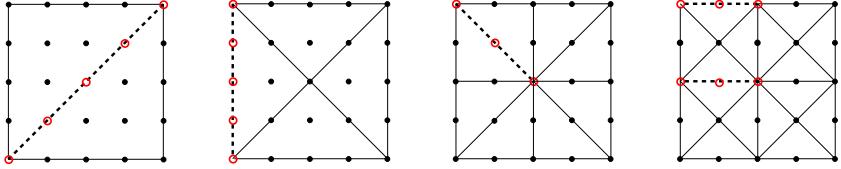


Fig. 2. Enumeration of edges for constructing quadratic approximation using longest-edge bisection. Circles indicate original function values used to compute approximating quadratic functions along each edge

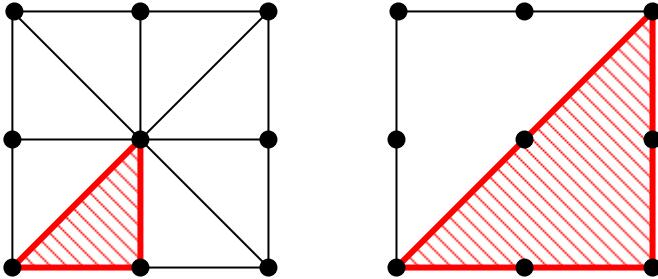


Fig. 3. Top: leaf tetrahedra for a mesh with linear tetrahedra. Bottom: leaf tetrahedra for a mesh with quadratic tetrahedra

mesh with quadratic tetrahedra is one level higher in the mesh than the leaf level for linear tetrahedra, see Fig. 3.

In summary, we construct a quadratic approximation of a volume data set as follows:

1. For each edge of the mesh hierarchy, approximate the data values along the edge with a quadratic function that passes through the endpoints.
2. For each tetrahedron in the hierarchy, construct a quadratic tetrahedron from the six quadratic functions along its edges.
3. Compute the approximation error e_a for each tetrahedron.

5 Contouring Quadratic Tetrahedra

We use the method described in [21] to extract and represent isosurfaces of quadratic tetrahedra. We summarize the main aspects of the method here. First, the intersection of the isosurface is computed with each face of the quadratic tetrahedron forming *face-intersection curves*. Next, the face-intersection curves are connected end-to-end to form groups of curves that bound various portions of the isosurface inside the tetrahedron, see Fig. 4. Finally, the face-intersection groups are “triangulated” with rational-quadratic patches to represent the various portions of the isosurface inside the quadratic tetrahedron.

Since intersections are conic sects. [14], the intersections between the isosurface and the faces produce rational-quadratic curves. We define a rational-quadratic curve

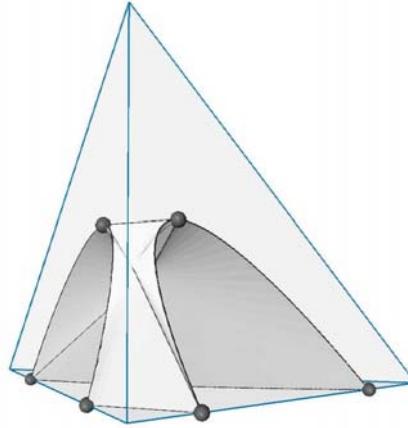


Fig. 4. Isosurface bounded by six face-intersection curves. Dark dots indicate endpoints of the curves

$Q(t)$ with control points p_i and weights w_i , $0 \leq i \leq 2$, as

$$Q(t) = \frac{\sum_{i=0}^2 w_i \mathbf{p}_i B_i^2(t)}{\sum_{i=0}^2 w_i B_i^2(t)} \quad (9)$$

By connecting the endpoints of the N face-intersection curves $Q_j(t)$, $0 \leq j \leq N - 1$, we construct M rational-quadratic patches $Q_k(u, v)$, $0 \leq k \leq M - 1$, to represent the surface. We define a rational-quadratic patch $Q(u, v)$ with six control points p_{ij} and six weights w_{ij} as

$$Q(u, v) = \frac{\sum_{j=0}^2 \sum_{i=0}^{2-j} w_{ij} p_{ij} B_{ij}^2(u, v)}{\sum_{j=0}^2 \sum_{i=0}^{2-j} w_{ij} B_{ij}^2(u, v)} \quad (10)$$

A patch $Q(u, v)$ is constructed from two or three face-intersection curves by using the control points of the curves as the control points for $Q(u, v)$. Four or more face-intersection curves require the use of a “divide-and-conquer” method that results in multiple patches, see [21].

6 Results

We have applied our algorithm to various volume datasets. The datasets are all byte-valued, and the quadratic coefficients along the edges are stored as signed shorts. In all examples, the mesh is refined to approximate the original dataset, according to the quadratic tetrahedra approximation, within a user specified error bound e_u . The resulting mesh consists of a set of quadratic tetrahedra which approximates the dataset within e_u . The isosurface, a set of quadratic bezier patches, is extracted from

Table 1. Error values, number of quadratic tetrahedra used for approximation, and number of quadratic patches extracted

Dataset	Size	Error	Tets	Patches
Buckyball	256^3	2.0	8560	4609
Buckyball	256^3	1.3	23604	10922
Buckyball	256^3	0.7	86690	32662
H-Atom	128^3	1.23	8172	3644
H-Atom	128^3	0.57	20767	7397

this mesh. Table 1 summarizes the results. It shows the error value, the number of quadratic tetrahedra needed to approximate the dataset to within the specified error tolerance, and the number of quadratic patches extracted from the mesh.

As discussed in Sect. 4.2, the error value indicates the maximum difference between the quadratic representation and the actual function values at the data points. On the boundaries, our quadratic representation is C^0 continuous with respect to the function value and discontinuous with respect to the gradient; thus the gradients used for shading are discontinuous at patch boundaries. This fact leads to the creases seen in the contours extracted from the quadratic elements. The patches which define the contour are tessellated and rendered as triangle face lists. A feature of the quadratic representation is the ability to vary both the patch tessellation factor and the resolution of the underlying tetrahedral grid. This gives an extra degree of freedom with which to balance isosurface quality and rendering speed.

The storage requirements of the linear and quadratic representations are summarized in Table 2. Storage costs of linear and quadratic representations with and without precomputed gradients are shown. When gradients are precomputed for shading, a gradient must be computed at each data location regardless of representation. When rendering linear surfaces, gradients are often precomputed and quantized to avoid the cost of computing them at runtime. For quadratic patches, gradients do not need to be precomputed because they can be computed from the analytical definition of the surface. However, if gradients are precomputed, they can be used directly.

Table 2. Storage requirements(bytes) for linear and quadratic representations for a dataset with 2^{3n} points. The linear representation consists of $L = 2^{3n}$ diamonds, and the quadratic representation consists of $\frac{L}{8} = 2^{3(n-1)}$ diamonds. R is the number of bytes used to store the error, min, and max values of a diamond, G is the number of bytes used to store a gradient, and C is the number of bytes used to store a quadratic coefficient

Type	Data	Gradients	Bézier Coeffs	Error/Min/Max	Total
Linear	L	0	0	RL	$L(1+R)$
Linear	L	GL	0	RL	$L(1+G+R)$
Quadratic	$\frac{L}{8}$	0	CL	$R\frac{L}{8}$	$L\frac{1+8C+R}{8}$
Quadratic	$\frac{L}{8}$	GL	CL	$R\frac{L}{8}$	$L\frac{1+8G+8C+R}{8}$

The difference between the leaf levels of linear and quadratic representations, as described in Sect. 4.2, implies that there are eight times as many diamonds in the linear representation than there are in the quadratic representation. We represent the quadratic coefficients with two bytes. The quadratic coefficients for the Buckyball dataset shown in Figs. 5 and 6 lie in the range $[-88,390]$. The representation of error, min, and max values is the same for both representations. They can be stored as raw values or compressed to reduce storage costs. The quadratic representation essentially removes three levels from the binary tree of the tetrahedral mesh reducing the number of error, min, and max values by a factor of eight compared with the linear representation.

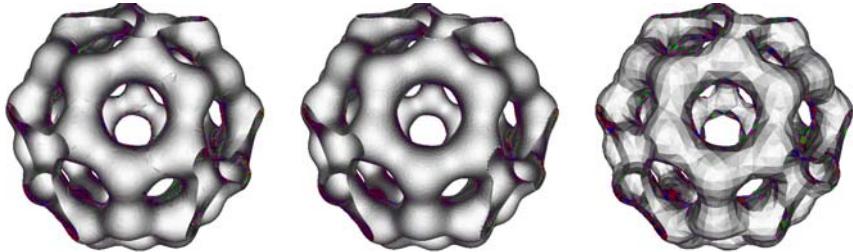


Fig. 5. *Left:* Isosurface of quadratic patches extracted using quadratic tetrahedra. *Middle:* Full resolution isosurface (1798644 triangles). *Right:* Isosurface of triangles extracted from the same mesh used to show the resolution of the tetrahedral grid. Isovalue = 184.4, Error = 0.7

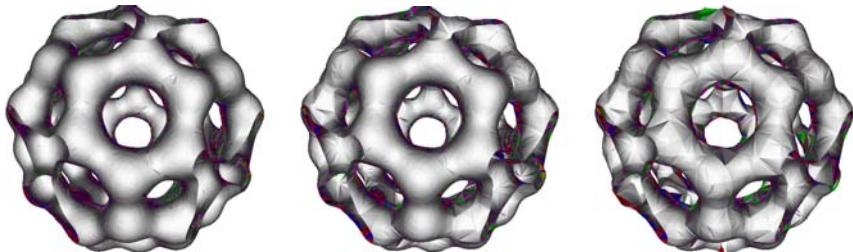


Fig. 6. Isosurfaces extracted using quadratic tetrahedra at different error bounds. *Top to Bottom:* Error = 0.7, 1.2, and 2.0. Number of Quadratic Patches = 32662, 10922, 4609

The first dataset is a Buckyball dataset made from Gaussian functions. Figure 5 compares contours extracted using quadratic and linear tetrahedra against the full resolution surface. The isosurfaces are extracted from the same mesh which consists of 86690 tets; it yields 32662 quadratic patches. Figure 6 shows three isosurfaces of the Buckyball from the same viewpoint at different resolutions. The images are created by refining the mesh using a view-dependent error bound. Thus, the middle image, for an error of 1.3 has more refinement in the region closer to the viewpoint and less refinement in the regions further from the viewpoint. For the Buckyball dataset, the

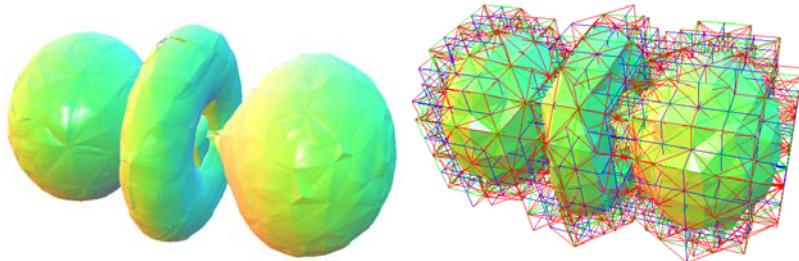


Fig. 7. Isosurface through the Hydrogen Atom dataset. The isosurface rendered using quadratic patches, and the tetrahedra from which the contours were extracted. Isovalue = 9.4, Error = 1.23, Number of patches = 3644

patches are tessellated with 28 vertices and 36 triangles. These images show how the quadratic representation can be effectively used to adaptively approximate a dataset. The second dataset is the Hydrogen Atom dataset obtained from www.volvis.org. The dataset is the result of a simulation of the spatial probability distribution of the electron in a hydrogen atom, residing in a strong magnetic field. Figure 7 shows the surfaces generated from the quadratic tetrahedra and the coarse tetrahedral mesh from which the contours are extracted.

Figure 8 is a closeup view of the dataset's interior. It shows a thin “hourglass-like” feature emanating from the probability lobe visible on the right. For the Hydrogen Atom dataset, the patches are tessellated with 15 vertices and 16 triangles. The isosurface extracted from the quadratic representation is compared with the linear isosurface to show how the quadratic representation accurately approximates the silhouette edges with a small number of elements.



Fig. 8. Closeup view of hydrogen atom dataset rendered with quadratic patches(*left*). As in Fig. 5, the isosurface extracted using linear elements(*right*) shows the resolution of the underlying tetrahedral grid. Isovalue = 9.4, Error = 0.566

7 Conclusions

We have presented an algorithm for approximating and contouring datasets with quadratic tetrahedra. Our algorithm uses hierarchically defined tetrahedral meshes to construct a multiresolution representation. This representation is used to approximate the dataset within a user specified error tolerance. Quadratic tetrahedra are created from this multiresolution mesh by constructing approximating quadratic functions along edges and using these functions to form quadratic tetrahedra. We have improved previous methods for visualizing quadratic elements by showing how to directly contour them without splitting them into a large number of linear elements. Comparisons of the storage costs of quadratic and linear representations show that quadratic elements can represent datasets with a smaller number of elements and without a large storage overhead.

Future work is planned in these areas:

- **Improving the quality and speed of the contour extraction and comparing the quality of the surfaces to those generated from linear tetrahedra.** Currently, our algorithm generates some small thin surfaces that are undesirable for visualization. Additionally we are working on arbitrary slicing and volume rendering of quadratic elements.
- **Improving the computation of the quadratic representation.** Our current algorithm, while computationally efficient, fails to capture the behavior of the dataset within a tetrahedron, and yields discontinuous gradients at the boundaries. It is desirable to have an approximation that is overall C^1 -continuous or C^1 -continuous in most regions and C^0 in regions where discontinuities exist in the data. A C^1 -continuous approximation might improve the overall approximation quality, allowing us to use fewer elements, and would improve the visual quality of the extracted contours.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48. This work was also supported by the National Science Foundation under contracts ACI 9624034 (CAREER Award), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, through the National Partnership for Advanced Computational Infrastructure (NPACI) and a large Information Technology Research (ITR) grant; the National Institutes of Health under contract P20 MH60975-06A2, funded by the National Institute of Mental Health and the National Science Foundation; and the Lawrence Livermore National Laboratory under ASCI ASAP Level-2 memorandum agreement B347878, and agreements B503159 and B523294; We thank the members of the Visualization and Graphics Research Group at the Institute for Data Analysis and Visualization (IDAV) at the University of California, Davis.

References

1. B.K. Bloomquist, Contouring Trivariate Surfaces, Masters Thesis, Arizona State University, Computer Science Department, 1990
2. P. Cignoni and P. Marino and C. Montani and E. Puppo and R. Scopigno Speeding Up Isosurface Extraction Using Interval Trees *IEEE Transactions on Visualization and Computer Graphics* 1991, 158–170
3. P. J. Davis Interpolation and Approximation Dover Publications, Inc., New York, NY, 2, 3
4. Klaus Engel and Rudiger Westermann and Thomas Ertl Isosurface Extraction Techniques For Web-Based Volume Visualization *Proceedings of IEEE Visualization 1999*, 139–146
5. Benjamin Gregorski, Mark Duchaineau, Peter Lindstrom, Valerio Pascucci, and Kenneth I. Joy Interactive View-Dependent Extraction of Large Isosurfaces *Proceedings of IEEE Visualization 2002*, 475–482
6. T. Gerstner Fast Multiresolution Extraction Of Multiple Transparent Isosurfaces, *Data Visualization 2001 Proceedings of VisSim 2001*
7. Thomas Gerstner and Renato Pajarola, Topology Preserving And Controlled Topology Simplifying Multiresolution Isosurface Extraction, *Proceedings of IEEE Visualization 2000*, 259–266
8. T. Gerstner and M. Rumpf, Multiresolution Parallel Isosurface Extraction Based On Tetrahedral Bisection, *Volume Graphics 2000*, 267–278
9. Leif P. Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel Feature-Sensitive Surface Extraction From Volume Data *SIGGRAPH 2001 Conference Proceedings*, 57–66
10. Y. Livnat and C. Hansen View Dependent Isosurface Extraction *Proceedings of IEEE Visualization 1998*, 172–180
11. Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren Dual contouring of hermite data *SIGGRAPH 2002 Conference Proceedings*, 339–346
12. Jian Huang, Yan Li, Roger Crawfis, Shao-Chiung Lu, and Shuh-Yuan Liou A Complete Distance Field Representation *Proceedings of Visualization 2001*, 247–254
13. Gerald Farin, Curves and Surfaces for CAGD, Fifth edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, 2001
14. A.J. Worsey and G. Farin, Contouring a bivariate quadratic polynomial over a triangle, *Computer Aided Geometric Design* 7 (1–4), 337–352, 1990
15. B. Hamann, I.J. Trott, and G. Farin *On Approximating Contours of the Piecewise Trilinear Interpolant using triangular rational-quadratic Bézier patches*, *IEEE Transactions on Visualization and Computer Graphics*, 3(3), 315–337 1997
16. Tom Roxborough and Gregory M. Nielson, Tetrahedron Based, Least Squares, Progressive Volume Models With Application To Freehand Ultrasound Data”, In *Proceedings of IEEE Visualization 2000*, 93–100
17. S. Marlow and M.J.D. Powell, A Fortran subroutine for plotting the part of a conic that is inside a given triangle, Report no. R 8336, Atomic Energy Research Establishment, Harwell, United Kingdom, 1976
18. R. Van Uitert, D. Weinstein, C.R. Johnson, and L. Zhukov Finite Element EEG and MEG Simulations for Realistic Head Models: Quadratic vs. Linear Approximations Special Issue of the Journal of Biomedizinische Technik, Vol. 46, 32–34, 2001.
19. David F. Wiley, H.R. Childs, Bernd Hamann, Kenneth I. Joy, and Nelson Max, Using Quadratic Simplicial Elements for Hierarchical Approximation and Visualization, *Visualization and Data Analysis 2002, Proceedings*, SPIE - The International Society for Optical Engineering, 32–43, 2002

20. David F. Wiley, H.R. Childs, Bernd Hamann, Kenneth I. Joy, and Nelson Max, Best Quadratic Spline Approximation for Hierarchical Visualization, *Data Visualization 2002, Proceedings of VisSym 2002*
21. D. F. Wiley, H. R. Childs, B. F. Gregorski, B. Hamann, and K. I. Joy Contouring Curved Quadratic Elements *Data Visualization 2003, Proceedings of VisSym 2003*
22. Jane Wilhelms and Allen Van Gelder Octrees for Faster Isosurface Generation *ACM Transaction in Graphics*, 201–227, July 1992

On the Convexification of Unstructured Grids from a Scientific Visualization Perspective

João L.D. Comba¹, Joseph S.B. Mitchell², and Cláudio T. Silva³

¹ Federal University of Rio Grande do Sul (UFRGS)
comba@inf.ufrgs.br

² Stony Brook University
jsbm@ams.sunysb.edu

³ University of Utah
csilva@cs.utah.edu

Summary. Unstructured grids are extensively used in modern computational solvers and, thus, play an important role in scientific visualization. They come in many different types. One of the most general types are non-convex meshes, which may contain voids and cavities. The lack of convexity presents a problem for several algorithms, often causing performance issues.

One way around the complexity of non-convex methods is to convert them into convex ones for visualization purposes. This idea was originally proposed by Peter Williams in his seminal paper on visibility ordering. He proposed to fill the volume between the convex hull of the original mesh, and its boundary with “imaginary” cells. In his paper, he sketches two algorithms for potentially performing this operation, but stops short of implementing them.

This paper discusses the convexification problem and surveys the relevant literature. We hope it is useful for researchers interested in the visualization of unstructured grids.

1 Introduction

The most common input data type in Volume Visualization is a *regular (Cartesian) grid of voxels*. Given a general scalar field in \Re^3 , one can use a regular grid of voxels to represent the field by regularly sampling the function at grid points $(\lambda i, \lambda j, \lambda k)$, for integers i, j, k , and some scale factor $\lambda \in \Re$, thereby creating a regular grid of voxels. However, a serious drawback of this approach arises when the scalar field is *disparate*, having nonuniform resolution with some large regions of space having very little field variation, and other very small regions of space having very high field variation. In such cases, which often arise in computational fluid dynamics and partial differential equation solvers, the use of a regular grid is infeasible since the voxel size must be small enough to model the smallest “features” in the field. Instead, *irregular grids* (or *meshes*), having cells that are not necessarily uniform cubes, have been proposed as an effective means of representing disparate field data.

Irregular-grid data comes in several different formats [37]. One very common format has been *curvilinear grids*, which are *structured* grids in computational space that have been “warped” in physical space, while preserving the same topological structure (connectivity) of a regular grid. However, with the introduction of new methods for generating higher quality adaptive meshes, it is becoming increasingly common to consider more general *unstructured* (non-curvilinear) irregular grids, in which there is no implicit connectivity information. Furthermore, in some applications *disconnected* grids arise.

Preliminaries

We begin with some basic definitions. A *polyhedron* is a closed subset of \Re^3 whose boundary consists of a finite collection of convex polygons (*2-faces*, or *facets*) whose union is a connected 2-manifold. The *edges* (*1-faces*) and *vertices* (*0-faces*) of a polyhedron are simply the edges and vertices of the polygonal facets. A bounded convex polyhedron is called a *polytope*. A polytope having exactly four vertices (and four triangular facets) is called a *simplex* (*tetrahedron*). A finite set S of polyhedra forms a *mesh* (or an *unstructured grid*) if the intersection of any two polyhedra from S is either empty, a single common vertex, a single common edge, or a single common facet of the two polyhedra; such a set S is said to form a *cell complex*. The polyhedra of a mesh are referred to as the *cells* (or *3-faces*). We say that cell C is *adjacent* to cell C' if C and C' share a common facet. The adjacency relation is a binary relation on elements of S that defines an *adjacency graph*.

A facet that is incident on only one cell is called a *boundary facet*. A *boundary cell* is any cell having a boundary facet. The union of all boundary facets is the *boundary* of the mesh. If the boundary of a mesh S is also the boundary of the convex hull of S , then S is called a *convex* mesh; otherwise, it is called a *non-convex* mesh. If the cells are all simplices, then we say that the mesh is *simplicial*.

The input to our problem will be a given mesh S . We let c denote the number of connected components of S . If $c = 1$, the mesh is *connected*; otherwise, the mesh is *disconnected*. We let n denote the total number of edges of all polyhedral cells in the mesh. Then, there are $O(n)$ vertices, edges, facets, and cells.

We use a coordinate system in which the viewing direction is in the $-z$ direction, and the image plane is the (x, y) plane. We let ρ_u denote the ray from the viewpoint v through the point u .

We say that cells C and C' are *immediate neighbors* with respect to viewpoint v if there exists a ray ρ from v that intersects C and C' , and no other cell $C'' \in S$ has a nonempty intersection $C'' \cap \rho$ that appears in between the segments $C \cap \rho$ and $C' \cap \rho$ along ρ . Note that if C and C' are adjacent, then they are necessarily immediate neighbors with respect to very viewpoint v not in the plane of the shared facet. Further, in a convex mesh, the *only* pairs of cells that are immediate neighbors are those that are adjacent.

A *visibility ordering* (or *depth ordering*), $<_v$, of a mesh S from a given viewpoint, $v \in \Re^3$ is a total (linear) order on S such that if cell $C \in S$ visually obstructs cell $C' \in S$, partially or completely, then C' precedes C in the ordering: $C' <_v C$. A visibility

ordering is a linear extension of the binary *behind* relation, “ $<$ ”, in which cell C is *behind* cell C' (written $C < C'$) if and only if C and C' are immediate neighbors and C' at least partially obstructs C ; i.e., if and only if there exists a ray ρ from the viewpoint v such that $\rho \cap C \neq \emptyset$, $\rho \cap C' \neq \emptyset$, $\rho \cap C'$ appears in between v and $\rho \cap C$ along ρ , and no other cell C'' intersects ρ at a point between $\rho \cap C$ and $\rho \cap C'$. A visibility ordering can be obtained in linear time (by topological sorting) from the behind relation, $(S, <)$, provided that the directed graph on the set of nodes S defined by $(S, <)$ is acyclic. If the behind relation induces a directed cycle, then no visibility ordering exists. Certain types of meshes, (e.g., Delaunay triangulations [16]) are known to have a visibility ordering from any viewpoint, i.e., they do not have cycles, and thus can be called *acyclic meshes*.

Spatial Decompositions

There is a rich literature in the computational geometry community on spatial decompositions. See Nielson, Hagen and Müller [25] for an overview of their importance in the context of visualization applications.

Spatial decomposition is an essential tool in finite element analysis and geometric modeling. Applications require high-quality mesh generation, in which the goal is to triangulate domains with elements that are “nice” in some well-defined sense (e.g., triangulations having no large angle [3]). See the recent surveys of Bern and Eppstein [4], Bern and Plassmann [5], and Bern [2], and the book of Edelsbrunner [16], for a comprehensive overview of the literature.

A problem extensively studied in the early years of computational geometry was the polygon triangulation problem, in which the goal was to decompose a simple polygon, or a polygon with holes, into triangles. A milestone result in two-dimensional triangulations was the discovery by Chazelle [6] of a linear-time algorithm for triangulating a simple polygon. Optimization problems related to decompositions of polygons into convex pieces have been studied in many variations; see Chazelle and Dobkin [7] and the survey of Keil [21].

In three or more dimensions, decomposition of polyhedral domains into “triangles” (tetrahedra) is substantially more complex. Ruppert and Seidel [27] have shown that it is NP-complete to decide if a (non-convex) polyhedron can be tetrahedralized without the addition of Steiner points. Chazelle and Palios [10] show that a (non-convex) polyhedron having n vertices and r reflex edges can always be triangulated (with the addition of Steiner points) in time $O(nr + r^2 \log r)$ using $O(n + r^2)$ tetrahedra (which is worst-case optimal, since some polyhedra require $\Omega(n + r^2)$ tetrahedra in any triangulation).

A *regular triangulation* in dimension d is the vertical projection of the “lower” side of a convex polytope in one higher dimension. The most widely studied regular triangulation is the Delaunay triangulation of a point set, which is the projection of the downward-facing facets of the convex hull of the lifted images of the input points onto the paraboloid in one higher dimension. An alternative characterization of a Delaunay triangulation is that the (hyper)sphere determined by the vertices of each triangle (simplex) of a Delaunay triangulation is “site-free,” not containing input

points in its interior. See Edelsbrunner [15], as well as the book of Okabe, Boots, and Sugihara [26] and the recent survey articles of Fortune [17].

Chazelle et al. [8] have examined how selectively adding points to an input set in three dimensions results in the worst-case size of the Delaunay triangulation being provably subquadratic in the input size, even though the worst-case size of a Delaunay triangulation of n points in space is $\Theta(n^2)$.

The meshes we study here are decompositions of polyhedral domains and piecewise-linear complexes, in which the decomposition is required to respect the facets of the input. A *constrained Delaunay triangulation* is a variation of a Delaunay triangulation that is constrained to respect the input shape, while being, in some sense, “as Delaunay as possible.” Such decompositions have desirable properties, favoring more regular tetrahedra over “skinny” tetrahedra. This makes them particularly appealing for interpolation, visualization, and finite element methods. Two-dimensional constrained Delaunay triangulations have been studied by, e.g., Chew [11], De Floriani and Puppo [14], and Seidel [29]. More recently, three-dimensional constrained Delaunay triangulations have been studied for their use in mesh generation; see the surveys mentioned above [2, 4, 5, 16], as well as Weatherill and Hassan [39]. Shewchuk [30–34] has developed efficient methods for three-dimensional constrained Delaunay triangulations, including, most recently [34], provable techniques of inserting constraints and performing “flips” (local modifications to the mesh) to construct constrained Delaunay and regular triangulations incrementally.

Exploiting Mesh Properties

Meshes that conform to properties such as “convexity” and “acyclicity” are quite special, since they simplify the algorithms that work with them. Here are three instances of visualization algorithms that exploit different properties of meshes:

- A classic technique for hardware-based rendering of unstructured meshes couples the Shirley-Tuchman technique for rendering a single tetrahedron [35] with Williams’ MPVO cell-sorting algorithm [41]. For the case of acyclic convex meshes, this is a powerful combination that leads to a linear-time algorithm that is provably correct, i.e., one is guaranteed to get the right picture.¹ When the mesh is not convex or contains cycles, MPVO requires modifications that complicate the algorithm and its implementation and lead to slower rendering times [13, 22, 36].
- A recent hardware-based ray casting technique for unstructured grids has been proposed by Weiler et al [40]. This is essentially a hardware-based implementation of the algorithm of Garrity [19]. Strictly speaking, this technique only works for convex meshes. Due to the constraints of the hardware, instead of modifying the rendering algorithm, the authors employ a process of “convexification”, originally proposed by Williams [41], to handle general cells.

¹The rendering technique of Shirley and Tuchman [35] requires certain modifications as proposed in Stein et al [38].

- The complexities of the simplification of unstructured grids has led some researchers to employ a convexification approach. As shown in Kraus and Ertl [23], this greatly simplifies the simplification algorithm, since it becomes much simpler to handle the simplification of the boundary of the mesh. Otherwise, expensive global operations are necessary to guarantee that the simplified mesh does not suffer from self intersections.

The “convexification” concept as proposed by Williams [41] is the process of turning a non-convex mesh into a convex one. The basic idea is that this process can be performed by adding a set of non-overlapping cells that fill up any holes or non-convex regions up to the bounding box of the original mesh. Also, Williams proposes that all the additional cells be marked “imaginary”. This is exactly the concept that is used in the works of Weiler et al [40] and Kraus and Ertl [23]. In [23, 40], the non-convex meshes were *manually* modified to be convex by the careful addition of cells. This approach is not scalable to larger and more complex data.

In this paper, we discuss the general problem of convexification. We start by reviewing Williams’ work, and discuss a number of issues. Then, we talk about two techniques for achieving convexification: techniques based on constrained and conforming Delaunay tetrahedralization, and techniques based on the use of a binary space partition (BSP). Finally, we conclude the paper with some observations and open questions. One of the goals of this paper is to help researchers be able to choose among tools and options for convexification solutions.

2 Williams’ Convexification Framework

In his seminal paper [41] on techniques for computing visibility orderings for meshes, Williams discusses the problem of handling non-convex meshes (Sect. 9). (Also related is Sect. 8, which contains a discussion of cycles and the use of Delaunay triangulations.) After explaining some challenges of using his visibility sorting algorithm on non-convex meshes, Williams says:

“Therefore, an important area of research is to find ways to convert non-convex meshes into convex meshes, so that the regular MPVO algorithm can be used.”

Williams proposes two solution approaches to the problem; each relies on “treating the voids and cavities as ‘imaginary’ cells in the mesh.” Basically, he proposes that such non-convex regions could be either **triangulated** or **decomposed** into convex pieces, and their parts marked as imaginary cells for the purpose of rendering. Implementing this “simple idea” is actually not easy. In fact, after discussing this general approach, Williams talks about some of the challenges, and finishes the section with the following remark:

“The implementation of the preprocessing methods, described in this section, for converting a non-convex mesh into a convex mesh could take a very

significant amount of time; they are by no means trivial. The implementation of a 3D conformed Delaunay triangulation is still a research question at this time.”

In fact, Williams does not provide an implementation of any of the two proposed convexification algorithms. Instead, he developed a variant of MPVO that works on non-convex meshes at the expense of not being guaranteed to generate correct visibility orders.

The first convexification technique that Williams proposes is based on triangulating the data using a conforming Delaunay triangulation. The idea here is to keep adding more points to the dataset until the original triangulation becomes a Delaunay triangulation. This is discussed in more details in the next section.

The second technique Williams sketches is based on the idea of applying a decomposition algorithm to each of the non-convex polyhedra that constitute the set $CH(S) \setminus S$, which is the set difference between the convex hull of the mesh and the mesh itself. In general, $CH(S) \setminus S$ is a union of highly non-convex polyhedra of complex topology. Each connected component of $CH(S) \setminus S$ is a non-convex polyhedron that can be decomposed into convex polyhedra (e.g., tetrahedra) using, for example, the algorithm of Chazelle and Palios [10], which adds certain new vertices (Steiner points), whose number depends on the number of “reflex” edges of the polyhedron. In general, non-convex polyhedra require the addition of Steiner points in order to decompose them; in fact, it is NP-complete to decide if a polyhedron can be tetrahedralized without the addition of Steiner points [27].

2.1 Issues

Achieving Peter Williams’s vision of a simple convexification algorithm is much harder than it appears at first. The problem is peculiar since we start with an existing 3D mesh (likely to be a tetrahedralization) that contains not only vertices, edges, and triangles, but also volumetric cells, which need to be respected. Furthermore, the mesh is not guaranteed to respect global geometric criteria (e.g., of being Delaunay). Most techniques need to modify the original mesh in some way. The goal is to “disturb” it as little as possible, preserving most of its original properties.

In particular, several issues need to be considered:

Preserving Acyclicity. Even if the original mesh has no cycles, the convexification process can potentially cause the resulting convex mesh to contain cycles. Certain techniques, such as constructing a conforming Delaunay tetrahedralization, are guaranteed to generate a cycle-free mesh. Ideally, the convexification procedure will not create new cycles in the mesh.

Output Size. For the convexification technique to be useful the number of cells added by the algorithm needs to be kept as small as possible. Ideally, there is a provable bound on the number of cells as well as experimental evidence that for typical input meshes, the size of the output mesh is not much larger than the input mesh (i.e., the set of additional cells is small).

Computational and Memory Complexity. Other important factors are the processing time and the amount of memory used in the algorithm. In order to be practical on the meshes that arise in computational experiments (having on the order of several thousand to a few million cells), convexification algorithms must run in near-linear time, in practice.

Boundary and Interior Preservation. Ideally, the convexification procedure adds cells only “outside” of the original mesh. Furthermore, the newly created cells should exactly match the original boundary of the mesh. In general, this is not feasible without subdividing or modifying the original cells in some way (e.g., to break cycles, or to add extra geometry in order to respect the Delaunay empty-circumsphere condition). Some techniques will only need to modify the cells that are at or near the original boundary while others might need to perform more global modifications that go all the way “inside” the original mesh. One needs to be careful when making such modifications because of issues related to interpolating the original data values in the mesh. Otherwise, the visualization algorithm may generate incorrect pictures leading to wrong comprehension.

Robustness and Degeneracy Handling. It is very important for the convexification algorithms to handle real data. Large scientific datasets often use floating-point precision for specifying vertices, and are likely to have a number of degeneracies. For instance, these datasets are likely to have many vertices (sample points) that are coplanar, or that lie on a common cylinder or sphere, etc., since the underlying physical model may have such features.

3 Delaunay-Based Techniques

Delaunay triangulations and Delaunay tetrahedralizations (DT) are very well known and studied geometric entities (see, e.g., [16, Chap. 5]). A basic property that characterizes this geometric structure is the fact that a tetrahedron belongs to the DT of a point set if the circumsphere passing through the four vertices is empty, meaning no other point lies inside the circumsphere. Under some non-degeneracy conditions (no 5 points co-spherical), this property completely characterizes DTs and the DT is unique.

Part of the appeal of Delaunay tetrahedralizations (see Fig. 1(b)) is the relative ease of computing the tetrahedralizations. As a well-studied structure, often used in mesh generation, standard codes are readily available that compute the DT. The practical need of forcing certain faces to be part of the tetrahedralizations led to the development of two main approaches: *conforming* Delaunay tetrahedralizations and *constrained* Delaunay tetrahedralizations. Here, we only give a high-level discussion on the intuition behind these ideas; for details see, e.g., [32].

Given a set of faces $\{f_i\}$ (Fig. 1(a)) that need to be included in a DT, the idea behind *conforming* Delaunay tetrahedralizations (Fig. 1(c)) is to add points to the original input set in order that the DT of the new point set (consisting of the original points *plus* the newly added points) is such that each face f_i can be expressed

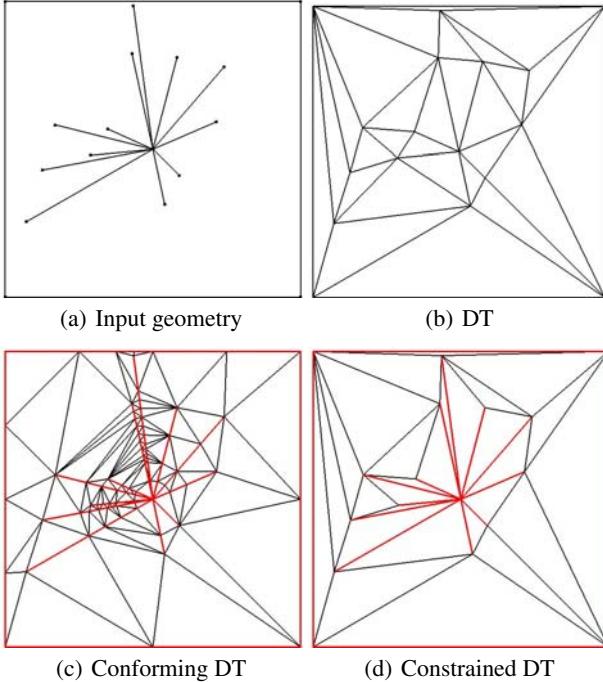


Fig. 1. Different triangulation techniques. (a) The input geometry; (b) the Delaunay triangulation; (c) a conforming Delaunay triangulation with input geometry marked in red – note how the input faces have been broken into multiple pieces; and, (d) the constrained Delaunay triangulation. Images after Shewchuk [31]

as the union of a collection of faces of the DT. The newly added points are often called *Steiner* points. A challenge in computing a conforming DT is minimizing the number of Steiner points and avoiding the generation of very small tetrahedra. While techniques for computing the traditional DT of point sites are well known, and reliable code exists, conforming DT algorithms are still in active development [12, 24]. The particular technique for adding Steiner points affects the termination of the algorithm, and also the number and quality of the added geometry.

For convexification purposes, the conforming DT seems to be a good solution upon first examination, and was one of the original techniques Williams proposed for the problem. One of the main benefits is that since a conforming DT is actually a DT of a larger point set, it must be acyclic. On closer inspection, we can see that conforming DTs have a number of potential weaknesses. First, if the original mesh was not a DT, we may need to completely re-triangulate it. This means that internal structures of the mesh, which may have been carefully designed by the modeler, are potentially lost. In addition, the available experimental evidence [12] suggests that a considerable number of Steiner points may be necessary. Part of the problem is that

when a face f_i is pierced by the DT, adding a *local* point p to resolve this issue can potentially result in modifications to the mesh deep within the triangulation, not just in the neighborhood of the point p . Another potential issue with using a conforming DT is the lack of available robust codes for the computation. This is an issue that we expect soon to be resolved, with advances under way in the computational geometry community.

The constrained DT (Fig. 1(d)) is a different way to resolve the problem of respecting a given set of faces. While in a conforming DT we only had to make sure that each given face f_i can be represented as the union of a set of faces in the conforming DT, for a constrained DT we insist that each face f_i appears exactly as a face in the tetrahedralization. In order to do this, we must *relax the empty-circumsphere criterion* that characterizes a DT; thus, a constrained DT is *not* (in general) a Delaunay tetrahedralization. The definition of the constrained DT requires a modification to the empty-circumsphere criteria in which we use the input faces $\{f_i\}$ as blockers of visibility and empty-circumsphere tests are computing taking that into account. That is, when performing the tests, we need to *discard* certain geometry when the sphere intersects one (or more) of the input faces. We refer the reader to [32] and [16, Chap. 2] for a detailed discussion. In regions of the mesh “away from” the input faces, a constrained DT looks very much like a standard DT. In fact, they share many of the same properties [30].

Because we are not allowed to add Steiner points when building a constrained DT, they have certain (theoretical) limitations. A particularly intriguing possibility is that it may not be possible to create one because some polyhedra cannot be tetrahedralized without adding Steiner points. (In fact, it is NP-complete to decide if a polyhedron can be tetrahedralized without adding Steiner points [27].) Further, constrained DTs suffer from some of the same issues as conforming DTs in that they may require re-triangulation of large portions of the original mesh. While it may be possible to maintain the Delaunay property on the “internal” portions of the mesh, away from the boundary faces, it is unclear what effect the non-Delaunay portions of the mesh near the boundary have on global properties, such as acyclicity, of the mesh. At this point, some practical issues related to constrained DTs are an area of active investigation [30, 33]; to our knowledge, there is no reliable code available for computing them.

Whether using a conforming or a constrained Delaunay tetrahedralization, the robust computation of the structure for very large point sets is not trivial. Even the best codes take a long time and use substantial amounts of memory. Some of the interesting non-convex meshes we would like to handle have on the order of ten million tetrahedra or more. In the case that the whole dataset needs to be re-triangulated, it is unclear if these techniques would be practical.

4 Direct Convexification Approaches Using BSP-trees

The Binary Space Partitioning tree (BSP-tree) is a geometric structure that has many interesting properties that can be explored in the convexification problem. For

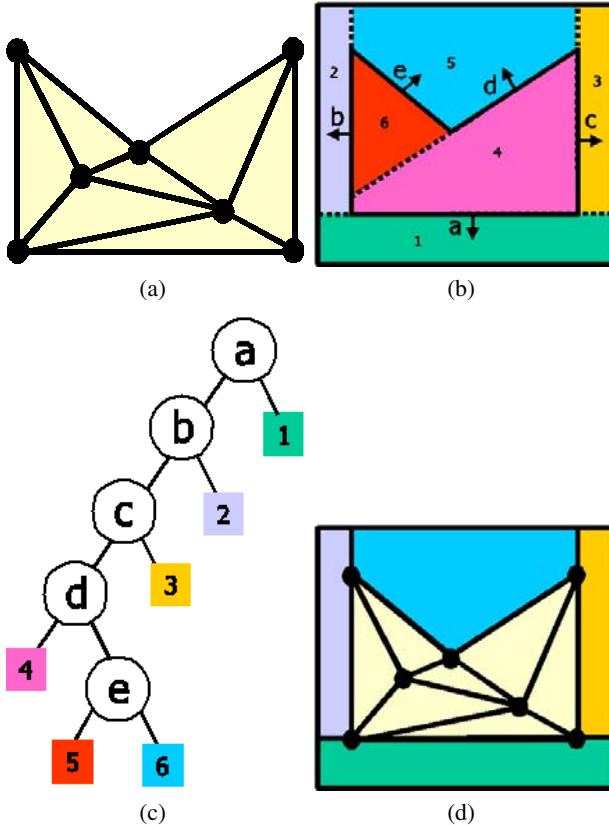


Fig. 2. Using BSP-trees to fill space. (a) The input non-convex mesh; (b) the BSP decomposition using the boundary facets of the input mesh; (c) the corresponding BSP tree; and, (d) the input mesh augmented with BSP cells

instance, the BSP-tree induces a hierarchical partition of the underlying space into convex cells that allows visibility ordering to be extracted by a priority-search driven by the viewpoint position (in a near-to-far or far-to-near fashion) [18]. In Fig. 2 we show how the BSP is used to capture the structure of the empty space.

4.1 Implicit BSP-Tree Regions

The visibility-ordering produced by the BSP-tree was explored in [13] to produce missing visibility relations in projective volume rendering. The approach relies on using the BSP-trees to represent the empty space surrounding a non-convex mesh. Since the empty space $CH(S) \setminus S$ and mesh S have a common intersection at the boundary facets of the mesh S , a BSP-tree was constructed using cuts along the supporting planes of the boundary facets. The construction algorithm starts with

the collection of boundary facets of the mesh, and uses an appropriate heuristic to choose a cut at each step to partition the space. The partition process associates each facet with the corresponding half-space (two half-spaces if a facet is split), storing the geometric representations of the boundary facets along the partitioning plane at the nodes of the BSP. The process is recursively repeated at each subtree until a stopping criterion is satisfied.

The resulting BSP-tree partitions the space into convex cells that are either internal or external to the mesh. If a consistent orientation for the boundary facet normals is used, these sets can be distinguished by just checking to which side a given leaf node is with respect to its parent (see Fig. 2).

In this approach no effort was made to enumerate explicitly the convex regions corresponding to the empty space in the BSP-tree. However, their implicit representation was used to help provide the missing visibility ordering information in the empty space surrounding the mesh.

Central to this approach is the extraction of visibility relations between interior regions (mesh cells) and exterior regions (the convex cells of the empty space induced by the BSP-tree). The boundary facets of the mesh S are the common boundary between these two types of regions. The approach used in [13] explores one way to obtain the visibility relations, using the visibility ordering produced by the BSP-tree to drive this process. This is done by using a visibility ordering traversal in the BSP-tree with respect to a given viewpoint (in a far-to-near fashion). When an internal node is visited we reach a boundary facet of the mesh. Only facets facing away from the viewing direction impose visibility ordering restrictions, and, for these, two situations can arise, as follows.

The first case happens when the facet stored at the node was not partitioned by the BSP-tree, and therefore is entirely contained in the hyperplane (visible). Visiting an entirely visible boundary facet allows the visibility ordering restriction imposed by this facet into the incident mesh cell to be lifted, which may lead to the inclusion of the cell in the visibility ordering if all restrictions to this cell were lifted.

The second case happens when the boundary facet is partially stored at the BSP node, which indicates that it was partitioned by another cut in the BSP. In this case it is not possible to lift the visibility ordering restriction, since other fragments were not yet reached by the BSP traversal (and therefore not entirely visible). At the moment that the last facet fragment is visited, a cell may be able to be included in the visibility ordering. The solution proposed in [13] uses a counter to accumulate the number of facet fragments created, decrementing this counter for each fragment visited, and lifting the conditions imposed by the fragment when the counter gets to zero.

However, the partition of boundary facets by cuts in the BSP-tree has additional side effects that need to be taken into consideration. In such cases, the BSP traversal is not enough to produce a valid visibility ordering for mesh cells. This happens because the BSP establishes a partial ordering between the convex cells it defines, and a mesh cell that is partitioned by a BSP cut lies in different convex cells of the BSP. In Fig. 3 we have an example in which a cell $C1$ cannot enter the visibility ordering because a partially visited cell has facet fragments that were not yet visited.

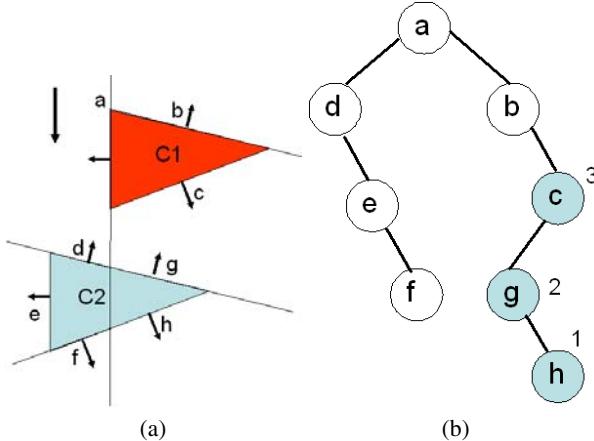


Fig. 3. Partially projected cells. Two cells, (a), and the corresponding BSP-tree, (b). The moment that the traversal reaches node c , cell $C1$ cannot be projected, but has to wait until a partially visited cell $C2$ has been projected

If the ordering to be produced is between the cell $C1$ and the two sub-cells of $C2$, then the BSP ordering suffices.

Cells that have partially visited facets need special treatment; the collection of all such cells at any given time is maintained in a partially projected cells list (PPC). It can be shown that a valid visibility ordering can be produced by the partial orderings provided by mesh adjacencies ($<_{ADJ}$), the ordering produced by the BSP-tree traversal ($<_{BSP}$), and an additional intersection involving cells in the PPC list ($<_{PPC}$). The PPC test increases the complexity of the algorithm; however, it is guaranteed not to generate cycles.

4.2 Explicit BSP-trees Regions

The implicit use of the convex regions induced by the BSP-tree in the previous approach required a BSP-traversal to drive the visibility ordering procedure. Another approach is to compute explicitly such convex regions (*filler cells*) and combine them with the mesh to form a convex mesh.

The construction of the BSP-tree uses, as before, partitioning cuts defined by the planes through the boundary facets, except that a different heuristic is used to select the cuts. The algorithm that computes the filler cells needs to perform the following tasks:

- **Computing the geometry of the filler cells:**

Extracting convex regions associated with nodes of the BSP is straightforward; it can be done in a top-down manner, starting at the root of the tree with a bounding box that is guaranteed to contain the entire model. In order to obtain the convex regions of the left and right children, the convex region associated with the node

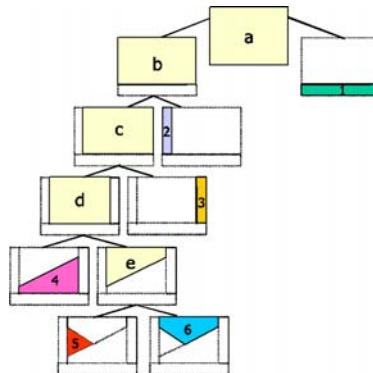


Fig. 4. Geometric computation of filler cells. Illustration of the recursive procedure that applies a partitioning operation to the cell of a node

is partitioned by the hyperplane. The resulting two convex regions are associated with the children nodes, and the process continues recursively. Figure 4 illustrates this process.

- **Computing topological adjacencies between mesh and filler cells:**

The extraction of topological information in the BSP is not as straightforward. One difficulty that arises is the fact that a cell may be adjacent, by a single facet, to more than one cell. (The cells do not form a cell complex.) The fact that the BSP has arbitrary direction cuts makes the task even harder, requiring an approach that handles numerical degeneracies. The topological adjacencies that need to be computed include filler-to-filler adjacencies, mesh-to-filler and filler-to-mesh adjacencies (see Fig. 5).

This convexification approach needs to satisfy the requirements posed before; we briefly discuss them in the context of this approach: **Preserving Acyclicity:**

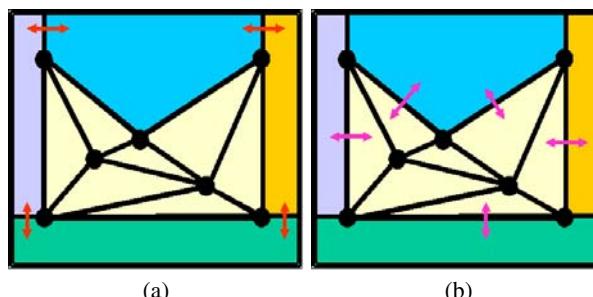


Fig. 5. Topological adjacencies. Filler-to-filler adjacency relations (a) and mesh-to-filler (and vice-versa) relations (b) that need to be computed

Although the internal adjacencies of the mesh may not lead to cycles in the visibility ordering, the addition of filler cells may lead to an augmented model (mesh plus filler cells) that contains cycles. Since the mesh is assumed acyclic, cycles do not involve only mesh cells, and from the visibility ordering property of BSP-trees, cycles do not involve only filler cells. Cycles will not involve runs of several filler to mesh cells (filler-mesh), or vice-versa (mesh-filler), since each one of the runs is acyclic. However, cycles can happen in filler-mesh-filler or mesh-filler-mesh cells.

It is still an open problem how to design techniques to avoid or to minimize the appearance of cycles. (See [1, 9] for theoretical results on cutting lines to avoid cycles.) Also, it would be interesting to establish bounds on the number of cells in a cycle.

Output Size: The number of cells generated is directly related to the size of the BSP-tree. Although the BSP can have worst-case $\Theta(n^2)$ in \Re^3 , in practice the use of heuristics reduces the typical size of a BSP to linear. Preliminary tests show that one can expect an increase of 5-10% in the number of cells produced.

Computational and Memory Complexity: The computational cost of the algorithm is proportional to the time required to build a BSP for the boundary faces. The extraction of geometric and topological information of the BSP is proportional to the time to perform a complete traversal of the BSP.

Boundary and Interior Preservation: The BSP approach naturally preserves the boundary and interior of the mesh, since it only constructs cells that are outside S . This requires that the mesh has the interior well defined, i.e., each connected component of the boundary is a 2-manifold. A consistent orientation of boundary facet normals allows an easy classification of which cells of the BSP are interior or exterior to the mesh.

Robustness and Degeneracy Handling: The fundamental operations used in the construction of BSP-trees are point-hyperplane classification and the partition of a facet by a hyperplane. The fact that geometric computations rely on only these two operations allows better control of issues of numerical precision and floating point errors. Of course, unless one uses exact geometric computation [28, 42], numerical errors are inevitable; however, several geometric and topological predicates can be checked to verify if a given solution is numerically consistent. The literature on solid modeling has important suggestions on how to do this [20], as in the problem of converting CSG solids to a boundary representation. The possibility of having nearly coplanar boundary facets needs to be treated carefully, since it may require the partition of a facet by a nearly coplanar hyperplane.

The filler cells obtained after a convexification algorithm need to be added to the non-convex mesh, with updates to the topological relationships. In particular, three new types of topological relationships need to be added: filler to filler adjacencies, filler to mesh adjacencies and mesh to filler adjacencies. This problem is complicated by the fact that adjacencies do not occur at a single facet (i.e., a cell can be adjacent to more than one cell, as the cells do not necessarily form a cell complex). Again, geometric and topological predicates that guarantee the validity of topological

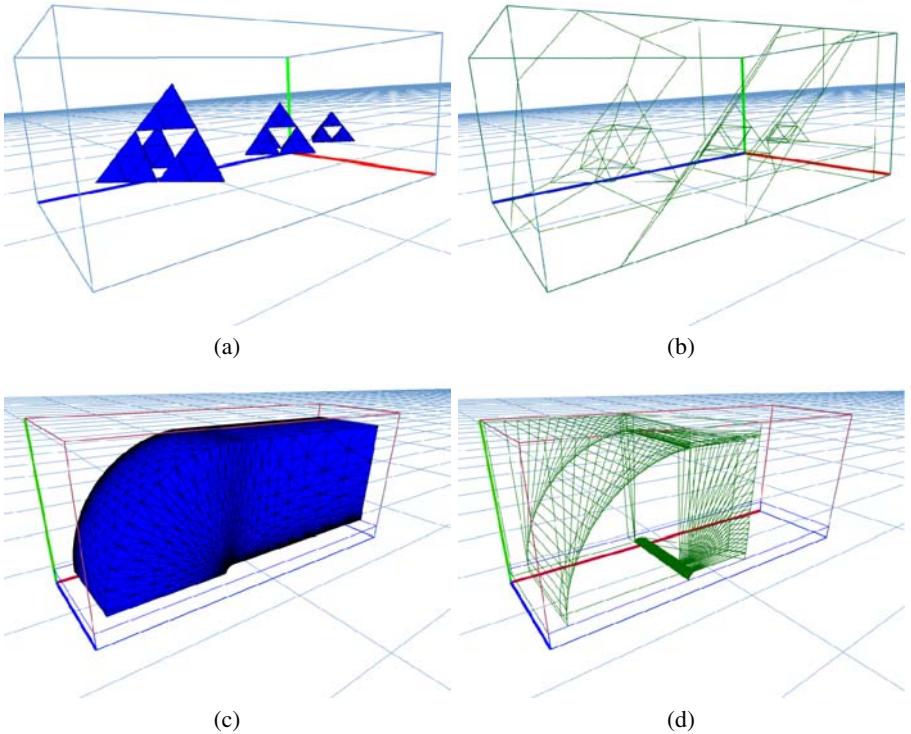


Fig. 6. Explicit BSP regions. Two sample meshes ((a) and (c)) and the correspond BSP-regions that fill space ((b) and (d))

relationships need to be enforced (e.g., if a cell c_i is adjacent to c_j by way of facet f_m , then there must exist a facet f_n such that c_j is adjacent to c_i by way of facet f_n).

5 Final Remarks

This work presents a brief summary of the current status of strategies to compute a convexification of space with respect to a non-convex mesh. We present a formal definition of the problem and summarize the requirements that one solution needs to fulfill. We discuss two possible solutions. The first is based in Delaunay triangulations; we point out some of the difficulties faced by this approach. We discuss the use of BSP-trees as a potentially better and more practical solution to the problem. However, many problems are still open. For example, what is a practical method for convexification that avoids the generation of cycles in the visibility relationship?

Acknowledgements

We thank Dirce Uesu for help in the preparation of images for this paper. Figure 1 was generated using Jonathan Shewchuk’s Triangle software. The work of João L. D. Comba is supported by a CNPq grant 540414/01-8 and FAPERGS grant 01/0547.3. Joseph S.B. Mitchell is supported by NASA Ames Research (NAG2-1325), the National Science Foundation (CCR-0098172), Metron Aviation, Honda Fundamental Research Lab, and grant No. 2000160 from the U.S.-Israel Binational Science Foundation. Cláudio T. Silva is partially supported by the DOE under the VIEWS program and the MICS office, and the National Science Foundation under grants CCF-0401498, EIA-0323604, and OISE-0405402.

References

1. M. de Berg, M. Overmars, and O. Schwarzkopf. Computing and verifying depth orders. *SIAM J. Comput.*, 23:437–446, 1994.
2. M. Bern. Triangulations and mesh generation. In J. E. Goodman and J. O’Rourke, editors, *Handbook of Discrete and Computational Geometry (2nd Edition)*, chapter 25, pp. 563–582. Chapman & Hall/CRC, Boca Raton, FL, 2004.
3. M. Bern, D. Dobkin, and D. Eppstein. Triangulating polygons without large angles. *Internat. J. Comput. Geom. Appl.*, 5:171–192, 1995.
4. M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 1 of *Lecture Notes Series on Computing*, pp. 23–90. World Scientific, Singapore, 1992.
5. M. Bern and P. Plassmann. Mesh generation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pp. 291–332. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
6. B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.
7. B. Chazelle and D. P. Dobkin. Optimal convex decompositions. In G. T. Toussaint, editor, *Computational Geometry*, pp. 63–133. North-Holland, Amsterdam, Netherlands, 1985.
8. B. Chazelle, H. Edelsbrunner, L. Guibas, J. Hershberger, R. Seidel, and M. Sharir. Selecting heavily covered points. *SIAM J. Comput.*, 23:1138–1151, 1994.
9. B. Chazelle, H. Edelsbrunner, L. J. Guibas, R. Pollack, R. Seidel, M. Sharir, and J. Snoeyink. Counting and cutting cycles of lines and rods in space. *Comput. Geom. Theory Appl.*, 1:305–323, 1992.
10. B. Chazelle and L. Palios. Triangulating a non-convex polytope. *Discrete Comput. Geom.*, 5:505–526, 1990.
11. L. P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4:97–108, 1989.
12. D. Cohen-Steiner, E. Colin de Verdière, and M. Yvinec. Conforming Delaunay triangulations in 3d. In *Proc. 18th Annu. ACM Sympos. Comput. Geom.*, 2002.
13. J. L. Comba, J. T. Klosowski, N. Max, J. S. Mitchell, C. T. Silva, and P. Williams. Fast polyhedral cell sorting for interactive rendering of unstructured grids. In *Computer Graphics Forum*, volume 18, pp. 367–376, 1999.
14. L. De Floriani and E. Puppo. A survey of constrained Delaunay triangulation algorithms for surface representation. In G. G. Pirooni, editor, *Issues on Machine Vision*, pp. 95–104. Springer-Verlag, New York, NY, 1989.

15. H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.
16. H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge, 2001.
17. S. Fortune. Voronoi diagrams and Delaunay triangulations. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry (2nd Edition)*, chapter 23, pp. 513–528. Chapman & Hall/CRC, Boca Raton, FL, 2004.
18. H. Fuchs, Z. M. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. *Comput. Graph.*, 14(3):124–133, 1980. Proc. SIGGRAPH '80.
19. M. P. Garrity. Raytracing irregular volume data. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):35–40, Nov. 1990.
20. C. Hoffmann. *Geometric and Solid Modeling*. Morgan-Kaufmann, San Mateo, CA, 1989.
21. J. M. Keil. Polygon decomposition. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pp. 491–518. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
22. M. Kraus and T. Ertl. Cell-projection of cyclic meshes. In *IEEE Visualization 2001*, pp. 215–222, Oct. 2001.
23. M. Kraus and T. Ertl. Simplification of Nonconvex Tetrahedral Meshes. In Farin, G. and Hagen, H. and Hamann, B., editor, *Hierarchical and Geometrical Methods in Scientific Visualization*, pp. 185–196. Springer-Verlag, 2002.
24. M. Murphy, D. M. Mount, and C. W. Gable. A point-placement strategy for conforming Delaunay tetrahedralization. In *Proc. 11th ACM-SIAM Sympos. Discrete Algorithms*, pp. 67–74, 2000.
25. G. M. Nielson, H. Hagen, and H. Müller. *Scientific Visualization: Overviews, Methodologies, and Techniques*. IEEE Computer Society Press, Washington, DC, 1997.
26. A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Chichester, UK, 1992.
27. J. Ruppert and R. Seidel. On the difficulty of triangulating three-dimensional non-convex polyhedra. *Discrete Comput. Geom.*, 7:227–253, 1992.
28. S. Schirra. Robustness and precision issues in geometric computation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 14, pp. 597–632. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
29. R. Seidel. Constrained Delaunay triangulations and Voronoi diagrams with obstacles. Technical Report 260, IIG-TU Graz, Austria, 1988.
30. J. R. Shewchuk. A condition guaranteeing the existence of higher-dimensional constrained Delaunay triangulations. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pp. 76–85, 1998.
31. J. R. Shewchuk. Constrained Delaunay tetrahedralizations, bistellar flips, and provably good boundary recovery. Talk slides; available from author's web page.
32. J. R. Shewchuk. Lecture notes on Delaunay mesh generation. Technical report, Department of Electrical Engineering and Computer Science, University of California at Berkeley, 1999.
33. J. R. Shewchuk. Sweep algorithms for constructing higher-dimensional constrained Delaunay triangulations. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pp. 350–359, 2000.
34. J. R. Shewchuk. Updating and constructing constrained Delaunay and constrained regular triangulations by flips. In *Proc. 19th Annu. ACM Sympos. Comput. Geom.*, pp. 181–190, 2003.

35. P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. In *San Diego Workshop on Volume Visualization*, volume 24 of *Comput. Gr*, pp. 63–70, Dec. 1990.
36. C. T. Silva, J. S. Mitchell, and P. L. Williams. An exact interactive time visibility ordering algorithm for polyhedral cell complexes. In *1998 Volume Visualization Symposium*, pp. 87–94, Oct. 1998.
37. D. Speray and S. Kennon. Volume probes: Interactive data exploration on arbitrary grids. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):5–12, November 1990.
38. C. Stein, B. Becker, and N. Max. Sorting and hardware assisted rendering for volume visualization. *1994 Symposium on Volume Visualization*, pp. 83–90, October 1994. ISBN 0-89791-741-3.
39. N. P. Weatherill and O. Hassan. Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. *International Journal for Numerical Methods in Engineering*, 37:2005–2039, 1994.
40. M. Weiler, M. Kraus, M. Merz, and T. Ertl. Hardware-Based Ray Casting for Tetrahedral Meshes. In *Proceedings of IEEE Visualization 2003*, pp. 333–340, 2003.
41. P. L. Williams. Visibility ordering meshed polyhedra. *ACM Transaction on Graphics*, 11(2):103–125, Apr. 1992.
42. C. Yap. Towards exact geometric computation. *Comput. Geom. Theory Appl.*, 7(1):3–23, 1997.

Brain Mapping Using Topology Graphs Obtained by Surface Segmentation

Fabien Vivodtzev¹, Lars Linsen²,
Bernd Hamann², Kenneth I. Joy², and Bruno A. Olshausen³

¹ Laboratoire GRAVIR (CNRS, INP Grenoble, INRIA, UJF).
`fabien.vivodtzev@imag.fr`

² Institute for Data Analysis and Visualization (IDAV), Department of Computer Science,
University of California, Davis. `{llinsen}@ucdavis.edu`,
`{hamann|joy}@cs.ucdavis.edu`

³ Center for Neuroscience, Department of Psychology, University of California, Davis.
`baolshausen@ucdavis.edu`

Summary. Brain mapping is a technique used to alleviate the tedious and time-consuming process of annotating brains by mapping existing annotations from brain atlases to individual brains. We introduce an automated surface-based brain mapping approach. After reconstructing a volume data set (trivariate scalar field) from raw imaging data, an isosurface is extracted approximating the brain cortex. The cortical surface can be segmented into gyral and sulcal regions by exploiting geometrical properties. Our surface segmentation is executed at a coarse level of resolution, such that discrete curvature estimates can be used to detect cortical regions. The topological information obtained from the surface segmentation is stored in a topology graph. A topology graph contains a high-level representation of the geometrical regions of a brain cortex. By deriving topology graphs for both atlas brain and individual brains, a graph node matching defines a mapping of brain cortex regions and their annotations.

1 Introduction

Annotating brains is a tedious and time-consuming process and can typically only be performed by an expert. A way to alleviate and accelerate the process is to take an already existing completely annotated brain and map its annotations onto other brains. The three-dimensional, completely annotated brain is called neuroanatomical brain atlas. An atlas represents a single brain or unified information collected from several “healthy” brains of one species. The digital versions of atlas brains are stored in databases [30]. Neuroscientists can benefit from this collected information by connecting to the database, accessing atlas brains, and mapping annotations onto their own data sets.

We propose an automated brain mapping approach that consists of several processing steps leading from three-dimensional imaging data to mapped cortical surfaces. Data sets are typically obtained in a raw format, which is the output of

some imaging technique, such as functional magnetic resonance imaging (fMRI), given as a stack of aligned two-dimensional images. Isosurface extraction is used to obtain a surface representation of the brain cortex.

The shape of the brain cortex is complex, having many winding folds and creases, but its main characteristic can be described by alternating convex and concave regions called *gyri* and *sulci*. We use a multiresolution surface representation, since fine details are not present at a coarse level of resolution, while more global gyral and sulcal brain structures still are. We detect gyri and sulci by exploiting discrete curvature estimates, and segment the cortical surface into distinct regions based on curvature. The curvature-based segmentation is independent of the size of the segments. Thus, it is capable of extracting cortical regions of varying size and of detecting corresponding cortical regions in atlas and user brains, even if a region's size varies substantially for two brains being compared.

Curvature-based segmentation leads to a topological characterization of the surface. A topology graph is used to store the topological surface information at a high level. The brain mapping is executed by generating topology graphs for atlas brain and user brains and finding node correspondences for the graphs, where each node represents a cortical region.

2 Related Work

First approaches in brain mapping used rigid models and spatial distributions. In [26], a stereotactic atlas is expressed in an orthogonal grid system, which is rescaled to a patient brain, assuming one-to-one correspondences of specific landmarks. Similar approaches are discussed in [2, 5, 11] using elastic transformations. The variation in brain shape and geometry is of significant extent between different individuals of one species. Static rigid models are not sufficient to describe appropriately such inter-subject variabilities.

Deformable models were introduced as a means to deal with the high complexity of brain surfaces by providing atlases that can be elastically deformed to match a patient brain. Deformable models use snakes [20], B-spline surfaces [24], or other surface-based deformation algorithms [8, 9]. Feature matching is performed by minimizing a cost function, which is based on an error measure defined by a sum measuring deformation and similarity. The definition of the cost function is crucial. Some approaches rely on segmentation of the main sulci guided by a user [4, 27, 29], while others automatically generate a structural description of the surface.

Level set methods, as described in [21], are widely used for convex shapes. These methods, based on local energy minimization, achieve shape recognition requiring little known information about the surface. Initialization must be done close to surface boundaries, and interactive seed placement is required. Several approaches have been proposed to perform automatically the seeding process and adapt the external propagation force [1], but small features can still be missed. Using a multiresolution representation of the cortical models, patient and atlas meshes are matched progressively by the method described in [16]. Folds are annotated according to size at a

given resolution. The choice of the resolution is crucial. It is not guaranteed that same features are present at the same resolution for different brains.

Many other automatic approaches exist, including techniques using active ribbons [10, 13], graph representations [3, 22], and region growing [18]. A survey is provided in [28]. Even though some of the approaches provide good results, the highly non-convex shape of the cortical surface, in combination with inter-subject variability and feature-size variability, leads to problems and may prevent a correct feature recognition/segmentation and mapping without user intervention.

Our approach is an automated approach that can deal with highly non-convex shapes, since we segment the brain into cortical regions, and with feature-size as well as inter-subject variability, since it is based on discrete curvature behavior. Moreover, isosurface extraction, surface segmentation, and topology graphs are embedded in a graphical system supporting visual understanding.

3 Brain Mapping

Our brain mapping approach is based on a pipeline of automated steps. Figure 1 illustrates the sequence of individual processing steps.

The input for our processing pipeline is discrete imaging data in some raw format. Typically, imaging techniques produce stacks of aligned images. If the images are not

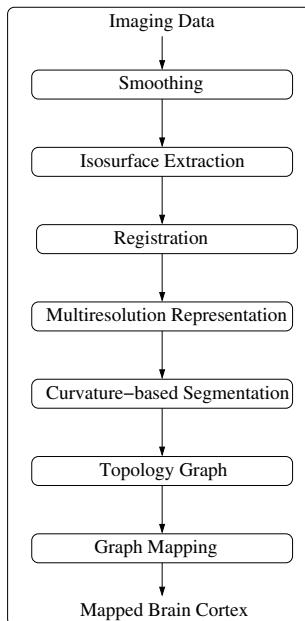


Fig. 1. Processing pipeline: from imaging data in raw format to mapped brain cortex surface

aligned, appropriate alignment tools must be applied [25]. Volumetric reconstruction results in a volume data set, a trivariate scalar field.

Depending on the used imaging technique, a scanned data set may contain more or less noise. We mainly operate on fMRI data sets, thus having to deal with significant noise levels. We use a three-dimensional discrete Gaussian smoothing filter, which eliminates high-frequency noise without affecting visibly the characteristics of the three-dimensional scalar field. The size of the Gaussian filter must be small. We use a $3 \times 3 \times 3$ mask locally to smooth every value of a rectilinear, regular hexahedral mesh. Figure 2 shows the effect of the smoothing filter applied to a three-dimensional scalar field by extracting isosurfaces from the original and filtered data set.

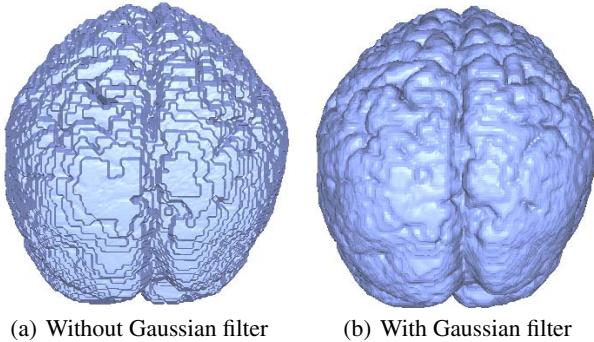


Fig. 2. Smoothing of volumetric scalar field visualized by extracting isosurfaces from original and filtered data

After this preprocessing step, we extract the geometry of the brain cortex from the volume data. The boundary of the brain cortex is obtained via an isosurface extraction step, as described in Sect. 4. If desired, isosurface extraction can be controlled and supervised in a fashion intuitive to neuroscientists.

Once the geometry of the brain cortices is available for both atlas brain and a user brain, the two surfaces can be registered. Since our brain mapping approach is feature-based, we perform the registration step by a simple and fast rigid body transformation. For an overview and a comparison of rigid body transformation methods, we refer to [6].

User-guided surface segmentation of the brain cortices is based on curvature estimates. Since curvature estimates are sensitive to high-frequency detail, a multiresolution approach is used, as described in Sect. 5. On a coarse level of resolution, only the main (low-frequency) features of the brain cortices are represented while the small (high-frequency) details are not present.

Curvature estimates on surfaces are used to distinguish between regions of different behavior [12]. We use Gaussian and mean curvatures to distinguish between elliptic and hyperbolic regions and between convex and concave regions, respectively.

The shape of a brain cortex is mainly defined by gyral and sulcal regions. We segment the surface based on these curvature characteristics, as described in Sect. 6.

Curvature-based surface segmentation describes the topological behavior of the surface, which we store in a topology graph, as described in Sect. 7. Nodes of the topology graph represent regions of the cortical surface. Neighborhood information of such regions is represented by edges in the graph.

The final brain mapping is performed on the high-level and abstract representation of a topology graph, as described in Sect. 8. We construct a topology graph for the atlas brain and a user brain and determine matching node correspondences.

4 Isosurface Extraction

Extracting the geometry of a brain cortex from a discrete trivariate scalar field can be done by standard isosurfacing techniques. We decided to use a marching cubes-like approach [19]. For the quality of brain mapping it is crucial to choose an “appropriate isovalue,” such that the extracted isosurface follows closely the geometrical shape of the brain cortex.

To validate the proper choice of an isosurface, we designed a tool that allows a user to supervise the isosurfacing procedure. Traditionally, neuroscientists segment data slice-by-slice in a two-dimensional setup. Thus, the supervision tool should allow them to inspect the original two-dimensional slices and an extracted segment boundary for that particular slice simultaneously.

Figure 3 shows an example of supervised isosurface extraction. The upper row shows isosurfaces extracted for various isovales. The two rows below show two original two-dimensional slices with overlaid cross sections (red contour) of the extracted isosurface. The left column shows the location of the slice with respect to the isosurface. In this particular example, the chosen isovale is a good one, since the red contours follow closely the gyri and sulci of the brain cortex.

Due to remaining noise in the data set, the isosurface extraction step produces one large main component and many small isolated components. The main component represents the brain cortex, while the small components should be removed. We use a surface-growing algorithm that generates a watertight triangular mesh in a half-edge data structure representing the largest component. The small components are removed.

5 Multiresolution Surface Representation

To obtain a multiresolution surface representation of a brain cortex, we start with the triangular isosurface mesh. To simplify the high-resolution triangular mesh we use a simplification algorithm based on progressive meshes [14]. We iteratively apply edge-collapse operations. Although collapsing an edge is a simple operation, it can modify topology and geometry. To ensure consistency of our mesh, we use consistency checks as described in [15], based on topological analysis in the neighborhood affected by a collapse operation.

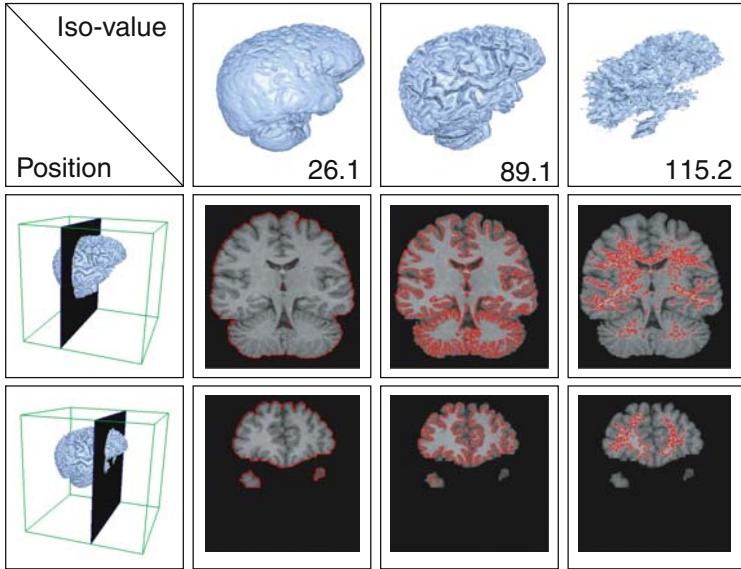


Fig. 3. Supervised isosurface extraction: overlaying original two-dimensional slices with cross sections (red contour) of extracted isosurface

For each edge of the mesh, an error corresponding to the cost of its collapse is computed and stored. According to this value an ordered heap of edges is created. During mesh simplification, the method identifies the top edge, checks for consistency, and, if possible, collapses it. This process is highly dependent on the error metric used to decide which edge to collapse next. Many metrics have been proposed for edge collapse algorithms over the past decade [7, 14, 17, 23]. Most of these metrics attempt to preserve “sharp” edges and details. Our objective, instead, is to remove detail even in regions of high curvature. Thus, our error metric is only based on edge length, and our main goal is to create a near-uniform distribution of vertices on the surface. After a valid collapse, the affected neighborhood is updated accordingly.

Topology (i.e., adjacency information of triangles) and geometry (i.e., positional information of the resulting “collapse” vertex) are modified by an edge collapse. An edge collapses to its midpoint. (We decided not to optimize the position to keep computation costs low.) Using midpoints also reduces the risk of self-intersections.

Figure 4 shows the result of simplifying a triangular mesh. The main gyral and sulcal features of the cortical surface are well preserved at the coarse level of resolution.

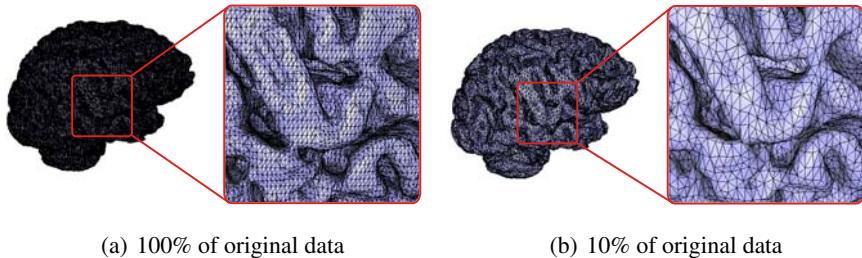


Fig. 4. Multiresolution surface representation

6 Surface Segmentation

6.1 Curvature-based Surface Characteristics

A surface can be divided into regions of elliptic and hyperbolic behavior. The regions of elliptic behavior can further be classified into convex and concave regions. When considering the a brain cortex, the gyri contain convex elliptic regions and the sulci contain concave elliptic regions. The blending areas between gyri and sulci are hyperbolic regions. This observation led to our decision to use curvature-based surface characteristics for user-guided surface segmentation. Discrete curvature estimates and their use for curvature-based surface segmentation were introduced in [31].

We use mean curvature estimates to distinguish between convex and concave regions. A discrete version of the mean curvature operator at a vertex \mathbf{x}_i of a triangular mesh can be defined by the length of a vector operator $\mathbf{K}(\mathbf{x}_i)$. For characterizing surface behavior with respect to mean curvature, we only need to use the direction of $\mathbf{K}(\mathbf{x}_i)$. Thus, we use a simplified operator $\mathbf{K}_{dir}(\mathbf{x}_i)$. The vector $\mathbf{K}_{dir}(\mathbf{x}_i)$ associated with a vertex \mathbf{x}_i is computed as a weighted sum of difference vectors emanating from \mathbf{x}_i and ending at the vertices being edge-connected with \mathbf{x}_i . The weight of the vector associated with edge e_{ij} between \mathbf{x}_i and its neighbor \mathbf{x}_j depends on the cotangents taken from the opposite angles of its adjacent faces. This operator is defined as

$$\mathbf{K}_{dir}(\mathbf{x}_i) = \sum_{j=1}^{N_i} (\cot \alpha_j + \cot \beta_j)(\mathbf{x}_j - \mathbf{x}_i),$$

where N_i is the number of neighbors constituting the set of edge-connected neighbor vertices of \mathbf{x}_i , and α_j, β_j are the opposite angles of e_{ij} with respect to its adjacent faces, see Fig. 5.

We use the operator $\mathbf{K}_{dir}(\mathbf{x}_i)$ to define the Boolean operator $mean(\mathbf{x}_i)$, which allows us to distinguish between convex and concave regions. It is defined as

$$mean(\mathbf{x}_i) = \begin{cases} \text{convex} & \text{if } \mathbf{K}_{dir}(\mathbf{x}_i) \cdot \mathbf{n}_i \leq 0 \\ \text{concave} & \text{if } \mathbf{K}_{dir}(\mathbf{x}_i) \cdot \mathbf{n}_i > 0, \end{cases}$$

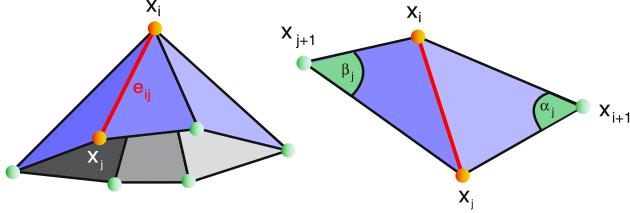


Fig. 5. Parameters used by mean curvature operator

where \mathbf{n}_i is a discrete approximation of the normal vector at \mathbf{x}_i . In concave areas, the operator $\mathbf{K}_{dir}(\mathbf{x}_i)$ and the normal vector \mathbf{n}_i are directed in roughly opposite directions, whereas in convex areas they are directed in roughly the same direction, see Fig. 6.

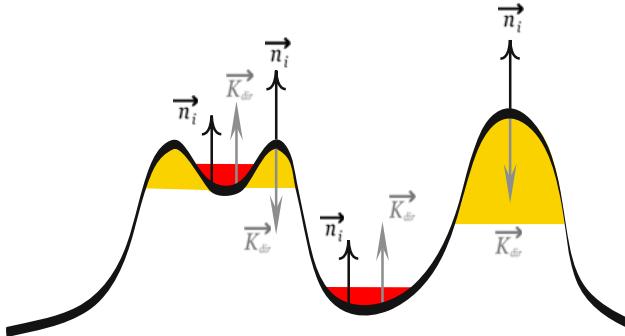


Fig. 6. Using mean curvature to distinguish between convex and concave regions.

To further distinguish between elliptic and hyperbolic regions, i. e., to separate local extrema from blending regions, we consider Gaussian curvature. A discrete version of the Gaussian curvature at a vertex \mathbf{x}_i of a triangular mesh can be defined by the length of an operator $\kappa_G(\mathbf{x}_i)$. This operator compares 2π with the sum of inner angles θ_j of all the adjacent faces of a vertex \mathbf{x}_i , see Fig. 7. In the planar case, the sum of the angles is 2π . When \mathbf{x}_i is an extremum, a plane through \mathbf{x}_i exists, such that all neighbor vertices of \mathbf{x}_i lie on one side of that plane, see Fig. 7. Thus, the angles sum to a value smaller than 2π . When \mathbf{x}_i is not an extremum and we compute the best fitting plane in the least-squares sense through \mathbf{x}_i , the neighbor vertices lie above and below that plane. In this situation, the angles sum up to a value larger than 2π . Hence, we consider only the sign of the operator $\kappa_G(\mathbf{x}_i)$, defined as

$$\kappa_G(\mathbf{x}_i) = 2\pi - \sum_{j=1}^{N_i} \theta_j ,$$

where θ_j is the angle between the difference vectors $\mathbf{x}_j - \mathbf{x}_i$ and $\mathbf{x}_{j+1} - \mathbf{x}_i$, emanating from vertex \mathbf{x}_i and ending at neighbors \mathbf{x}_j and \mathbf{x}_{j+1} , respectively, see Fig. 7.

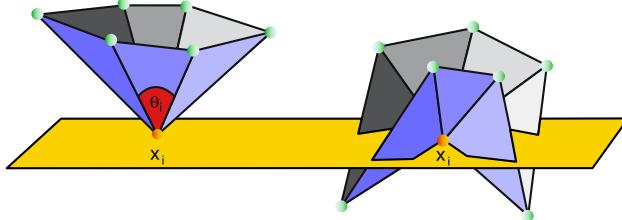


Fig. 7. Using Gaussian curvature to distinguish between elliptic and hyperbolic regions

We use $\kappa_G(\mathbf{x}_i)$ to define another Boolean operator $Gauss(\mathbf{x}_i)$, which is true if the vertex \mathbf{x}_i is a local extremum:

$$Gauss(\mathbf{x}_i) = \begin{cases} \text{elliptic} & \text{if } \kappa_G(\mathbf{x}_i) > 0 \\ \text{hyperbolic} & \text{if } \kappa_G(\mathbf{x}_i) \leq 0 \end{cases}.$$

6.2 Curvature-based Segmentation

By combining the operators *mean* and *Gauss*, we can generate an initial surface segmentation consisting of regions of the same type of curvature. Figure 8 shows how a cortical surface of a human brain is partitioned into elliptical convex regions (yellow), elliptical concave regions (red), hyperbolic convex regions (green), and hyperbolic concave regions (blue).

Figure 9 shows the effect of the multiresolution surface representation by segmenting the high- and low-resolution surfaces from Fig. 4 with respect to mean curvature. Only for the low-resolution surface, the cortical regions are detected as desired.

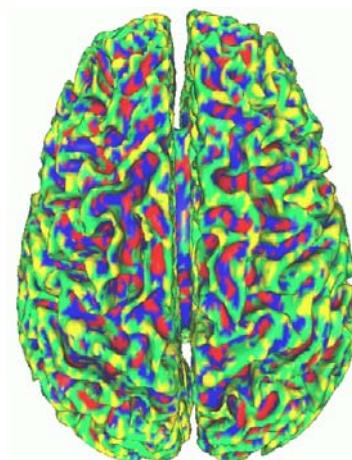


Fig. 8. Curvature-based segmentation of cortical surface

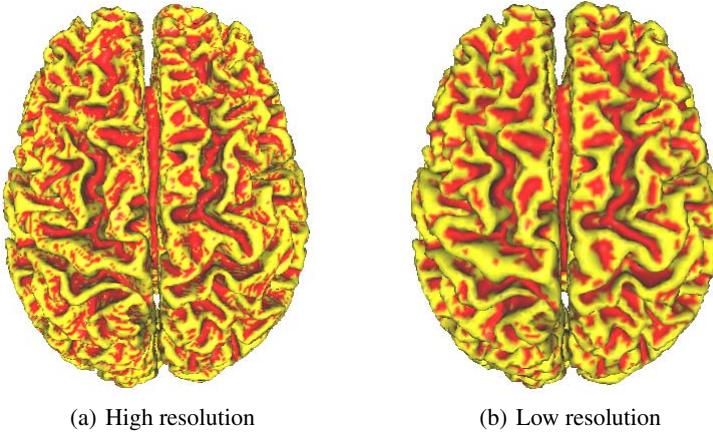


Fig. 9. Surface segmentation at different levels of resolution

7 Topology Graph

Curvature-based surface segmentation implies a topology for the surface. We construct a graph that stores the topology information. The nodes in the graph represent regions of a certain curvature type, and the edges in the graph represent neighborhood information of the surface regions.

For cortical surfaces, gyral regions cover larger parts of the brain. Their segmentation into smaller functional regions cannot be done automatically, since it is not based on geometrical properties. Sulcal regions instead remain local. Thus, we decided to use sulci only for the construction of topology graphs.

Each node in the topology graph represents one sulcus. The node representing a certain sulcus is generated by collapsing all vertices of the triangulated surface that are characterized by the surface segmentation procedure as belonging to that sulcus. The position of the node is determined by averaging the positions of the collapsed vertices.

To determine neighborhood information for sulci on the cortical surface, we use a contour-growing algorithm. Starting from a polygonal contour that describes the boundary of a sulcus on the triangulated surface, we grow the contour iteratively by one triangle in all directions, i. e., after one iteration step, the new contour encloses all vertices of the old contour plus all its neighbors. If the contour of a sulcus, when growing, intersects another sulcus, then these sulci are considered neighbors, and the nodes representing these sulci in the topology graph are connected by an edge. The number of iteration steps depends on the resolution of the triangulated surface.

Figure 10 shows the generation of a topology graph for a cortical surface of a human brain. Figure 10(a) shows the segmented surface, where the detected sulci are rendered using random colors. Figure 10(b) shows the topology graph generated from the segmented surface, where nodes are shown in red and edges in blue. We

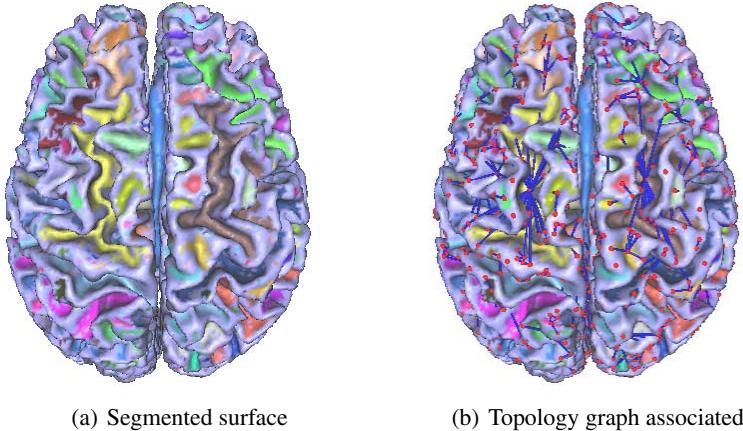


Fig. 10. Topology graph from surface segmentation

applied surface segmentation at a resolution of 50%, and used four iteration steps for generating the edges in the graph.

8 Graph Mapping

We prepare the brain mapping step by generating a topology graph representation for both atlas brain and a user brain. The mapping is performed by matching graph nodes. In addition to node position and edge connectivity information, the graphs also store, for each node, the size of the associated sulcus. Since edges in the triangular mesh have nearly the same length, the size of a sulcus can be estimated well by the number of vertices of the triangular mesh that are classified to belong to the sulcus.

For each node \mathbf{n}_u of the topology graph representing a user brain, we identify a node \mathbf{n}_a in the topology graph representing the atlas brain that provides a best match in terms of location and size. To find a best match for \mathbf{n}_u , we search for a node \mathbf{n}_a , representing the sulcus, whose size is closest to the size of the sulcus represented by \mathbf{n}_u . The search is restricted by limiting the Euclidean distance from \mathbf{n}_u and the topological distance in the graph to not being beyond a certain threshold.

Figure 11 shows the result of a graph-based brain mapping. Figure 11(a) shows the atlas brain, and Fig. 11(b) shows the user brain. Colors of the sulci indicate which sulci of the atlas brain are associated with which sulci of the user brain. Regions consisting of less than a certain number of vertices are not considered as being useful and are not mapped (indicated by red in Fig. 11(b)).

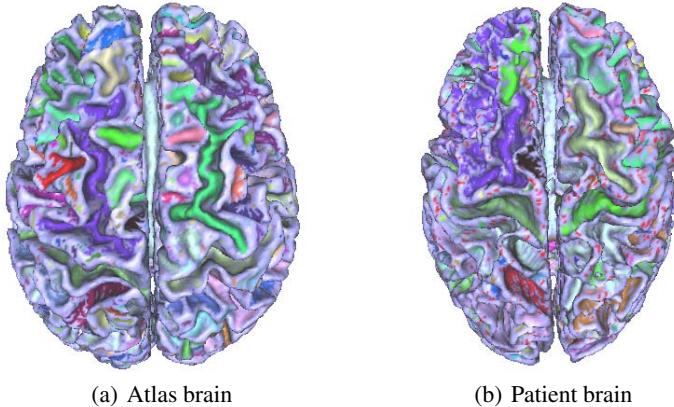


Fig. 11. Brain mapping based on topology graphs

9 Conclusions and Future Work

We have presented an automated approach for brain mapping to map annotations of the cortical surface from a brain atlas to individual brains. After reconstructing trivariate scalar fields from raw imaging data, isosurfaces are extracted approximating brain cortices. A cortical surface is segmented into gyral and sulcal regions by exploiting geometrical properties. Our surface segmentation step is performed at a coarse level of resolution, such that discrete curvature estimates can be used to detect cortical regions. The topological information obtained from the surface segmentation step is stored in a topology graph. A topology graph contains a high-level representation of the geometrically distinct regions of a brain cortex. By deriving topology graphs for both atlas brain and user brain, a high-quality brain mapping is obtained by mapping graph nodes.

We plan to extend the node matching process in a way that further exploits region neighborhood information. Moreover, we would like to develop a more sophisticated registration method, which we should lead to further improvement of node matching results.

Acknowledgments

This work was supported by the National Science Foundation under contracts ACI 9624034 (CAREER Award) and ACI 0222909, through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, and through the National Partnership for Advanced Computational Infrastructure (NPACI); the National Institute of Mental Health and the National Science Foundation under contract NIMH 2 P20 MH60975-06A2; and the Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878

and under Memorandum Agreement B503159. We thank the members of the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

References

1. C. Baillard and C. Barillot. Robust 3d segmentation of anatomical structures with level sets. In G.-P. Bonneau, S. Hammann, and Charles D. Hansen, editors, *Proceedings of Medical Image Computing and Computer-Assisted Intervention, MICCAI'00, LNCS 1935*, pages 236–245, 2000.
2. F.L. Bookstein. Thin-plate splines and the atlas problem for biomedical images. In A. Colchester and D. Hawkes, editors, *12th Internat. Conf. Information Processing in Medical Imaging, vol. 511 of Lecture Notes in Computer Science*, pages 326–342, 1991.
3. P. Cachier, J.F. Mangin, X. Pennec, D. Rivi  re, D. Papadopoulos-Orfanos, J. Regis, and N. Ayache. Multisubject non-rigid registration of brain MRI using intensity and geometric features. In W.J. Niessen and M.A. Viergever, editors, *4th Int. Conf. on Medical Image Computing and Computer-Assisted Intervention (MICCAI'01), vol. 2208 of Lecture Notes in Computer Science*, pages 734–742, 2001.
4. D.L. Collins, G. Le Goualher, and A.C. Evans. Non-linear cerebral registration with sulcal constraints. In *First International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), LNCS 1496*, pages 974–984, 1998.
5. D.L. Collins, T.M. Peters, and A.C. Evans. An automated 3d nonlinear image deformation procedure for determination of gross morphometric variability in human brain. In *Proc. Conf. Visualization in Biomedical Computing, SPIE 2359*, pages 180–190, 1994.
6. D.W. Eggert, A. Lorusso, and R.B. Fisher. Estimating 3-d rigid body transformation: a comparison of four major algorithms. *Machine Vision and Applications*, 9:272–290, 1997.
7. M. Garland and P.S. Heckbert. Surface simplification using quadric error metrics. *Computer Graphics, 31st Annual Conference Series*, pages 209–216, 1997.
8. R.D. Rabbitt G.E. Christensen and M.I. Miller. 3-d brain mapping using a deformable neuroanatomy. *Physics in Medicine and Biology*, 39:609–618, 1994.
9. J.C. Gee, M. Reivich, and R. Bajcsy. Elastically deforming 3-d atlas to match anatomical brain images. *Journal of Computer Assisted Tomography*, 17:225–236, 1993.
10. G. Le Goualher, E. Procyk, L. Collins, R. Venegopal, C. Barillot, and A. Evans. Automated extraction and variability analysis of sulcal neuroanatomy. *IEEE Transactions on Medical Imaging, TMI*, 18(3):206–217, 1999.
11. T. Greitz, C. Bohm, S. Holte, and L. Eriksson. A computerized brain atlas: Construction, anatomical content, and some applications. *Journal of Computer Assisted Tomography*, 15:26–38, 1991.
12. B. Hamann. Curvature approximation for triangulated surfaces. In G. Farin, H. Hagen, and H. Noltemeier, editors, *Geometric Modelling, Computing Suppl. 8*, pages 139–153. Springer-Verlag, 1993.
13. P. Hellier and C. Barillot. Coupling dense and landmark-based approaches for non-rigid registration. *IEEE Transactions on Medical Imaging*, 22:974–984, 2003.
14. H. Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH 1996*, pages 99–108. ACM Press, 1996.
15. H. Hoppe, T.D. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Proceedings of SIGGRAPH 1993*, pages 19–26. ACM Press, 1993.

16. S. Jaume, B. Macq, and S.K. Warfield. Labeling the brain surface using a deformable multi-resolution mesh. In *Proceedings of Medical Image Computing and Computer-Assisted Intervention, MICCAI 2002*, pages 451–458, 2002.
17. P. Lindstrom and G. Turk. Fast and efficient polygonal simplification. In *Proceedings of IEEE Conference on Visualization 1998*, pages 279–286. IEEE Computer Society Press, 1998.
18. G. Lohmann and D.Y. von Cramon. Automatic labeling of the human cortical surface using sulcal basins. *Medical Image Analysis*, 4(3):179–188, 2000.
19. William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH 1987*, pages 163–169. ACM Press, 1987.
20. A. Witkin M. Kass and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.
21. R. Malladi, J.A. Sethian, and B.C. Vemuri. Shape modeling with front propagation: A level set approach. *IEEE Transactions on PAMI*, 17(2):158–175, 1995.
22. D. Rivière, J.F. Mangin, D. Papadopoulos, J.M. Martinez, V. Frouin, and J. Regis. Automatic recognition of cortical sulci using a congregation of neural networks. In *Third International Conference on Medical Robotics, MICCAI'00, Imaging and Computer Assisted Surgery*, pages 40–49, 2000.
23. R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum, Proceedings of Eurographics 1996*, 15(3), 1996.
24. S. Sandor and R. Leahy. Surface-based labeling of cortical anatomy using a deformable atlas. *IEEE Transaction on Medical Imaging*, 16(1):41–54, 1997.
25. D. Shulga and J. Meyer. Aligning large-scale medical and biological data sets: Exploring a monkey brain. In *Visualization, Imaging and Image Processing (VIIP 2001)*, pages 434–439. The International Association of Science and Technology for Development (IASTED), 2001.
26. J. Talairech and P. Tournoux. *Co-Planar Stereotaxic Atlas of the Human Brain, 3-Dimensional Proportional System: An Approach to Cerebral Imaging*. Thieme Medical Publisher, Inc., Georg Thieme Verlag, 1988.
27. P. Thompson and A.W. Toga. Detection, visualization and animation of abnormal anatomic structure with a deformable probabilistic brain atlas based on random vector field transformation. *Medical Image Analysis*, 1(2):271–294, 1996.
28. P.M. Thompson, R.P. Woods, M.S. Mega, and A.W. Toga. Mathematical/computational challenges in creating deformable and probabilistic atlases of the human brain. *Human Brain Mapping*, 9:81–92, 2000.
29. M. Vailland and C. Davatzikos. Hierarchical matching of cortical features for deformable brain image registration. In *Proceeding of IPMI'99, LNCS 1613, Springer-Verlag, Berlin, Germany*, pages 182–195, 1999.
30. D.C. Van Essen, J. Harwell, D. Hanlon, and J.P.M. Dickson. Surface-based atlases and a database of cortical structure and function. In S.H. Koslow and S. Subramaniam, editors, *Databasing the Brain: From Data to Knowledge (Neuroinformatics)*, John Wiley & Sons, 2003.
31. Fabien Vivodtzev, Lars Linsen, Georges-Pierre Bonneau, Bernd Hamann, Kenneth I. Joy, and Bruno A. Olshausen. Hierarchical isosurface segmentation based on discrete curvature. In G.-P. Bonneau, S. Hahmann, and Charles D. Hansen, editors, *Proceedings of VisSym '03, Eurographics-IEEE TVCG Symposium on Visualization*, 2003.

Computing and Displaying Intermolecular Negative Volume for Docking

Chang Ha Lee and Amitabh Varshney

Department of Computer Science and UMIACS, University of Maryland, College Park, MD 20742, USA, xs
`{chlee, varshney}@cs.umd.edu`

Summary. Protein docking is a Grand Challenge problem that is crucial to our understanding of biochemical processes. Several protein docking algorithms use shape complementarity as the primary criterion for evaluating the docking candidates. The intermolecular volume and area between docked molecules is useful as a measure of the shape complementarity. In this paper we discuss an algorithm for interactively computing intermolecular negative volume and the area of docking site using graphics hardware. We also present the design considerations for building an interactive 3D visualization tool for visualizing intermolecular negative volumes.

1 Introduction

Several drug development processes have so far begun with large-scale random screening of candidate inhibitors. These initial discoveries are improved through well-defined approaches to find new drugs. As molecular structure determination techniques and computational methods progress, protein docking methods using structure-based molecular complementarity have become an important substitute for random screening in the drug design process [10].

Among many factors involved in protein-protein interactions such as electrostatics, hydrophobicity, and hydrogen bonding, shape complementarity is of major importance for protein docking. Purely geometric approach can restrict the time-consuming calculations of interaction energy to be performed only for those cases that have a good geometric fit. Geometric methods can also be used as foundations for more complete approaches considering chemical and energetic characteristics [2]. A complete search of all possible geometric fits of two flexible molecules takes too much time because of the extremely large degrees of freedom. Therefore, molecules have been often assumed as rigid bodies. Even with the rigid body assumption, finding accurate shape complementarities remains a challenging problem. Most existing methods provide a list of candidates sorted by complementarity criteria and the final decision by human is needed. Therefore, an interactive tool for visualizing the shape complementarity would be useful.

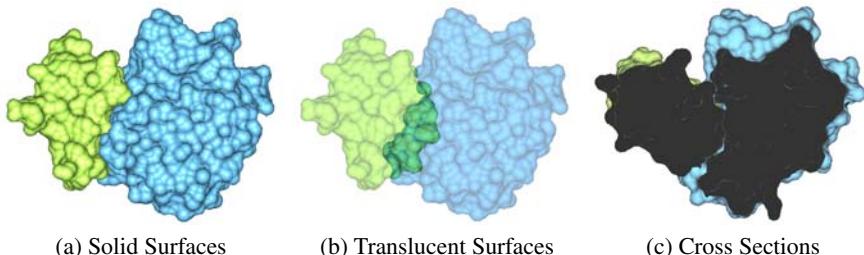


Fig. 1. Traditional Complementarity Visualization Methods

There are many methods for visualizing the steric fit between molecules. These include visualization using solid solvent-accessible smooth molecular surfaces, translucent molecular surfaces, and cross-sections of molecular surfaces (See Fig. 1). The solid molecular surface representation is unsuitable for complementarity visualization because the interface between molecules is difficult to observe due to occlusions from the solid surfaces. Translucent molecular surfaces allow the visualization of the interface between molecules. However, visual interference from other parts of the molecules prevents a clear visualization of the intermolecular interface. The cross-section method, also called the Z-clip method in graphics, visualizes the molecular interface by displaying cross sections of the molecules at varying depths from the viewer. Although the interface can be visualized clearly using two-dimensional cross sections, it is difficult to construct a mental model of the three-dimensional spatial structure of the interface. Therefore, in addition to the visualization of molecular surfaces, we need new and more informative methods for the visualization of the interface between molecules.

In this paper we present a method that computes *the negative volume* between molecules to visualize their interface. Our method leverages the recent advances in the 3D graphics hardware to achieve interactive rates of performance. Using this method scientists can interactively study various possible docking conformations and visualize the quality of the steric fit.

The remaining paper is organized as follows. In Sect. 2, we give an overview of the previous work. The concept of intermolecular negative volume and the description of the algorithm for computing intermolecular negative volume are given in Sects. 3, 4, and 5. The algorithm for computing the area and volume of the docking site is described in Sect. 6. In Sect. 7, we describe our interactive 3D visual tool to assist protein docking. We conclude this paper and discuss future work in Sect. 8.

2 Previous Work

The early research on drug design focused on geometric shape complementarity. Connolly [2] has proposed a protein docking algorithm based on geometric shape complementarity. He defines a molecule's *shape function* parameterized by scale R , at a surface point p as the volume of the molecule that lies inside a sphere of radius

R centered at p . He defines the *knobs* as the local minima in the shape function and *holes* as the local maxima. His method finds the transform to dock two proteins by finding matches between quartets of knobs and holes on the two proteins. Katchalski-Katzir et al. [8] have proposed a Fourier-transform-based geometric recognition algorithm for molecular surface complementarity.

Edelsbrunner et al. [4, 5] have defined a *pocket* as a region in the complement if it can be reached only via narrow pathways. They have proposed an algorithm to compute pockets in a protein. They have also proposed a method to measure properties of surface pockets such as volume and area. They have applied their method to discover the binding sites between molecules.

Word et al. [20] have proposed a method to measure the goodness-of-fit for molecular interfaces. They have described small-probe contact dots for measuring and visualizing the atomic contacts inside or between molecules. Their algorithm is similar to the Connolly's algorithm [1] for computing solvent-accessible molecular surfaces, in that a probe sphere is rolled over the spherical model of a molecule. The difference is that they leave a dot when the probe touches atoms of two molecules. Quantitative measure for goodness-of-fit is defined by the volume measured by the length between dots.

Wintner and Moallemi [19] have proposed the concept of *Quantized Surface Complementarity Diversity*, QSCD, for measuring complementarity between molecules. Diversity is defined as the measure of the difference, or similarity, between small molecules. They have defined a set of theoretical target surfaces that approximate all possible binding pockets with a volume limited by a predefined threshold. Each target surface is formed by cubic units carved out of the surface. These cubic units represent negative space that a potential ligand could occupy. To measure the complementarity of a molecule, the molecule is also quantized into a set of cubic units, and the quantized cubes are compared with the target surfaces. In this paper we discuss a shape complementarity definition based on the ratio of the negative volume to area of the interface between two molecules.

Several researchers have worked on analyzing and classifying molecular interfaces and interactions. Kuntz [10] has proposed strategies for drug design based on the structure of molecules. He finds possible docking sites by locating the grooves on the surface and creating their negative images by using spheres. Then he matches the ligand and the receptor by placing the ligand into the site using the isomorphic subgraph matching algorithms. Jones and Thornton [7] have analyzed protein complexes for better understanding of the principles of protein–protein interactions. They have defined the protein–protein interface based on the changes in the solvent-accessible surfaces when going from a monomeric to a dimeric state. They have examined structural properties of protein–protein interfaces as well as the biochemical properties. Specifically, they have measured the complementarity between surfaces using the *gap index*. The gap index is defined as the gap volume between molecules divided by the interface area. They calculate the gap volume using a method by Laskowski [11]. Laskowski defines the *gap sphere* as the largest sphere which can be placed between two atoms from each molecule without penetrating either of the molecular surfaces. He computes the gap volume by adding the volumes of all gap spheres. This is a

good estimate for the intermolecular volume, however: 1) gap spheres might overlap resulting in possible overestimation of the gap volume, 2) the gap spheres might not cover the entire intermolecular volume resulting in possible underestimation of the gap volume, 3) isotropic spheres might not be adequate to measure an anisotropic intermolecular region, and 4) this method could take quadratic time if it considers all possible pairs of atoms. Our approach computes the intermolecular volume accurately to any desired level of precision and runs in linear time.

Nadassy et al. [14] have measured how compactly atoms are packed in molecular interfaces compared to the internal spaces by computing the atomic volumes in double-stranded DNA and in protein–DNA interfaces. They have also used two measures for assessing packing in interfaces of macromolecular complexes: (a) the gap volume index as defined above, and (b) the *shape correlation index* by Lawrence and Colman [12]. The shape correlation index is derived from the distance between two points on the surfaces of interacting molecules and the angle between normal vectors of these points. This is an intuitive estimate for the shape complementarity and could be implemented to run in linear time with a good data structure for identifying nearest points. However, their metric is insensitive to the area of the intermolecular interface.

Varshney et al. [18] have proposed an analytic approach for computing and visualizing molecular interfaces in linear time. However, their primary goal is to visualize the interface surfaces, not the intermolecular negative volume. Also, they do not use the graphics hardware to accelerate the computation of the interface surfaces. To the best of our knowledge, no previous work has been done in interactively computing and visualizing intermolecular negative volume using linearly scalable algorithms with user-specifiable accuracy.

Domik and Fels [3] have developed a visual tool for studying molecular docking. Their system enables users to visually determine prospective binding sites by visualizing collision detections. Recently Olson et al. [15] have developed an augmented reality tool to study molecules. As the user rotates and translates a 3D printed replica of a molecule, their system tracks the molecule’s movements and mimics them in conjunction with a virtual molecule on the screen. This provides the users a compelling sense of the shape of a molecule and how it relates to other molecules. This can be valuably used in shape complementarity studies. Another powerful tool for protein visualization has been recently developed by Kreylos *et al.* [9]. Their tool allows the users to design proteins ab-initio using primary, secondary, and tertiary structures. Their tool also allows inverse kinematics and interactive visualization of proteins using a variety of motif visualizations as well as Ramachandran plots and intra-molecular collisions.

Interactivity is crucial for task-completion in 3D visualization applications. This has been proven by several researchers including Smets and Overbeeke [17] and Hawkes *et al.* [6]. To achieve interactivity while studying shape complementarity for a pair of molecules, we have developed an algorithm for computing and visualizing the intermolecular negative volume and the area of docking site using graphics hardware. Our work complements previous work on protein visualization in that it provides a new way to interactively visualize molecular interfaces.

3 Defining the Intermolecular Negative Volume

The smooth molecular surface, first proposed by Richards [16], is defined as the surface which an external probe sphere touches as it is rolled over the spherical atoms of a molecule. This representation is useful for studying interaction between molecules as it provides a smooth surface approximation to a molecule while retaining its most important shape features. Specifically, this surface representation is useful for studying shape complementarity since the two molecular surfaces are approximately coincident in the interfacial region [2]. We use the smooth solvent-accessible molecular surface to represent a molecule.

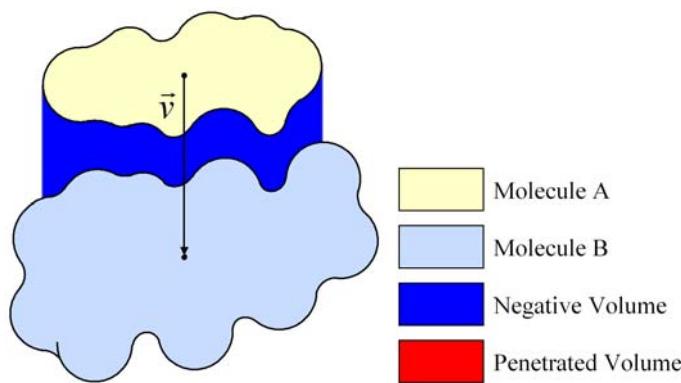


Fig. 2. Computing Intermolecular Negative Volume

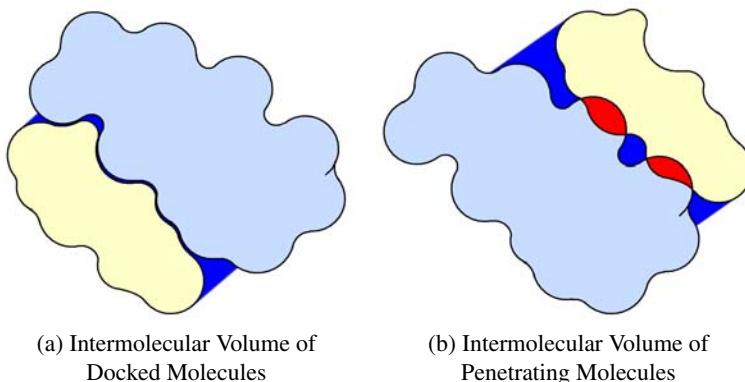


Fig. 3. Intermolecular Negative Volume

Given two molecules A and B , let the set of points on the molecular surfaces be represented by A_o and B_o respectively. Here the subscript o denotes that the points in these are defined in the object (world) coordinate system. Further, let the cardinalities of A_o and B_o be given by n and m , respectively. The centers of A_o and B_o are defined as the average of their respective surface points: $\mathbf{c}_A = \frac{1}{n} \sum_{j=0}^n \mathbf{x}_{oj}$ and $\mathbf{c}_B = \frac{1}{m} \sum_{k=0}^m \mathbf{x}_{ok}$, where $\mathbf{x}_{oj} \in A_o, \mathbf{x}_{ok} \in B_o$.

We define the *aligning direction* \mathbf{d} between molecules A and B as $(\mathbf{c}_B - \mathbf{c}_A)/|\mathbf{c}_B - \mathbf{c}_A|$. The aligning direction is the unit vector from the center of A to the center of B . We define the aligning direction this way only as a heuristic. Our system can accept user-defined aligning directions as well. We define two mutually orthogonal vectors \mathbf{u}, \mathbf{v} , perpendicular to the vector (d) to construct the *interface coordinate system*. In this coordinate system x axis is considered to be along \mathbf{u} , y along \mathbf{v} , and z along (d). Now consider an axis-aligned bounding box in the interface coordinate system that contains both molecules. We assume the origin lies at the center of that face of the bounding box which is parallel to the x - y plane and below the molecule A as shown in Fig. 4. This assumption makes all z values in the interface coordinate system positive. We shall use the subscript i to denote the interface coordinate system. Let the matrix to transform a point from the object coordinate system to the interface coordinate system be given by M_i . The point sets in the interface coordinate system A_i and B_i can be defined as $A_i = \{\mathbf{x}_i | \mathbf{x}_i = M_i \mathbf{x}_o, \mathbf{x}_o \in A_o\}$, and $B_i = \{\mathbf{x}_i | \mathbf{x}_i = M_i \mathbf{x}_o, \mathbf{x}_o \in B_o\}$.

Let $\mathcal{P}(A_i, \mathbf{d})$ and $\mathcal{P}(B_i, \mathbf{d})$ be the parallel projections of A_i and B_i onto the x - y plane of the interface coordinate system. Then, we define the *intersection of projected regions*, $P(A_i, B_i, \mathbf{d}) = \mathcal{P}(A_i, \mathbf{d}) \cap \mathcal{P}(B_i, \mathbf{d})$. This region is shown in Fig. 4 in dark green color between the two molecules.

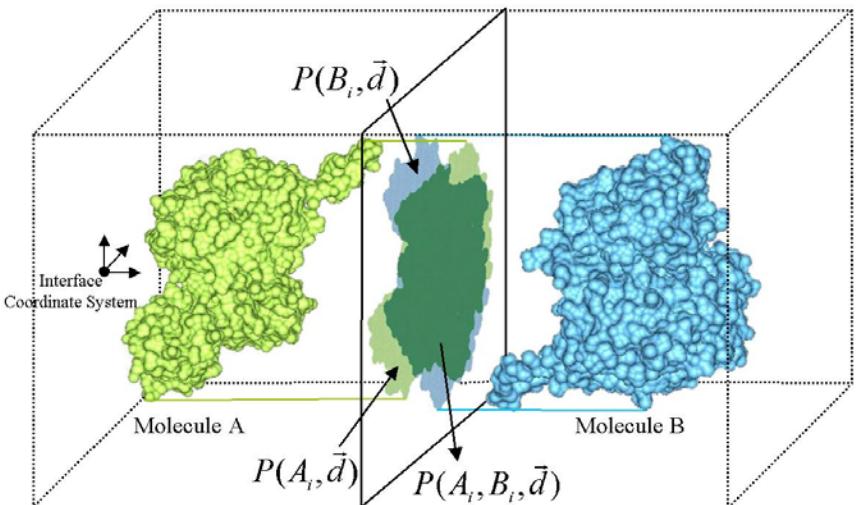


Fig. 4. The Interface Coordinate System and the Molecular Projection Regions

The docking region of a molecule is the region of the molecular surface which borders the intermolecular negative volume. We define the *docking region of A* as $R_A = \{(x_i, y_i, z_i) | (x_i, y_i) \in P(A_i, B_i, \mathbf{d}) \wedge z_i = \max\{z | (x_i, y_i, z) \in A_i\}\}$. Similarly, the *docking region of B* is defined as $R_B = \{(x_i, y_i, z_i) | (x_i, y_i) \in P(A_i, B_i, \mathbf{d}) \wedge z_i = \min\{z | (x_i, y_i, z) \in B_i\}\}$. The intermolecular negative volume between R_A and R_B is defined in the interface coordinate system as $V_i = \{(x_i, y_i, z_i) | (x_i, y_i) \in P(A_i, B_i, \mathbf{d}) \wedge z_i \in [R_A(x_i, y_i), R_B(x_i, y_i)]\}$, where $R_A(x_i, y_i) = \{z_i | (x_i, y_i, z_i) \in R_A\}$ and $R_B(x_i, y_i) = \{z_i | (x_i, y_i, z_i) \in R_B\}$.

The *intermolecular negative volume* between molecules A and B is transformed back to the object coordinate system as $V(A, B) = \{\mathbf{x}_0 | \mathbf{x}_0 = M_i^{-1} \mathbf{x}_i, \mathbf{x}_i \in V_i\}$. Figure 2 shows the intermolecular negative volume in blue.

4 Computing the Intermolecular Negative Volume

We compute the intermolecular negative volume using the graphics hardware. The graphics hardware depth-testing functionality can compute the distance from the viewing plane to the nearest (or the farthest) surface for each pixel in the viewing plane. For instance, in OpenGL the depth-testing option to select the surface closest to the viewing plane is *GL_LESS*, and the option to select the farthest surface is *GL_GREATER*. We can also change the level of detail of the interface by simply changing the resolution of the viewing plane.

The algorithm for computing intermolecular negative volume using depth buffer is as follows. First, we set the viewing direction as the aligning direction \mathbf{d} and draw molecule A with the depth-testing option to select the farthest depth coordinate. The depth buffer now contains the distance from the viewing plane to the farthest surface of A (the surface that defines one side of the intermolecular volume). The depth buffer is read-back and saved as D_A and the buffer is reset. Second, with the viewing direction still \mathbf{d} we draw molecule B with the depth-testing option to select the nearest depth coordinate. The depth buffer now contains the distance from the viewing plane to the nearest surface of B (which is the surface that defines the second side of the intermolecular volume). The depth buffer is again read back and saved this time as D_B .

In D_A , the region that lies outside the molecule has the largest possible value (1 in OpenGL), and in D_B , the region outside the molecule has the smallest possible value (0 in OpenGL). For each (x, y) value, if $D_A(x, y)$ is equal to 1 or $D_B(x, y)$ is equal to 0, we set both $D_A(x, y)$ and $D_B(x, y)$ to 0. This gives us the intersection of projected regions $P(A_i, B_i, \mathbf{d})$ and sets the remaining region to 0 in D_A and D_B . Now, D_A and D_B store R_A and R_B , respectively, which are the docking regions of A and B as defined in the previous section. Therefore, the volume between D_A and D_B is the intermolecular negative volume between A and B. The overview of our algorithm is shown in Fig. 5. To visualize the intermolecular negative volume, we build a triangle mesh using D_A and D_B . The method for triangulating the intermolecular negative volume is described in the next section.

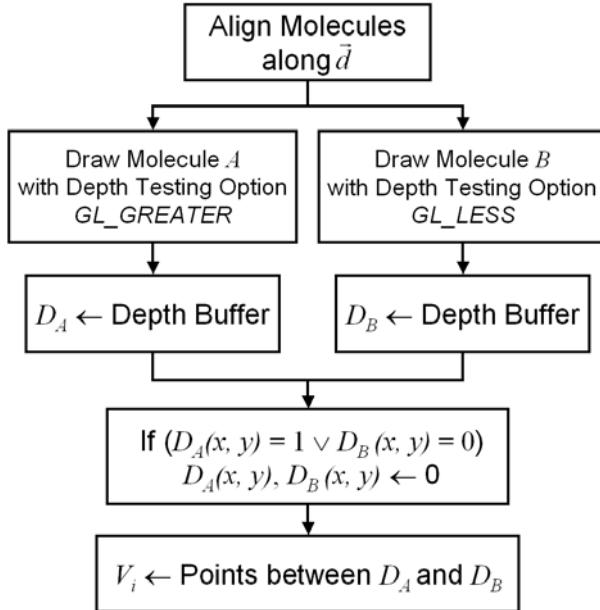


Fig. 5. Algorithm for Computing Intermolecular Negative Volume

5 Visualizing the Intermolecular Negative Volume

The intermolecular negative volume is the set of voxels between two 2D pixelated depth buffers, D_A and D_B . An obvious choice to extract an isosurface from volume data is the Marching Cubes algorithm [13]. However, we have additional information in this case that we can use. Since the non-zero regions of D_A and D_B are the same, we can reduce the marching cubes algorithm to its two-dimensional analog, the *Marching Squares* algorithm. For each vertex in the triangle mesh, we produce x and y coordinates using the marching squares algorithm and get the z coordinate from D_A and D_B .

There are six cases for the squares according to the values of four corner points. In Fig. 6, the white dots are points with zero values, and the black dots are points with non-zero values. For each square (pixel) of D_A , we repeat the following process. We create a gray point in the middle of an edge that connects a zero (white) point to a non-zero (black) point. The black and gray points are added to the vertex list of the resulting triangle mesh. The z coordinate of a black point is the value of D_A at the point, and the z coordinate of the gray point is same as that of the adjacent black point. We then generate triangles connecting black and gray dots, which are the shaded regions in Fig. 6. These triangles are part of the docking region of A . Similarly, we can produce triangles with same x and y coordinates and the z values from D_B , and these triangles form the docking region of B .

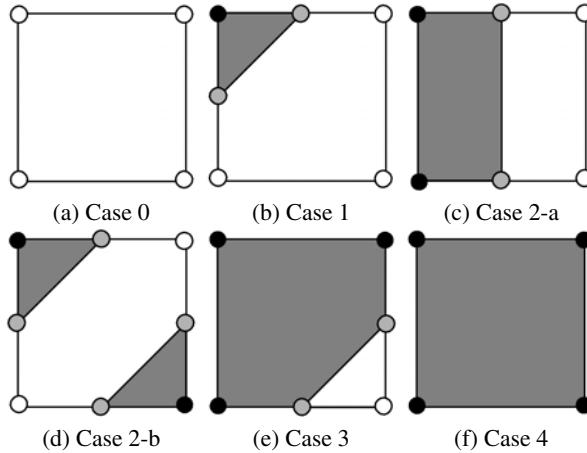
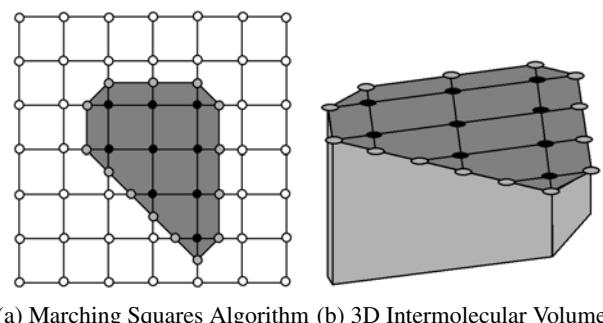


Fig. 6. Cases for building Triangle Meshes : *White dots* are zero points, *black dots* are non-zero points, and *gray dots* are middle points added by the algorithm. *Shaded regions* show the polygonal mesh

Next we connect the boundaries of the two surfaces representing docking regions of A and B . We define a *gray edge* as the edge formed by two adjacent gray points. The boundary of the docking region consists of gray edges. Therefore, we create two triangles to connect two gray edges with same x and y coordinates and different z coordinates from D_A and D_B . Figure 7 shows the construction of the mesh. Figure 8 shows the mesh of the intermolecular negative volume between the Proteinase and its Inhibitor in the Protein Data Bank complex 4SGB.



(a) Marching Squares Algorithm (b) 3D Intermolecular Volume

Fig. 7. A Mesh Construction Using Marching Squares Algorithm

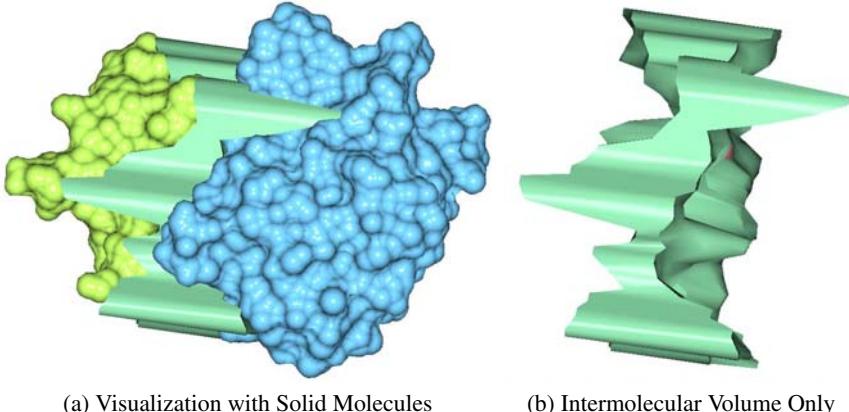


Fig. 8. Interactive Visualization of the Intermolecular Negative Volume between Proteinase and Inhibitor (4SGB)

6 Modifications for More Accurate Computation

6.1 Intermolecular Negative Volume with a Threshold

The algorithm for computing intermolecular negative volume described in Sect. 4 might not produce desirable results when the docking site is relatively small. The actual docking site might be smaller than the intersection of projected region as shown in Fig. 9. This problem causes the thick borders in the intermolecular negative volume that you can see in Fig. 8.

We have extended our algorithm to trim the thick borders that do not, in general, define the intermolecular volume. We exclude the regions where the distance between surfaces is greater than a certain threshold ε when we compute the intersection of projected regions. Specifically, we compute the difference of z values $Dist(x,y) = D_A(x,y) - D_B(x,y)$ for each (x,y) in the depth buffers D_A and D_B . Only if the absolute value of the difference is less than a threshold ε , $|Dist(x,y)| < \varepsilon$, we

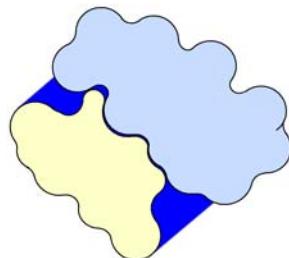


Fig. 9. Intermolecular Negative Volume with a Small Docking Site

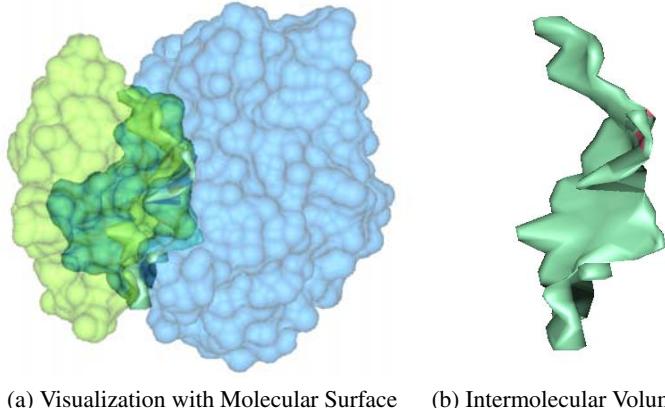


Fig. 10. The Intermolecular Negative Volume between Proteinase and Inhibitor (4SGB) with a threshold $\epsilon = 2.8 \text{ \AA}$

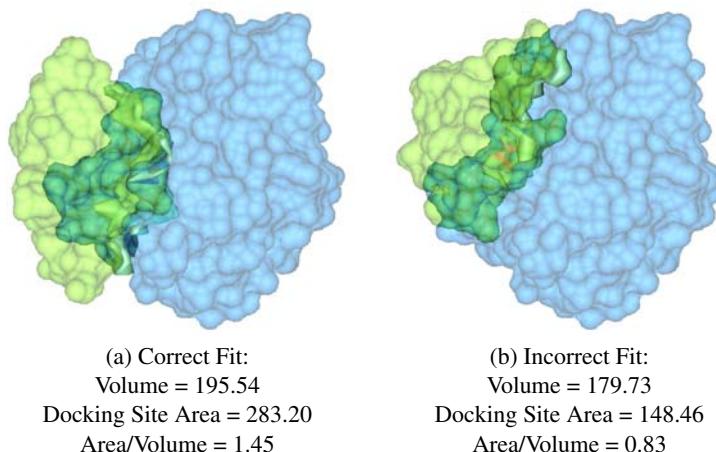
add (x, y) to the docking site. The rest of algorithm is same as described in Sect. 4. Figure 10 shows the visualization of the intermolecular negative volume modified from Fig. 8. We have currently set the ϵ to be the diameter of a water molecule, 2.8 \AA to make the intermolecular volume solvent inaccessible.

6.2 Computing the Area of the Docking Site

The intermolecular negative volume is often not enough to characterize the docking site. For instance, in Fig. 11, the correct fit (a) has larger area-volume ratio even though the incorrect fit (b) has smaller volume. We observe that the ratio of the area of the interface to the intermolecular volume is a much better heuristic than just using the intermolecular volume. We compute the area of the molecular interface by simply adding up the areas of the mesh triangles in the intermolecular volume that are defined by the surfaces of one of the two molecules A and B . The fit between molecules is considered good when the volume between them is small *and* the docking site area is large. We propose the ratio of the area of the interface and intermolecular volume as a criterion for characterizing the goodness-of-fit for protein docking as well as rational drug design. The larger this ratio, the better the fit between molecules.

7 Interactive Visualization

We have developed a 3D visualization tool for visualizing the intermolecular negative volume interactively, which can be used as a complementary tool to existing drug-design systems. Users can manipulate the molecules together or separately while the intermolecular negative volume is computed and rendered for every frame at interactive rates. Our system also provides various visualization options. The intermolecular

**Fig. 11.** Intermolecular Negative Volume between Proteinase and Inhibitor (4SGB)

negative volume can be visualized alone, or together with molecules as the images in this paper show. The molecules can be visualized in solid surfaces or translucent surfaces.

Table 1 shows the intermolecular volumes, the interface area contributions from the two molecules, and the ratio of the average interface area to the intermolecular volume for a number of naturally occurring molecular complexes. Table 2 shows the times for computing intermolecular negative volumes for the same molecular complexes. This includes the time to generate and render the triangle mesh for visualizing

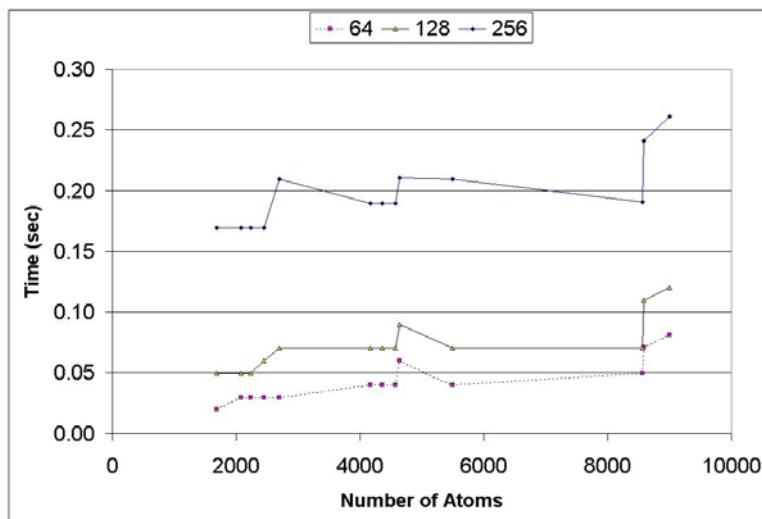
Table 1. Intermolecular Negative Volume and Interface Area

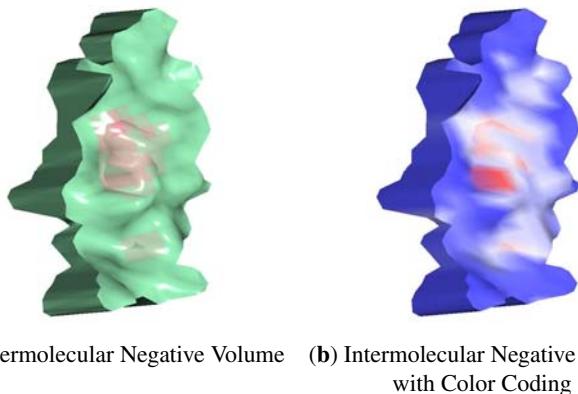
Complex Names	Volume (Å ³)	Interface Area (Å ²)	Ratio (Area/Volume)
Protease/Inhibitor	538.60	443.46 / 510.64	0.89
alpha-chymotrypsinogen/Trypsin inhibitor	539.75	602.76 / 651.60	1.16
beta-trypsinogen/Trypsin inhibitor	521.61	479.96 / 493.38	0.93
Subtilisin Novo/Chymotrypsin inhibitor 2	666.67	605.26 / 679.02	0.96
Subtilisin BPN/Subtilisin inhibitor	553.15	536.00 / 609.72	1.04
Barnase/Barstar	1224.65	1229.19 / 1171.00	0.98
Acetylcholinesterase/Inhibitor	955.03	837.64 / 810.42	0.86
Ribonuclease inhibitor/Ribonuclease A	536.34	406.89 / 401.39	0.75
IgG1 D44.1 Fab Fragment/Lysozyme	603.93	353.73 / 363.46	0.59
IgG1 E8 Fab Fragment/Cytochrome C	892.38	637.04 / 648.74	0.72
Antibody Hulysll Fv/Lysozyme	1674.65	1174.88 / 1140.83	0.69
CDK2 cyclin-dependant kinase 2/Cyclin	1534.82	1512.47 / 1705.81	1.05
Methylamine dehydrogenase/Amicyanin	2017.78	1517.69 / 1503.73	0.75

Table 2. Timing Information for Various Complexes

Complex Names	Number of Atoms	Time for 64 ³ (msec)	Time for 128 ³ (msec)	Time for 256 ³ (msec)
Protease/Inhibitor	1310/380	20	50	170
alpha-chymotrypsinogen/Trypsin inhibitor	1799/440	30	50	170
beta-trypsinogen/Trypsin inhibitor	1629/454	30	50	170
Subtilisin Novo/Chymotrypsin inhibitor 2	1938/513	30	60	170
Subtilisin BPN/Subtilisin inhibitor	1938/764	30	70	210
Barnase/Barstar	2581/2059	40	70	190
Acetylcholinesterase/Inhibitor	4116/460	40	70	190
Ribonuclease inhibitor/Ribonuclease A	951/3411	40	70	190
IgG1 D44.1 Fab Fragment/Lysozyme	4291/4291	60	90	211
IgG1 E8 Fab Fragment/Cytochrome C	3340/823	40	70	210
Antibody Hulysl Fv/Lysozyme	3488/2002	50	70	191
CDK2 cyclin-dependant kinase 2/Cyclin	4796/4202	71	110	241
Methylamine dehydrogenase/Amicyanin	4280/4281	81	120	261

the intermolecular volume. We have computed the intermolecular volume at various resolutions: 64³, 128³, and 256³. The computing time of our algorithm is linearly related to the number of atoms and runs at interactive rates as shown in Fig. 12. We have obtained these results on a Dell Precision Workstation with 1.5 GHz Pentium 4, 1 GB RAM, and a nVidia GeForce FX 5900 graphics card.

**Fig. 12.** Time for Computing Intermolecular Negative Volumes



(a) Intermolecular Negative Volume **(b)** Intermolecular Negative Volume
with Color Coding

Fig. 13. Color Coding of Intermolecular Negative Volume between Proteinase and Inhibitor (4SGB). The *red* color shows overlap between the two molecules of the 4SGB complex

In the study of intermolecular negative volumes, the variation of the distance between the two molecular surfaces is also important in addition to aggregate information such as area and volume. We have encoded the distance information into the color of the intermolecular negative volume as shown in Fig. 13. Deep blue shows larger interface distance and light blue shows a smaller interface distance.

8 Conclusions and Future Work

Our algorithm for computing intermolecular negative volume and the area of the molecular interfaces can be used for computing criteria for shape complementarity. Our 3D visualization tool for visualizing intermolecular negative volume interactively can be used as a complementary tool for existing protein docking and rational drug design systems. The visualization tool can be used to select the best fit among the candidates, and to improve the result by moving the molecules interactively to produce a better fit. Currently, we have only considered the geometric shape complementarity as a way of characterizing the goodness-of-fit between two molecules. Shape complementarity is an important criterion, but not the only one. It would be very helpful to include other criteria such as electrostatics, hydrophobicity, hydrogen bonding in developing a comprehensive metric for characterizing good molecular interfaces that can be used in protein docking and rational drug design applications.

Acknowledgements

We would like to thank Sergei Sukharev at the Department of Biology at the University of Maryland, Ron Unger at the Bar-Ilan University in Israel, and John Moult

at the Center for Advanced Research in Biotechnology at Rockville, Maryland for valuable discussions and for providing the data sets for testing. This work has been supported in part by the NSF grants: IIS 00-81847, CCF 04-29753, and CNS 04-03313.

References

1. M. L. Connolly. Solvent-accessible surfaces of proteins and nucleic acids. *Science*, 221:709–713, 1983.
2. M. L. Connolly. Shape complementarity at the hemoglobin a1b1 subunit interface. *Biopolymers*, 25:1229–1247, 1986.
3. G. Domik and G. Fels. HotDock: An interactive approach to molecular docking, 1996. <http://www.uni-paderborn.de/lst/HotDock/>.
4. H. Edelsbrunner, M. A. Facello, and J. Liang. On the definition and the construction of pockets in macromolecules. *Discrete Applied Mathematics*, 88(4):83–102, 1998.
5. H. Edelsbrunner, J. Liang, and C. Woodward. Anatomy of protein pockets and cavities: Measurement of binding site geometry and implications for ligand design. *Protein Science*, 7(9):1884–1897, 1998.
6. R. Hawkes, S. Rushton, and M. Smyth. Update rates and fidelity in virtual environments. *Virtual Reality: Research, Applications and Design*, 1(2):46–51, 1995.
7. S. Jones and J. M. Thornton. Principles of protein–protein interactions. In *Proc. Natl. Acad. Sci. USA*, volume 93, pp. 13–20, 1996.
8. E. Katchalski-Katzir, I. Shariv, M. Eisenstein, A. A. Friesem, C. Aflalo, and I. A. Vakser. Molecular surface recognition: determination of geometric fit between proteins and their ligands by correlation techniques. In *Proceedings of the National Academy of Sciences of United States of America*, volume 89, pp. 2195–2199, March 1992.
9. O. Kreylos, N. L. Max, B. Hamann, S. N. Crivelli, and E. W. Behel. Interactive protein manipulation. In *Proceedings of the IEEE Visualization*, pp. 581–588, Seattle, Washington, October 2003.
10. I. D. Kuntz. Structure-based strategies for drug design and discovery. *Science*, 257:1078–1082, 1992.
11. R. A. Laskowski. SURFNET: a program for visualizing molecular surfaces, cavities, and intermolecular interactions. *Journal of Molecular Graphics*, 13(5):323–330, 1995.
12. M. C. Lawrence and P. M. Colman. Shape complementarity at protein/protein interfaces. *Journal of Molecular Biology*, 234(4):946–950, 1993.
13. W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3d surface construction algorithm. In *Proceedings of SIGGRAPH*, pp. 163–169, July 1987.
14. K. Nadassy, I. Tomas-Oliveira, I. Alberts J., Janin, and S. J. Wodak. Standard atomic volumes in double-stranded DNA and packing in protein–DNA interfaces. *Nucleic Acids Research*, 29(16):3362–3376, 2001.
15. A. Olson. Tangible interfaces for molecular biology. In *Demos at the IEEE Visualization*, page D12, Seattle, Washington, October 2003.
16. F. M. Richards. Areas, volumes, packing and protein structures. In *Annual Review of Biophysics and Bioengineering*, volume 6, pp. 151–176, 1977.
17. G. J. F. Smets and K. J. Overbeeke. Trade-off between resolution and interactivity in spatial task performance. *IEEE Computer Graphics and Applications*, 15(5):46–51, 1995.

18. A. Varshney, F. P. Brooks Jr., D. C. Richardson, W. V. Wright, and D. Manocha. Defining, computing, and visualizing molecular interfaces. In *IEEE Visualization*, pp. 36–43, October 1995.
19. E. A. Wintner and C. C. Moallemi. Quantized surface complementarity diversity (QSCD): A model based on small molecule-target complementarity. *Journal of Medicinal Chemistry*, 43:1993–2006, 2000.
20. J. M. Word, S. C. Lovell, T. H. LaBean, H. C. Taylor, M. E. Zalis, B. K. Presley, J. S. Richardson, and D. C. Richardson. Visualizing and quantifying molecular goodness-of-fit: Small-probe contact dots with explicit hydrogen atoms. *Journal of Molecular Biology*, 285(4):1711–1733, 1999.

Optimized Bounding Polyhedra for GPU-Based Distance Transform

Ronald Peikert and Christian Sigg

Swiss Federal Institute of Technology, CH-8092 Zürich, Switzerland
peikert@inf.ethz.ch, sigg@inf.ethz.ch

Many problems in areas such as computer graphics, scientific visualization, computational geometry, or image processing require the computation of a distance field. The distance field indicates at each point in space the shortest distance to a given object. Depending on the problem setting, the object is described either by a voxel attribute within a volume data set or by a surface representation such as a triangle mesh. The two cases require separate approaches, and only the case of the triangle mesh is studied in this paper. Often, the distance field is needed as a regular grid of samples. The samples can be computed either in image space or object space, referring to the outer loop of the algorithm, which iterates over all samples or all triangles of the mesh, respectively. Object space methods can be competitive, especially for higher resolutions. An ideal object space method would compute a generalized Voronoi diagram (GVD) of the mesh and then scan convert its cells. At each sample location, the distance to the Voronoi site associated with the cell would yield the field value. A practical method however, avoids the expensive GVD computation and instead works with bounding polyhedra for the Voronoi cells. In this paper, we propose a new type of bounding polyhedra. This reduces the number of polyhedra and simplifies their geometry. The choice of these bounding polyhedra pays off especially if scan conversion is run on graphics hardware.

1 Introduction

For any set S of points in \mathbb{R}^n , the distance field u is a unique scalar function defined in \mathbb{R}^n . At each point, u equals the distance to the closest point on S . If S is a closed and orientable manifold of dimension $n - 1$, the space is divided into inner and outer parts. Therefore, a signed distance field can be defined. A positive sign is chosen outside the surface and a negative sign inside. Thus, the gradient of the distance field on the surface is equivalent to the surface normal.

The type of distance metric which is chosen depends on the application. Common choices are chessboard, chamfer and Euclidean distance [14]. We will restrict

ourselves to the Euclidean distance, which is probably the most meaningful, but it is also the most expensive to compute.

The signed distance field u is the solution to the Eikonal equation $|\nabla u| = 1$ with boundary condition $u|_S = 0$. The boundary condition shows that the definition of S as a subset of \mathbb{R}^n and the signed distance function are equivalent descriptions. The manifold corresponds to the zero-set of the signed distance function: $S = \{x|u(x) = 0\}$. Therefore, the signed distance transform converts an explicit surface representation to an implicit one.

Signed or unsigned distance fields have many applications in computer graphics, scientific visualization and related areas, such as implicit surface representation [6] [7], object metamorphosis [3], collision detection and robotics [6], skeletonization [1] [19], accelerated volume raytracing [17], camera path planning and image registration [4]. Depending on the application, the distance field is required on a full pixel or voxel grid or only within a band of width d around the objects.

2 Related Work

The problem of computing a 3D Euclidean distance transform exists in two varieties, distinguished by the type of object representation. The object can either be given as data on a voxel grid or in vector representation. The latter is typically a triangle mesh in the case where the object is a surface. Both problems have been studied extensively and fast methods have been developed for both of them. It is reasonable to treat the two problems separately. If the goal is to sample the exact distance to a triangle mesh, the problem cannot be stated in voxel space. Likewise, there is usually no advantage to transform the problem from voxel representation to vector representation. For triangle meshes, time complexity must depend on the number M of surface primitives (faces, edges, and vertices). Therefore, algorithms for the two different problem settings cannot be directly compared.

A method [12] which has been recently presented, finds the distance transform in voxel data in $O(N)$ time, where N is the number of voxels. In the same paper, a good overview of earlier methods is given. Essentially, methods fall into two categories, propagation methods and methods based on Voronoi diagrams.

In propagation methods, the distance information is carried over to neighbor voxels, either by sweeping in all grid dimensions, or by propagating a contour. A well-known example of the latter is the Fast Marching Method (FMM) [15], an upwind scheme which can solve the Eikonal Equation $|\nabla u| = 1/f$ in a single iteration and in $O(N \log N)$ operations. A signed distance field is obtained by using a constant propagation function f . However, due to the finite difference scheme, FMM is not an exact method.

Besides the distance, additional information can be stored in the distance field. Such information can be the vector pointing to the nearest object point, known as the vector distance transform [13]. Alternatively, the index of the nearest surface primitive can be attributed to each point, the resulting field is called a complete distance field representation [9]. By propagating this type of additional information, FMM

and similar propagation methods can be turned into exact distance transform algorithms [2, 5, 18].

For the second type of problem setting where the distance field of a polyline or triangle mesh is sought, a brute force algorithm would compute the distance of each grid point to each primitive. If the triangle mesh consists of a large number of triangles and the sampling grid is large, this approach is impractical. For an efficient algorithm, one needs to reduce the number of distances calculated per grid point or alternatively, per primitive.

To achieve this goal, a spatial data structure such as a BSP tree can be used for storing the primitives. When computing the distance field value for a given sample, a primitive can be excluded from the calculation if it is known that a closer primitive exists. By using this data structure, one can quickly find the closest primitive to a point: While the tree is scanned for the closest primitive, one can give an upper limit of the final distance. At the same time, a lower bound of the distance can be computed for any subtree. If the lower bound of a subtree is larger than the current upper bound of the final distance, the subtree can be excluded from the search. This leads to an algorithm logarithmic in the number of primitives of the input mesh.

An alternative to such an image space approach are object space methods, i.e. methods based on scan conversion. Here, the distance field is obtained by scan converting a number of polyhedra related to the triangle mesh and by conditionally overwriting the computed voxel values. The advantage of object space methods is their sub-pixel accuracy. However, it is obvious that the relative performance degrades if the average size of the polyhedra shrinks to the size of a single voxel. It has been shown that for distance fields of triangle meshes, methods based on scan conversion are competitive.

Optimally, only distances to grid points contained in the Voronoi cell of the corresponding primitive are calculated. If the primitives are not restricted to points, but include edges and triangles, a generalized Voronoi diagramm (GVD) is required. Once a GVD is computed, the distance field can easily be caluculated as the distance to the respective site. However, the computation of Voronoi diagrams is not easier than the computation of distance fields. The time for generating a diagram with M sites is $O(M \log M)$. Generalized Voronoi diagrams can be computed on graphics hardware [8] by rendering local distance fields as $n + 1$ -dimensional function graphs and using the z-buffer for minimization. A disadvantage of this method is that it requires accurate rendering of curved surfaces, requiring tessellations in the order of 100 triangles for a cone. The 3D version even requires doubly curved surfaces which strongly limits the number of primitives that can be handled.

Nevertheless, if a point is known to lie outside of a Voronoi cell, the distance to its base primitive does not need to be calculated. This led to the idea of using polyhedra bounding a Voronoi cell instead of the Voronoi cells themselves. The first such algorithm was presented by Mauch [11]. It will be shortly explained in Sect. 3.3. It is possible to run the scan conversion part of that algorithm on graphics hardware. Although the scan conversion runs faster on the graphics hardware, the overall speed up gained is minimal because of the large amount of geometry that has to be send

to the graphics card. In Sect. 3.4, we present an optimized type of polyhedra [16] giving significantly better performance on the graphics hardware.

3 Distance Field Methods for Triangle Meshes

For the rest of the paper, we focus our attention to the distance field computation for a triangle mesh. We shortly describe the more theoretical method based on scan conversion of the generalized Voronoi cells. We also outline Mauch’s method of using bounding polyhedra. In Sect. 3.4, we describe how the method can be made more suited to be run on a programmable graphics card by using an optimized type of bounding polyhedron.

3.1 Vertex Classification

The vertices of a closed and oriented triangle mesh can be classified into convex, concave and saddle vertices, depending on their incident edges. If all of them are convex (concave), the vertex is convex (concave), if both types occur, it is a saddle. Because convex edges become concave and vice-versa when we flip the orientation of the surface, we only distinguish convex/concave vertices and saddle vertices.

Besides the topological consistency, we assume also a geometric regularity requirement for mesh: At saddle points, all incident faces must keep their orientation when viewed from the normal direction. The normal direction in a vertex is defined simply by averaging all incident face normals. Failure of this assumption would indicate a poor triangulation. It can be fixed by subdividing triangles.

3.2 Voronoi Diagrams of Triangle Meshes

The Voronoi diagram of a finite set of points is a partitioning of space into cells. Each site (i.e. point) is associated with a cell containing all points for which this site is the nearest one. Points on cell boundaries have more than one nearest site.

A straightforward extension is to allow sites to be manifolds than just points, leading to generalized Voronoi diagrams. For our purpose, sites will be restricted to the points, edges and faces of a closed and oriented triangle mesh.

If the GVD of such a mesh was known, it would be a simple task to compute the distance field. For a given sample point, one would first identify the cell in which it is contained and then calculate the distance to the associated site. If a full grid of samples is needed, one would use an object-space approach, i.e. loop over the cells, which is known as scan conversion.

In computer graphics, scan conversion is a key operation in the rendering pipeline and is efficiently performed by standard graphics cards. By reading back the frame buffer data, the computing power of graphics cards becomes available for more purposes than just rendering. In recent years, the programmability of graphics cards made it possible to adapt the scan conversion operation. In particular, nonlinear interpolation functions can be programmed.

3.3 GVD-based Bounding Polyhedra

To actually compute a GVD is not only expensive, it can also produce arbitrarily complex polyhedra. Therefore, Mauch [11] replaced the cells by bounding volumes. As bounding volumes he used polyhedra which are possibly larger but of simpler geometric shape than the cells. The distance field can again be calculated by looping over the polyhedra. To correctly treat regions where two or more polyhedra overlap it is sufficient to take the minimum of all computed values.

For reasons of efficiency, only local information is used for constructing bounding polyhedra. This requires the introduction of a maximal distance d up to which the distance field is computed on either side of the surface. The following types of bounding polyhedra are used, depending on the type of site

- three-sided orthogonal prism for faces (“tower” of height $2d$ extruded from the triangle in both directions),
- three-sided orthogonal prism for edges (“wedge” of height d , filling the space between towers)
- n -sided pyramid for convex/concave vertices of degree n (of height d , filling the space left by towers and wedges, see Fig. 1).

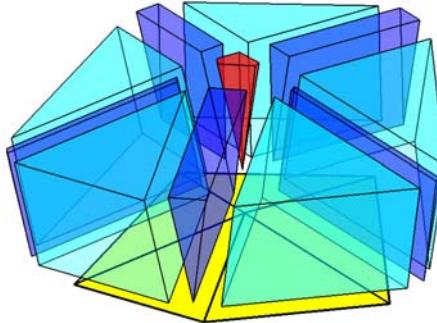


Fig. 1. Polyhedra constructed on one side of a (yellow) one-ring of the mesh: (cyan) towers, (blue) wedges, and a (red) cone. The polyhedra are moved away from the surface for better visibility

The case of the saddle vertex is not mentioned in [11]. However, a possible solution would be to construct an n -sided pyramid in the same way as for a convex/concave vertex, but on both sides of the surface, and then taking the convex hull of each pyramid.

3.4 Optimized Bounding Polyhedra

While these bounding polyhedra work well for scan conversion done purely in software, the large number of polyhedra is not ideal for a hardware-based scan

conversion method. The reason is that the overhead per polyhedron is larger for the hardware-based method because the geometry data has to be sent to the graphics card.

In order to reduce the number of polyhedra, our approach uses a different type of bounding polyhedra for the faces, such that their union completely covers space. This eliminates the need for wedges and pyramids. The price to pay is a slightly more complicated distance field computation: Each polyhedron no more represents a single site, but seven sites, namely a face, its three boundary edges, and its three vertices. In principle, the minimum of the distances to the seven sites must be calculated. However, we showed in [16] that this can be done quite efficiently, requiring little more operations than for a single distance calculation.

Now, in order to construct the new bounding polyhedra, we must divide wedges and pyramids among the neighbor polyhedra. Two observations can be made:

- Any way of splitting a wedge is allowed because the Voronoi site which it represents is also represented by the new polyhedra.
- On the convex side of the surface, i.e. opposite the wedge, the original towers overlap. This overlap can be eliminated by using the bisector plane of the dihedral angle as a divider. This is exactly where the two Voronoi cells meet.

Taking both observations into account, we can now use the angle bisector planes as the three lateral boundaries of the new polyhedron. Adding two planes parallel to the face at distances $\pm d$ (the limiting distance used for the distance field computation), a three-sided pyramid frustum is obtained (see Fig. 2). This bounding polyhedron has the advantage of having a single topological type and only five faces, all of them planar.

While the bounding polyhedra match along the edges of the mesh, this is not true near the mesh vertices in general. Near mesh vertices, the polyhedra can overlap. This is not a problem, it just leads to repeated distance calculations. But the polyhedra can

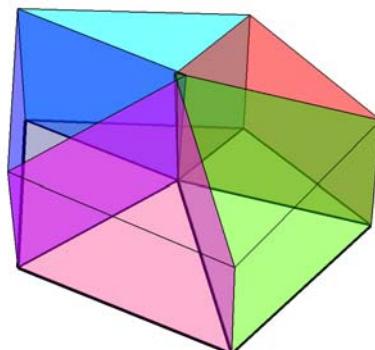


Fig. 2. Optimized bounding polyhedra for a one-ring of the mesh. The polyhedra extend to the other side of the surface, too, which is not shown in this figure. Some non-adjacent pairs of polyhedra are seen to overlap

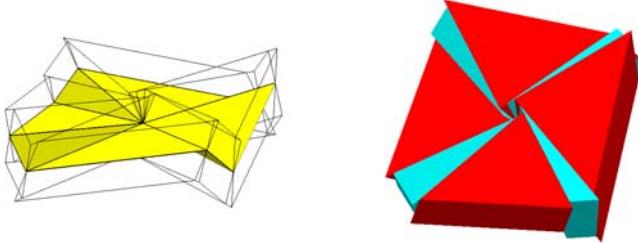


Fig. 3. Example of a saddle vertex with eight incident triangles. Bounding polyhedra are outlined (*left*) and filled (*right*). A gap in the shape of an eight-sided double-pyramid is visible in the center

also leave a gap. An example is shown in Fig. 3. In such cases, the gap must be closed by making some of the polyhedra slightly larger. We show in the appendix, that gaps can occur only for saddle vertices.

In order to study the situation near a mesh vertex, we introduce a few notations, see Fig. 4. Let c denote the vertex, x_0, \dots, x_{n-1} its neighbor vertices, $F_i = \langle c, x_i, x_{i+1} \rangle$ the incident faces (all indices are meant modulo n), and P_i the polyhedron constructed for the face F_i . That means that P_{i-1} and P_i are separated by the angle bisector plane of F_{i-1} and F_i which we denote by A_i . We denote by F the union of the F_i and by P the union of the P_i (for $i = 0, \dots, n-1$).

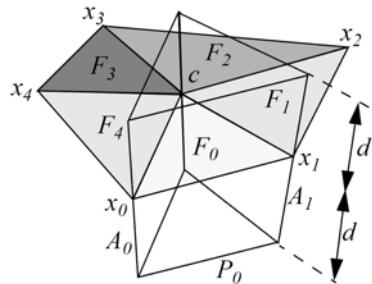


Fig. 4. A vertex c with neighbor vertices x_i , faces F_i , angle bisector planes A_i , and polyhedra P_i

If P completely covers a neighborhood of c , this means that any test point y near c is contained in at least one of the P_i . The point y is contained in P_i if it lies on the right hand side of A_i and on the left hand side of A_{i+1} . We also observe that in the reverse case (i.e. left of A_i and right of A_{i+1}), the antipodal point $2c - y$ is contained in P_i . Because in the cycle $A_0, A_1, \dots, A_n = A_0$ there are as many left-right transitions as right-left transitions, it follows, perhaps surprisingly, that the covering is point-symmetric w.r.t. the center c . The point symmetry holds for the multiplicity of the covering, not for each single polyhedron.

Therefore, if we can verify, that y lies neither on the left hand side of all A_i nor on the right hand side of all A_i , it follows that both y and its antipodal point are covered by P . For the practical test, it is sufficient to use one point on each intersection line $A_i \cap A_{i+1}$. Each test point must lie on the left of at least one A_j with $j \neq i$ and on the right of at least one such. Points lying exactly on a plane should pass the test, too. Also, it has to be noted that full planes can be used for the test, thus there is no need to bother with half-planes.

If the test fails for some of the test points, this means that the corresponding polyhedra must be made larger to avoid a gap. A possible way to do this is to take the centroid of the test points. Polyhedra must be enlarged just as much that they contain this centroid and its antipodal point. We want the polyhedra to remain pyramid frusta, therefore we restrict the modifications to parallel shifts of edges.

4 Results

In Sect. 4.1, the amount of vertices where the bounding polyhedra leave holes is analysed. In Sect. 4.2, the performance results of our algorithm are presented.

4.1 Saddle Vertices and Gaps

By looking at a few typical triangle meshes, it can be noticed that there are often more saddle vertices than convex/concave vertices. This can be caused by the geometry itself, but also by the triangulation. Especially, if a quadrilateral mesh is subdivided to a triangle mesh, the diagonals can turn a convex vertex into a saddle vertex. This is why the torus mesh has more than the expected 50% of saddle vertices.



Fig. 5. Datasets used for experiment

As mentioned, saddle vertices can lead to gaps between the bounding polyhedra and the extra effort to fill them. However, our experiment showed that gaps occur only for some of the saddle vertices. Depending on the mesh characteristics, the percentage of saddle vertices leading to gaps can be quite small (see Table 1).

4.2 Performance of the Hardware-based Distance Transform

The way to make use of the GPU's computing power is to work on layers of sample locations. Each layer defines a slicing plane which is intersected with the bounding

Table 1. Number of vertices, saddle vertices and vertices with incomplete covering by the unmodified bounding polyhedra

Mesh	Vertices	Saddles	Gaps
sphere6	16386	0	0
torus	3000	1788	0
knot	1440	1378	674
seashell	915	843	148
bunny	34834	30561	516

polyhedra. A list of active edges is used to avoid empty intersections. The resulting slices are sent to the GPU for scan conversion, where a fragment program is used to compute the local distance field. The local distance field is the distance field of only the seven sites (one triangle, three edges, three vertices) associated with the bounding polyhedron. The final distance field is obtained at each fragment by taking the minimum of the computed local distance field values. An outline of the fragment program can be found in [16].

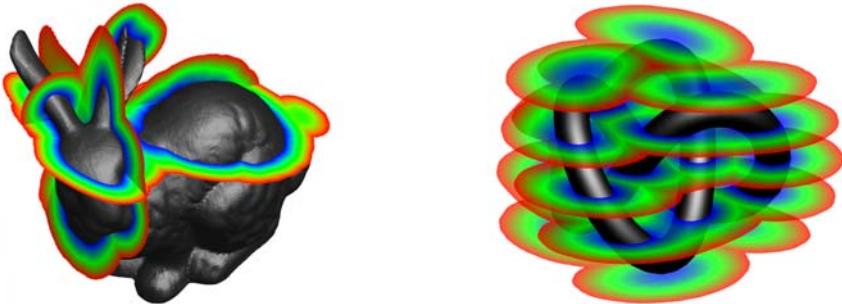


Fig. 6. Slices of the distance fields of bunny and knot data set, computed by the HW-based algorithm

As a basis for comparison of performance, we used the software scan conversion algorithm, which we downloaded from the URL [10]. We then re-implemented this algorithm such that the scanconversion part was done on the GPU. The machine at our disposition was a 2.4 GHz Pentium 4 equipped with 2 GB of RAM and an ATI Radeon 9700 PRO graphics card. It turned out only a negligible speedup could be obtained by this hardware-based program. In addition, the range of parameters (resolution, width of computational band) where a speedup could be measured, was rather narrow. This performance problem could be tracked down to the overhead caused by rendering too many small polygons.

When using our optimized bounding polyhedra, the speedup delivered on the same machine was significant for a wide range of resolutions and widths. When choosing a band of 10% of the model extent and a resolution of 256^3 samples, we measured an average speedup close to 5 for the sphere6, knot and bunny models.

For higher resolution as well as for wider bands, the speedup improved. But also for extremely low sample grid resolutions, the hardware-assisted program performed well. For instance, in the case of a mesh with 131072 triangles of average area less than 2 on the voxel scale, we measured a speedup of 3.30. However, it is obvious that the scan conversion approach, with or without hardware support, is no more an efficient strategy if sampling density is further decreased. For such problems, an image space method combined with a spatial data hierarchy would obviously be more adequate.

The advantage of the scan conversion approach also degrades when the narrow band is large in comparison to the volume the surface encloses. Because the bounding polyhedra for one triangle is computed using its neighboring triangles only, the bounding volumes tend to overlap on the convex side of the surface. The amount of overlap grows superlinearly with the thickness of the narrow band. In order to compute the distance transform in a dense volume around the surface, the fastest solution would be a combination the CSC and the FMM approach. While the CSC algorithm is faster in computing the distance in the narrow band, the FMM algorithm can then be used to compute the distance in regions further away from the surface.

5 Conclusion

We have shown that today's graphics hardware is suitable for supporting the signed distance field computation. A GPU implementation has a larger overhead per polyhedron while sampling the distance field using scan conversion is faster. By reducing the amount of polyhedra to approximately one third, we were able to get a significant speed up in comparison to the CPU implementation. It was proven that the polyhedra cover the area around triangles, edges and convex or concave vertices up to a user definable distance. However, the polyhedra can leave a hole in special configurations at saddle vertices. These holes are filled by shifting the sides of the polyhedra outward until they cover the normal of the saddle vertex. This procedure increases the amount of overlap and therefore introduces a certain overhead. But it was shown that gaps don't appear very often for common meshes. For an implementation using graphics hardware, the speed up gained by the reduced amount of geometry outweighs the extra cost of additional distance samples.

Acknowledgment

This work was partially funded by Schlumberger Cambridge Research.

References

1. Ingmar Bitter, Arie E. Kaufman, and Mie Sato. Penalized-distance volumetric skeleton algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):195–206, 2001.

2. D. Breen, S. Mauch, and R. Whitaker. 3D scan conversion of csg models into distance, closest-point and colour volumes. In M. Chen, A.E. Kaufman, and R. Yagel, editors, *Volume Graphics*, pp. 135–158, 2000.
3. Daniel Cohen-Or, Amira Solomovici, and Levin Levin. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141, April 1998. ISSN 0730-0301.
4. Olivier Cuisenaire. *Distance Transformations: Fast Algorithms and Applications to Medical Image Processing*. PhD thesis, Université Catholique de Louvain, Louvain-La-Neuve, Belgium, January 1999.
5. Hinnik Eggers. Two fast Euclidean distance transformations in Z^2 based on sufficient propagation. *Computer Vision and Image Understanding: CVIU*, 69(1):106–116, January 1998.
6. Sarah F. Frisken, Ronald N. Perry, Alyn Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In Kurt Akeley, editor, *Siggraph 2000 Proceedings*, pp. 249–254. ACM SIGGRAPH, 2000.
7. Sarah F. F. Gibson. Using distance maps for accurate surface reconstruction in sampled volumes. In *Proceedings of the 1998 Symposium on Volume Visualization (VOLVIS-98)*, pp. 23–30, New York, October 19–20 1998. ACM Press.
8. Kenneth Hoff, Tim Culver, John Keyser, Ming Lin, and Dinesh Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. In Alyn Rockwood, editor, *Siggraph 99 Proceedings*, pp. 277–286, N.Y., August 8–13 1999. ACM SIGGRAPH.
9. Jian Huang, Yan Li, Roger Crawfis, S.C. Lu, and Shu Liou. A complete distance field representation. In Thomas Ertl, Ken Joy, and Amitabh Varshney, editors, *Proceedings Visualization 2001*, pp. 247–254. IEEE Computer Society Technical Committee on Visualization and Graphics Executive Committee, 2001.
10. Sean Mauch. A fast algorithm for computing the closest point and distance transform, 2000.
<http://www.acm.caltech.edu/~seanm/projects/cpt/cpt.html>.
11. Sean Mauch. *Efficient Algorithms for Solving Static Hamilton-Jacobi Equations*. PhD thesis, Caltech, Pasadena CA, April 2003.
12. Calvin R. Maurer, Jr., Rensheng Qi, and Vijay Raghavan. A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(2):265–270, 2003.
13. James C. Mullikin. The vector distance transform in two and three dimensions. *Computer Vision, Graphics, and Image Processing. Graphical Models and Image Processing*, 54(6):526–535, November 1992.
14. A. Rosenfeld and J. L. Pfalz. Distance functions on digital pictures. *Pattern Recognition*, 1:33–61, 1968.
15. J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proc. Nat. Acad. Sci.*, 94:1591–1595, 1996.
16. Christian Sigg, Ronald Peikert, and Markus Gross. Signed distance transform using graphics hardware. In R. Moorhead, G. Turk, and J. van Wijk, editors, *Proceedings of IEEE Visualization '03*, pp. 83–90. IEEE Computer Society Press, October 2003.
17. M. Sramek and A. Kaufman. Fast ray-tracing of rectilinear volume data using distance transforms. In Hans Hagen, editor, *IEEE Transactions on Visualization and Computer Graphics*, volume 6 (3), pp. 236–252. IEEE Computer Society, 2000.
18. Yen-hsi Richard Tsai. Rapid and accurate computation of the distance function using grids. Technical report, Dept. of Mathematics, University of California, Los Angeles, 2000.

19. Ming Wan, Frank Dachille, and Arie Kaufman. Distance-field based skeletons for virtual navigation. In Thomas Ertl, Ken Joy, and Amitabh Varshney, editors, *Proceedings of the Conference on Visualization 2001 (VIS-01)*, pp. 239–246, Piscataway, NJ, October 21–26 2001. IEEE Computer Society.

Appendix: Proof for Complete Covering Around Convex/Concave Vertices

Let c be a convex/concave vertex of a closed and oriented triangle mesh. Using the notation of Sect. 3.4, we want to show that a small neighborhood of c is completely covered by the union P of polyhedra. This can be seen best by taking intersections with a small sphere S centered at c (see Fig. 7). We use the overbar symbol to denote the intersection with S . It follows that \bar{F}_i are great circles, and their union \bar{F} is a convex spherical polygon. Also the \bar{A}_i are great circles, which can be oriented consistently towards the interior of \bar{F} . Finally, \bar{P}_i are spherical lunes, because we can assume that the diameter of S is smaller than all edges.

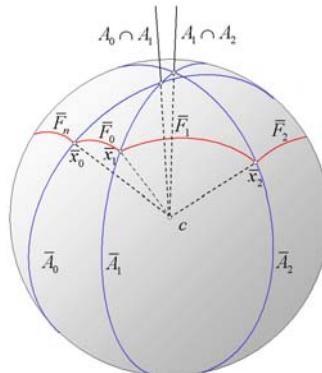


Fig. 7. Intersections of faces and angle bisector planes with the sphere

The conjecture is now that S is completely covered by \bar{P} , the union of the lunes. By the argument given in Sect. 3.4, it is sufficient to show that the northern hemisphere is covered.

Let y be a test point on S and on the convex side of the surface, i.e. an interior point of \bar{F} . We choose coordinates in such a way that y is the north pole of the sphere. By connecting the vertices of the spherical polygon with the north pole, we get n spherical triangles which add up to a full 2π angle at the north pole. For the i -th triangle, let γ_i be the angle at the north pole, and α_i and β_i the angles to the meridians (see Fig. 8).

Let us now assume that the north pole lies on the left of all \bar{A}_i which can be expressed as

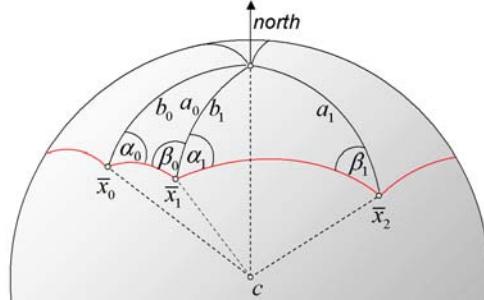


Fig. 8. Spherical triangles above mesh triangles

$$\alpha_{i+1} > \beta_i \quad (1)$$

Convexity implies that

$$\alpha_{i+1} + \beta_i \leq \pi \quad (2)$$

From (1), (2) and $0 < \alpha_{i+1}, \beta_i < \pi$ follows that

$$0 < \frac{\sin \beta_i}{\sin \alpha_{i+1}} < 1 \quad (3)$$

Taking the product yields

$$\prod_{i=0}^{n-1} \frac{\sin \beta_i}{\sin \alpha_i} = \prod_{i=0}^{n-1} \frac{\sin \beta_{i+1}}{\sin \alpha_i} < 1 \quad (4)$$

On the other hand, we can derive

$$\prod_{i=0}^{n-1} \frac{\sin \beta_i}{\sin \alpha_i} = \prod_{i=0}^{n-1} \frac{\sin b_i}{\sin a_i} = \prod_{i=0}^{n-1} \frac{\sin b_i}{\sin b_{i+1}} = 1 \quad (5)$$

making use of the spherical law of sines, the fact that $a_i = b_{i+1}$ because triangles fit together, and finally $b_n = b_0$.

From this contradiction follows that the test point is covered by \bar{P} , and so the interior of the spherical polygon \bar{F} .

Because of convexity, it is possible to choose an interior point of \bar{F} as the north pole such that all of \bar{F} lies in the northern hemisphere. It remains to show that \bar{P} not only covers \bar{F} but the whole hemisphere. Any spherical lune must have one of its end points below the equator, and because of convexity, this is the one on the concave side. But this means that along the equator, the sequence of lunes $\bar{P}_0, \dots, \bar{P}_{n-1}, \bar{P}_0$ can't have any gaps, and therefore the hemisphere is completely covered, which was the conjecture.

Generating, Representing and Querying Level-Of-Detail Tetrahedral Meshes

Leila De Floriani^{1,2} and Emanuele Danovaro¹

¹ Department of Computer Science, University of Genova, Genova, Italy

² Department of Computer Science, University of Maryland, College Park, MD, USA

Summary. In this paper, we survey techniques for building, encoding and querying Level-Of-Detail (LOD) models of three-dimensional scalar fields based on a domain decomposition into tetrahedral meshes. We focus on continuous LOD models, and we classify them into unstructured (irregular) and regular nested LOD models depending on the mesh subdivision pattern and on the distribution of the data points. Within each class, we review data structures, construction algorithms, as well as techniques for extracting adaptively refined field representations from an LOD model.

1 Introduction

Level-Of-Detail (LOD) models have been proposed to control and adapt the accuracy in the representation of large-size volume data sets. LOD models encode in a compact data structure the steps performed by a refinement process applied to a coarse representation of a scalar field, or by a decimation process applied to a full-resolution representation. A large numbers of simplified meshes can be extracted from an LOD model, in which the resolution (i.e., the density of the cells) of the simplified mesh may vary in different parts of the field domain, or in the proximity of interesting field values. The extraction of a simplified representation from an LOD model is called a *selective refinement*. The challenge in designing an LOD model is represented by the trade-off between the efficiency of the selective refinement algorithms and the storage cost of the representation.

This paper reviews techniques proposed in the literature for encoding, generating and performing selective refinement on an LOD model. We focus on so-called *continuous* LOD models, from which a virtually continuous simplified adaptive representations can be extracted. Discrete (non-continuous) LOD models consist of a (usually small) collection of representations at different LODs and only representations of the scalar field at uniform resolutions can be extracted from them [6].

The remainder of this paper is organized as follows. Section 2 introduces some background notions on tetrahedral meshes and discusses data structures for encoding them. Section 3 introduces the basic elements of an LOD model and the most

common update operations through which an LOD model is generated. Section 4 reviews incremental refinement and coarsening techniques used to generate unstructured LOD models. Section 5 discusses data structures for encoding unstructured LOD models. Section 6 reviews techniques for encoding and querying nested LOD models. Section 7 presents some comparisons of performances of unstructured and nested LOD models in extracting adaptively-refined meshes, and discusses some open research issues.

2 Background

A *volume data set* S consists of a set V of points in the three-dimensional Euclidean space, and of one or several field values associated with the points of V . The points in V can be regularly spaced, i.e., they are the vertices of a regular, rectilinear grid, or irregularly spaced. In the former case, we will call S a *structured*, or a *regular*, data set, while, in the latter case, we will call it an *unstructured*, or an *irregular* data set.

A *tetrahedral mesh* Σ is a connected set of tetrahedra such that the union of all tetrahedra in Σ covers a domain \mathcal{D} in 3D space and any two distinct tetrahedra have disjoint interiors. A tetrahedral mesh Σ is called a *conforming* mesh if the intersection of the boundaries of any two tetrahedra σ_1 and σ_2 of Σ , which have a non-empty intersection, consists of lower dimensional simplexes (vertices, edges, or triangles) that belong to the boundary of both σ_1 and σ_2 . Conforming meshes have a well-defined combinatorial structure in which each tetrahedron is adjacent to exactly one other tetrahedron along each of its faces. This is important when a tetrahedral mesh is used as a decomposition of the domain of a volume data set.

We call *nested meshes* those meshes which are defined by the uniform subdivision of a tetrahedron into scaled copies of it. In particular, we will consider nested regular meshes, in which the vertices are a subset of the vertices of a regular grid. A mesh which is not nested is called *irregular*, or *unstructured*. A mesh is called *stable* if the tetrahedra forming it satisfy some measure of non-degeneracy. Measures commonly used in the finite element literature are the *circumradius-to-shortest-edge* ratio (where the circumradius is the radius of the circumsphere of a tetrahedron), and the *minimum solid angle* associated with a tetrahedron [39].

The most common data structure for encoding a tetrahedral mesh Σ is the so-called *indexed data structure*, which, for each tetrahedron σ of Σ , stores the indexes of the four vertices of σ . If n denotes the number of vertices of Σ , the storage cost of this data structure is equal to 102 n bytes by assuming to encode indexes on 4 bytes, and coordinates on 2 bytes. The *indexed data structure with adjacencies* [31] generalizes the indexed data structure by encoding, for each tetrahedron σ of Σ , also the indexes of the four tetrahedra which are adjacent to σ along a face. The storage cost of this data structure is equal to 198 n bytes. This latter data structure has been extended to be able to retrieve all tetrahedra incident at a vertex efficiently, by storing, for each vertex v , the index of just one tetrahedron incident at v , leading to a storage cost of 202 n bytes.

3 LOD Models

The basic elements of an LOD model of a spatial object are a *base mesh*, that defines the coarsest approximation to the object, a set of *updates*, that, when applied to the base mesh, provide variable resolution mesh-based representations of the spatial object, and a *dependency relation* among updates, which allows combining them to extract consistent intermediate representations as well as the mesh at full resolution, that we call the *reference mesh* [11].

Intuitively, an *update* on a tetrahedral mesh Σ consists of replacing a connected set of tetrahedra Σ_1 in Σ with another set of tetrahedra Σ_2 in such a way that the result is still a mesh (see [11] for a formal definition). An update is *conforming* if, when applied to a conforming mesh, it produces a conforming mesh as result. An update is either described explicitly, as the set of tetrahedra involved in Σ_1 and Σ_2 , or implicitly, by encoding the operation which produces it. An update is called a *refinement update* if Σ_2 contains more tetrahedra than Σ_1 , a *coarsening update*, otherwise.

The *dependency relation* can be an inclusion relation or a representation of the possible orders in which updates can be performed to extract conforming meshes at different resolutions. When the base mesh is a nested regular mesh and all updates involve replacing one tetrahedron σ with a set of tetrahedra whose union cover σ , we call the resulting LOD model a *nested LOD model*, we call it an *unstructured LOD model* otherwise. In a nested LOD model, the dependency relation is an inclusion relation, since it captures the nestedness property of the underlying subdivision.

The basic operation on an LOD model, called *selective refinement*, consists of extracting a conforming mesh satisfying some application-dependent requirements based on level of detail, such as approximating a scalar field with a certain accuracy which can be uniform or variable in space. We consider a Boolean function τ , that we call an *LOD criterion*, defined over the tetrahedra of an LOD model, which returns a value *true* if a tetrahedron satisfies the requirements of the query, a value *false* otherwise. The general *selective refinement query* on an LOD model can be thus formulated as follows: given an LOD criterion τ , extract from the model the mesh M of minimum size that satisfies τ . In general, the LOD criterion is based on some approximation error, like the *field error*, which measures the accuracy with which a scalar field approximation provided by a simplified domain decomposition approximates the original field data, or the *isosurface error*, which measures the accuracy with which the isosurfaces extracted from a simplified representation of the field approximate the isosurfaces extracted by the representation at full resolution (see [5] for a thorough analysis of the different errors).

3.1 Updates in Unstructured LOD Models

In this Subsection, we briefly review the most common update for LOD models based on unstructured tetrahedral meshes, namely *edge collapse/vertex split*. Vertex insertion/removal, which is commonly used in the case of triangle meshes, is much less common for tetrahedral meshes, because of the theoretical problems involved in removing and in inserting a vertex in a tetrahedral mesh with a non-convex

domain [38, 42]. Tetrahedron collapse to a new vertex has also been used to simplify tetrahedral meshes [4]. Note that a tetrahedron collapse can be expressed also as a sequence of three edge collapses.

A *full-edge collapse* consists of contracting an edge e with extreme vertices v' and v'' to a new vertex v . The tetrahedra incident at v' or v'' become incident at v , and the tetrahedra incident at both v' and v'' collapse into triangles (see Fig. 1 from left to right). The reverse operation, called a *full-vertex split*, expands a vertex v into an edge e having its endpoints at v' and v'' . A full-vertex split partitions the tetrahedra incident at v into two subsets, which are separated by a fan T of triangles incident at v . Tetrahedra of the two subsets are deformed to become incident at v' and v'' , respectively. Triangles belonging to fan T become tetrahedra incident at both v' and v'' (see Fig. 1 from right to left).

A *half-edge collapse* consists of contracting an edge $e = (v, w)$ to one of its extreme vertices, say w (see Fig. 2 from left to right). The reverse update of a half-edge collapse is a *half-vertex split*, which expands a vertex w into an edge e by inserting the other extreme vertex v of e (see Fig. 2 from right to left). A half-edge collapse modifies the set of tetrahedra incident at v , while a half-edge split only a face-connected subset of tetrahedra incident at w . Our experiments have shown that, on average, a full-edge collapse replaces 33 tetrahedra with 27, while a half-edge collapse replaces 11 tetrahedra with 16 [8].

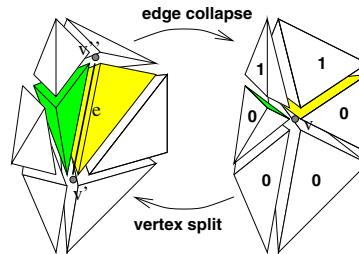


Fig. 1. Update of a tetrahedral mesh through a full-edge collapse and a full-vertex split. On the left, tetrahedra that degenerate into triangles after full-edge collapse are shaded. On the right, tetrahedra marked with 0 and with 1 result from the deformation of tetrahedra incident at v' and at v'' , respectively

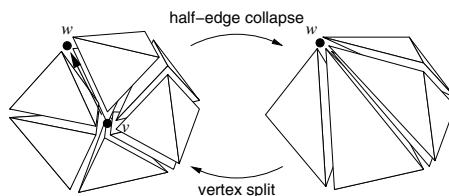


Fig. 2. Update of a tetrahedral mesh through a half-edge collapse and a half-vertex split

3.2 Updates in Nested LOD Models

Updates in nested LOD models consist of refining a three-dimensional cell σ with a set of three-dimensional cells whose union covers σ (*nestedness property*). The refinement rule is specific to each update rule. The most common refinement rules : *tetrahedron bisection*, *regular tetrahedron subdivision*, and *tetrahedron/octahedron subdivision*. Note that all such refinement rules generate non-conforming updates.

Tetrahedron Bisection

Tetrahedron bisection consists of replacing a tetrahedron σ with the two tetrahedra obtained by splitting σ at the middle point of its longest edge through the plane passing through such point and the opposite edge in σ [24, 30, 36, 39, 40]. It is applied recursively to an initial decomposition of a cubic domain into six tetrahedra (see Fig. 3a). This gives rise to three congruent tetrahedral shapes, that we call *1/2 pyramids*, *1/4 pyramids* and *1/8 pyramids*, respectively (see Fig. 3). The tetrahedra in the initial cube subdivision are *1/8 pyramids*. In general, if we consider as level 0 in the recursive subdivision the one corresponding to the domain subdivision, we have *1/8 pyramids* at any level $i = 3j$, *1/2 pyramids* at any level $i = 3j + 1$, *1/4 pyramids* at any level $i = 3j + 2$, $j = 0, 1, \dots$. Note that all three shapes satisfy stability measures.

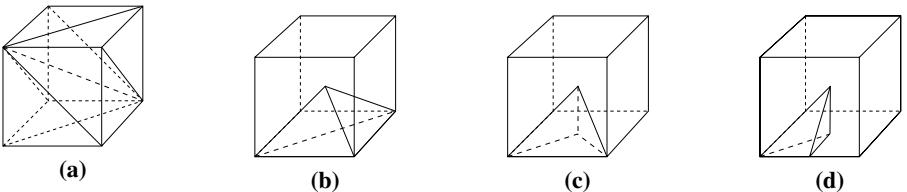


Fig. 3. (a) Subdivision of the cubic domain into six tetrahedra. Examples of (b) a 1/2 pyramid, (c) a 1/4 pyramid, and (d) a 1/8 pyramid

In order to guarantee that a conforming mesh is generated when applying tetrahedron bisection, all tetrahedra that share their longest edge with the tetrahedron being split must be split at the same time. Such tetrahedra form a *diamond* (sometimes called a *cluster*) [19, 26, 34]. The conforming update defined by tetrahedron bisection consists of replacing all the tetrahedra in a diamond with the tetrahedra obtained by splitting them along their longest edge. The vertex used to split a diamond is called the *split vertex* of the diamond. There are three types of diamonds, generated by the three congruent tetrahedral shapes, that we call *plane-aligned*, *axis-aligned* and *non-aligned* diamonds, which are formed by 1/2, 1/4 and 1/8 pyramids, respectively (see Fig. 4).

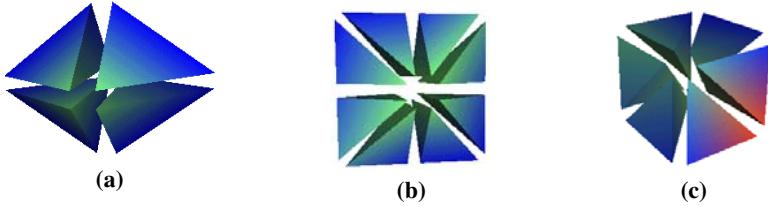


Fig. 4. (a) A plane-aligned diamond. (b) An axis-aligned diamond. (c) A non-aligned diamond

Regular Tetrahedron Refinement

The *regular tetrahedron refinement rule* subdivides a tetrahedron σ into eight tetrahedra by bisecting all edges of σ [1, 22, 45]. Four tetrahedra in regular tetrahedron refinement correspond to the vertices of σ , since each of them is obtained by cutting off the corresponding corner. These four tetrahedra are congruent with σ . The other four tetrahedra are obtained by splitting the octahedron generated when cutting off the four corner tetrahedra from σ (see Fig. 5). The octahedron is split into two pyramids, each of which is partitioned into two tetrahedra. The splitting of the octahedron into four tetrahedra is not unique (there are three possible choices for a diagonal) and a wrong choice can generate unstable meshes. The method proposed by Bey [1] generates three congruence classes.

To generate a conforming mesh, when a tetrahedron has to be split, all the tetrahedra at the same level of subdivision have to be split at the same time. In this way, adaptive meshes, i.e., meshes in which the size of the tetrahedra varies in different parts of the domain, cannot be obtained. To overcome this problem, the above subdivision rule, also called a *red refinement* rule, is combined with a refinement rule which generates irregular tetrahedra, called *green refinement* rule [1, 22, 45] (see Subsect. 6.2).

Tetrahedron/Octahedron Refinement

The *tetrahedron/octahedron refinement rule* [20] combines the regular refinement rule for a tetrahedron which subdivides a tetrahedron σ into four tetrahedra and one

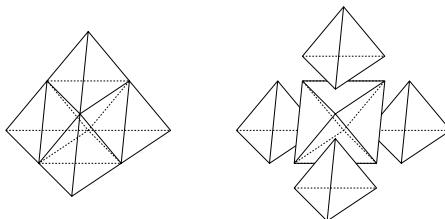


Fig. 5. Splitting of a tetrahedron into four tetrahedra and an octahedron

octahedron, with a refinement rule for an octahedron which subdivides an octahedron γ into eight tetrahedra and six octahedra by connecting the edge midpoints of each face of γ and by connecting all edge midpoints to the barycenter of octahedron γ (see Fig. 6). To generate a conforming mesh, when a tetrahedron or an octahedron at a certain level has to be split, all the tetrahedra and octahedra at the same level of subdivision have to be split at the same time. Thus, the only conforming meshes are those including all the tetrahedra and octahedra at the same level. To generate adaptive meshes, an irregular refinement rule must be applied (see Subsect. 6.2).

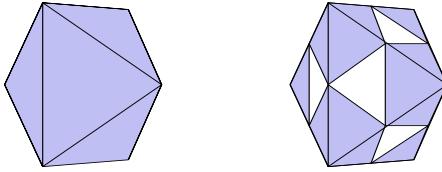


Fig. 6. Splitting of an octahedron into eight tetrahedra and six octahedra.

4 Generating LOD Models

LOD models are generated through simplification algorithms. Many simplification methods have been developed in the literature for dealing with triangulated surfaces (see, e.g., [16] for a survey). The most effective solutions to the simplification problem have been provided by *incremental* techniques, based on either *coarsening (decimation)*, or *refinement* strategies. Incremental techniques are especially interesting for generating LOD models, since the intermediate steps of the process produce meshes at decreasing (or increasing) LODs.

4.1 Coarsening Algorithms

Most coarsening algorithms simplify a volume data set by performing an edge-collapse. They all work on irregular tetrahedral meshes. Gross and Staadt [21] present a decimation technique based on collapsing an edge to an arbitrary interior point, and they introduce various cost functions to drive the collapsing process. Cignoni et al. [5] propose an algorithm based on half-edge collapse, in which the decimation process is driven by a combination of the geometric error introduced in simplifying the shape of the domain and of the error introduced in approximating the scalar field with fewer points. Trott et al. [44] perform half-edge collapse as well. They control the quality of the simplified mesh by estimating the deviation of the simplified scalar field from the original one, and by predicting the increase in deviation caused by a collapse.

Renze and Olivier [38] use vertex removal on a Delaunay mesh as a decimation criterion: their algorithm removes a vertex v only if the hole left in the mesh can be

re-triangulated without inserting new points. Vertex removal cannot always be performed on a Delaunay tetrahedral mesh: there is no guarantee that the star-shaped polyhedron generated by vertex removal can be re-triangulated without adding extra points. Chopra e Meyer [4] present a decimation algorithm based on iteratively collapsing a tetrahedron σ into a new vertex. This technique produces a higher decimation rate per step compared with methods based on edge collapse.

4.2 Refinement Algorithms

Refinements algorithms are used for both regular and irregular meshes. Refinement techniques for regular meshes consists of recursively applying to the base mesh a refinement update, defined through a refinement rule satisfying the nestedness property.

Refinement techniques for irregular meshes are based on incremental point insertion. In [23], Hamann and Chen describe a refinement technique for an irregular mesh with a convex boundary, which performs a top-down selection of relevant points, based on curvature. Significant data are identified by large absolute curvatures obtained through a local least-square approximation method. When a point is inserted into the mesh, local modifications are performed in order to minimize a local approximation error, thus leading to a data-dependent triangulation of the domain. In [6], an incremental refinement technique based on inserting a point in a Delaunay tetrahedral mesh with a convex domain is described. The point with the largest error with respect to the original volume data set is selected. LOD models based on vertex insertion can be generated only for unstructured meshes with a convex domain, or for meshes generated by warping of a regular grid [6]. This is due to the intrinsic theoretical difficulties in generating a constrained Delaunay tetrahedral mesh in three dimensions by using just the vertices in the data set.

5 Data Structures for Unstructured LOD Models

Unstructured LOD models are based on conforming updates and on a *dependency relation* defined among updates as follows: an update u is *dependent* on another update u' if and only if u deletes some tetrahedron introduced by u' . This rule imposes a strict partial order on the set of updates, which encodes all and only those dependencies that are necessary to perform either refinement or coarsening updates in the same situation in which they were performed during construction, thus guaranteeing that the result is a conforming mesh.

The data structures for encoding unstructured LOD models can be classified into *explicit data structures*, which explicitly represent the tetrahedra in the LOD model, and into *implicit data structures*, which encode the operations through which the LOD model is generated (see also [10]). Adaptive meshes are extracted through top-down refinement or through alternating incremental refinement and coarsening steps by traversing the data structure which implements the dependency relation. In this Section, we briefly review an explicit data structure, and three implicit ones based on edge collapse (see [10] for an implicit data structure encoding vertex-based updates).

5.1 An Explicit Data Structure

An LOD model based on d -dimensional simplicial meshes, called a *Multi-Tessellation (MT)*, has been defined in [13], which is independent of the dimension of the complex and of the specific strategy through which the model is built. This model extends the *Multi-Triangulation*, proposed in [37], to three and higher dimensions. A data structure implementing the three-dimensional MT describes the dependency relations as a DAG (Directed Acyclic Graph) and represents each tetrahedron explicitly as the 4-tuple of its vertex indexes [10]. The total cost of the 3D MT data structure, when built through half-edge collapses, has been shown to be equal to $360n$ bytes, which is about 3.5 times the cost for encoding the reference mesh as an indexed data structure, and about 1.8 times the cost for encoding the reference mesh as an indexed data structure with adjacencies.

5.2 A Data Structure based on Full-Edge Collapse

In [7], an LOD data structure for unstructured LOD models, called a *Full-Edge Multi-Tessellation (MT)*, has been developed to efficiently encode an LOD model generated through full-edge collapses.

The dependency relation between pair of updates is encoded implicitly through a data structure proposed in [14], called a *view-dependent tree*. A view-dependent tree consists of a binary forest of vertices, enriched with a vertex numbering scheme. In the binary forest, the roots are vertices of the base mesh Σ_0 , and the two children of a vertex v are the vertices v' and v'' generated by splitting v , i.e., the endpoints of the edge collapsed into v . Vertices of the reference mesh are assigned labels from 1 to n in arbitrary order. Vertices created in simplification are assigned consecutive integer labels. In order to check whether or not a split [collapse] can be performed, it is sufficient to compare the label of the vertex to be split [extreme vertices of the edge to collapse] with the labels of its [their] adjacent vertices. This guarantees that all splits and collapses applied in selective refinement will be performed in the same situation as in the simplification algorithm.

An internal node of the view-dependent tree corresponds to the collapse of an edge $e = (v', v'')$ to a vertex v (see Fig. 1) and to its inverse vertex split. It contains offset values, used to find the positions and the field values at vertices v' and v'' from that of v , and vice-versa, the approximation error introduced by performing the edge collapse, and a bit mask, used to partition the set of tetrahedra incident at v by labeling each tetrahedron with one bit. Vertex v must be replaced with v' in all tetrahedra marked with 0, while v must be replaced with v'' in all tetrahedra marked with 1. Each triangular face shared by two differently marked tetrahedra must be expanded into a tetrahedron incident at both v' and v'' (see Fig. 1).

The total cost of the Full-Edge MT data structure is equal to $32n$ bytes, which is about 1/3 of the cost for encoding the reference mesh as an indexed data structure, and about 19% of the cost for encoding the reference mesh as an indexed structure with adjacencies [7].

5.3 Data Structures based on Half-Edge Collapse

A compact data structure for encoding a *Half-Edge Multi-Tessellation (MT)*, i.e., an LOD model generated through half-edge collapses, is described in [8]. The dependency relation is encoded as a DAG by using the technique proposed by Klein and Gumhold [25], in which each arc of the DAG is encoded only once. A half-edge collapse of an edge (v, w) to a vertex w and its corresponding half-vertex split are encoded by storing: the coordinates of vertex v and the field value at vertex v , the approximation error introduced by performing the half-edge collapse, an implicit encoding of vertex w and a compact encoding of the set of tetrahedra incident in w which are affected by inserting vertex v (that we call the *region of influence* of the update). This encoding consists of an initial tetrahedron plus a traversal of the region of influence represented as a string of bits. The bit string is generated by traversing the tetrahedra involved in the vertex split in a breadth-first fashion, starting from the given tetrahedron. Any tetrahedral face f crossed during the traversal is labeled with 1 or 0, depending on whether it is an internal face or it is on the boundary of the region of influence (see Fig. 2). The total cost of this DAG-based Half-Edge MT data structure (including the cost of encoding the direct dependencies) is equal to $56n$ bytes [8], which is about half of the cost for encoding the reference mesh as an indexed data structure, and about 28% of the cost for encoding the reference mesh as an indexed structure with adjacencies.

A more space-efficient data structure for a Half-Edge MT has been introduced in [43]. In this data structure, the direct dependency relation is encoded implicitly through an extension of the view-dependent tree. The vertices of the reference mesh still correspond to the leaves of the binary forest, but, if an edge (v, w) is collapsed to vertex w , then a renamed copy w' of w appears in the binary tree as the parent of vertices v and w : vertex w is called a *false child* of w' . For each false child q , a *true parent* of q is defined as the true parent of the nearest ancestor \bar{q} of q , such that \bar{q} is a true child. Updates are encoded in a similar way as in the previous DAG-based data structure. The total cost is $33n$ bytes, which is about 60% of the cost of encoding the DAG-based representation.

Table 1 shows experimental comparisons between a Half-Edge MT and a Full-Edge MT based on mesh extractions at a uniform as well as at a variable LOD. For extractions at a uniform LOD, the error with which the extracted mesh approximates the reference mesh is required to be less or equal a threshold value, expressed as a percentage of the range of the field values. For extractions at variable LODs, we require the approximation error to be less or equal to a threshold value in a focus set (which is a box in the experiments presented here) and arbitrarily large outside the focus set. The ratio between the size of any extracted mesh and the reference mesh is shown in the table.

The experiments shown have been performed on three different data sets, namely: *Plasma* (Visual Computing Group, CNR, Italy), which is a regular synthetic data set representing Perlin noise, with 274,625 vertices and 1,310,720 tetrahedra, *Blunt Fin* (C.M. Hung and P.G. Buning, NASA), which is a curvilinear data set representing an airflow over a flat plate and has 40,948 vertices and 222,414 tetrahedra, and *San*

Table 1. Size of the meshes extracted from Plasma, Blunt Fin and San Fernando data sets at a uniform LOD and at a variable LOD with a box as focus set. The values represent the size of the extracted mesh with respect to the size of the reference mesh

Error	Uniform LOD				Variable LOD – Box			
	Half-Edge MT		Full-Edge MT		Half-Edge MT		Full-Edge MT	
	Blunt Plasma	SF	Blunt Plasma	SF	Blunt Plasma	SF	Blunt Plasma	SF
0.00%	1.00	1.00	1.00	1.00	0.55	0.06	0.25	0.77
0.03%	0.91	0.98	0.84	0.95	0.99	0.91	0.52	0.06
0.09%	0.83	0.97	0.77	0.89	0.99	0.85	0.49	0.06
0.30%	0.63	0.95	0.67	0.72	0.97	0.77	0.35	0.06
0.90%	0.40	0.72	0.51	0.51	0.75	0.61	0.20	0.05
3.00%	0.26	0.18	0.41	0.33	0.21	0.45	0.09	0.02
9.00%	0.24	0.02	0.24	0.33	0.03	0.32	0.03	0.01
20.00%	0.20	0.01	0.02	0.28	0.01	0.32	0.03	0.00

Fernando (D.R. O’Hallaron and J.R. Shewchuk, Quake project), which is an irregular data set representing the simulation of an earthquake in the San Fernando Valley in Southern California and has 378,747 vertices and 2,067,739 tetrahedra. The results show that, on average, the size of the meshes extracted from a Half-Edge MT is about 16% less than the size of those extracted from a Full-Edge MT at a uniform LOD, while this percentage increases to 51% for extractions at variable LOD with a box as focus set.

6 Encoding and Querying Nested LOD Models

Most data structures for nested LOD models maintain a table of field values and, possibly, the approximation errors associated with the tetrahedra (or octahedra) generated at intermediate steps in the subdivision. In this latter case, approximation errors are encoded in an array which provides a pointer-less representation of the hierarchy describing the nested mesh. The hierarchy consists of a forest of tetrahedra (and possibly octahedra), in which the roots correspond to the initial domain subdivision, any other node describes a tetrahedron, or an octahedron, and the children of a node are tetrahedra (and octahedra) subdividing the corresponding cell. The leaf nodes, which correspond to tetrahedra (or octahedra) in the reference mesh, are not encoded, since they have a null error. Note that the forest of tetrahedra does not need to be explicitly encoded unless approximation errors or other attribute information are attached to the tetrahedra (or octahedra) in the forest.

In this Section, we review data structures which have been developed for nested LOD models generated through *tetrahedron bisection*, and we discuss techniques for extracting adaptive conforming meshes from such representations.

6.1 Data Structures

The data structures for encoding nested regular meshes produced through tetrahedron bisections encode (implicitly or explicitly) either a binary forest of tetrahedra, called a *Hierarchy of Tetrahedra (HT)*, or a *DAG of diamonds* with their direct dependencies.

An HT consists of a field table containing the field values at the data points plus an array containing the approximation errors associated with the tetrahedra corresponding to the internal nodes in the forest. Alternatively, error information can be associated with the vertices. Each split vertex v of a diamond would have an approximation error associated with which is equal to the maximum of the approximation errors associated with the tetrahedra in the diamond. In this case, the hierarchy is not encoded and the inclusion relation is implicit. If n is the number of points in the reference mesh, $2n$ bytes are necessary for the field table, while between $2n$ and $12n$ bytes are required for the error values, assuming two bytes per error and depending on whether errors are associated with tetrahedra in the hierarchy or just to the vertices $=$. If the error and the field values are quantized as in [19] to one byte, the storage cost of the data structure ranges from $2n$ to $7n$ bytes.

A different, but equivalent, representation is defined by considering the conforming updates, i.e. the diamonds, generated by the tetrahedron bisection process and their dependency relations [19, 34]. We call such representation a *DAG of diamonds*. It extends similar representations used for describing nested LOD models based on triangle meshes [28, 29, 33]. Given a diamond D , the *parents* of D are those diamonds that must be split to create the tetrahedra of D . The diamonds that are created when D is split are the *children* of D . Note that a plane-aligned diamond has two parents and four children, an axis-aligned diamond has four parents and eight children, and a non-aligned diamond has three parents and six children. In a DAG of diamonds G_D , the root is the initial subdivision of the cube (a non-aligned diamond), any other node is a diamond and the arcs describe the parent-child relation. Note that the diamonds at level i are non-aligned diamonds, those at level $i + 1$ are plane-aligned diamonds, while those at level $i + 2$ are axis-aligned diamonds, where $i = 3j, j = 0, 1, \dots, max$.

In [19], a compact data structure for encoding a DAG of diamonds is described in which the DAG structure has not to be explicitly recorded. Diamond information as well as error information are attached to each vertex together with its field value. Only three bytes per diamond are used to store the pre-computed error information for each diamond.

6.2 Extracting Conforming Meshes

In order to compute adaptive conforming meshes from a nested LOD model built through tetrahedron bisection, it is necessary to apply conforming updates to the currently extracted mesh. In a selective refinement algorithm, when a tetrahedron σ must be refined, all tetrahedra belonging to the same diamond as σ must be refined as well. When a DAG of diamonds is maintained, the conforming updates are pre-computed. Otherwise, the diamonds must be computed while extracting a mesh at a

certain LOD. Two techniques have been used in order to apply conforming updates at each step: one is based on *error saturation* [18, 32, 46], and the other one is based on *neighbor finding* [24, 26].

One way to ensure that all tetrahedra belonging to the same diamond are refined at the same time is to assign all of them the same error value, which is equal to the maximum of their original error values. The saturation condition states that approximation error associated with each tetrahedron must be greater than or equal to the error associated with its children. The saturation condition can be applied to any error metric as shown in [17] for the case of terrains.

An alternative method consists of computing the diamond to which a given tetrahedron σ belongs during selective refinement by finding the tetrahedra sharing their longest edge with σ . The neighbor finding algorithms is based on assigning a *location code* to each tetrahedron, with a mechanism similar to that adopted for linear encoding of triangle quadtrees [27], or hierarchies of right triangles [15]. Location codes are not stored, but they are computed when extracting a mesh during selective refinement. A *location code* for a tetrahedron σ consists of a pair of numbers, in which the first denotes the level of σ in the tree, and the second indicates the path from the root of the tree to σ . The location code for a tetrahedron is defined on the basis of a labeling scheme for its children and for the vertices of such children in the forest [26].

The neighbor finding algorithm sketched in [26] uses an approach similar to the one defined in [41] for region quadtrees. It consists of two steps. Given an input tetrahedron σ and a face $v_i v_j v_k$, the first step identifies the nearest common ancestor of σ and its neighbor σ' along face $v_i v_j v_k$ by scanning the location code from right to left. The second step updates the location code for the neighbor by using the information obtained while finding the nearest common ancestor: just the one bit corresponding to the child of the nearest common ancestor needs to be inverted. In [26], an implementation is proposed which performs neighbor finding in worst-case constant time. The algorithm makes use of the carry property of addition to find a neighbor efficiently without specifically searching for a nearest common ancestor. In [24], Hebert proposes a technique for computing parents, children, and neighbors of a tetrahedron in a symbolic way, but finding neighbors still takes time proportional to the depth of the subdivision.

Experimental comparisons between saturation-based and neighbor finding techniques, discussed in [12], performed on the basis of mesh extractions at uniform resolution, have shown that the meshes extracted using error saturation have, on average, 5% more tetrahedra than those extracted with the neighbor finding algorithm. On the other hand, the computing times for mesh extraction are the same for the saturated and non-saturated versions.

Extracting uniformly-refined conforming meshes from nested LOD models based either on regular tetrahedron bisection, or on tetrahedron/octahedron subdivision just requires to refine all tetrahedra or octahedra at the same level of subdivision at the same time. The extracted meshes will be just those corresponding the different subdivision levels. Extracting adaptive meshes requires combining the regular refinement

rules, discussed in Subsection 3.2, with irregular refinement rules, which refine a tetrahedron by bisecting it on one to five edges.

The refinement algorithm performs all the regular (red) refinements (as defined in Subsection 3.2) in a top-down fashion to meet a certain LOD criterion. Then the resulting mesh Σ is refined by still applying the regular refinement rule, until any pair of edge-adjacent 3-cells (tetrahedra or octahedra) differ at most in one subdivision level. Thus, any 3-cell in the resulting refined mesh Σ' is at a level i or at a level $i - 1$. Any octahedron at level $i - 1$ in a tetrahedron/octahedron subdivision is first split by connecting the vertices to its centroid. Then, the irregular refinement rule is applied to all tetrahedra at level $i - 1$. A tetrahedron has 64 irregular edge-refinement patterns, which can be grouped into 9 patterns due to symmetry considerations [20].

7 Concluding Remarks

In [9], we have compared the performance of the Half-Edge MT, and of a model based on tetrahedron bisection on regular volumetric data sets, encoded as a Hierarchy of Tetrahedra (HT), by considering a set of queries at a uniform LOD and at variable LODs. We have used two regular data sets, namely the *Buckminster Fullerene (Buckyball)* data set (Robert Haimes, MIT), which consists of 262,144 vertices and 1,250,235 tetrahedra and the *Plasma* data set, previously described.

The graph in Fig. 7 (a) shows the results of extractions at uniform LOD, i.e., the number of tetrahedra in the meshes extracted at a uniform LOD for different error thresholds. The graph in Fig. 7 (b) shows the results of extractions at variable LOD, i.e., the number of tetrahedra in the meshes extracted at a variable LOD for different error thresholds inside a box and arbitrary large error outside. At a uniform LOD, the meshes extracted from the HT have roughly 20% more tetrahedra than those extracted from the Half-Edge MT, since in the Half-Edge MT there is more flexibility in choosing which tetrahedra are to be split, while in the HT these are determined by the fixed recursive decomposition rule. On the other hand, extractions at variable LOD show the opposite behavior: the meshes extracted from an HT have fewer tetrahedra than those extracted from an Half-edge MT (about 1/4 for extractions with maximum resolution in a box).

There are several open research issues related to data structures for LOD models. Out-of-core issues, like simplification of meshes on secondary memory, as well as data structures and querying algorithms for LOD models stored on secondary memory, are very important both in the regular and irregular cases to be able to handle meshes of large size efficiently (see [35] for a proposal for regular tetrahedral meshes). Inserting information about the morphological structure of the underlying scalar field in its LOD model could enhance the accuracy and the adaptivity of the extracted representations. Some proposals exist for simplifying a 3D scalar field by maintaining the topology of the isosurfaces [3], and for an LOD model of a 2D scalar field built by starting with a decomposition of the domain of the scalar field defined by its critical points and separatrix lines and progressively canceling critical points [2].

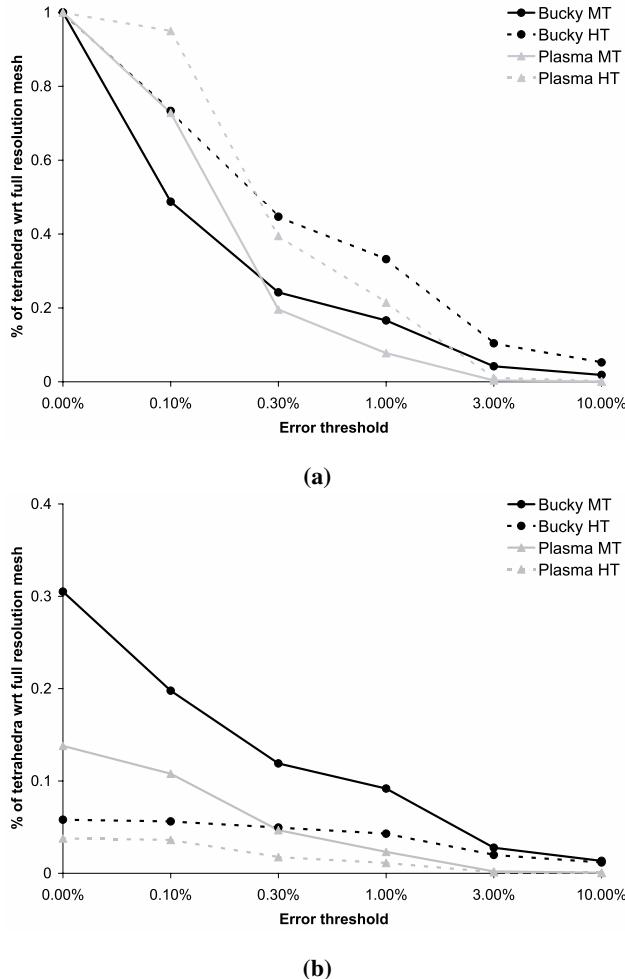


Fig. 7. Comparison between HT and MT at uniform LOD (a) and at variable LOD with a box as focus set (b) on the Buckyball and Plasma data sets

Acknowledgments

This work has been partially supported by two projects funded by the Italian Ministry of Education, University, and Research (MIUR) *Algorithmic and Computational Methods for Geometric Object Representation* (MACROGeo) and *Representation and Management of Spatial Data on the WEB* (SPADA-WEB).

References

1. J. Bey. Tetrahedral mesh refinement. *Computing*, 55:355–378, 1995.
2. P.-T. Bremer, H. Edelsbrunner, B. Hamann, and V. Pascucci. A multi-resolution data structure for two-dimensional Morse functions. In J. van Wijk G.Turk and R.Moorhead, editors, *Proceedings IEEE Visualization 2003*, pp. 139–146. IEEE Computer Society, October 2003.
3. Y.-J. Chiang and X. Lu. Progressive simplification of tetrahedral meshes preserving all isosurface topologies. *Computer Graphics Forum*, 22(3), 2003.
4. P. Chopra and J. Meyer. Ttfusion: an algorithm for rapid tetrahedral mesh simplification. In *Proceedings IEEE Visualization 2002*, pp. 133–140. IEEE Computer Society, October 2002.
5. P. Cignoni, D. Costanza, C. Montani, C. Rocchini, and R. Scopigno. Simplification of tetrahedral volume data with accurate error evaluation. In *Proceedings IEEE Visualization 2000*, pp. 85–92. IEEE Computer Society, 2000.
6. P. Cignoni, L. De Floriani, C. Montani, E. Puppo, and R. Scopigno. Multiresolution modeling and visualization of volume data based on simplicial complexes. In *Proceedings 1994 Symposium on Volume Visualization*, pp. 19–26, October 1994.
7. P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):525–537, November 2003.
8. E. Danovaro and L. De Floriani. Half-Edge Multi-Tessellation: a compact representations for multiresolution tetrahedral meshes. In *Proceedings 1st International Symposium on 3D Data Processing Visualization and Transmission*, pp. 494–499. IEEE Computer Society, 2002.
9. E. Danovaro, L. De Floriani, M. Lee, and H. Samet. Multiresolution tetrahedral meshes: an analysis and a comparison. In *Proceedings 2002 International Conference on Shape Modeling*, pp. 83–91, May 2002.
10. E. Danovaro, L. De Floriani, P. Magillo, and E. Puppo. Data structures for 3D Multi-Tessellations: an overview. In F.H. Post, G.P. Bonneau, and G.M. Nielson, editors, *Proceedings Dagstuhl Scientific Visualization Seminar*, pp. 239–256. Kluwer Academic Publishers, 2002.
11. L. De Floriani and P. Magillo. Multi-resolution mesh representation: models and data structures. In M. Floater, A. Iske, and E. Quak, editors, *Principles of Multi-resolution in Geometric Modeling*, Lecture Notes in Mathematics, Springer Verlag, Berlin (D), 2002.
12. L. De Floriani and M.Lee. Selective refinement on nested tetrahedral meshes. In B.Hamann G.Brunett and H.Mueller, editors, *Geometric Modelling for Scientific Visualization*, pp. 329–344. Springer Verlag, 2003.
13. L. De Floriani, E. Puppo, and P. Magillo. A formal approach to multiresolution modeling. In R. Klein, W. Straßer, and R. Rau, editors, *Geometric Modeling: Theory and Practice*, pp. 302–323. Springer Verlag, 1997.
14. J. El-Sana and A. Varshney. Generalized view-dependent simplification. *Computer Graphics Forum*, 18(3):C83–C94, 1999.
15. W. Evans, D. Kirkpatrick, and G. Townsend. Right-triangulated irregular networks. *Algorithmica*, 30(2):264–286, 2001.
16. M. Garland. Multi-resolution modeling: Survey & future opportunities. In *Eurographics '99 – State of the Art Reports*, pp. 111–131. Eurographics Association, 1999.
17. T. Gerstner. Top-down view-dependent terrain triangulation using the octagon metric. Technical report, Institut für Angewandte Mathematik, University of Bonn, Bonn, Germany, 2003.

18. T. Gerstner and M. Rumpf. Multiresolutional parallel isosurface extraction based on tetrahedral bisection. In *Proceedings 1999 Symposium on Volume Visualization*. IEEE Computer Society, 1999.
19. B. Gregorski, M. Duchaineau, P. Lindstrom, V. Pascucci, and K. Joy. Interactive view-dependent rendering of large isosurfaces. In *Proceedings IEEE Visualization 2002*. IEEE Computer Society, October 2002.
20. G. Greiner and R. Grosso. Hierarchical tetrahedral-octahedral subdivision for volume visualization. *The Visual Computer*, 16:357–369, 2000.
21. M.H. Gross and O.G. Staadt. Progressive tetrahedralizations. In *Proceedings IEEE Visualization '98*, pp. 397–402, Research Triangle Park, NC, 1998. IEEE Computer Society.
22. R. Gross, C. Luerig, and T. Ertl. The multilevel finite element method for adaptive mesh optimization and visualization of volume data. In R. Yagel and H. Hagen, editors, *Proceedings IEEE Visualization '97*, pp. 387–394, Phoenix, AZ, October 1997.
23. B. Hamann and J.L. Chen. Data point selection for piecewise trilinear approximation. *Computer Aided Geometric Design*, 11:477–489, 1994.
24. D. J. Hebert. Symbolic local refinement of tetrahedral grids. *Journal of Symbolic Computation*, 17(5):457–472, May 1994.
25. R. Klein and S. Gumhold. Data compression of multiresolution surfaces. In *Visualization in Scientific Computing '98*, pp. 13–24. Springer Verlag, 1998.
26. M. Lee, L. De Floriani, and H. Samet. Constant-time neighbor finding in hierarchical tetrahedral meshes. In *Proceedings International Conference on Shape Modeling & Applications*, pp. 286–295, Genova, Italy, May 2001.
27. M. Lee and H. Samet. Navigating through triangle meshes implemented as linear quadtrees. *ACM Transactions on Graphics*, 19(2):79–121, April 2000.
28. P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. A. Turner. Real-time continuous level of detail rendering of height fields. In *Proceedings SIGGRAPH'96*, pp. 109–118, New Orleans, August 1996.
29. P. Lindstrom and V. Pascucci. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):239–254, 2002.
30. J. M. Maubach. Local bisection refinement for n -simplicial grids generated by reflection. *SIAM Journal on Scientific Computing*, 16(1):210–227, January 1995.
31. G. M. Nielson. Tools for triangulations and tetrahedralizations and constructing functions defined over them. In G. M. Nielson, H. Hagen, and H. Müller, editors, *Scientific Visualization: Overviews, Methodologies and Techniques*, chapter 20, pp. 429–525. IEEE Computer Society, Silver Spring, MD, 1997.
32. M. Ohlberger and M. Rumpf. Adaptive projection operators in multiresolution scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):74–93, 1999.
33. R. Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. In D. Ebert, H. Hagen, and H. Rushmeier, editors, *Proceedings IEEE Visualization'98*, pp. 19–26, Research Triangle Park, NC, October 1998. IEEE Computer Society.
34. V. Pascucci. Slow Growing Subdivisions (SGSs) in any dimension: towards removing the curse of dimensionality. *Computer Graphics Forum*, 21(3), 2002.
35. V. Pascucci. Multi-resolution indexing for hierarchical out-of-core traversal of rectilinear grids. In G. Farin, H. Hagen, and B. Hamann, editors, *Hierarchical and Geometrical Methods for Scientific Visualization*, Springer Verlag, Heidelberg, Germany, 2003.
36. V. Pascucci and C. L. Bajaj. Time critical isosurface refinement and smoothing. In *Proceedings IEEE Symposium on Volume Visualization*, pp. 33–42, Salt Lake City, UT, October 2000. IEEE Computer Society.

37. E. Puppo. Variable resolution triangulations. *Computational Geometry Theory and Applications*, 11(3-4):219–238, December 1998.
38. K.J. Renze and J.H. Oliver. Generalized unstructured decimation. *IEEE Computational Geometry & Applications*, 16(6):24–32, 1996.
39. M. Rivara and C. Levin. A 3D refinement algorithm for adaptive and multigrid techniques. *Communications in Applied Numerical Methods*, 8:281–290, 1992.
40. T. Roxborough and G. Nielson. Tetrahedron-based, least-squares, progressive volume models with application to freehand ultrasound data. In *Proceedings IEEE Visualization 2000*, pp. 93–100. IEEE Computer Society, October 2000.
41. H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990.
42. J. Shewchuk. Tetrahedral mesh generation by Delaunay refinement. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, pp. 86–95, Minneapolis, Minnesota, June 1998. Association for Computing Machinery.
43. N. Sokolovsky, E. Danovaro, L. De Floriani, and P. Magillo. Encoding level-of-detail tetrahedral meshes based on half-edge collapse. In *MINGLE'2003 Proceedings*, pp. 91–102. Springer Verlag, Cambridge, UK, September 2003.
44. I.J. Trotts, B. Hamann, and K.I. Joy. Simplification of tetrahedral meshes with error bounds. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):224–237, 1999.
45. S. Zhang. Successive subdivision of tetrahedra and multigrid methods on tetrahedral meshes. *Houston J. Mathematics*, 21:541–556, 1995.
46. Y. Zhou, B. Chen, and A. Kaufman. Multiresolution tetrahedral framework for visualizing regular volume data. In R. Yagel and H. Hagen, editors, *Proceedings IEEE Visualization '97*, pp. 135–142, Phoenix, AZ, October 1997.

Split 'N Fit: Adaptive Fitting of Scattered Point Cloud Data

Gregory M. Nielson¹, Hans Hagen², Kun Lee³ and Adam Huang⁴

¹ Arizona State University

nielson@asu.edu

² University of Kaiserslautern

hagen@informatik.uni-kl.de

³ Handong University

kunlee@handong.edu

⁴ National Institute of Health

hhuang@cc.nih.gov

1 Problem and Basic Strategy

In this paper, we describe a new technique for computing a triangular mesh surface approximation to a point cloud of unorganized 3D data points. We should point out that the problem of point cloud fitting should be distinguished from that of scattered data modeling [13] Even though many of the basic techniques and tools from CAGD and multivariate approximation theory apply to both problems, they are basically different. The term “scattered data” was coined by Schumaker in his 1976 paper [27] and there was a great deal of interest and published research on (mainly) bivariate problems in the 70s and 80s. With the advent of scientific visualization along with volume visualization in the 90s, there has been growing interest in trivariate scattered data modeling [19, 21] and interest in this area continues to grow. In many respects, the problem of scattered point cloud fitting is more difficult because it is

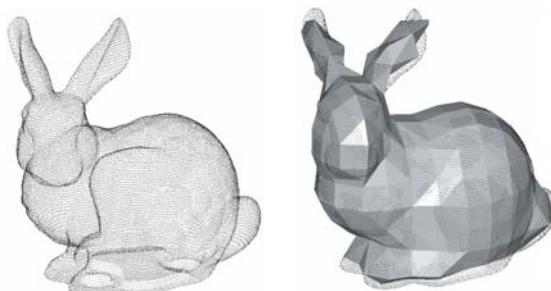


Fig. 1. Illustrating the process of fitting a point cloud with a triangular mesh surface

less understood, but the widespread and strong need for practical and effective methods make this an important problem. One fundamental connection between scattered data fitting and scattered point cloud fitting is through some type of parameterization of the point cloud [12] whether this be implicit or explicit, but this relationship is not well and completely understood today. Currently, there is widespread interest in the problem of point cloud fitting and many algorithms, methods and techniques are being proposed. See the survey [16] and the methods of [5, 9, 11, 17, 26, 28–30].

Some methods require normal vectors (see, for example, [8]) and without a scheme for computing these estimates the method is really incomplete as many data sets from practical applications do not include normal vectors. The most difficult part of estimating normals is to obtain a consistent orientation of inside or outside pointing normals. In some cases, knowing the origin of the point cloud data can aid this determination. For example; if the data is range data from some scanner, then the method of collection can yield the proper orientation. The method we describe here does not require or assume that normal estimates are available.

A number of methods involve the signed distance function, $D(P)$, which is a trivariate function defined to be zero on the surface S , negative interior to S and positive outside of S . Typically, $D(P)$ is sampled on a 3D rectilinear grid and a method like the marching cubes algorithm [15, 18, 22] is used to extract a triangular mesh surface approximation. Once it is decided what the metric or the definition of distance from a surface to a point cloud is to be, it is usually not too difficult to develop algorithms for the efficient computation of the distance function. The particular difficulty here is getting the sign correct; that is, to be able to efficiently and effectively determine when a point is inside the surface or outside. Typical of the methods based upon distance functions is that of [14] where the sign is based upon local least squares estimates of the normal vector of the surface and a consistent orientation (in or out) is sought using estimates of global properties. One of the drawbacks to this method is the heuristics of the signed distance function calculation may lead to gaps in the surface and the difficulty of choosing the proper resolution for the marching cubes voxel grid can have detrimental effects on the success of the method.

Another important concept involved in point cloud fitting problems is the Delaunay Tetrahedrization and its dual, the Dirichlet tessellation and Voronoi diagram [20]. The method based upon alpha shapes of [10] is a typical and early example. Here the first step is the Delaunay tetrahedrization. The second step is to apply the alpha-erasure to remove tetrahedra, triangles and edges whose minimum surrounding sphere is not contained in the alpha-erasure sphere. The result is called the alpha-shape. In the third step, triangles for the final surface are selected so that a sphere of radius alpha containing the triangle does not contain any other point cloud points. The main negative aspect of this approach is the choice of a suitable value of alpha. Too big of a choice leads to poor approximations not utilizing many of the points of the point cloud and too small a choice leads to gaps and fragmented surfaces. Even though the methods of [6] and [7] guarantee for sufficiently dense samplings a surface that is homeomorphic and geometrically close to the point clouds resulting from sampling, in practice, it can be the case that point density varies considerably limiting the success for these methods for certain applications. [4] suggest

a postprocessing, topological clean up phase based upon linear programming “that can (to some extent) reconstruct non-smooth or undersampled surfaces.” We mention another potential drawback to these types of methods based upon DT in general for certain applications. The resulting triangular mesh surface has vertices that are points of the original point cloud. For noisy data or overlapping data resulting from imperfect registration of scanned data, this may not be undesirable. Rather than interpolated the point cloud (or a subset), it is potentially more desirable to approximate it for some applications. In addition it would be desirable to take advantage of the fact the many point cloud data sets have varying density. This leads us to the present method.

The method we describe here uses the general strategy of a top-down, adaptive least squares approach that allows for the complexity of the approximating surface to conform to the complexity of the data. We discuss the two main aspects of the algorithm consisting of the refinement strategy and the surface fitting criteria. Triangles are selected for refinement based upon a global error distribution are subdivided so as to optimize triangle shape properties while allowing overall quality fitting surfaces. Selecting vertex positions for optimal fitting surfaces is based upon a least squares method of measuring the error between surfaces and scattered point clouds.

We need the following notation in the subsequent sections. Let $P_i = (x_i, y_i, z_i)$, $i = 1, \dots, M$ denote the point cloud and $V_i = (u_i, v_i, w_i)$; $i = 1, \dots, N$ will denote the vertices of the triangular mesh surface that constitutes the approximation to the point cloud. In addition to the *geometry* of a triangular mesh surface S , consisting of a list of vertices $V_i = (x_i, y_i, z_i)$, $i = 1, \dots, N$, we have the *topology* consisting of a list of triple indices (i_n, j_n, k_n) , $n = 1, \dots, N_T$ indicating that the surface contains the triangles with vertices V_{i_n}, V_{j_n} and V_{k_n} . The list of edge indices is denoted by $e = \{(i, j) : \overline{V_i V_j} \in S\}$. If V_i is a vertex of S , then the *star* of V_i consists of all triangles of S that include V_i . We denote the star by $(V_i)^*$. The list of indices of the vertices of $(V_i)^*$ connected by edges to V_i is called the contiguity list which is denoted by $(i)^o$. The list of vertices $V_i, i \in (i)^o$ is called the one-ring of V_i . Sometimes the topology is specified by giving the contiguity list for each vertex rather than the list of triangles. These two methods are equivalent. It is also useful to define $(i)^* = \{i\} \cup (i)^o$ and the collection of triple indices that comprise the triangles of the star of a vertex, $\Delta^*(n) = \{(i, j, k) : T_{i,j,k} \subset (V_n)^*\}$.

2 Best Approximation and Characterization

A common method of measuring the distance between a collection of points, PtC , and a triangular mesh surface, S , is

$$\rho^2(S, PtC) = \sum_{i=1}^N \min_{V \in S} \|V - P_i\|^2 \quad (1)$$

which is illustrated in Fig. 3.

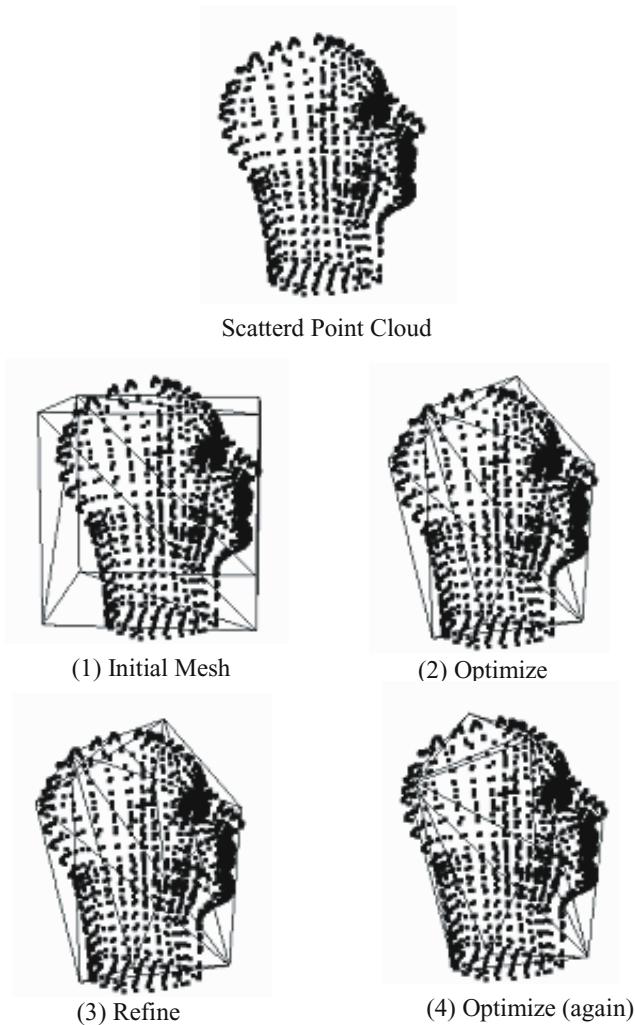


Fig. 2. The basic steps of the adaptive fitting scheme

While the method of (1) is useful for some applications, it has some problems in this context. This can be observed in the 2D diagram of Fig. 4. The polygon can actually be quite far from the point cloud and yet the distance computed by (1) is relatively small.

In order to alleviate this problem, we add some additional terms to our measure of distance which account for the distance from the polygon back to the scattered point cloud. See Fig. 5. Let

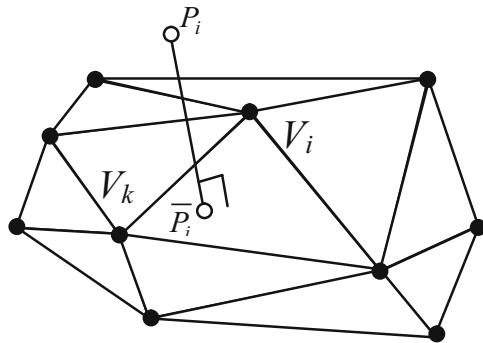


Fig. 3. The point \bar{P}_i is the closest point of S to P_i

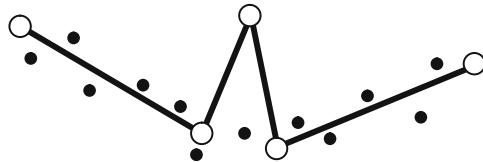


Fig. 4. This 2D example illustrates the problem of using (1) as a measure of distance between a scattered point cloud and a triangular mesh surface

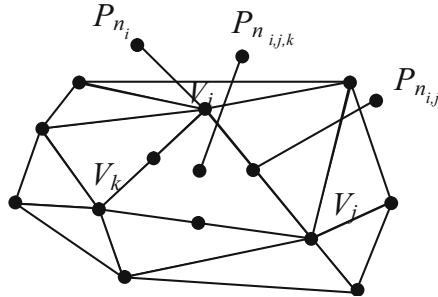


Fig. 5. Additonal points for error metric

P_{n_i} be the closest point of the point cloud to V_i ;

$P_{n_{i,j}}$ be the closest point to the midpoint $\frac{V_i + V_j}{2}$

$P_{n_{i,j,k}}$ be the closest point to the centroid $\frac{V_i + V_j + V_k}{3}$,

and define the error measure as

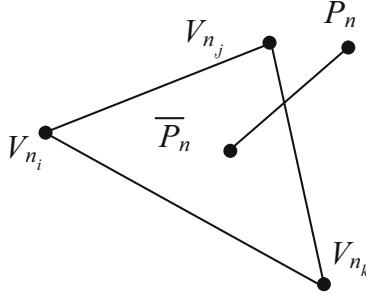


Fig. 6. Illustrating the notation for the triangle containing the point \bar{P}_n that is the closest point on S to P_n

$$\begin{aligned} \rho^2(S, PtC) = & \sum_{i=1}^N \min_{V \in S} \|V - P_i\|^2 + w_1 \sum_{i=1}^N \|V_i - P_{n_i}\|^2 \\ & + w_2 \sum_{i,j \in E} \left\| \frac{V_i + V_j}{2} - P_{n_{i,j}} \right\|^2 \\ & + w_3 \sum_{i,j,k \in T} \left\| \frac{V_i + V_j + V_k}{3} - P_{n_{i,j,k}} \right\|^2 \end{aligned} \quad (2)$$

We now discuss our approach to solving the problem

$$\min_{V_i} \rho^2(S, PtC). \quad (3)$$

A necessary conditions is

$$\frac{\partial \rho^2(S, PtC)}{\partial V_i} = 0. \quad (4)$$

Let \bar{P}_n be the closest point on S to P_n so that $\|\bar{P}_n - P_n\| \leq \|V - P_n\| \forall V \in S$ and let $(i)^\perp$ be the set of all indices whose projected point \bar{P}_n falls into the star of V_i , that is $(i)^\perp = \{n : \bar{P}_n \in (V_i)^\ast\}$. Let V_i, V_{n_j}, V_{n_k} be the triangle of the star of V_i that contains $\bar{P}_n, n \notin (i)^\perp$. We can utilize barycentric coordinates and write

$$\bar{P}_n = \alpha_{n,n_i} V_i + \alpha_{n,n_j} V_{n_j} + \alpha_{n,n_k} V_{n_k} \quad (5)$$

Using (5) we can see that necessary conditions of the vertices of a triangular mesh surface that minimizes the error to a point cloud given by (2) are

$$\begin{aligned} V_i = W \left\{ \sum_{n \in (i)^\perp} (P_{n_i} - \alpha_{n,n_j} V_{n_j} - \alpha_{n,n_k} V_{n_k}) \alpha_{n,n_i} \right. \\ \left. w_1 P_{n_i} + \frac{w_2}{2} \left(P_{n_{i,j}} - \frac{V_j}{2} \right) + \frac{w_3}{3} \left(P_{n_{i,j,k}} - \frac{V_j + V_k}{3} \right) \right\} \end{aligned} \quad (6)$$

where

$$W = \frac{1}{\sum (\alpha_{n,n_i} + w_1 + \frac{w_2}{4} + \frac{w_3}{9})}$$

In order to solve the equations of (6) we use a straightforward iterative method. That is, we successively compute

$$V_i^{(k+1)} = \Phi \left(V_1^{(k)}, \dots, V_{i-1}^{(k)}, V_{i+1}^{(k)}, \dots, V_M^{(k)} \right), \quad i = 1, \dots, M \quad (7)$$

where Φ represents the right hand side of (6).

We now summarize the overall strategy of our fitting algorithm.

1. Initialize $V_i^{(0)}$
2. Compute $P_{n_i}, P_{n_{i,j}}$ and $P_{n_{i,j,k}}$ as defined in (2)
3. Compute \tilde{P}_n and the barycentric coordinates defined in (5)
4. Do K steps of (6) or iterate to prescribed convergence tolerance.
5. Compute error contribution of each triangle. If the error distribution has a variance less than a user specified tolerance, δ , then a uniform refinement step is made else the triangle with the largest error is subdivided and the next step is 2.

In order to complete the description of the new adaptive fitting scheme, we need to describe what type of refinement strategy we will use. This is covered in the next section.

3 Refinement Strategy

We use two types of refinement strategies; one uniform and one adaptive. In the early stages of the fitting process it is more efficient to invoke the adaptive refinement scheme. As the fit progresses, the error tends to be more uniformly distributed over the mesh approximation and it can be more efficient to use a uniform refinement strategy.

3.1 Uniform Refinement

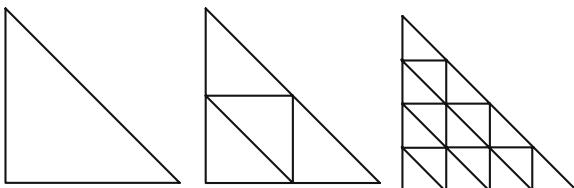


Fig. 7. Uniform refinement

3.2 Adaptive Refinement

The refinement strategy that we use is illustrated in Fig. 8 for a single triangle. It has previously been discussed by [24] and [25]. The order of the refinements of Fig. 8 are from left to right and from top to bottom. When a triangle is refined it is always split along its longest edge. If a neighboring triangle must also be refined in order to avoid the cracking problem, this neighbor is also refined by splitting its longest edge even if the common edge to its just split neighbor is not its longest edge. Splitting one triangle may cause refinement to permeate out through the triangular grid. The number of permeations is bound by the number of levels present in the triangular mesh. See [23] for more details and implementation strategies on this type of refinement.

The overall oracle for deciding whether to use adaptive refinement (and which triangle to refine) or uniform refinement is a variable in our general approach. For the examples of this paper, we use a greedy strategy of adaptively refining the triangle with the largest error contribution from (4). A uniform refinement is made if the variance of the error contributions over all triangles is less than a user specified parameter, δ .

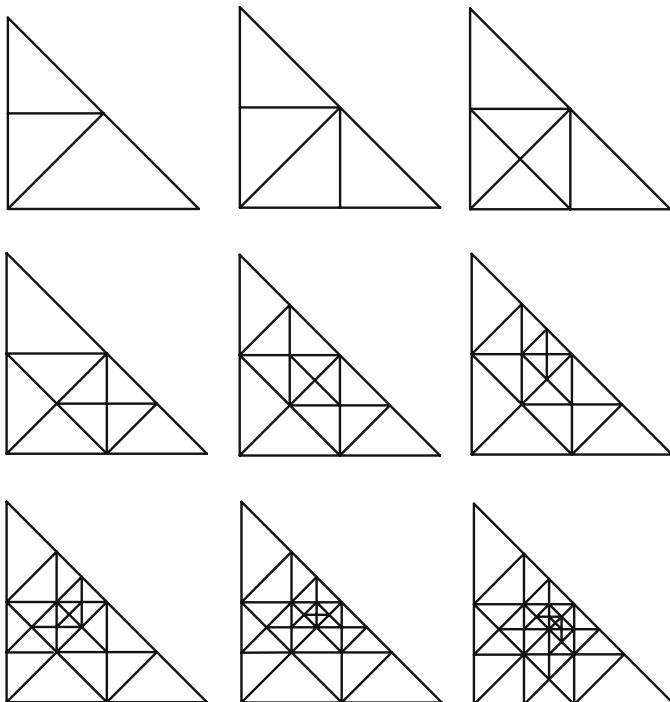


Fig. 8. Illustrating the adaptive refinement strategy

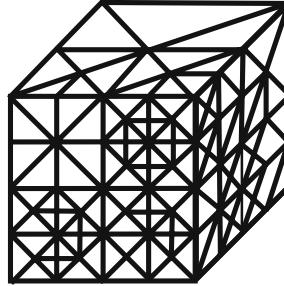


Fig. 9. An example of the adaptive refinement strategy with a cube as the initial domain

4 Examples

In this section we illustrate our adaptive fitting technique with several examples. The first two examples are “proof of concept” examples and use contrived data. The third example uses data that we collected in our own lab using an LDI scanner. The data of the fourth example is made available by Geomagic and illustrates the ability of our new method to handle noisy data with missing regions of data and data that is not very uniformly sampled.

Example 1. Simple, Genus Zero, Geometric Object

For this first example, the scattered point cloud consists of 64K, randomly perturbed points sampled from a simple geometric object shown in the upper left image of Fig. 10. The initial triangular mesh approximation with 52 vertices is shown in the left image of the middle row of Figure N. Using the values $w_1 = 1.0$, $w_2 = 0.1$, $w_3 = 1.0$, $K = 5$, $\delta = 0.1$ to obtain the approximations with 150, 300 and 700 vertices shown in the right image of the middle row and the bottom row of Fig. 10 respectively. From this example we clearly see how the complexity of the approximation conforms to the complexity of the data.

Example 2. Geometric Object With Genus 3

This next example utilizes data sampled from an object with genus three. The geometric object shown in the upper left image of Fig. 11 is sampled and perturbed to yield the scattered point cloud shown in the upper right image of Fig. 11. The initial approximation of the left image of the center row of Fig. 11 has 90 vertices. The subsequent three stages of the fitting process have 300, 500 and 2000 vertices respectively. Here we have used $w_1 = 1.0$, $w_2 = 1.0$, $w_3 = 1.0$, $K = 3$, $\delta = 0.01$

Example 3. Rubber Duck Example

For this next example, we have gathered a scattered point cloud by using a free-hand LDI scanner. The data collection process is illustrated in Fig. 12. The laser scanner

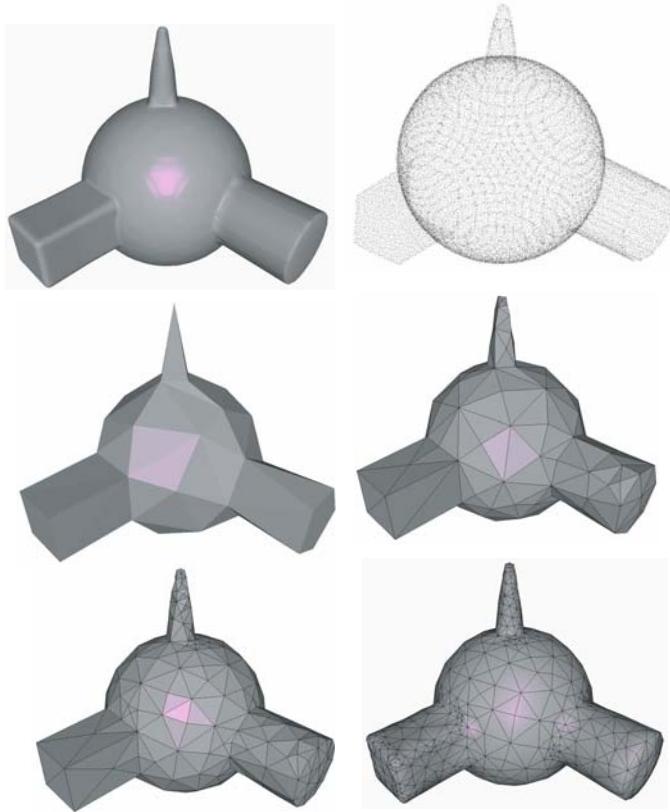


Fig. 10. Images of the *top* row show object and sampled, perturbed scattered point cloud. The initial approximation is shown in the left image of the *center* row. Approximations with 150, 300 and 700 vertices are shown in the right image of the center row and the *last* row, respectively

collects a rectangular grid of data points representing distances to the object. We do not know the position nor orientation of the data relative to the object or any other coordinate system. We arbitrarily pick one scan and use this as the base 3D coordinate system. All other scans are mapped to this coordinate system by registering the data. We do not go into the details of the registration process here as what we did was not particularly innovative. Mainly it was an interactive, by-hand process. The overall quality of the data is certainly dependent upon how well the registration is done, but since we are using a least squares fitting process, a certain amount of bounded error is tolerable. This is one of the nice properties of the present method that is not shared by some of the other fitting strategies.

The initial approximation, which is shown in the left image of Fig. 13, was obtained in the following manner. A voxel grid of resolution $N \times M \times K$ is placed over the domain of the scattered point cloud domain. If a voxel contains points of the scat-

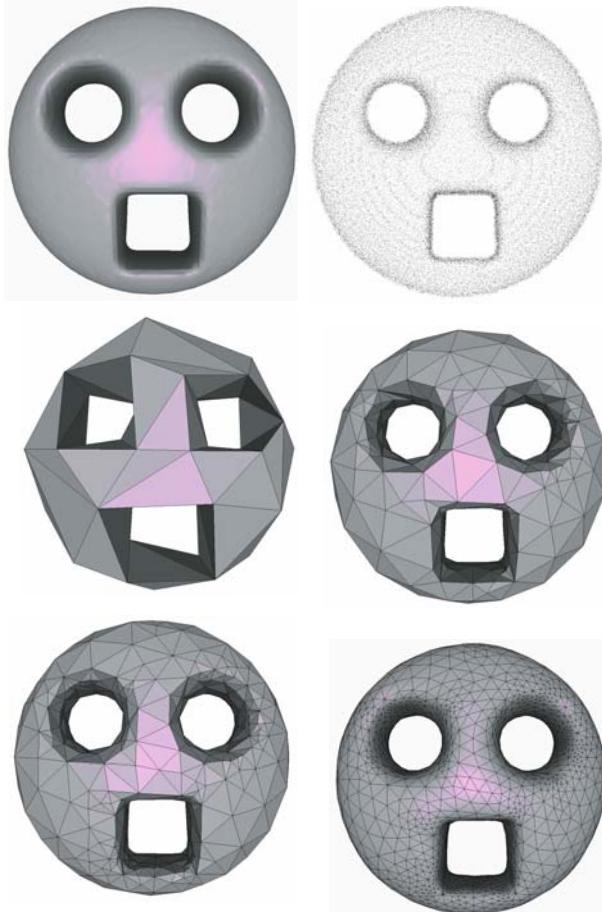


Fig. 11. The images of the *top* row show the object of genus three and points, randomly sampled and perturbed from this object. The initial triangular mesh approximation is shown in the left image of the *middle* row, followed by approximations with 300, 500 and 2000 vertices

tered point cloud is marked as “occupied” and given a value of 1. All other voxels are marked “empty” and give a value of 0. The marching cubes isosurface that separated the occupied voxels from the empty voxels serves as the initial triangular mesh surface. In the middle image of Fig. 13, the first iteration of the fitting process is shown. The converged triangular mesh with the same topology as the initial approximation of the left image is shown in the right image. The success of this automated method of obtaining an initial approximation is, of course, dependent upon the selection of the voxel grid. We make this selection interactively and the whole process works rather well.

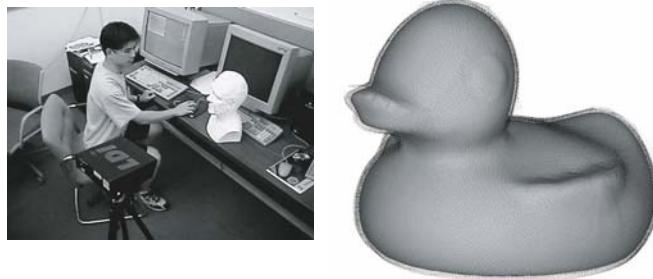


Fig. 12. Data collection and the resulting scattered point cloud

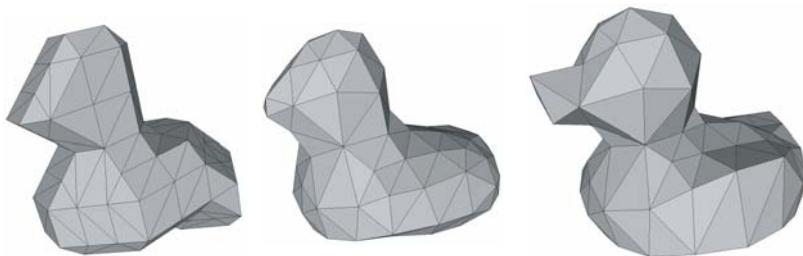


Fig. 13. Initial approximation, one fitting iteration and converged approximation

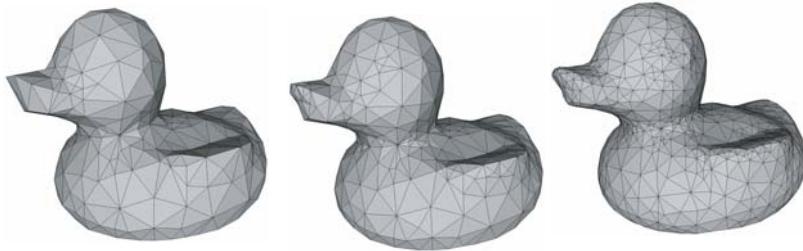


Fig. 14. Steps of the adaptive fitting process with models having 250, 500 and 1000 vertices

Example 4. Noisy Foot Example

This example serves to illustrate two additional properties of the new fitting scheme; namely that holes or missing data can be handled and the uniform splitting step of Fig. 7. In this example we have set the parameter for doing a uniform split at a fairly large value and so that it is more likely to do uniform split. A close examination of the point cloud shown in Figure 16. will reveal the missing data at the heel and near the balls of the foot. Also the data is not very uniformly distributed. All of these properties are not obstacles to the present method of fitting. The approximations shown in Fig. 17 utilize $w_1 = 9.13$, $w_2 = .43$ and $w_3 = 1.0$, $K = 2$ and $\delta = 0.5$.

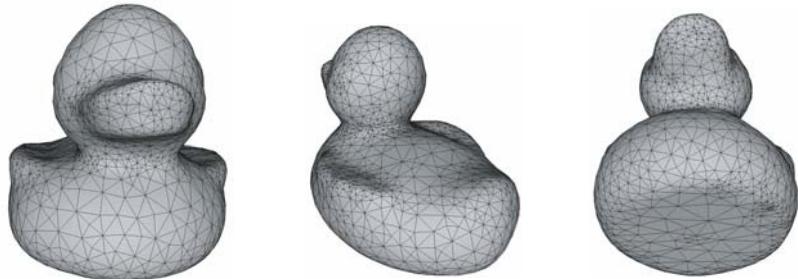


Fig. 15. Three views of the final approximation with 2000 vertices illustrating the property that the complexity of the model conforms to the complexity of the shape implied by the scattered point cloud data



Fig. 16. This data is courtesy of Geomagic. The data is noisy, the sampling density is not uniform and there are regions of missing data near the heel, the balls and the ankle

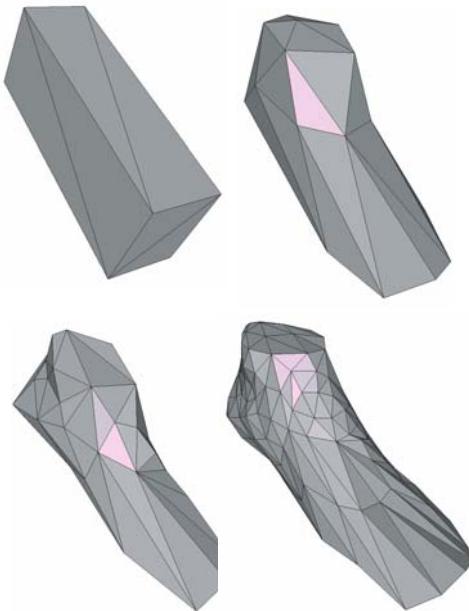


Fig. 17. The initial approximation consists of a triangulated parallelepiped. The *upper right* is the best fitting surface with 20 vertices and the lower left is the optimal fit with 36 vertices. A uniform split occurs after this fit and the optimal fit with 138 vertices is shown in the *lower right* image

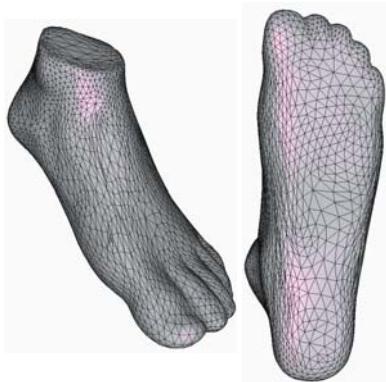


Fig. 18. Two views of the approximation with 3002 vertices. Note how the regions with missing data have been filled in because the topology does not change from the initial approximation

Acknowledgments

We wish to acknowledge the support of the Office of Naval Research (N00014-97-1-0243 & N00014-00-1-0281), the National Science Foundation (NSF IIS-9980166 & ACI-0083609) and DARPA (MDA972-00-1-0027). Thanks to T. Roxborough and R Holmes for their help. Thanks to Geomagic for the noisy foot data.

References

1. Kagan, A.M., Linnik, Y.V., Rao, C.R.: Characterization Problems in Mathematical Statistics. Wiley, New York (1973)
2. Meyer, P.A.: A short presentation of stochastic calculus. In: Emery, M. (ed) Stochastic Calculus in Manifolds. Springer, Berlin Heidelberg New York (1989)
3. Miller, B.M., Runggaldier, W.J.: Kalman filtering for linear systems with coefficients driven by a hidden Markov jump process. *Syst. Control Lett.*, **31**, 93–102 (1997)
4. Adamy, U., Giesen, J., John, M. 2000. New techniques for topologically correct surface reconstruction, In: Proceedings of IEEE Visualization 2000, IEEE Press, 372–380.(2000)
5. Algorri, M.-E. and Schmitt, E. Surface reconstruction from unstructured 3D data, *Computer Graphics Forum* **19**, No. 3, Proceedings of EUROGRAPHICS, 69–81. (2000)
6. Amenta, N. and Bern, M., Surface reconstruction by Voronoi filtering, *Discrete & Computation Geometry* **22**, 4, 481–504. (1999)
7. Amenta, N., Bern, M., and Kamvysselis, M. A new Voronoi-based surface reconstruction algorithm, In: Proceedings of SIGGRAPH 1998, ACM Press/ACM SIGGRAPH, New York. Computer Graphics Proceedings, Annual Conference Series, ACM, 39–48. (1998)
8. Bernardini, F., Silva, C., Mittleman, J., Rushmeier, H. and Taubin, G. The ball-pivoting algorithm for surface reconstruction, *IEEE Transactions on Visualization and Computer Graphics* **5**, 4, 123–143. (1999)
9. Curless B. and Levoy, M. A volumetric method for building complex models from range images, In: Proceedings of SIGGRAPH 1996, ACM Press/ACM SIGGRAPH, New York. Computer Graphics Proceedings, Annual Conference Series, ACM, 303–312. (1996)
10. Edelsbrunner, H. and Mücke, E.P. 1994. Three-dimensional alpha shapes, *ACM Transactions on Graphics* **13**, No. 1, 43–72. (1994)
11. Guo, B., Surface reconstruction: From points to splines, *CAGD* **29**, No. 2, 269–277. (1997)
12. Floater, M.S. Parameterization of triangulations and unorganized points, In: *Tutorials on Multiresolution in Geometric Modelling*, eds. A. Iske, E. Quak and M.S. Floater, Springer, 287–316. (2002)
13. Franke, R. and Nielson, G.M. Scattered data interpolation and applications: A tutorial and survey, In: *Geometric Modelling: Methods and Their Applications*, eds. H. Hagen & D. Roller, Springer, 131–160. (1990)
14. Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. and Stuetzle, W. Surface reconstruction from unorganized points, In: Proceedings of SIGGRAPH 1992, ACM Press/ACM SIGGRAPH, New York. Computer Graphics Proceedings, Annual Conference Series, ACM, 71–78. (1992)
15. Lorensen, W.E. and Cline, H.E. 1987. Marching cubes: A high resolution 3D surface construction algorithm, In: Proceedings of SIGGRAPH 87, ACM Press/ACM SIGGRAPH, New York. Computer Graphics Proceedings, Annual Conference Series, ACM, 163–169. (1987)

16. Müller, H. Surface Reconstruction – An introduction, In: Scientific Visualization, H. Hagen, G. Nielson, F. Post, eds. IEEE Computer Society Press, 239–242. (1999)
17. Morse, B., Yoo, T.S., Rheingans, P., Chen, D.T. and Subramanian K. R. Interpolating Implicit Surfaces from Scattered Surface Data Using Compactly Supported Radial Basis Functions. In: Proceedings of International Conference on Shape Modeling and Applications '01, IEEE Computer Society Press, 89–98. (2001)
18. Nielson, G.M., and Hamann, B. The asymptotic decider: Resolving the ambiguity in marching cubes, In: Proceedings of Visualization '91, IEEE Computer Society Press, 83–91. (1991)
19. Nielson, G.M. Scattered data modelling, Computer Graphics and Applications **13**, *1*, IEEE Press, 60–70. (1993)
20. Nielson, G.M. Tools for Triangulations and Tetrahedrizations, Scientific Visualization, Volume modeling, In: Scientific Visualization, G. Nielson, H. Hagen & H. Mueller, Eds., IEEE CS Press, 429–526. (1997)
21. Nielson, G.M. Volume modeling, In: Volume Graphics, M. Chen, A.E. Kaufman and R. Yagel, Eds., Springer, 29–48. (2000)
22. Nielson, G.M. On Marching Cubes, IEEE Transactions on Visualization and Computer Graphics N, Vol. **9**, No. **3**, IEEE Press, 283–297. (2003)
23. Nielson, G.M. and Roxborough, T., Tetrahedron based, least squares, progressive volume models with applications to freehand ultrasound data, In: Proceedings of the conference on Visualization '00, IEEE Computer Society Press, pp. 93–100. (2000)
24. Maubach, J.M. Local bisection refinement for N-simplicial grids generated by reflection, SIAM Journal of Scientific Computing **16**, 210–227. (1995)
25. Rivara, M.C. Local Modification of meshes for adaptive and/or multigrid finite element methods, Journal of Computaiton and Applied Mathematics **36**, 79–89. (1991)
26. Savchenko, V.V., Pasko, A.A., Okunev, O.G., and Kunii, T.L., Function representation of solids reconstructed from scattered surface points and contours, Computer Graphics Forum, Vol. **14**, No. **4**, 181–188. (1995)
27. Schumaker, L. Fitting surfaces to scattered data, In: Aprorixmation Theory II, G.G. Lorentz, C.K. Chui and L.L. Schumaker, eds., 203–268. (1976)
28. Turk, G. and Levoy, M. Zippered polygon meshes from range images, In: Proceedings of SIGGRAPH 1994, ACM Press/ACM SIGGRAPH, New York. Computer Graphics Proceedings, Annual Conference Series, ACM, 311–318. (1994)
29. Zhao, H.-K., Osher, S., Fedkiw, R., Fast Surface Reconstruction Using the Level Set Method, In: 1st IEEE Workshop on Variational and Level Set Methods, in conjunction with the 8th International Conference on Computer Vision (ICCV), Vancouver, Canada, 194–202. (2001)
30. Xie, H., Wang, J. Hua, J., Qin, J., Kaufman, A., Piecewise C1 continuous surface reconstruction of noisy point clouds via local implicit quadric regression, In: Proceedings of IEEE Visualization 2003, IEEE Press, 91–98.

Part II

Volume Visualization and Medical Visualization

Ray Casting with Programmable Graphics Hardware

Manfred Weiler¹, Martin Kraus², Stefan Guthe³,
Thomas Ertl¹, and Wolfgang Straßer³

¹ Institute of Visualization and Interactive Systems, University of Stuttgart

² PURPL, Purdue University

³ WSI/GRIS, University of Tübingen

Summary. The flexible programmability introduced with the latest graphics chip generations has led to a new class of hardware-accelerated volume rendering algorithms implementing ray casting on the graphics processor. Three independent publications have presented this idea in the context of volume rendering for uniform meshes and tetrahedral meshes. This paper provides a more general survey of the topic. In reviewing the background, extracting the basic techniques and mechanisms common to all three approaches, and highlighting the differences, we hope to provide a comprehensive discussion of the subject.

1 Introduction

Interactive volume rendering has always been one of the most challenging techniques in computer graphics; therefore, algorithms suited for a hardware-accelerated implementation have been and still are of particular interest. For example, the linear interpolation provided by triangle rasterization is exploited by the Shirley-Tuchman cell projection [21] and hardware-accelerated texture mapping is exploited by texture-based volume rendering [3].

Today, programmable graphics hardware is available allowing us to implement much more complex algorithms directly on the graphics hardware, thus, removing the bottleneck of the communication between the graphics hardware and the CPU. Moreover, modern graphics hardware offers a high degree of parallelism that is very well suited for particular algorithms. For example, the fragments of slices for texture-based volume rendering are processed independently; therefore, this technique benefits directly from the parallel architecture of modern graphics hardware. In contrast, an efficient parallel implementation of the visibility sorting required for cell projection is rather difficult to achieve.

Ray casting is an efficient volume rendering technique, which is also very well suited for parallel implementations since the rays are processed independently. However, the basic algorithm requires certain optimizations (e.g., early ray termination, empty space leaping, adaptive sampling distance, etc.) in order to work efficiently. Unfortunately, hardware-based implementations of these optimizations are usually

impossible. (In fact, texture-based volume rendering may be considered as a very simple ray casting algorithm with constant sampling distance, no empty space leapfrogging, and no early ray termination.) Therefore, ray casting in volume rendering was restricted to software-based, non-interactive implementations [5] or custom hardware designs [15] in the past. This situation, however, has dramatically changed recently.

Modern programmable graphics hardware allows us to implement many of the most important ray casting optimizations, in particular early ray termination, empty space leapfrogging, and adaptive sampling distance. It also allows us to employ improved ray integration schemes, e.g., pre-integrated classification. Moreover, ray casting on programmable graphics hardware is not restricted to uniform grids but may also be applied to unstructured meshes, e.g., tetrahedral meshes.

This paper summarizes three approaches to the implementation of ray casting on programmable graphics hardware, published in [8, 17, 22]. The common concept is to trace all rays in parallel by rasterizing many screen-filling polygons. Several optimization techniques are implemented in very similar ways, e.g., early ray termination by early depth tests, while other techniques are implemented differently, e.g., the computation of the sampling distance. Furthermore, ray casting in tetrahedral meshes offers several additional challenges, e.g., non-convex mesh boundaries and non-trivial mesh connectivity.

In Sect. 2, we present the basic ideas and concepts of ray casting algorithms for volume visualization, in particular the volume rendering integral, classification schemes, and sampling strategies. We also compare ray casting with other volume rendering algorithms. This comparison shows the importance of optimizations for volume rendering algorithms since ray casting algorithms are not necessarily more efficient than, for example, object-order algorithms. Sect. 3 gives a first overview of the three approaches to ray casting of volume data using programmable graphics hardware. Optimization techniques for ray casting algorithms and their implementation on programmable graphics hardware are discussed in Sect. 4.

2 Ray Casting for Volume Visualization

As depicted in Fig. 1, the basic idea of ray casting algorithms for volume visualization is to trace viewing rays through a volumetric scalar field. Along each ray, colors and opacities are calculated at discrete sampling points and composited in order to compute the total color of each ray. Usually, at least one ray is traced for each pixel and the final image is generated pixel by pixel, i.e., in *image-order*. Ray casting algorithms are among the earliest volume rendering algorithms; see for example [5, 10].

2.1 Volume Rendering Integral

The basic task of any volume renderer is an (approximate) evaluation of the *volume rendering integral* for each pixel, i.e., the integration of attenuated colors and extinction coefficients along each viewing ray. Although this is obvious for ray casting algorithms, note that the volume rendering integrals do not necessarily have to

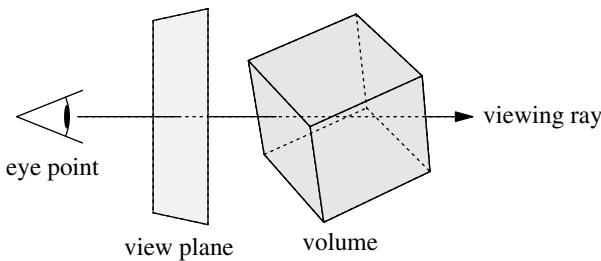


Fig. 1. Direct volume rendering with ray casting: One viewing ray is traced for each pixel

be evaluated one by one; rather the integrals for all pixels may be evaluated simultaneously; therefore, ray casting algorithms are particularly well suited for parallel computers.

We assume that the viewing ray $\mathbf{x}(\lambda)$ is parameterized by the distance λ to the eye point. A scalar data value at point $\mathbf{x}(\lambda)$ is denoted as $s(\mathbf{x})$. Color and extinction densities are computed from this scalar value by applying transfer functions $\tilde{c}(s)$ and $\tau(s)$, i.e., the color density at point \mathbf{x} is $\tilde{c}(s(\mathbf{x}))$ and the extinction density is $\tau(s(\mathbf{x}))$. This process is often referred to as *classification* of the scalar data. The units of color and extinction densities are color intensity per length and extinction strength per length, respectively. However, we will refer to them as colors and extinction coefficients when the precise meaning is clear from the context.

We write the volume rendering integral as

$$I = \int_0^D \tilde{c}\left(s(\mathbf{x}(\lambda))\right) \exp\left(-\int_0^\lambda \tau\left(s(\mathbf{x}(\lambda'))\right) d\lambda'\right) d\lambda . \quad (1)$$

with the maximum distance D , i.e., there is no color density for λ greater than D . In words, color is emitted at each point \mathbf{x} according to the function $\tilde{c}(s(\mathbf{x}))$, and attenuated by the integrated extinction coefficients $\tau(s(\mathbf{x}))$ between the eye point and the point of emission.

2.2 Pre- and Post-Classification

Direct volume rendering techniques differ considerably in the way they evaluate (1). One important and very basic difference is the computation of $\tilde{c}(s(\mathbf{x}))$ and $\tau(s(\mathbf{x}))$. The scalar field $s(\mathbf{x})$ is usually defined by a mesh with scalar values s_i specified at each vertex \mathbf{v}_i of the mesh in combination with an interpolation scheme.

The ordering of the two operations, interpolation and the application of transfer functions, defines the difference between *pre-* and *post-classification*. Post-classification is characterized by the application of the transfer functions *after* the interpolation of $s(\mathbf{x})$ from the scalar values at several vertices (as suggested by (1));

while pre-classification is the application of the transfer functions *before* the interpolation step, i.e., colors $\tilde{c}(s_i)$ and extinction coefficients $\tau(s_i)$ are calculated in a pre-processing step for each vertex v_i and then used to interpolate $\tilde{c}(s(\mathbf{x}))$ and $\tau(s(\mathbf{x}))$ for the computation of the volume rendering integral.

2.3 Numerical Integration

The most common numerical approximation of the volume rendering integral in (1) is the calculation of a Riemann sum for n equal ray segments of length $d := D/n$. (See Fig. 2a and Section IV.A in [13].) It is straightforward to generalize the following considerations to unequally spaced ray segments.

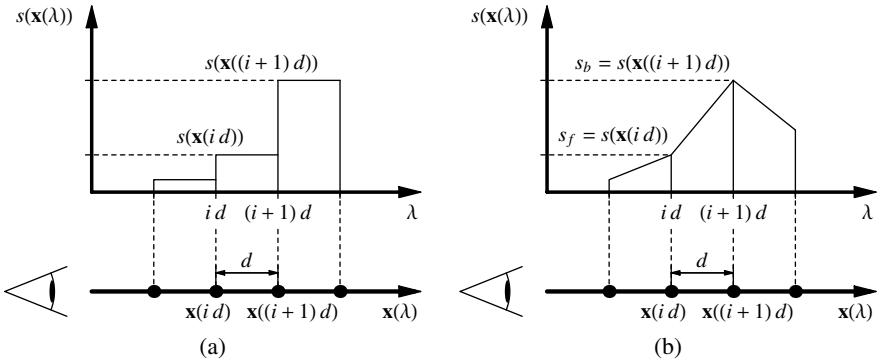


Fig. 2. (a) Piecewise constant approximation of the function $s(\mathbf{x}(\lambda))$ along a viewing ray
(b) Piecewise linear approximation of $s(\mathbf{x}(\lambda))$

We define the *opacity* α_i of the i -th ray segment by

$$\alpha_i := 1 - \exp\left(-\int_{i d}^{i(d+1)} \tau(s(\mathbf{x}(\lambda'))) d\lambda'\right) \quad (2)$$

and approximate it by

$$\alpha_i \approx 1 - \exp\left(-\tau(s(\mathbf{x}(i d))) d\right). \quad (3)$$

This approximation assumes a piecewise constant value of $s(\mathbf{x}(\lambda))$ as illustrated in Fig. 2a. The result is often further approximated to

$$\alpha_i \approx \tau(s(\mathbf{x}(i d))) d. \quad (4)$$

$1 - \alpha_i$ will be called the *transparency* of the i -th ray segment.

Similarly, the color \tilde{C}_i emitted in the i -th ray segment is defined by

$$\tilde{C}_i := \int_{id}^{i(d+1)} \tilde{c}(s(\mathbf{x}(\lambda))) \exp\left(-\int_{id}^{\lambda} \tau(s(\mathbf{x}(\lambda'))) d\lambda'\right) d\lambda . \quad (5)$$

Neglecting the self attenuation within a ray segment \tilde{C}_i may be approximated by

$$\tilde{C}_i \approx \tilde{c}(s(\mathbf{x}(id))) d . \quad (6)$$

Thus, the approximation of the volume rendering integral in (1) is

$$I \approx \sum_{i=0}^{\lfloor D/d \rfloor} \tilde{C}_i \prod_{j=0}^{i-1} (1 - \alpha_j) . \quad (7)$$

Therefore, a front-to-back compositing algorithm will implement the equation

$$\tilde{C}'_i = \tilde{C}'_{i-1} + (1 - \alpha'_{i-1}) \tilde{C}_i \quad \text{and} \quad \alpha'_i = \alpha'_{i-1} + (1 - \alpha'_{i-1}) \alpha_i , \quad (8)$$

where the color accumulated in the first i ray segments is denoted by \tilde{C}'_i and the accumulated opacity is denoted by α'_i .

Substituting $\tilde{c}(s)$ by $\tau(s)c(s)$ and employing the approximation

$$C_i \approx \tau(s(\mathbf{x}(id))) c(s(\mathbf{x}(id))) d \quad (9)$$

will result in the more common approximation

$$I \approx \sum_{i=0}^n \alpha_i C_i \prod_{j=0}^{i-1} (1 - \alpha_j) \quad (10)$$

with the corresponding front-to-back compositing equations

$$\tilde{C}'_i = \tilde{C}'_{i-1} + (1 - \alpha'_{i-1}) \alpha_i C_i \quad \text{and} \quad \alpha'_i = \alpha'_{i-1} + (1 - \alpha'_{i-1}) \alpha_i . \quad (11)$$

This compositing equation indicates that \tilde{C} corresponds to a *pre-multiplied color* αC ; which is also called *opacity-weighted color* (see [25]) or *associated color* (see [2]).

The compositing of colors described by (8) and (11) is often terminated as soon as α'_i has reached some threshold close to 1 because the contribution of colors from more distant samples is negligible if they are multiplied with $(1 - \alpha'_i) \approx 0$. This acceleration technique is usually referred to as early ray termination or adaptive ray termination; see for example [11].

The sampling distance (or sampling length) does not need to be constant. Often, the sampling distance is adaptively increased in order to quickly skip empty space. This technique is known as empty space leaping; see for example [11].

For unstructured meshes, the sampling points are usually chosen at the intersections of the viewing ray with cell boundaries (see for example [5]); thus, the sampling distances depend on the geometry of the unstructured mesh; see Fig. 3b.

2.4 Pre-Integrated Classification

The discrete approximation of the volume rendering integral will converge to the correct result for $d \rightarrow 0$, i.e., for high sampling rates $1/d$. According to the sampling theorem, a correct reconstruction is only possible with sampling rates greater than the Nyquist frequency. However, non-linear features of transfer functions may considerably increase the sampling rate required for a correct evaluation of the volume rendering integral as this sampling rate depends on the product of the Nyquist frequencies of the scalar field $s(\mathbf{x})$ and the maximum of the Nyquist frequencies of the two transfer functions $\tilde{c}(s)$ and $\tau(s)$ (or of the product $c(s)\tau(s)$; see also [20]).

In order to overcome this problem, the approximation of the volume rendering integral has to be improved. With *pre-integrated classification* high sampling frequencies are avoided by reconstructing a piecewise linear, continuous scalar function along the viewing ray (see Fig. 2b), and evaluating the volume rendering integral between each pair of successive samples of the scalar field by lookups in a precomputed table. This allows us to avoid the problematic product of Nyquist frequencies since the sampling rate for the reconstruction of the scalar function along the viewing ray does not depend on the transfer functions.

Pre-integrated classification was first published for cell projection in [18] by Röttger et al. and was later adapted for texture-based volume rendering in [4] by Engel et al. The first application to ray casting was published by Knittel in [6].

One drawback of this technique introduced by the relatively expensive computation of the pre-integration table is the lacking possibility to interactively change the transfer function, which was addressed by an approximative computation in [4]. Recently an acceleration technique has been published by Lum et al. [12] that allows for the accurate computation of the precomputed table at interactive rates without any approximation.

2.5 Comparison with Other Volume Rendering Algorithms

In contrast to image-order algorithms, *object-order* algorithms process part after part of the volumetric data. Each part, usually a cell of a volumetric mesh, can contribute to the color of many pixels; however, there are very efficient ways to compute all contributions of a single cell of the mesh. One of the most important object-order algorithms in volume rendering is cell projection, in particular the projection of tetrahedra as first suggested by Shirley and Tuchman in [21]. However, cells of other shapes may also be projected to the view plane. Object-order volume rendering algorithms also include splatting (suggested by Westover in [24]) and the shear-warp algorithm (see for example [9]).

The availability of hardware-accelerated texture mapping led to texture-based volume rendering algorithms, see for example Cabral et al. in [3]. Although texture-based volume rendering using two-dimensional textures is conceptually very similar to the shear-warp algorithm, it has to be classified as an image-order algorithm since the number of texture lookups depends on the image resolution instead of the number of voxels. If the slice distance is chosen to correspond to the size of a voxel, we

might think of the algorithm as working in object-order in one dimension and in image-order in the other two (image) dimensions.

The time complexity of a naive object-order projection of all cells of a mesh is linear in the number of cells. Moreover, the time complexity of most object-order volume rendering algorithms, e.g., splatting or the Shirley-Tuchman cell projection, is also linear in the number of rasterized fragments. However, some object-order algorithms are only linear in the number of pixels covered by the final image; for example, the shear-warp algorithm, which requires only one warp of the final image. Also note that the constants for the linear dependency on the number of fragments or pixels is usually very small since very efficient rasterization techniques can be employed.

The time complexity of image-order algorithms, in particular ray casting and texture-based approaches, does not depend on the total number of cells but on the number of cells actually intersected by viewing rays, assuming a constant time access to the data. Thus, for large meshes image-order algorithms are in theory preferable. Unfortunately, for implementations based on graphics hardware, all the data should fit into texture memory; otherwise, these approaches are far less efficient. Moreover, the theoretical advantage of image-order algorithms is in practice often compensated by choosing a higher resolution for the generated images for larger meshes. Thus, neither object-order nor image-order volume rendering algorithms are very efficient for large meshes. Therefore, optimization techniques are of particular interest.

Empty space leaping is a standard technique for image-order algorithms. For object-order algorithms it corresponds to culling empty cells, for example by simply testing each cell and skipping the projection of completely transparent cells. Hierarchical data structures may also be employed to avoid any empty cell processing. Thus, image-order and object-order algorithms benefit similarly from empty space leaping in this wider sense.

Early ray termination is also a standard technique for image-order algorithms. In general, it is very difficult to apply it to object-order algorithms; the only exception being the shear-warp algorithm. Thus, for volume visualizations with many opaque regions, e.g., opaque isosurfaces, image-order algorithms can often perform considerably better than object-order algorithms.

Hierarchical data structures for different level-of-detail representations of volume data have been published for software implementations of image-order and object-order volume rendering algorithms. However, for hardware-based implementations any additional data structures are usually problematic since they also require scarce texture memory. Note that an adaptive computation of sampling distances combined with pre-integrated classification can achieve a similar effect for image-based volume rendering algorithms.

3 Ray Casting Based on Programmable Graphics Hardware

3.1 Common Concepts

The basic principle of a ray casting implementation based on programmable graphics hardware is to trace all viewing rays from front to back at the same time in order to exploit the parallel architecture of modern rasterization hardware; see Fig. 3. In an initialization step, typically performed by rendering the non-occluded boundary faces of the mesh, the first intersection of each ray with the mesh is computed. After this, the rays are consecutively propagated from one sampling point to the next until the whole volume has been processed. Note that several (or even all) of these propagation steps can be combined in one rasterization pass depending on the capabilities of the graphics hardware.

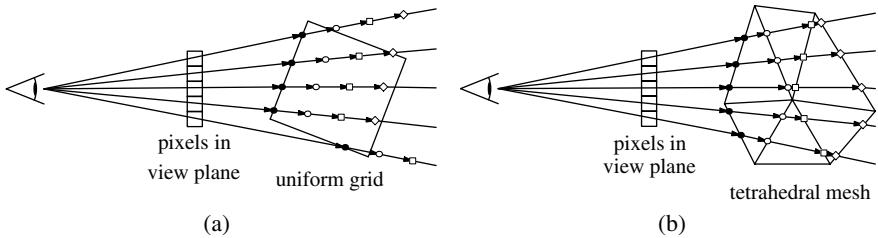


Fig. 3. (a) Ray casting in uniform grids: The initial intersections with the boundary are marked with dots (●); further sampling points with circles (○), squares (□), and diamonds (◇). (b) Same as (a) for tetrahedral meshes: The sampling points are at the intersections of the viewing rays with the cell boundaries

Intermediate information on the current sampling point of each viewing ray is stored in the frame buffer and in several two-dimensional textures of the same dimensions as the frame buffer, which are used as additional output targets for the rasterization. Each of the viewing rays corresponds to exactly one pixel of the frame buffer and one texel in the supplemental textures. Computations for all viewing rays are performed by rasterizing screen-filling rectangles into the texture maps. The per-pixel computations are described by so called fragment shaders or fragment programs, which are loaded onto the graphics hardware. They implement the following main tasks:

1. Initialization of the textures with the first intersection of the viewing rays with the boundary of the mesh.
2. Computation of the next sampling point for each texel.
3. Sampling of the mesh data, classification of the scalar value, and compositing of the colors for each texel based on the computed extinction coefficient.

This basic scheme is the same for all published ray casters implemented on programmable graphics hardware [8, 17, 22]. In fact, it is also similar to ray tracing with

programmable graphics hardware as suggested by Purcell et al. in [16]. Moreover, all three ray casters implement early ray termination by early depth tests.

3.2 Overview of the Differences

Although the three approaches reviewed in this paper are all based on the ray casting algorithm and employ similar graphics hardware, they differ considerably in several ways. Looking at Table 1, the most important difference is the kind of volumetric meshes these systems visualize. While the ray caster published by Weiler et al. [22] works on tetrahedral meshes; both, the ray casters described by Röttger et al. [17] and the system published by Krüger and Westermann [8], render uniform meshes.

Table 1. Overview of the differences and common features of the presented ray casting algorithms

Algorithm	Grid Type	Adaptive Sampling (Sects. 4.1 and 4.2)	Classification (Sect. 4.3)	Early Ray Termination (Sect. 4.4)
Krüger [8]	uniform	fixed distance + empty space leaping	post-classification	yes
Röttger [17]	uniform	data dependent	pre-integration	yes
Weiler [22]	tetrahedral	intersection with cell boundaries	pre-integration	yes

Therefore, the sampling distance is determined in different ways: The system by Weiler et al. computes intersections with cell boundaries in order to determine the sampling points, while the ray caster described by Röttger et al. computes an adaptive sampling distance depending on the data at the sampling point. The system by Krüger and Westermann implements a fixed sampling distance but optimized by empty space leaping.

Furthermore, the three systems employ different classification schemes. Weiler et al.'s system employs pre-integrated classification, which is the obvious choice for tetrahedral meshes since the linear interpolation of scalar data values between two sampling points on the boundary of one tetrahedron is well justified. Although the system published by Röttger et al. computes the sampling distance differently and trilinear interpolation is employed for the interpolation of data values, it also implements pre-integrated classification in order to improve the image quality. Both systems use similar methods to compute the required look-up tables efficiently; a detailed description is given in [22]. Meanwhile, an even faster acceleration technique based on incremental computation has been published in [12]. The system by Krüger and Westermann also employs trilinear interpolation for the scalar data but implements traditional post-classification.

All approaches store volumetric data in the local memory of the graphics adapter. However, uniform meshes are stored as three-dimensional texture maps similar to classical texture-based volume rendering algorithms, whereas for the representation of tetrahedral meshes several texture maps are used for storing connectivity, neighborhood information, and vertex properties.

4 Optimization Techniques

This section presents the different implementations of the most relevant optimization techniques described in [8, 17, 22] in more detail than in Sect. 3. Of course, the original publications are even more detailed. However, by directly comparing the three systems we hope to offer a more comprehensive discussion of the subject.

With these optimization techniques the system by Weiler et al. achieves 2.0 – 5.0 fps, corresponding to 280K – 760K tetrahedra per second, the system by Krüger et al. between 13 and 24 fps and the system by Röttger et al. between 0.3 and 1.4 fps. Note that all reported timings were taken for a 512×512 viewport and that Röttger et al. employ a four times finer sampling than Krüger et al. (see the original papers for more details about the data sets and the test setups).

4.1 Empty Space Leaping

Apart from the size of the output image, the second most dominant factor for the performance of a ray casting algorithm is the number of samples that are taken along each viewing ray. Avoiding the sampling of areas without contribution to the final image can therefore significantly improve the rendering speed.

In the work of Krüger and Westerman [8] empty space leaping is incorporated by the help of an auxiliary data structure allowing to determine whether a block of $8 \times 8 \times 8$ voxels of a uniform mesh contains relevant information or not. This data structure consists of a texture map 1/8 the width, height, and depth of the original data texture and holds the minimum and the maximum data value of each voxel block. The visibility of a block depends on the applied transfer function but can easily be decided by a dependent lookup in a texture, storing for each pair of minimum/maximum values whether the transfer function maps to visible colors or not. Only this relatively small dependent texture has to be recalculated on updates of the transfer function.

The rendering is then performed by rasterizing a polygon of at least the size of the projected volume several times. A fragment shader capable of evaluating 8 sampling points is applied to the rasterized polygon but its execution is suspended with the help of the early depth test if the sampling points lie within an invisible block. The speedup is gained from the fact that fragments blocked by the early depth test will not be shaded at all. Thus, no operations will be spent by the GPU on those fragments.

The empty block test is performed in an additional rendering pass between two sampling passes since the early depth test is only effective for fragment shaders not modifying the depth value. Finally note that although not clearly stated in [8] the

auxiliary data structure stores the minimum/maximum value not only of the voxels in the corresponding 8^3 voxel block. Actually, the minimum/maximum is computed with respect to the current block and all 26 neighboring blocks. This reflects the fact, that an 8-sample ray segment may start anywhere inside a voxel block and may, therefore, very likely reach into an adjacent block.

Incorporating empty space leaping into a ray caster for tetrahedral meshes is more sophisticated since here the sampling is guided by the geometry of the tetrahedral mesh. Therefore, an adapted mesh geometry would be required, recomputed on every transfer function update.

4.2 Adaptive Sampling Distance

Beside from empty space skipping, other improvements can be added to the ray casting algorithm in order to increase the performance. One of them which is used by Röttger et al. [17] is the variation of the sampling distance. Along the viewing ray, we can observe the following scenarios:

- constant sample values
- linear changing sample values
- non-linear changing sample values

For constant sample values we can adopt a similar scheme as for empty space leaping, i.e., we skip the homogeneous region during the ray sampling by increasing the sampling distance. However, we need to acquire at least one sample inside the homogeneous region and account for the correct attenuation along the sampling ray. Thus, if we increase the sampling distance from d to d' , we also have to increase the opacity α_i to $1 - (1 - \alpha_i)^{d'/d}$. If the sample values vary linearly along the viewing ray we can use the same approach as for cell projection rendering, i.e., we have to both increase the opacity and account for the changing self-attenuation. In the case of non-linear varying sample values the sampling distance has to be decreased in order to improve the quality of the rendered image.

If we add lighting effects to the volume rendered image, we are not only interested in the sample values but in the gradients as well. However, these only change for non-linearly varying sample values so we are still on the safe side, if we use the lighting algorithm proposed by Meißner et al. [14], i.e., also pre-integrate the lighting over a certain interval.

In areas with varying gradients the lighting can be computed as proposed by Lum et al. [12]. For each ray segment they look up two pre-integrated lighting terms based on the two gradients at consecutive sample points which are assumed to be constant along the segment. These values can be blended by choosing different pre-integration tables for the previous and the current sample point and adding both contributions. Similar to Gouraud shading this resembles an interpolation of the lighting value rather than an interpolation of the lighting normals.

For more aggressive optimizations to the sampling distance in non-linearly varying areas, we also have to take the transfer function into account. Disregarding lighting, we can also increase the sampling distance in areas that, after applying the

transfer function, share the same color and opacity value. Taking lighting into account is more sophisticated in this case since the gradients change even in those areas where the color stays constant. Therefore, we can only increase the sampling distance if ambient lighting is dominant in this area.

In order to implement the changing sampling distances efficiently, we build a texture map that for each voxel of the original data set contains the maximum allowed sampling distance in any direction. This texture has to be updated whenever the transfer function or the volume data changes. It is created using a simple three pass algorithm. The first pass marks all voxels that have the same gradient in their immediate 26 voxel neighborhood, i.e., the data varies linearly in space. The second pass marks with a different marker voxels with all neighbors mapped to the same ambient color. Finally, the third pass then calculates the oversampling factor for each unmarked voxel based on the non-linearity of voxel values in that region or stores the distances to the next differently marked or unmarked voxel as sampling distance for the marked voxel.

With the help of this sampling distance texture, adaptive ray sampling works as follows: In each rendering pass the current sampling position for each pixel is read from a supplemental texture map. Then the volume data acquired from this position is used for ray integration, and a new sampling position is written back to the supplemental texture map. However, instead of adding a constant length to the current position in order to compute the next sampling position, a scaling factor is retrieved from the sampling distance texture with the current sampling position as texture coordinates. This factor is used to compute the next sampling position by adding the scaled distance between two voxels to the current position.

4.3 Ray Integration

In all surveyed papers the ray integration is performed on a per-sample basis. For each newly computed sampling point the color and the opacity contribution for the ray segment between the previous and the current sampling point is evaluated and composited with the already accumulated color and opacity in front-to-back order (see Sect. 2.3). The accumulated colors are written into a hardware-accelerated off-screen buffer, which allows to access the results of a previous rendering pass by binding the off-screen buffer as a texture map. Precision issues are a second reason for not using the framebuffer and regular OpenGL alpha blending for color accumulation since the provided 8-bit color format is insufficient in particular for ray casting tetrahedral grids where the contribution of a single tetrahedron tends to be very small. The floating-point off-screen buffers used instead provide sufficient accuracy. A disadvantage of this approach is the use of two off-screen buffers and the ping-pong rendering, which is required due to hardware restrictions.

In order to reduce the number of required rendering passes, both Krüger and Westermann and Röttger et al. perform several sampling and blending steps at once in the fragment shader before writing the result to the off-screen buffer.

As mentioned previously there are two different approaches in the surveyed papers for determining the color and opacity contribution of a ray segment. Krüger and

Westermann employ traditional post-classification, i.e., they determine the color and opacity of the ray segment only from the data value at the current sampling point. Röttger et al. and Weiler et al. both apply pre-integrated classification which also requires the data value from the previous sample point along with the length of the ray segment for the lookup of the volume rendering integral in a three-dimensional pre-integration texture map. This requires additional information to be stored in the off-screen buffers. Apart from the data value at the current sampling position Röttger et al. also write the distance to the next sampling point whereas Weiler et al. compute the ray segment length on the fly from the current sampling position and the stored coordinates of the previous sample.

The most sophisticated part in ray casting on tetrahedral meshes is the computation of the sampling points. Given the current cell and an entering intersection point, the next sampling point can be computed by intersecting the viewing ray with all remaining faces of the tetrahedron and taking the intersection on the face – closest to the viewer – that has a normal pointing in the view direction. The next cell can then be determined from the neighborhood information. In [22] all this is performed within a fragment shader accessing the required mesh data from several three-dimensional texture maps. This iterative traversal, of course, only works for convex meshes since there is no information about the next cell for viewing rays leaving the mesh. One way to overcome this limitation is to add virtual tetrahedra between the boundary of the mesh and its convex hull, which are traced just like ordinary tetrahedra but without adding a contribution to the color and opacity. Alternatively, as has been demonstrated in [1, 23], reentries of the viewing rays can be handled by a rendering technique similar to depth peeling.

Note that the pre-integrated classification only guarantees artifact-free renderings in the presented ray casting on tetrahedral meshes since here the assumption of linear data variation along the ray holds. Trilinear interpolation in uniform grids as in Röttger et al. yields a cubic behavior of the data values which has to be compensated by a more dense sampling. Thus, with respect to rasterization, pre-integration provides only limited benefits compared to the traditional post-classification approach.

4.4 Early Ray Termination

The basic idea behind early ray termination is to avoid the processing of occluded samples that do not contribute to the final color, leading to a speedup due to saved fragment operations. The processing of a pixel can be aborted if either the opacity exceeds a certain threshold or the corresponding viewing ray has left the volume. The latter case can be mapped to the first case by assigning an opacity of 1 to the fragments corresponding to those viewing rays.

For a tetrahedral mesh, leaving rays can be easily determined from the neighborhood information by tagging faces with no neighbor with a special index, which can be checked for in the fragment shader. In the case of a uniform mesh, leaving rays have to be determined geometrically by checking the current sample position against the bounding box of the volume.

Preventing the processing for finished viewing rays is also implemented by exploiting the early depth test of modern graphics adapters. In an intermediate rendering pass that reads the accumulated colors and writes only to the depth buffer, the depth value of all fragments with an accumulated opacity greater than the threshold is modified in such a way that the corresponding pixel is blocked from further shading operations. A depth value of z_{near} or z_{far} is written depending on whether the regular rendering is performed with the depth test set to `GL_LESS` or `GL_GREATER`.

However, unless a fixed number of rendering passes can be performed as in [8], we still have to detect when all viewing rays have been fully processed and the polygon rendering can be stopped. This information can be provided via the occlusion query functionality of modern graphics adapters. Issued with a rendering pass, the occlusion query returns the number of pixels that passed the z-test. In [17] and [22] occlusion queries are, therefore, emitted from time to time. If the result of one such query reaches a count of zero fragments, the frame has been completed and the rendering can be terminated. Unfortunately, the result of an occlusion query is delivered asynchronously with some delay which typically leads to additional rendering passes according to [22]. However, this effect can be neglected compared to the delay caused by waiting for the results.

4.5 Compression of Volume Data

Efficient ray casting of volume data requires an extremely fast access to the volume data at any point. Thus, this access should not only be performed in constant time, but the constant should also be very small. Since all three systems reviewed in this paper store the volume data in texture maps, they are limited by the size of texture memory on the graphics hardware. If this limit is exceeded and texture data has to be transferred between the CPU and the graphics hardware for individual texture lookups, the performance of the three systems will drop significantly.

The most straightforward way to provide a high performance for large uniform volume meshes that do not fit into texture memory is the compression of volume textures combined with an on-the-fly decoding in constant time performed by programmable texture lookups as published in [7] and [19]. While the integration of these compression techniques with ray casters for uniform meshes requires longer fragment programs, it does not pose any crucial difficulties.

Although a compression of tetrahedral meshes with on-the-fly decoding in constant time is less straightforward, Weiler et al. have published a data structure based on tetrahedral strips in [23] that allows the ray caster presented in [22] to work directly on the tetrahedral strips, i.e., directly on the compressed mesh.

Of course, an additional decoding step for each lookup of volume data reduces the overall performance of any ray caster. For large meshes that have to be stored in texture memory, however, it avoids the bottleneck of transferring volume data between the CPU and the graphics hardware. Thus, the compression of volume data appears to be a particularly important extension of the three systems reviewed in this paper.

5 Conclusions

Ray casting on programmable graphics hardware is an interesting way of exploiting the parallel architecture of modern graphics hardware. Moreover, the almost simultaneous publication of the discussed implementations also indicates that the programmability of graphics hardware has reached a critical level, which now allows us to implement many more algorithms on modern graphics hardware. With respect to hardware-accelerated ray casting algorithms, these include, for example, more complex – possibly hierarchical – data structures, higher order interpolation schemes, and more elaborate computations of sampling distances.

References

1. F. F. Bernardon, C. A. Pagot, J. L. D. Comba, and C. T. Silva GPU-based Tiled Ray Casting using Depth Peeling *Techreport UUSCI-2004-006, SCI Institute, University of Utah*, 2004.
2. J. F. Blinn. Jim Blinn's Corner – Compositing, Part I: Theory. *IEEE Computer Graphics and Applications*, 14(5): 83–87, 1994.
3. B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *Proceedings 1994 Symposium on Volume Visualization*, pp. 91–98, 1994.
4. K. Engel, M. Kraus, and T. Ertl. High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. In *Proceedings Graphics Hardware 2001*, pp. 9–16, 2001.
5. M. P. Garrity. Raytracing Irregular Volume Data. *ACM Computer Graphics (Proceedings San Diego Workshop on Volume Visualization)*, 24(5): 35–40, 1990.
6. G. Knittel. Using Pre-Integrated Transfer Functions in an Interactive Software System for Volume Rendering. In *Proceedings Short Presentations EUROGRAPHICS 2002*, pp. 119–123, 2002.
7. M. Kraus and T. Ertl. Adaptive Texture Maps. In *Proceedings SIGGRAPH/EG Graphics Hardware Workshop '02*, pp. 7–15, 2002.
8. J. Krüger and R. Westermann. Acceleration Techniques for GPU-based Volume Rendering. In *Proceedings Visualization 2003*, pp. 287–292, 2003.
9. P. Lacroute and M. Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. In *Proceedings SIGGRAPH 94*, pp. 451–458, 1994.
10. M. Levoy. Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*, 8(3): 29–37, 1988.
11. M. Levoy. Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics*, 9(3): 245–261, 1990.
12. E. B. Lum, B. Wilson, and K.-L. Ma. High-Quality Lighting and Efficient Pre-Integration for Volume Rendering. In *Proceedings of the Joint EUROGRAPHICS - IEEE TVCG Symposium on Visualization 2004*, pp. 25–34, 2004.
13. N. Max. Optical Models for Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2): 99–108, 1995.
14. M. Meißner, S. Guthe, W. Straßer. Interactive Lighting Models and Pre-Integration for Volume Rendering on PC Graphics Accelerators. In *Proceedings of Graphics Interface 2002*, pp. 209–218, 2002.

15. M. Meißner, U. Kanus, G. Wetekam, J. Hirche, A. Ehlert, W. Straßer, M. Doggett, P. Forthmann, and R. Proksa. VIZARD II: A Reconfigurable Interactive Volume Rendering System. In *Proceedings Graphics Hardware 2002*, pp. 137–146, 2002.
16. T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan. Ray Tracing on Programmable Graphics Hardware. *ACM Transactions on Graphics (Proceedings SIGGRAPH 2002)*, 21(3): 703–712, 2002.
17. S. Röttger, S. Guthe, D. Weiskopf, T. Ertl, and W. Straßer. Smart Hardware-Accelerated Volume Rendering. In *Proceedings Symposium on Visualization, VisSym 2003*, pp. 231–238, 2003.
18. S. Röttger, M. Kraus, and T. Ertl. Hardware-Accelerated Volume and Isosurface Rendering based on Cell-Projection. In *Proceedings Visualization 2000*, pp. 109–116, 2000.
19. J. Schneider and R. Westermann. Compression Domain Volume Rendering. In *Proceedings IEEE Visualization 2003*, pp. 293–300, 2003.
20. J.P. Schulze, M. Kraus, U. Lang, and T. Ertl. Integrating Pre-Integration into the Shear-Warp Algorithm. In *Proceedings Third International Workshop on Volume Graphics*, pp. 109–118, 2003.
21. P. Shirley and A. Tuchman. A Polygonal Approximation to Direct Scalar Volume Rendering. *ACM Computer Graphics (Proceedings San Diego Workshop on Volume Visualization)*, 24(5): 63–70, 1990.
22. M. Weiler, M. Kraus, M. Merz, and T. Ertl. Hardware-Based Ray Casting for Tetrahedral Meshes. In *Proceedings Visualization 2003*, pp. 333–340, 2003.
23. M. Weiler, P. N. Mallón, M. Kraus, and T. Ertl. Texture-Encoded Tetrahedral Strips. In *Proceedings Symposium on Volume Visualization 2004 (to appear)*, 2004.
24. L. Westover. Footprint Evaluation for Volume Rendering. *ACM Computer Graphics (Proceedings SIGGRAPH '90)*, 24(4): 367–376, 1990.
25. C. M. Wittenbrink, T. Malzbender, and M. E. Goss. *Opacity-Weighted Color Interpolation for Volume Visualization*. In *Proceedings 1998 Symposium on Volume Visualization*, pp. 135–142, 1998.

Volume Exploration Made Easy Using Feature Maps

Klaus Mueller, Sarang Lakare*, and Arie Kaufman¹

Center for Visual Computing, Computer Science Department, Stony Brook University, Stony Brook, NY 11794, USA.

{mueller,lsarang,ari}@cs.sunysb.edu

We present a framework that enables an intuitive, feature-centric exploration of segmented volumetric datasets. Our system is geared towards users familiar with the basic elements of volume rendering, but who seek to conduct volume exploration in a guided fashion. It provides the infrastructure to organize features and objects extracted from a volume dataset, via segmentation or otherwise, and provides the functionality to view these features with standard volume rendering tools. A novel aspect of our system is that it does not require a separate binary tag volume to indicate the presence of a feature (or object). Instead, we mark a feature by migrating its density range, including its smooth boundary, to a private interval. This avoids the aliasing problems associated with binary tag volumes as well as the extensive run-time costs incurred to resolve these. In addition, since the smooth boundaries of the features are preserved, any volume renderer can be used for data visualization, without modification.

1 Introduction

The extraction of features in volumetric datasets remains a hard task, and these difficulties are among the main impediments in making volume rendering a main-stream data investigation tool for general users, such as medical doctors and other clinical personnel, computational scientists, and even data miners. A main advantage of volume visualization is that it allows users to “play” with the data, exploring different aspects in an engaging interrogative experience. This is usually done via modifying the transfer functions that map raw volume data to visual attributes, such as color and transparency. Transfer functions give users the flexibility to “sculpt” a visualization from the raw volume data, including densities and their derivatives. Transfer functions allow users to show certain features as soft, semitransparent gel-like materials or as accentuated surfaces. A number of elaborate tools have become available that enable users to assist in this endeavor. One such framework is the dual-domain

*Currently at Siemens Medical Solutions, Malvern, PA, USA

interaction tool by Kniss et al. [8], which builds on earlier work by Kindlmann and Durkin [7]. Here, users can find and accentuate features by probing the volume to find critical points in a 2.5D histogram that plots the magnitudes of first and second derivatives over voxel densities. Users can then place so-called transfer function widgets directly into the histogram to visually accentuate the probed features. Tenginakai and Machiraju extended the range of data signatures from the first and second-order to higher-order moments and their derivatives, such as skew and kurtosis [23], which essentially extends the dimensionality of the transfer functions further. Other related work includes that of Pekar [14] who applied cumulative Laplacian-weighted gray value histograms.

While these tools are undoubtedly extremely valuable for visualization experts, they are likely too involved for users who are only marginally experienced in the theoretical underpinnings of feature exploration using data signatures. Here, it does not help either that the tools become quite difficult, and perhaps even inadequate, to use once the density distributions in the data grow more complex, such as for MRI volumes, fine-scale computational data, and others. An example for such a configuration is illustrated in Fig. 1, which shows the visible human's foot. In this dataset, both the muscle and the bone-marrow have very similar data signatures, and the corresponding region voxels will all fall into the same portion of the density-signature histogram. Thus, their given visual attributes will overlap as well, and as a consequence, the two features will not be visually distinguishable.

It has become a trend, in particular in medical visualization, to create very simple interfaces for clinical practitioners. These applications have just a few buttons and sliders, to allow a fast and target-oriented visualization of the patient data for diagnosis and planning. In the typical case, there are a number of task-specific feature extraction tools, such as a vessel segmentation tool sensitive to tubular structures, or a lung nodule tool sensitive to spherical objects of certain densities [19]. Beyond these capabilities, there are typically only a set of navigation facilities, such as zoom,



Fig. 1. Features with similar data signatures are hard to distinguish since they map to the same visual attributes. (a) Muscle and bone marrow are both mapped to gray when using transfer functions. (b) Our method maps the bone marrow's density interval to a private range, which allows different visual attributes to be assigned for it (compared to muscle), while using the same transfer function interface as in (a)

rotate, tilt, and slice. No transfer functions are usually available to change visual attributes, rather, users can choose among a few provided colormaps to colorize the data.

Thus, these highly-specialized tools are on one end of the spectrum of data visualization frameworks, while the data-signature tools mentioned earlier are on the other. In this paper, we suggest a system that is somewhere inbetween. It is geared towards a user who is comfortable in using low-dimensional transfer functions for volume exploration, and would like to enjoy the benefits of applying density-based histograms to create custom visualizations of a dataset. This user either lacks the expertise, time, or motivation to engage into a session with a complex transfer function-based data explorer that operates directly on the raw volume data. Instead, our system provides what one might call a “groomed” data exploration experience, that is, the data are converted into a representation in which exploration with transfer functions is still possible, yet the transfer functions have low dimensionality and are therefore easy to manage and to interact with.

Our system takes as its input a segmented dataset, which has either been prepared by an automated segmentation method or an experienced “senior-user” with an advanced interactive segmentation tool. Here, the segmentation could have been obtained via seed-growing, watershed algorithms, snakes, balloons, live-wires, statistical methods, level sets, deformable models, feature-tracking, and the like. In this regard, our tool may also proof useful to fine-tune a prior automated pre-segmentation. Our system represents the extracted features as a graph, which allows grouping and selection of individual features. However, contrary to other methods of this nature, it does not represent the features as binary objects captured in a tag volume. The problem with tag volumes is that they must invoke tedious and time-consuming algorithms when a ray sample point falls into the proximity of a boundary interface, to determine the object at that position for visual property look-up [24]. When more naive algorithms are used instead, staircasing artifacts may be visible in the resulting image. Our system, on the other hand, maintains objects in their original fuzzy boundary representation, which avoids all visual artifacts and allows any available volume renderer to be used unchanged to produce the visualization.

Our paper is structured as follows. First, in Sect. 2, we will overview related work and preliminaries. The following sections will then focus on our new contribution. Here, Sect. 3 will describe our approach of feature migration, in which we port the density intervals of segmented features to private intervals, and Sect. 4 will describe how these features can be managed using transfer functions in conjunction with feature maps. Finally, Sect. 5 will end with final conclusions and give an outlook onto future work.

2 Preliminaries and Related Work

Our method takes as its input regions of voxels that have been associated with a particular feature or region of interest. Here, a feature may just be one of the skeletonized toes in a foot dataset, the entire skeleton, the muscles, or any other region.

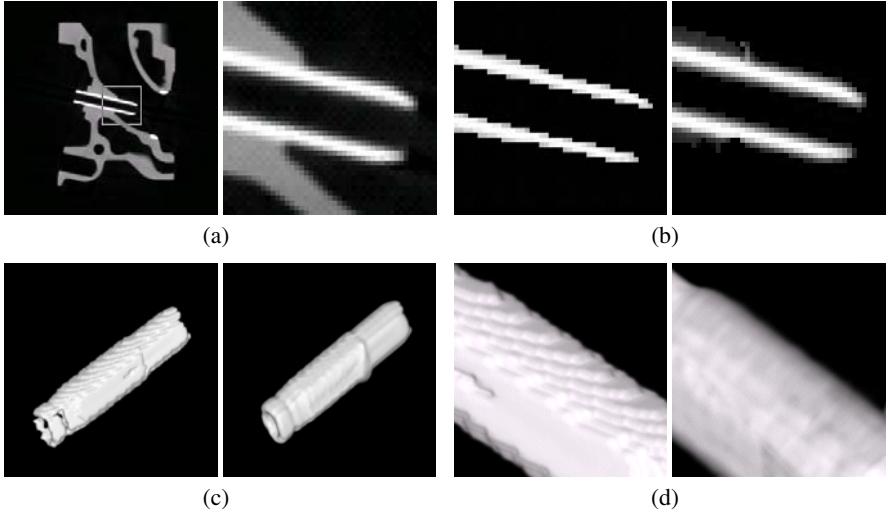


Fig. 2. Feature extraction (the valve) from the Engine dataset: (a) original slice (*left*) and zoomed into the valve (*right*), (b) binary extraction (*left*) and fuzzy extraction (*right*), (c) and (d) volume renderings of the extracted valve: binary extraction (*left*) and fuzzy extraction (*right*)

This feature-centric approach makes it possible for visualization users to manipulate the feature in isolation of other, previously signature-similar, regions, which could not be distinguished with the transfer function interface before.

We refer to the process of extracting the voxels belonging to a particular region of interest as volume extraction. This term has been used previously in the literature [1, 22], but it has always referred to the extraction of 3D surfaces from a volume. In contrast, we actually extract voxels from one volume and port them into another volume, which we call the feature volume. This feature volume is the data, which the end-user interacts with. Here, volume regions, which were not specifically tagged as features may simply be copied into the new volume at verbatim, without change.

One of our main goals is to allow high quality volume rendering of these feature regions. To achieve this, we need to avoid artifacts caused by aliasing, since any aliasing in the data results in subsequent aliasing effects in the volume rendered image. One of the locations where aliasing often occurs in extracted volumes is at the boundary of the segmented region. We shall illustrate this by ways of an example. Consider Fig. 2a (left), where we show a cross-sectional image of the Engine dataset, while Fig. 2b (left) shows the result of a segmentation via thresholding, followed by a simple extraction of the valve in which the intensities of all voxels that do not belong to the valve are set to the air intensity. Consider now Fig. 2a (right) which shows the zoomed-in images of the valve. We observe that the object boundaries were originally fuzzy, and not binary. That is, there is a smooth intensity variation as one moves from one region to another due to the partial volume effect. In the

segmented image (see Fig. 2b (left)), however, the fuzziness is not present. Instead there is a sharp contrast between the intensities of the segmented valve and the surrounding air. This sharp contrast causes an aliasing effect, mostly due to the poor gradient estimation it affords (see the rendered images Figs. 2c and 2d (left)). The intensity-flipping algorithm described in [8] restores the fuzziness at the boundaries of an extracted feature, and rendering results are shown in Figs. 2b, c and d (right).

The effect of the intensity flipping algorithm of [11] is illustrated in Fig. 3a. Here, the boundary profile (labelled before flipping) separates two objects. The goal is to remove the object with the higher density (the one on the right), but leave the boundary characteristics (the position of the zero-crossing of the second derivative and the location of the maximum of the first derivative) unchanged since these will determine the location of the iso-surface in the volume rendering (while the fuzziness will determine gradients and the overall look of the volume rendered surface). Figure 3b shows the intensity/density transfer function. Here, δ_1 is the density of the object we would like to keep, while δ_2 is the density of the object we would like to remove. The range $\delta_2 - \delta_1$ is the density range that the boundary bridges. The transfer function maps this density range linearly to the range $\delta_1 - \delta_{AIR}$ such that the new density of the other side of the boundary to δ_1 is set to δ_{AIR} . The algorithm described in [11] performs this mapping using local measures of δ_1 and δ_2 , thus the mapping function adapts to the local density statistics of the boundary. The current work borrows from this technique, when merging the extracted features into the newly constructed feature volume, preventing aliasing in the process.

Work related to our intensity-flipping technique includes the smooth boundary enhancement of voxelized geometric objects [1, 21] as well as adaptive distance fields (ADFs) [4, 5]. While the former methods append a smoothly (linearly) decaying intensity seam to the binary density field generated by voxelization algorithms, the latter stores a distance field in the non-occupied voxels. Both help alleviate the aliasing problems associated with binary and near-binary objects. In contrast, our method is not geared towards voxelized objects and does not seam an object with a static, artificial function. Rather, it restores the original smooth density-falloff at the boundary of the sampled object, which may vary with spatial location. These boundary effects may be partially due to the partial volume effect and the lowpassing of the sampling process, but may also be due to the object characteristic itself.

Another approach to reach our goal might be gleaned from the soft-segmentation technique [16]. Soft-segmentation seeks to overcome the fundamental problem in image segmentation. For many images there is no way to uniquely and correctly determine the object boundaries. Instead of the usual crisp segmentation where a fixed boundary is derived for a region, soft-segmentation proposes an approach where a pixel/voxel can belong to more than one region. This results in voxels around the region boundary being assigned partially to each of the neighboring regions. However, such an approach would require either non-scalar volumes for storage of these mixed-membership voxels or an extra set of tags, one for each mixed class. We will achieve similar effects with a single scalar volume, allowing users to enhance and visualize mixed regions by ways of the transfer function.

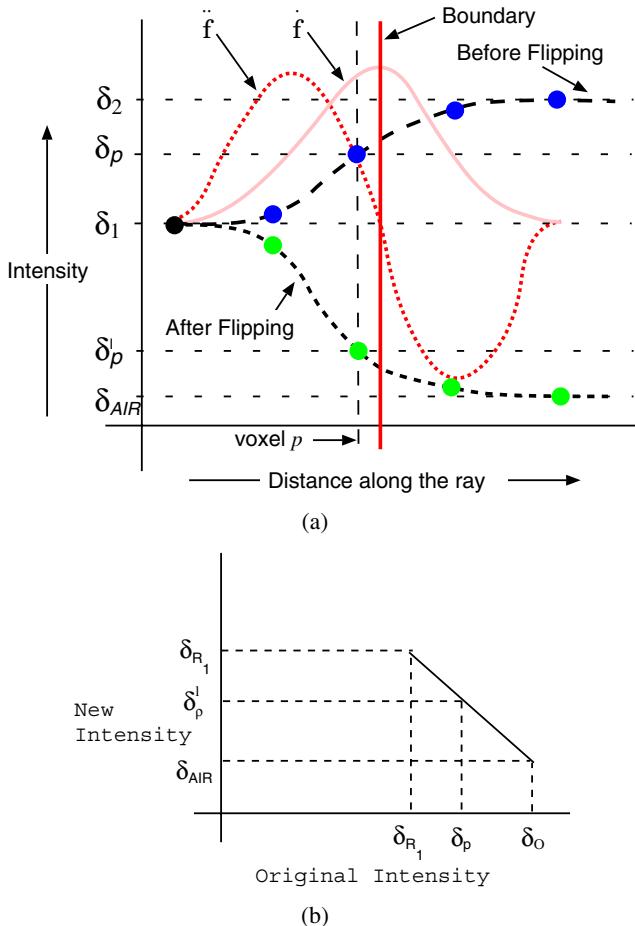


Fig. 3. Effect of the intensity flipping algorithm. (a) The old and new boundary profile with the location of the maximum of the first derivative and the zero-crossing of the second derivative left intact. (b) The corresponding density/intensity transfer function

3 Density Range Migration

In the following, we will use the term region [2] to refer to a connected group of voxels with similar properties (e.g., a bone or a muscle), while we use the notion region of interest (ROI) to describe a group of one or more connected regions that we are interested in. The ROI is the feature to which range shifting is applied so that it can be assigned optical properties without interfering with other objects or features. Our algorithm is composed of four steps. In the first step, we define the ROI. In the second step, we separate the data voxels into different categories in order to perform different actions on them. In the third step we move the scalar value range

of the ROI so that it now occupies a different space in the histogram, and in the final step we modify the boundary between the ROI and its neighboring regions to reflect the change in the ROI range. We shall now describe these four steps in turn.

3.1 Defining the ROI

The ROI is defined by a mask volume, which marks all voxels that are part of the ROI, as well as a list of voxels that form the boundary of the ROI. A voxel is said to be on the boundary if one of its neighboring voxels is not part of the ROI. The mask volume, which defines the ROI can be generated using any suitable segmentation algorithm (as mentioned in the introduction). Feature tracking has also been employed for time-varying datasets [13, 20]. Fortunately, the recent advances in computer graphics hardware [10, 18] make interactive, user-assisted methods, such as seed growing [9, 17], level set methods [12], or snakes/balloons [6], a viable solution. Using such an interactive segmentation system with immediate visual feedback on the segmentation result, ROI-mask voxels can be quickly labeled. We shall illustrate our algorithm using the Lobster CT dataset as an example. Figure 4a show a cross-section of this dataset. The goal of our example application is to apply range shifting to one of the claws, which forms the ROI (shown in a box) in our example. We segment the claw using an interactive seed growing algorithm. The result of the segmentation is shown in Fig. 4b. The voxels which lie on the boundary of the ROI are highlighted in Fig. 4c.

3.2 Categorizing the Voxels

After selecting the ROI, we divide the data voxels into separate categories depending on the actions we perform on them in the later stages. From the previous step we have identified the voxels that are part of the ROI and a list of those that lie on the ROI boundary (Fig. 4b, c). Each ROI boundary voxel can be of one of two types: (1) adjacent to at least one non-boundary ROI voxel, and (2) not adjacent to any non-boundary ROI voxel.

In Fig. 4d, the voxels of the first and second have been painted in differnt shades of gray. The significance of the voxels of the second type (shown in light gray on the periphery) is that they are part of a very thin ROI. The ROI is so thin that all ROI voxels are also boundary voxels.

A volumetric dataset obtained from a scanning modality, such as CT, MRI or PET, displays a natural fuzziness at the object boundaries. This can be attributed to the partial volume effect which occurs due to a finite resolution sampling of a continuous signal by the scanners and the reconstruction operators. The ROI boundary voxels are also part of a naturally fuzzy boundary between the ROI and its neighboring regions. The only exception occurs when the ROI boundary voxels are of the second type. In that case, the ROI boundary voxels are not part of the fuzzy boundary, however, the voxels that surround them are.

We now attempt to find all voxels that form this fuzzy boundary. We assume that the width of the fuzzy boundary is between 3-4 voxels. This can be justified by the

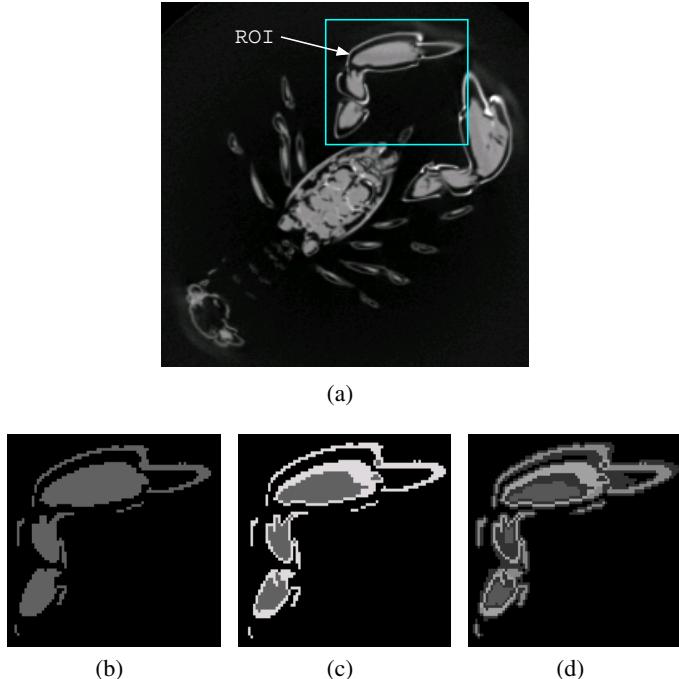


Fig. 4. The Lobster dataset serves as an example to illustrate parts of our algorithm. **(a)** a slice, **(b)** segmenting the ROI (the left claw), **(c)** marking the boundary voxels, **(d)** categorizing the voxels

fact that commonly the radius of the Gaussian kernel used in 3D reconstruction from medical and other image data is about 1.5 times the size of the voxel. To mark the fuzzy boundary voxels, we grow the ROI boundary region once in all directions using an 18-neighbor region grow, which is a 3D version of the 8-neighbor (in 2D) region grow [15]. Next, we categorize the data voxels into the following four categories:

1. Voxels that form the fuzzy boundary between the ROI and its neighboring regions.
2. Voxels of a thin ROI.
3. The remaining voxels of the ROI.
4. The rest of the voxels in the dataset.

We begin with voxels which belong to category 1. In this we include the *ROI* boundary voxels of the first type and the fuzzy boundary voxels we found by region growing. The ROI boundary voxels of the second type are the category 2 voxels. The remaining voxels of the ROI belong to category 3. All remaining voxels in the dataset belong to category 4 (shown in black in Fig. 4).

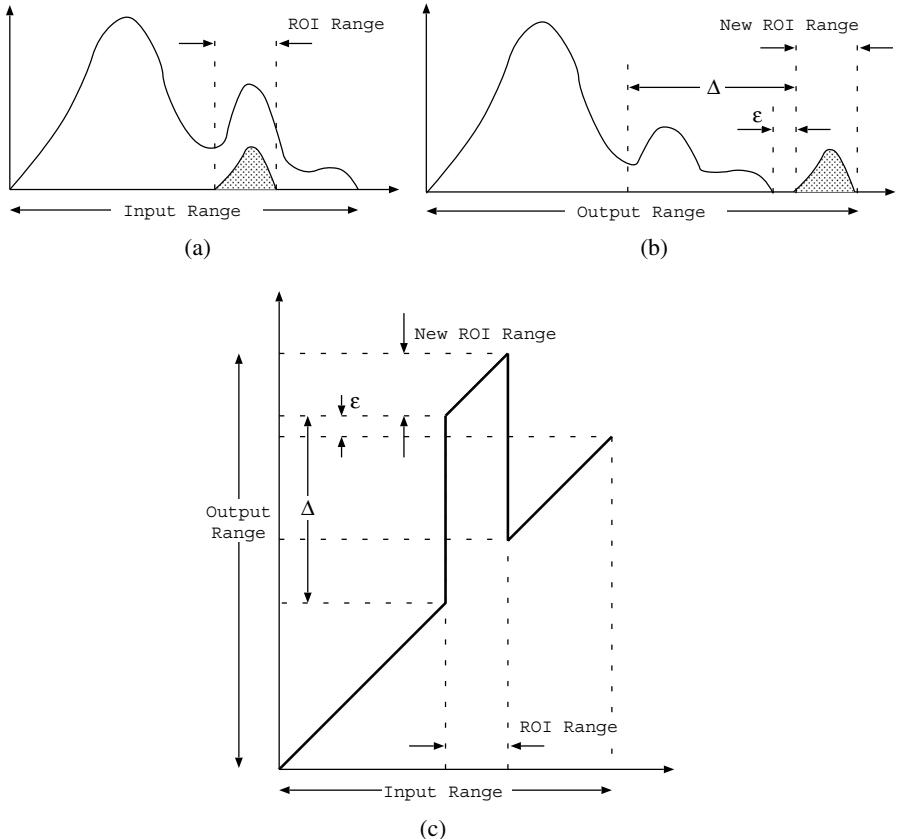


Fig. 5. Migration of the ROI densities: (a) the histogram of the input volume, (b) the histogram of the constructed feature volume, after importing the migrated (shifted) ROI density interval, (c) the density mapping function that achieves this

3.3 Moving the ROI Density Range

In this step, we change the scalar value of the *ROI* voxels, which effectively moves the density range of the *ROI*. We start by finding the current density range of the *ROI*. We define the range as a pair of the lowest and the highest scalar values present, and the width of the range as their difference. The range can be easily computed by looping over the voxels once to find the highest and the lowest scalar values. Migrating the entire density interval preserves the fine density fluctuations within the object. These can be meaningful in later visualizations. For example, NPR (Non-Photo-Realistic) techniques can be applied to accentuate even minute density variations to produce more insight into the micro-scale definition of an object [3].

In the following, we assume that the range of the input data is: (IN_{low}, IN_{high}) . We find the range of the *ROI* by taking into consideration all category 2 and category

3 voxels. We denote this range by (ROI_{low}, ROI_{high}) and its width by ROI_{width} . Some of the ROI voxels (those on the ROI boundary that now belong to category 1) are not considered when finding the range because they are part of the partial volume and we want to avoid the partial volume being part of the range.

As our goal is to move the range of the ROI , we have to select the new range. The new range should be such that no other voxel in the input data has values in that range. There are two possibilities for the new range. First, the new range can be placed within IN_{low} and IN_{high} if there are ROI_{width} consecutive levels of the range that are unused in the input data. In this case, the range of the resultant data remains the same as that of the input data. However, in practice, such cases are rare as the voxel values usually cover the entire range. The second possibility is to place the new range outside the initial range of the input data. This effectively increases the range of the resultant data to:

$$Out = (IN_{low}, IN_{high} + ROI_{width}) \quad (1)$$

We focus on this second case since it is always possible to increase the data range to accommodate the new range of the ROI . Assuming that the ROI range is to be moved beyond the current data range, the new range for the ROI is:

$$ROI' = (IN_{high} + \epsilon, IN_{high} + ROI_{width} + \epsilon) \quad (2)$$

where ϵ is a small number to provide a gap between the ROI and the rest of the data in the histogram (Fig. 5a,b). This gap makes transfer function assignment easier when focussing on the ROI . The starting position of the new ROI range is:

$$ROI'_{low} = IN_{high} + \epsilon \quad (3)$$

Thus, the shift in ROI range from the original position to the new position is given by:

$$\Delta = IN_{high} + \epsilon - ROI_{low} \quad (4)$$

We can now write the equation for moving the ROI range as:

$$\delta' = \begin{cases} (\delta + \Delta) & \text{if category 2 or 3 voxel} \\ \delta & \text{otherwise} \end{cases} \quad (5)$$

where δ is the original scalar value of the voxel and δ' is the new scalar value for the voxel. This transformation for the category 2 and 3 voxels is shown by the graph in Fig. 5c. Figure 5a shows a hypothetical histogram of a dataset with the ROI histogram shown by a dotted region. The result after applying (5) to the histogram in Fig. 5a is shown in Fig. 5b.

3.4 Reconstructing the Fuzzy Boundary

The final step consists of reconstructing the fuzzy boundary such that the fuzziness reflects the new region intensities around the boundary. This procedure has two steps.

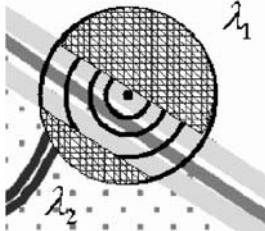


Fig. 6. Region-grow with voxel p at the center

In the first step, we compute a local average intensity of the regions surrounding each voxel in the fuzzy boundary. In the second step, we use these local averages to compute the new intensity for the fuzzy boundary voxel. The following two steps are repeated for each voxel p that belongs to category 1.

Step 1. With p as the center, in the original dataset, a region-grow is performed such that the region includes voxels which are nearest to the central voxel and belong to categories 2, 3 or 4. All category 1 voxels are ignored (Fig. 6). The region-grow continues until the following two conditions are satisfied:

- The region includes at least l voxels from categories 2 or 3 OR γ voxels from category 2.
- The region includes at least l voxels from category 4.

where l is a number small enough to compute a reliable local average intensity of the region, and γ is a number small enough to compute a reliable local average intensity of the ROI voxels in the thin ROI (category 2). A large value for l or γ can result in a wrong average intensity for the surrounding regions. For our experiments, we chose a value of 5 for l and 3 for γ . The region-grow is stopped after 3 iterations even if the above conditions are not satisfied. This is done to avoid going past a 3-voxel radius. Voxels beyond a radius of 3 are unlikely to affect the intensity at voxel p and should not be considered. The idea behind the region-grow is to find the voxels that are nearest to voxel p. Some of these belong to categories 2 or 3 and some to category 4. These neighboring voxels represent the regions that surround p, and are responsible for influencing the intensity value at p. We therefore calculate the average intensity of each of these sets of neighboring voxels. We assume that at the end of the region grow, we have l_i voxels that belong to categories 2 or 3 (or g_i voxels that belong to category 2) and l_o voxels that belong to category 4. The subscript i denotes that the voxels belong to the ROI and the subscript o denotes that the voxels are outside the ROI.

We compute the average intensity of the ROI voxels nearest to p , denoted by δ_i , using the following:

$$if(\gamma_i = \gamma) \quad \delta_i = \frac{\sum_{k=1}^{\gamma_i} \delta_k}{\gamma_i} \quad \text{else} \quad \delta_i = \frac{\sum_{k=1}^{l_i} \delta_k}{l_i} \quad (6)$$

The condition $\gamma = \gamma_i$ checks if the nearest ROI voxels belong to a thin ROI and in that case γ voxels are used to compute the average intensity rather than the l voxels. This change allows our algorithm to work even when the ROI is extremely thin. We similarly compute the average intensity of the non-ROI voxels nearest to p , denoted by δ_o using:

$$\delta_o = \frac{\sum_{k=1}^{\lambda_o} \delta_k}{\lambda_o} \quad (7)$$

Equations (6) and (7) assume that the values of γ_i , λ_i , and λ_o are non-zero. This is however not always true. An important condition under which one of the values is zero (except γ which can be zero when p is not near a thin ROI), is when the region grow stops after going past the 3-voxel radius. At this stage, we make an assumption that since there is no voxel from one of the regions within a 3 voxel radius, the voxel was probably categorized incorrectly as a category 1 voxel and does not belong to the fuzzy boundary. We re-categorize the voxel based on the region whose voxels were included in the region grow. If λ_i is non-zero and λ_o is zero, we consider the voxel as a category 2 or 3 voxel that belongs to the ROI and change its intensity based on 5. In the other case, the voxel is considered part of category 4 and its intensity remains unchanged. Another degenerate case occurs when either λ_i or λ_o are not equal to λ at the end of the region grow. In those cases, we use 6 and 7 to compute the average intensities.

In addition to λ_i and λ_o we also need to compute the new average intensity of the ROI voxels nearest to p after the shifting of the ROI range. We compute the new average intensity without performing another region grow. A region grow performed from p on new values of ROI would include the same voxels that were included in the first region grow performed at p , since we do not move the location of any voxel. Also, all of the ROI voxels included (either γ_i or λ_i) have values shifted by D in the previous step as they belong to category 2 or 3. Thus, the new average intensity of these ROI voxels is shifted by D , and is given by:

$$\delta'_i = \delta_i + \Delta \quad (8)$$

Step 2. The goal in this step is to find the new intensity at voxel p , after the ROI range has moved. In Fig. 7a we show an intensity profile along a hypothetical ray that would pass through voxel p , moving along the approximate direction of the gradient, but traversing through voxel centers. Before intensity flipping, the ray profile would be as shown in Fig. 7a with the points which are the voxels that the rays traverses. As we go across voxel p (dashed vertical line), we see that the intensity gradually changes from δ_o , to δ_i . This intensity profile basically depicts how the intensity changes at the boundary of the region (solid vertical line), as we move across the boundary.

The goal of intensity flipping is to preserve the boundary location while changing the intensities of the voxels along the ray. The basic idea is that if the new voxel intensities are all scaled by the same ratio, the gradient magnitude would increase, but the maxima, which defines the boundary, will remain at the same location. The

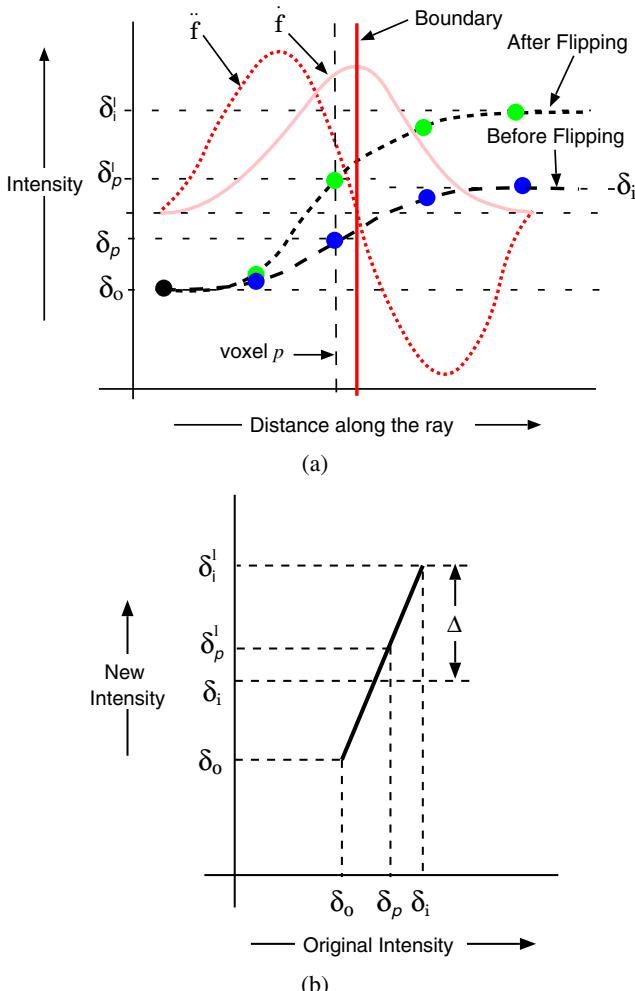


Fig. 7. Migrating a density interval. **(a)** The old and new boundary profile with the location of the maximum of the first derivative and the zero-crossing of the second derivative left intact. **(b)** The corresponding density/intensity transfer function

assumption here is that the voxel intensities along the ray lie inbetween those of the surrounding regions (δ_i and δ_o).

We now present the generic intensity flipping equation when one of the regions around the voxel p has changed its average intensity. Assuming that δ_i is the new average intensity of the region whose original average intensity was δ_i , the intensity flipping equation will be given by:

$$\delta'_p = \delta_o + \frac{\delta_p - \delta_o}{\delta_i - \delta_o} (\delta'_i - \delta_o) \quad (9)$$

where δ'_o is the new intensity for the voxel p. Substituting δ'_o from (8) into (9) we get:

$$\delta'_p = \delta_o + \frac{\delta_p - \delta_o}{\delta_i - \delta_o} (\delta_i + \Delta - \delta_o) \quad (10)$$

The intensity flipping curve corresponding to (10) is shown in Fig. 7b. When this intensity flipping is applied to all the voxels along the ray, the intensity profile of our hypothetical ray changes, and is shown by the curve labelled “after flipping” in Fig. 7a. The points are the intensities of the same voxels along the ray but now with different values than before. As a result of this flipping, the first derivative changes, but the maximum of the first derivative (the boundary) remains at the same spatial location. Similarly, the second derivative zero-crossing remains unchanged. This nice property gives us an unchanged boundary location, even though one of the regions around the boundary has changed.

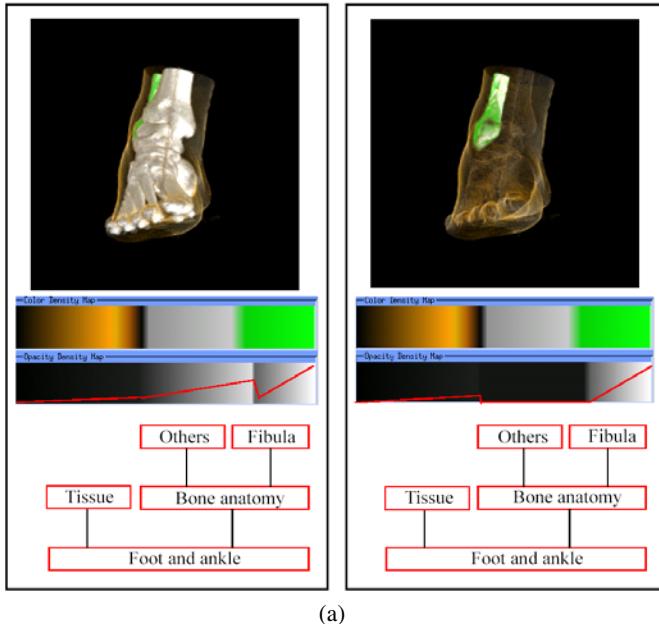
Equation (10) is based on the assumption that δ_p is somewhere inbetween δ_i and δ_o . However, there are cases when this is not true. This is either due to noise in the dataset or due to an incorrect location of the fuzzy boundary. We solve this degenerate case by suggesting that whenever such a case occurs, the voxel should not be part of the fuzzy boundary, and was categorized incorrectly to category 1. We compute the new intensity for this voxel by re-categorizing the voxel based on its original intensity δ_p . If the original intensity is beyond the intensity δ_i , then we shift its intensity as if it were a category 2 or 3 voxel using (5). Otherwise, we leave its intensity unchanged as if it were a category 4 voxel.

We apply the previous two steps on all category 1 voxels as mentioned before. All the category 4 voxels are left untouched.

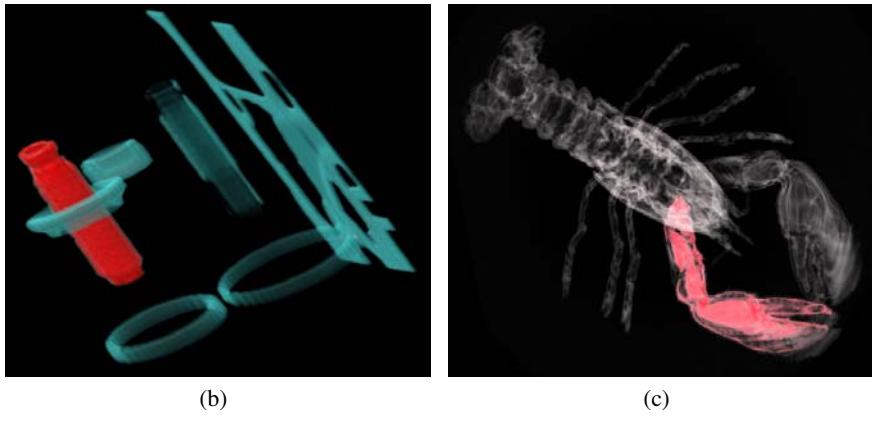
4 Volume Exploration with Feature Maps

The process described above yields the feature volume in which each feature (or ROI) takes up a private density interval (but with the original density statistics), surrounded by a smooth boundary. In essence, the feature volume can also be regarded as a segmentation notebook where features, once captured, are collected in a common environment. In practice, we first copy the original volume into the feature volume and then perform the segmentation and migration there. In this way, unsegmented volume portions are kept “as-is”. To keep track of the extracted objects and their assigned density intervals, we log them in a hierarchical organization, which we call a feature map. Displaying this feature map directly below the transfer function window enables users to easily navigate the volume. Descriptive labels and text may also be associated with each map node, and this labeling may be done either at the segmentation stage or later by the user in the general viewing stage.

A simple example for such a user interface is depicted in Fig. 8a. Here, the fibula of the Visible Human Foot was extracted and its density interval migrated to the



(a)



(b)

(c)

Fig. 8. (a) A feature volume of the Visible Human Foot with opacity and color transfer functions and a relatively simple hierarchical feature map for navigation. On the left, the user was painted the fibula green, and the rest of the skeleton white. On the right, the user decided to eliminate the occluding bone structures and just show the fibula, along with a faint outline of the tissues. **(b)** The Engine with one valve separated from the rings. **(c)** The lobster with one claw separated from the remaining body structures

upper portion of the density range. The new density layout is communicated below in form of the feature map hierarchy. The user may now use the indicated density intervals (or brackets) as a guide to quickly give each feature the desired look. The color transfer function uses a color palette along with a brightness (luminance) curve,

while the opacity transfer function works with the familiar 1D curve. While on the left panel the user decided to show the fibula colored in green and the remaining bones in white, on the right panel he/she chose to eliminate all other bones to reveal the fibula in full view, still colored in green. Finally, Fig. 8b shows the Engine with one of the valves extracted as a separate feature, while Fig. 8c shows the Lobster with one claw separated from the remaining tissue and shell.

5 Conclusions

We have presented a framework that enables an intuitive, feature-centric exploration of (possibly automatically) segmented volumetric datasets. This approach is geared towards users who are familiar with the basic elements of volume rendering, but who seek to conduct volume exploration in a guided fashion. These users could be scientists, medical personnel, or students. Our system provides the infrastructure to organize features and objects extracted from a volume dataset, via segmentation or otherwise, and provides the functionality to view these features with standard volume rendering tools. Our method does not require a separate binary tag volume to indicate the presence of a feature (or object). Instead, we migrate the entire density range of the feature to a private interval, including its smooth boundary. There are several advantages to this approach. First, the aliasing problems associated with binary tag volumes are avoided, as well as the run-time costs incurred to resolve these. In essence, we defer these costs to the density migration stage, which, however, typically runs in a matter of seconds. Since in our method the smooth boundaries of the features are preserved, any volume renderer can be used for dataset visualization, without modification. On the other hand, since the density statistics of the objects are preserved as well, any sophisticated volume visualizer, such as an NPR renderer, can be employed to enhance these small-scale fluctuations.

Another highlight of our system is the hierarchical feature map that captures and organizes the structural knowledge that has been gathered about the dataset. While hierarchical feature maps are also possibly used in conjunction with tag volumes, the density migration makes it possible to use them within the familiar transfer function interface to control color and opacity as a function of density. This enables users to apply the typical volume rendering effects, such as smooth semi-transparencies and gradient modulation. This luxury, however, does not come for free. In all but the simplest cases, the density range provided by 8-bit datawords will not be sufficient and longer datawords (in most cases 16-bit) will be needed to house the extended density intervals. This, however, may not be a huge problem since even GPUs can nowadays process volumes as large as 32 bits per voxel. On the other hand, tag volumes double the memory consumption as well, in addition to requiring algorithmic changes to the renderer. But in any case, future work will seek to investigate methods that can provide a dynamic range compression of the density intervals, in order to make the extended range fit into the 8-bit range. Another notable point is that due to the range migration and the associated steeper density curves at boundaries, care must

be taken when rendering with gradient modulation. We have resolved this issue by using a modulation curve that saturates gradients above a certain threshold.

Acknowledgements

This work has been supported by grants from NIH #CA82402, NSF Career grant ACI-0093157, NSF CCR-0306438, CAT Biotechnology, NYSTAR, and ONR # N000110034. The engine dataset is courtesy of GE. The Visible Human foot dataset is courtesy of the Visible Human project. The authors wish to thank Manjushree Lakare and the visualization lab members for their help.

References

1. F. Dachille and A. Kaufman. Incremental Triangle Voxelization. In *Proc. Graphics Interface*, pp. 205–212, May 2000.
2. E. R. Dougherty and C. R. Giardina. *Matrix Structured Image Processing*, chapter Topological Operations, pp. 140–148. Prentice-Hall, 1987.
3. D. Ebert and P. Rheingans. Volume Illustration: Non-Photorealistic Rendering of Volume Models. In *Proc. IEEE Visualization*, pp. 195–202, 2000.
4. S. Frisken, R. Perry, A. Rockwood, and T. Jones. Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics. In *Proc. SIGGRAPH*, pp. 249–254, 2000.
5. S. Frisken Gibson. Using Distance Maps for Accurate Surface Representation in Sampled Volumes. In *Symposium on Volume Visualization*, pp. 23–30, 1998.
6. M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active Contour Models. *International Journal of Computer Vision*, 1(4):321–331, 1988.
7. G. Kindlmann and J. Durkin. Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering. In *Proc. of Symposium on Volume Visualization*, pp. 79–86, 1998.
8. J. Kniss, G. Kindlmann, and C. Hansen. Multi-Dimensional Transfer Functions for Interactive Volume Rendering. *IEEE Trans. on Visualization and Computer Graphics*, pp. 270–285, 2002.
9. Kevin Kreeger and Arie Kaufman. Interactive Volume Segmentation with the PAVLOV Architecture. In *IEEE Parallel Visualization and Graphics Symposium*, pp. 61–68, October 1999.
10. J. Krüger and R. Westermann. Acceleration Techniques for GPU-Based Volume Rendering. In *Proc. IEEE Visualization*, pp. 287–292, 2003.
11. S. Lakare and A. Kaufman. Anti-Aliased Volume Extraction. In *Data Visualization 2003, Proc. of Eurographics/IEEE TCVG Visualization Symposium*, pp. 113–122, May 2003.
12. A. Lefohn, J. Kniss, C. Hansen, and R. Whitaker. Interactive Deformation and Visualization of Level Set Surfaces Using Graphics Hardware. In *Proc. IEEE Visualization*, pp. 75–82, 2003.
13. J. Ming, R. Machiraju, and D. Thompson. A Novel Approach to Vortex Core Detection. In *Proc. of Euro./IEEE Visualization Symp.*, pp. 217–225, 2002.
14. V. Pekar, R. Wiemker, and D. Hempel. Fast Detection of Meaningful Isosurfaces for Volume Data Visualization. In *Proc. IEEE Visualization*, pp. 223–230, 2001.
15. W. K. Pratt. *Digital Image Processing*. A Wiley-Interscience, second edition, 1991.

16. D. Prewer and L. J. Kitchen. Soft Image Segmentation by Weighted Linked Pyramid. *Pattern Recognition Letters*, 22(2):123–132, 2001.
17. K.-L. Ma R. Huang, P. McCormick, and W. Ward. Visualizaing Industrial CT Volume Data for Nondestructive Testing Applications. In *Proc. IEEE Visualization*, pp. 547–554, 2003.
18. 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. SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pp. 109–118, 2000.
19. Y. Sato, C. Westin, A. Bhalerao, S. Nakajima, N. Shiraga, S Tamura, and R. Kikinis. Tissue Classification Based on 3D Local Intensity Structures for Volume Rendering. *IEEE Tran. on Visualization and Computer Graphics*, 6(2):160–180, 2000.
20. D. Silver and X. Wang. Tracking Scalar Features in Unstructured Datasets. In *Proc. IEEE Visualization*, pp. 79–86, 1998.
21. M. Sramek and A. E. Kaufman. Alias-Free Voxelization of Geometric Objects. *IEEE Trans. on Visualization and Computer Graphics*, 5(3):251–267, July 1999.
22. Ikuko Takanashi, Shigeru Muraki, Akio Doi, and Arie Kaufman. 3D Active Net - 3D Volume Extraction. *Journal of the Institute of Image Information and Television Engineers*, 51(12):2097–2106, 1997.
23. S. Tenginakai, J. Lee, and R. Machiraju. Salient Iso-Surface Detection with Model-Independent Statistical Signatures. In *Proc. IEEE Visualization*, pp. 231–238, 2001.
24. U. Tiede, T. Schiemann, and K. H. Höhne. High Quality Rendering of Attributed Volume Data. In *Proc. IEEE Visualization*, pp. 255–262, 1998.

Fantastic Voyage of the Virtual Colon

Arie Kaufman and Sarang Lakare*

Center for Visual Computing, Computer Science Department, Stony Brook University, Stony Brook, NY 11794, USA.

{ari,lsarang}@cs.sunysb.edu

Abstract. We pioneered a visualization-based alternative to conventional optical colonoscopy, called virtual colonoscopy (VC), for screening patients for colonic polyps, the precursor of colon cancer. Unlike optical colonoscopy, VC is patient friendly since the patient undergoes a less rigorous bowel preparation. VC is also a fast, non-invasive, more accurate, and cost-effective procedure for mass screening of colon polyps. In VC, the patient's abdomen is imaged by a helical or multi-slice computed tomography (CT) scanner during a 40-second single-breath-hold. Supine and prone scans are acquired, each typically consists of 350 to 700 axial 512×512 images of sub-millimeter resolution. A 3D model of the colon is then reconstructed from the CT scan by automatically segmenting the colon out of the rest of the abdomen and employing an electronic cleansing algorithm for computer-based removal of the residual material and accurate reconstruction of the soft surface behind the removed material. This is accomplished using a novel segmentation rays algorithm. The visualization software, running on a PC, allows the physician to interactively navigate through the colon using a physically-based navigation system with fast volume rendering for visualization and the center line of the colon as a guide for the navigation. An intuitive user interface with customized tools supports measurements and virtual biopsy to inspect suspicious regions.

1 Introduction

Colorectal cancer currently ranks as the third most common human malignancy and the second leading cause of cancer-related deaths [6]. The overall risk of developing the disease is approximately 5% over a lifetime. In both recent years there were approximately 130,000 new cases of colorectal cancer and 57,000 deaths each year in the US [6]. Most colon cancer arises from adenomatous polyps growing at about 1mm a year, which can take five to fifteen years for malignant transformation. The risk of developing carcinoma from a polyp is directly related to its size: essentially 0% risk if the polyp is less than 5mm, 1% risk if size is 5–10 mm, 10% risk with size 10–20 mm, and at least 30% risk with polyps larger than 20mm. Survival rates from colon cancer are related directly to the pathologic staging of the disease, and

*Currently at Siemens Medical Solutions, Malvern, PA, USA

are over 90% when cancers are limited to the bowel wall. Whereas 75% of cancers found by screening in asymptomatic patients are confined to the bowel wall, more than half of those with symptoms had a more advanced stage.

The American Cancer Society recommends that screening begins at age 50 for asymptomatic, average-risk patients. Standard optical colonoscopy employs a long fiberoptic-based medical instrument, called endoscope, which can perform a biopsy and/or remove detected polyps, and is generally considered the “gold standard” for colon screening, although miss rates of 18% of adenomas larger than 6mm have been reported on back-to-back optical colonoscopies. It is also expensive, uncomfortable, requiring harsh colon cleansing and sedation, time consuming, and invasive, with a small risk of perforation and death (colonic perforation in one in 500–1,000 cases and death in one in 2,000–5,000 cases), examines only 75–80% of the colon surface, fails to examine the entire colon in up to 10% of patients, and is ineffective in examining areas of the colon blocked by masses or in areas of severe narrowing.

A visualization based alternative, known as virtual colonoscopy (VC) or CT colonography (CTC), is rapidly gaining popularity. It was concurrently developed by us at Stony Brook University [7, 8] and at a few other institutions (e.g., [16]). The distended colon (inflated with carbon dioxide or room air through a tiny rectal tube) is imaged by a helical or multi-slice computed tomography (CT) scanner during a 40-second single-breath-hold. Supine and prone abdominal CT scans are acquired, each typically consists of 350–600 axial images of 512×512 sub-millimeter resolution, providing excellent contrast between the colon wall and the lumen. A 3D model of the colon is then reconstructed from the CT scan by automatically segmenting the colon out of the rest of the abdomen and employing an electronic cleansing algorithm for computer-based removal of the residual material. The visualization software, running on a PC, allows the physician to interactively navigate through the colon using a physically-based navigation system with fast volume rendering for visualization and the centerline of the colon as a guide for the navigation. An intuitive user interface with customized tools supports measurements and virtual biopsy to inspect suspicious regions. Unlike optical colonoscopy, VC is patient friendly since the patient undergoes a less rigorous bowel preparation consisting of a modified diet with oral agents to “tag” the residual stool and fluid. VC is also a fast, non-invasive, more accurate than optical colonoscopy [15], and cost-effective procedure for mass screening of colon polyps.

2 Segmentation and Electronic Cleansing

An effective colonoscopy is only possible if there are no residual materials inside the colon that could be falsely interpreted as polyps. Most current approaches, either optical or virtual, involve complete physical bowel cleansing through either washing the colon with large amounts of liquids and/or administrating laxative medications and enemas to induce bowel movements. Our approach is much preferred by patients as it replaces physical cleansing of the human bowel with virtual cleansing of the CT scan data. This process, called electronic cleansing, relies on a new bowel preparation

scheme which increases the density of the residual material in the CT scan followed by automatic identification and removal of the residual material [9, 12]. As part of the bowel preparation, the patient is asked to remain on a soft-food diet (yogurt, cereals, mashed potatoes etc.) for an entire day preceding the examination date. A bottle of “banana smoothie”, that is, density-enhancing fluid with barium is taken by the patient with each of the three meals, resulting in tagging of residual stool and fluid. Before administrating the CT scan, the patient’s colon is distended with carbon dioxide (CO_2 gas of approximately 1000cc) via a tiny rectal tube. Although others use room air, we found that CO_2 is more tolerated by the patients. The distention avoids colon wall collapses and provides a good view of the colon surface.

The CT dataset obtained after scanning is very complex due to the large amounts of residual fluid and stool inside the colon. Although these unwanted residual materials are enhanced due to our special bowel preparation, they cannot be simply subtracted from the CT images. First, due to the partial volume effect (Fig. 1), the voxels on the boundary of the enhanced residual material get incorrectly classified resulting in incorrect coloring and shading during volume rendering. Second, the enhanced material does not have uniform intensity, making it difficult to accurately identify the residual material.

In order to solve the partial volume problem, we introduced a novel segmentation technique that is based on segmentation rays [12]. These are named so because they assist in the segmentation. The basic idea behind our approach is that the intersection of two distinct-density regions possesses a unique intensity profile as we move in a direction normal to the intersection. When these rays traverse through the volume, they compare their intensity profile with some pre-defined ones. If a ray crosses an intersection between two regions in an approximately normal direction, it finds a match and then performs certain tasks of classification and reconstruction at the intersection. Depending on the application, the rays can be programmed to detect certain specific intersections and perform certain specific tasks. This leads to a very fast and effective segmentation approach that successfully eliminates the partial volume effect. For our application, we program the segmentation rays to detect and remove the partial volume at the boundary of the enhanced residual material [9].

In Fig. 2 we show a portion of an axial CT image obtained after scanning a patient who has followed our bowel preparation. It can be observed that the contrast for the residual material varies substantially. In order to detect residual material with low contrast, we have designed a new segmentation technique that combines classification with thresholding [10]. Our technique has two steps. In the first step, we perform a vector quantization based classification [3] that classifies the data voxels into different classes. A 23 dimensional neighborhood is used to perform the classification. The different classes obtained are arranged in a decreasing order of average intensity. All the classes with an average intensity above a certain threshold are considered to be part of the residual material. The threshold itself is automatically detected by analyzing the histogram of the dataset.

After detecting the enhanced material, we want to cleanse or remove it from the dataset. However, just setting the identified residual material voxels to zero intensity would produce a discontinuous intensity jump. This jump creates aliasing in

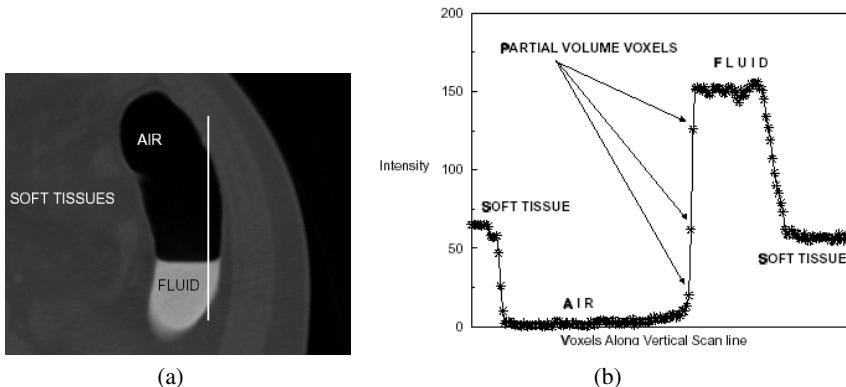


Fig. 1. The partial volume voxels between the air and fluid regions have the same intensity as the soft tissue voxels. (a) A portion of an axial CT image with labelled material; (b) Density profile along the vertical line shown in (a)

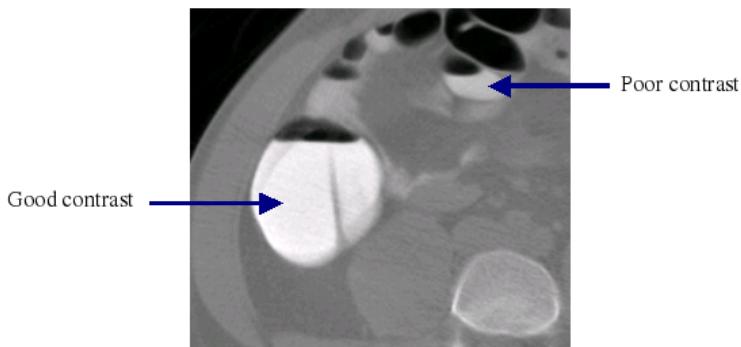


Fig. 2. A portion of an axial CT image showing non-uniform intensity of the enhanced residual material

the dataset, that shows up as staircase artifacts during volume rendering [11]. Those transition voxels contain the partial volume averaging of soft tissue and intensified material, where in a physically-cleansed colon they would contain lower values due to the partial volume averaging of soft tissue and air. Hence, we not only remove high-intensity voxels, but also properly reconstruct the surrounding tissue by inversion of the density profile. This transition region reconstruction allows us to render images of the reconstructed colon wall in high quality (Fig. 3). Figure 4 shows an axial CT image before and after electronic cleansing, which effectively turns the CT scan model of a non-cleansed patient into a dataset of a patient with complete physical bowel cleansing.

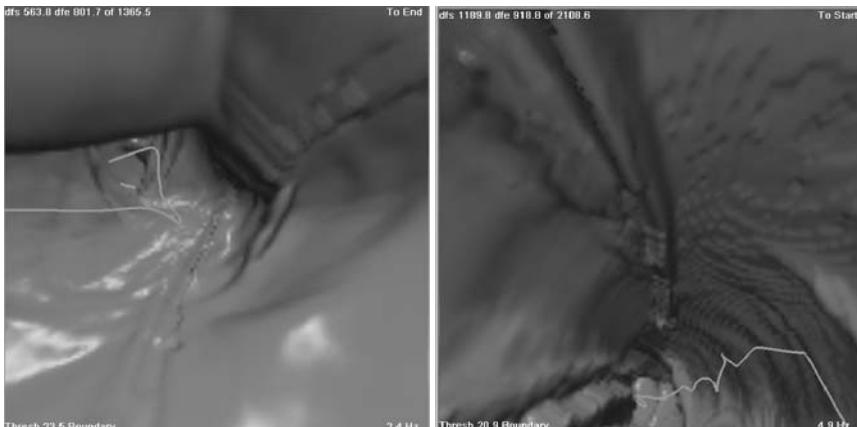


Fig. 3. Removing residual material without reconstruction (*left*) and with our reconstruction (*right*)

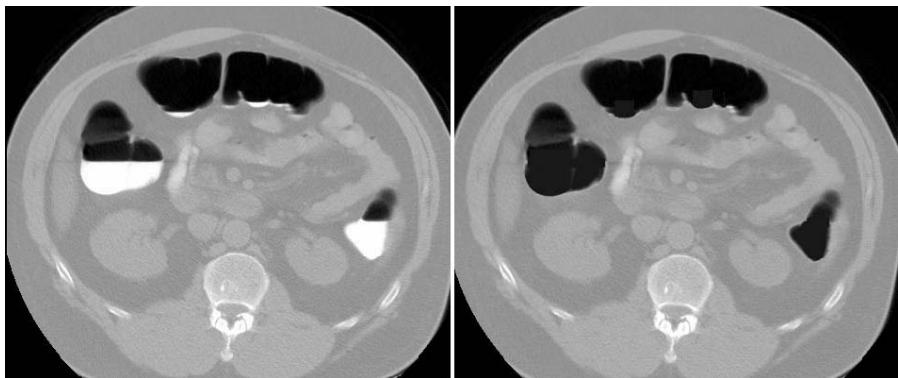


Fig. 4. An axial CT image before (*left*) and after (*right*) electronic cleansing

3 Colon Centerline

On many VC systems the CT data is presented as a sequence of 2D axial images or at best in its raw 3D form and the radiologist has to manually browse through the axial images or follow the winding path of the colon in search for abnormalities. Our system preprocesses the CT data automatically into multiple layers of meta data to make the subsequent data analysis as convenient as possible. First, it segments the colon lumen from the CT volume. This process includes the finding of multiple colon segments that may be separated due to large concentration of tissues (masses) or a colon collapse as well as automatic connection of these segments in the most likely order. Next, our system computes potential fields, which are used for guiding the navigation and for collision avoidance during interactive navigation inside the colon [7, 8]. For guiding the navigation inside the colon we use the colon center-

line. In our early work [7, 8], this centerline was computed through an onion peeling algorithm that starts with the complete segmented object and the two endpoints of the centerline. It then repeatedly removes the outermost layer of voxels that does not violate topology constraints. Through these constraints it is guaranteed that the end points do not get removed and that we always keep a singly connected component. When no more voxels can be removed, all onion layers are peeled off and only the centerline remains. Unfortunately, the constraint voxel removal is extremely slow and the resulting centerline may not be intuitive in some special degenerate cases.

More recently, we devised a faster and degeneracy-free method to compute the centerline [2]. In this approach, we first compute the gradient of the distance field for each voxel in the colon. Next, we flag those colon voxels that are part of a $2 \times 2 \times 2$ cell in which the distance field gradients are not uniformly pointing in a similar direction. This is the case at local distance minima and maxima, which are good candidates for voxels on or close to the centerline. We then start at each flagged voxel and traverse a sequence of neighboring voxels along the local distance gradient direction until we reach another flagged voxel. All voxels along the sequence are also flagged. This procedure connects the local minima to the local maxima along a path of potential centerline voxels. All future centerline computations need to consider only the set of now flagged voxels, which are only about 20% of the total colon voxels.

The next step is to automatically find one of the centerline endpoints. We do this through accumulation of the piecewise Euclidian distance from one arbitrary flagged source voxel to all other flagged voxels. A Dijkstra shortest path algorithm computes this accumulation incrementally. The voxel with the largest piecewise Euclidian distance must be at the very extreme of the colon and is one of the two endpoints. We could repeat this step to find the other endpoint and track back in the piecewise Euclidian distance field to create a centerline. Unfortunately, this centerline would scrape along the colon wall while “cutting corners” and thus not be suited as a flight path. Instead we generate a penalized distance field that again uses the Dijkstra algorithm to incrementally compute a field of minimum cost values. However, this time the cost increment to reach a neighbor voxel is the sum of its Euclidian distance and a penalty that is high at the colon boundary and low in the colon center. This penalty is computed at each voxel as a function of the initial distance from boundary volume. The farthest voxel in this penalized distance field is the other centerline endpoint. Backtracking to the source voxel in this field yields the desired well-centered discrete centerline. A last step of constraint smoothing creates a smooth continuous centerline that we use as the flight path for automatic navigation. This centerline is displayed in light gray in the endoscopic view at the center of Fig. 5.

4 Virtual Navigation

Traditionally, CT scans were interpreted by the radiologist by viewing the separate hard-copy film images in a 2D matrix against a light box mounted on a wall. More recently, the 2D images were viewed on a computer by sequencing the images on



Fig. 5. The user interface for the virtual colonoscopy colon module of Viatronix, Inc

the computer display. Only very recently have 3D reconstructed images begun to be utilized to view the CT data. Researchers have shown that if the entire colon lumen surface is seen, 3D endoscopic navigation has a higher sensitivity of polyp detection than viewing only 2D axial images [1, 15].

Figures 5 and 6 show two different user-interfaces for virtual colonoscopy. The one in Fig. 6 is the Vikon system that we use for our research. The interface in Fig. 5 is the commercially sold system from Viatronix Inc. Both these interactive interfaces provides multiple views of the patient data. Specifically, in Fig. 5, 2D axial, saggital and coronal cross sections are shown down the right hand side. An oblique reformatted slice perpendicular to the colon centerline is shown in the middle left hand side. The upper left corner shows an outside 3D overview map of the patient's colon with indication of current virtual position and orientation and possible bookmarks of suspicious regions. In the center is the 3D volume rendered endoscopic view using standard perspective projection. A 10.2mm polyp is clearly seen in this view, along with its measurement. These 2D and 3D images are all correlated and interlinked so that a position in 3D is overlaid on the 2D images and positions of the 2D slices are overlaid on the 3D images. This provides a quick and simple means to easily analyze suspicious areas in both 2D and 3D.

One of the primary advantages of VC over optical colonoscopy is its ability to interactively and freely pan and zoom the virtual camera for close examinations, and

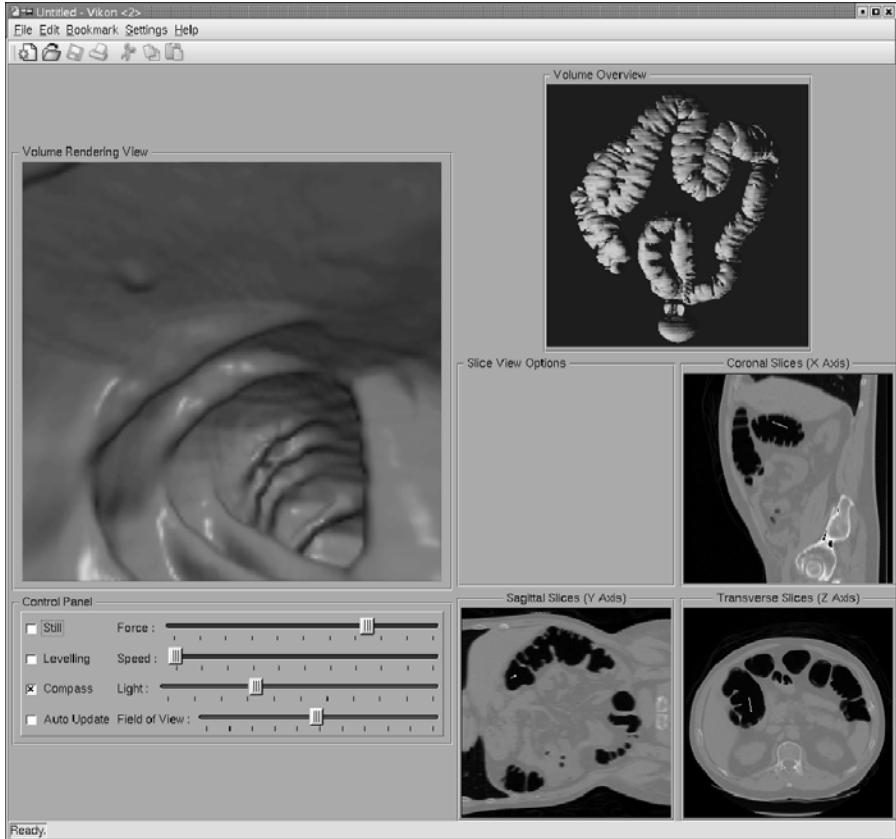


Fig. 6. The user interface for our virtual colonoscopy Vikon system that is used for research purposes

further turn it around (in essence 180 degrees) to view behind folds and sharp bends. This allows us to view a much larger percentage of the colon lumen surface compared to optical colonoscopy. In our system, we keep track of those surfaces on the colon wall that have been displayed to the user so far.

In fact, simulated optical colonoscopy (a flight in one direction through the colon) viewed an average of 74% of the colon surface, primarily missing the back sides of haustral folds and around sharp bends. VC, with non-interactive flythroughs along the centerline, both forward (antegrade) and backward (retrograde), viewed an average of 89% of the surface. With interactive navigation, as discussed below, VC can achieve 100% coverage [20]. For each rendered image, any voxels that contribute sufficiently to a pixel of the image is marked as visualized. The ratio of the amount of currently visualized voxels to total wall voxels gives an excellent quantitative measure of the lumen surface percentage seen so far. We also can paint all previously visualized wall voxels with a unique green color in a special volume rendering

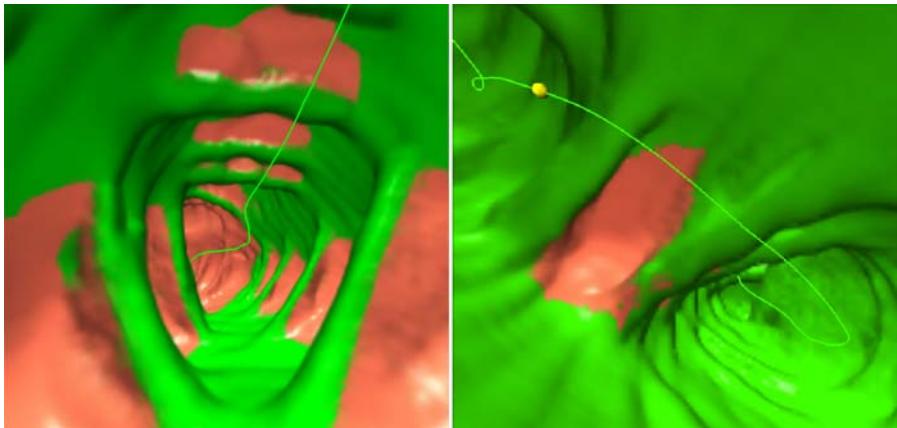


Fig. 7. Endoscopic view of painted information after a forward (antegrade) flythrough (*left*); and an example of a missed patch after both antegrade and backward (retrograde) flythroughs (*right*). The dark gray areas were visualized, while the light gray areas were missed

display of visualized versus un-visualized surfaces. Example images are shown in Fig. 7.

In our system, an automated surface area detection system is utilized. Once the user has flown along the centerline in both directions, the visualized-marking information is processed and a list of missed patches is created. The patches are sorted by size (voxel count) and displayed to the user in a sorted (by size) list box allowing the user to go through the list to view each patch. After stepping through the missed patches, the viewer can simply and efficiently achieve 100% coverage of the colon lumen [20].

5 Interactive Volume Rendering

A primary technological advantage of our system is that the 3D volume rendered endoscopic views are created on the fly, while the user is interacting with the system. This has two important consequences. First, the user does not have to wait for a movie to be generated, which often times takes 15-30 minutes. Second, the user is not only able to auto-fly along the centerline to get a good view of most of the colon surface, but also interactively fly off the centerline similar to computer game navigation. This allows the user to get a better view of suspicious structures, examining and analyzing them from practically any angle. Previously, volume rendering was a much too expensive technique to provide fast enough frame rates to allow interactive navigation. For this reason, many early VC systems utilized a surface rendering approach for which hardware acceleration was available. It is generally accepted that volume rendering provides a much more accurate representation of the true surface of the colon lumen since it does not force piecewise planar approximation to the

surface creating artifacts which are not present in the data while, at the same time, removing small details.

Recently, interactive volume rendering can be implemented on PC class machines as well as true real-time rendering is available using the VolumePro [13] hardware acceleration board, which is based on our Cube-4 architecture [14]. While the VolumePro board provides 30 frames per second, it only provides parallel projection, not perspective projection as required for any virtual endoscopy application (the newer VolumePro 1000 is supposed to have such capability). As an alternative, we have developed a method to provide high-quality images at interactive rates for perspective projections using a multipass approach with the VolumePro rendering card. Now that volume rendering is available from either highly optimized PC solutions or hardware acceleration add-ons, it is much preferred compared to the lower accuracy surface rendering approach.

An important performance measure for software-based interactive navigation is frame rate. Our system relies on years of volume rendering research at Stony Brook University (e.g., [8, 14, 18]) to achieve at least 15 rendered frames per second. At this rate the interactive response is perceived as natural and smooth. Previous systems only achieved between several seconds per frame to a few frames per second, resulting in cumbersome interactivity that most users gave up using them.

One of the techniques we use for accelerating our volume rendering is empty space leaping [19, 21]. The colonic interior is empty space or air that does not contribute to the volume rendered image as air does not absorb or emit light. Our space leaping acceleration technique is illustrated in Fig. 8. Point P_1 is the camera location inside the colon. When a ray R_1 is cast from P_1 , we detect the distance d_1 that is the distance to the closest colon surface. The ray R_1 is leaped ahead by distance d_1 to its next position P_2 . The point P_2 is guaranteed to not cross the colon surface because d_1 is the distance to the closest location on the colon surface. We repeat the leaping step at P_2 by first detecting the distance to the closest colon surface from P_2 and moving the ray ahead by that much distance. Once the distance to the closest surface goes below the regular sampling distance, we stop the space leaping and start sampling the ray at regular intervals. For rays that only grace the surface of the colon (ray R_2 for example), the space leaping starts again as the ray moves away from the surface inside the colon, and the distance to the nearest colon surface is larger than the sampling distance. It is important to note that we pre-compute the distance to the colon surface for every voxel inside the colon and use that distance at run-time.

Because we perform volume rendering, the user is not limited to a surface view of the colon lumen. We provide a translucency feature, called electronic biopsy [17], shown in Fig. 9. When the user is navigating and viewing the colon wall as an opaque surface, only its shape can be analyzed. In essence, the user is viewing the geometry of the colon surface and can make observations such as “here is a bump in the wall”. The electronic biopsy allows the user to see behind the colon wall and analyze the structure of suspected abnormalities. In this way, the user can evaluate not just shape, but also texture, or density-make-up, of an abnormality, confirming that it is indeed a polyp and thus reducing the number of false positives.

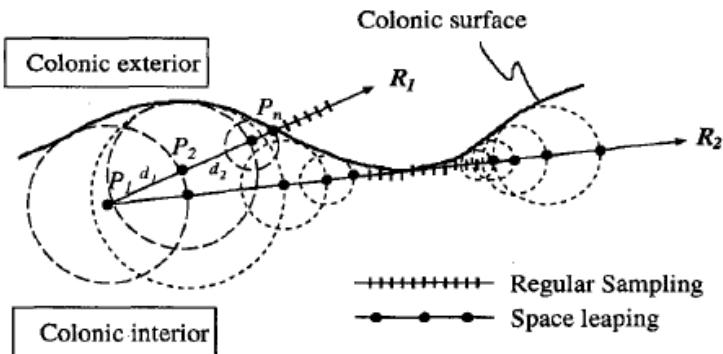


Fig. 8. Fast ray traversal in the colonic interior

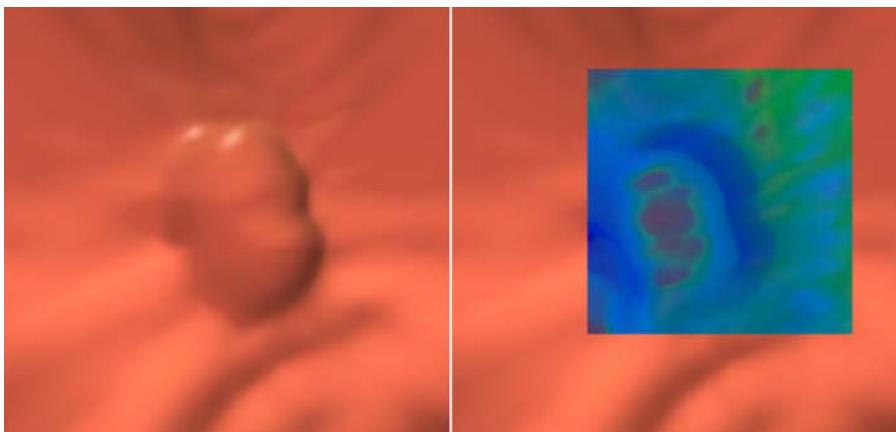


Fig. 9. A surface view (*left*) and an electronic biopsy (*right*) of a polyp

6 Clinical Results

Since the inception of VC about a decade ago, more than 20 clinical trials, including one at Stony Brook University Hospital, with a total of several thousand patients have been published in the medical literature. The performance has been very encouraging, with sensitivity (percentage of true polyps that were found with VC) and specificity (percentage of cases where a polyp was detected which does not actually exist) for polyps larger than 10mm, ranging in 75-91% and 90-93%, respectively, as reported by per polyp comparisons [4, 5, 22]. More recently, the largest-ever multi-center independent clinical trial was conducted in the National Naval Medical Center, Walter Reed Army Medical Center, and the Naval Medical Center San Diego. The 1233 asymptomatic participants in the trial received a virtual colonoscopy examination followed the same day by optical colonoscopy. The results of the study [15] show 93.9% sensitivity and 96.0% specificity for polyps 8mm and larger. These results

show that VC performance compared favorably with that of optical colonoscopy, the accepted “gold standard”. VC is poised to become the procedure of choice for mass screening for colon polyps, the precursor of colon cancer. If all patients 50 years of age and older will participate in such a screening program, over 92% of colorectal cancer will be prevented.

Acknowledgements

This work has been partially supported by NIH grant CA79180, ONR grant N000149710402, NSF grant CCR-0306438, New York Center for Biotechnology, New York State Strategic Partnership for Industrial Resurgence (SPIR) grants, and Viatronix Inc. The patients’ data are courtesy of Stony Brook University Hospital. Thanks are due to numerous people who worked on the VC project: Jerome Liang, Mark Wax, Ming Wan, Dongqing Chen, Ingmar Bitter, Frank Dachille, Kevin Kreeger, and many others.

References

1. C. F. Beaulieu, R. B. Jeffrey, and C. Karadi. Display Modes for CT Colonography Part II: Blinded Comparison of Axial CT and Virtual Endoscopic and Panoramic Endoscopic Volume-Rendered Studies. *Radiology*, 212:203–212, 1999.
2. I. Bitter, A. Kaufman, and M. Sato. Penalized-Distance Volumetric Skeleton Algorithm. *IEEE Trans. on Visualization and Computer Graphics*, 7(3):195–206, July-Sept. 2001.
3. D. Chen, Z. Liang, M. Wax, Lihong Li, B. Li, and A. Kaufman. A Novel Approach to Extract Colon Lumen from CT Images for Virtual Colonoscopy. *IEEE Transactions on Medical Imaging*, 19(12):1220–1226, Dec 2000.
4. H. M. Fenlon, D. P. Nunes, and P. C. Schroy. A Comparison of Virtual and Conventional Colonoscopy for the Detection of Colorectal Polyps. *New England J. of Med.*, 341:1496–1503, 1999.
5. J. G. Fletcher, C. D. Johnson, T. J. Welch, and R. L. MacCarty. Optimization of CT Colography Technique: Prospective Trial in 180 Patients. *Radiology*, 216:704–711, 2000.
6. R. T. Greenlee, R. Murray, S. Bolden, and P. A. Wingo. Cancer Statistics 2000. *Cancer J Clin*, 50:7–33, 2000.
7. L. Hong, A. Kaufman, Y. Wei, A. Viswambharn, M. Wax, and Z. Liang. 3D Virtual Colonoscopy. In *Proc. Symposium on Biomedical Visualization*, pp. 22–32, 1995.
8. L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He. Virtual Voyage: Interactive Navigation in the Human Colon. In *Proc. SIGGRAPH*, pp. 27–34, 1997.
9. S. Lakare, D. Chen, L. Li, A. E. Kaufman, M. Wax, and Z. Liang. Electronic Colon Cleansing Using Segmentation Rays for Virtual Colonoscopy. In *Proc. SPIE Medical Imaging, Physiology and Function from Multidimensional Images*, volume 4683, pp. 412–418, Feb. 2002.
10. S. Lakare, D. Chen, L. Li, A. E. Kaufman, M. Wax, and Z. Liang. Robust Colon Residue Detection Using Vector Quantization Based Classification for Virtual Colonoscopy. In *Proc. SPIE Medical Imaging, Physiology and Function from Multidimensional Images*, volume 5031, pp. 515–520, Feb. 2003.

11. S. Lakare and A. Kaufman. Anti-Aliased Volume Extraction. In *Data Visualization 2003, Proc. of Eurographics/IEEE TCVG Visualization Symposium*, pp. 113–122, May 2003.
12. S. Lakare, M. Wan, M. Sato, and A. Kaufman. 3D Digital Cleansing Using Segmentation Rays. In *Proc. IEEE Visualization*, pp. 37–44, Oct. 2000.
13. H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The VolumePro Real-Time Ray-Casting System. In *Proc. SIGGRAPH*, pp. 251–260, Aug 1999.
14. H. Pfister and A. Kaufman. Cube-4 - A Scalable Architecture for Real-Time Volume Rendering. In *Proc. Symposium on Volume Visualization*, pp. 47–ff., 1996.
15. P. J. Pickhardt, J. R. Choi, I. Hwang, J. A. Butler, M. L. Puckett, R. K. Wong H. A. Hildebrandt, P. A. Nugent, P. A. Mysliwiec, and W. R. Schindler. Computed Tomographic Virtual Colonoscopy to Screen for Colorectal Neoplasia in Asymptomatic Adults. *New England Journal of Medicine*, 349:2191–2200, Dec. 2003.
16. D. J. Vining, D. W. Gelfand, R. E. Bechtold, and et. al. Technical Feasibility of Colon Imaging with Helical CT and Virtual Reality (abst.). *Am. J. Roentgenol*, 1994.
17. M. Wan, F. Dachille, K. Kreeger, S. Lakare, A. Kaufman, M. Wax, and J. Liang. Interactive Electronic Biopsy for 3D Virtual Colonoscopy. In *Proc. SPIE Medical Imaging, Physiology and Function from Multidimensional Images*, volume 4321, pp. 483 – 488, Feb. 2001.
18. M. Wan, A. Kaufman, and S. Bryson. High Performance Presence-Accelerated Ray Casting. In *Proc. IEEE Visualization*, pp. 379–386, 1999.
19. M. Wan, Q. Tang, A. Kaufman, Z. Liang, and M. Wax. Volume Rendering Based Interactive Navigation within the Human Colon. In *Proc. IEEE Visualization*, pp. 397–400, 1999.
20. M. Wax, K. Kreeger, and J. Anderson. Endoscopic View in Virtual Colonoscopy: Achieving Complete Surface Visualization (abst.). In *RSNA*, page 307, Nov 2001.
21. R. Yagel and Z. Shi. Accelerating Volume Animation by Space-Leaping. In *Proc. IEEE Visualization*, pp. 62–69, 1993.
22. J. Yee, G. A. Akerkar, and R. K. Hung. Colorectal Neoplasia: Performance Characteristics of CT Colonography for Detection in 300 Patients. *Radiology*, 219:685–692, 2001.

Volume Denoising for Visualizing Refraction

David Rodgman¹ and Min Chen²

¹ ARM Ltd., 110 Fulbourn Road, Cambridge CB1 9NJ, United Kingdom
dave.rodgman@arm.com

² University of Wales Swansea, Singleton Park, Swansea SA2 8PP, United Kingdom
m.chen@swansea.ac.uk

1 Introduction

Refraction is all around us. Without it we would not be able to see. The failure of the lenses in our eyes to refract incoming light would render the world an indecipherable blur. Externally, we also see refractive objects everywhere. Our vision system is accustomed to the visual effects caused by refraction, and is able to benefit from the visual effects present in refractive scenes by acquiring additional visual cues from these effects.

Figure 1 shows a volumetric refractive scene, which is rendered using ray casting (a) without refraction, and (b) with refraction assuming uniform refractive index for the “water” block. Although (a) may be able to fulfill a basic illustrative function in many circumstances, it offers viewers an incorrect and unrealistic visualization. (b) gives a typical refractive visualization which can be achieved by many surface rendering systems, a small number of which are also able to simulate heterogeneous refractive materials using attribute mapping techniques. It can be seen that (b) gives greatly improved perception of depth (of the rod) and shape (of the ‘water’). In order to keep up with the rapid advances in volumetric data capture and volumetric scientific simulation, Rodgman and Chen recently proposed and studied a collection of methods for rendering refraction in discrete ray tracing [22]. By associating a volume object with a refractive index field, one can use discrete ray tracing to directly render complex refractive conditions. This development is particularly important to volume visualization where the intermixing of translucent and opaque objects is commonplace.

In order to produce high quality images featuring volume objects which are refractive, we must concern ourselves with the quality of the datasets from which the image is synthesized. For these purposes, mathematically defined scalar fields provide unsurpassed quality: they are free from both noise and discretization errors, and therefore allow us to analytically derive precise normals. Consequently, the images produced are of excellent quality.

The situation is somewhat different, however, with discrete datasets. Even if the dataset has been sampled above the Nyquist limit, theoretically permitting a perfect

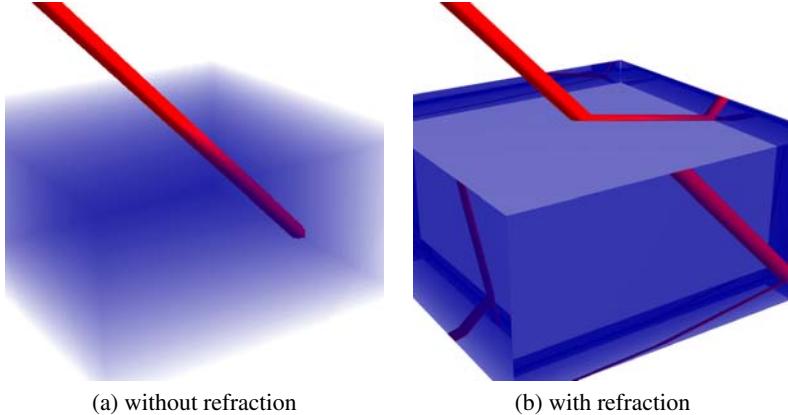


Fig. 1. A water container and a rod, rendered using volume ray casting

reconstruction of the original signal, in practice aliasing errors are hard to avoid. This is particularly the case where a low rendering time is desirable, since it imposes limits on the complexity and degree of the interpolation function. Further, if the dataset has been acquired from a real world source, such as an MRI scanner or similar, the introduction of noise in the scanning process poses considerable problems specific to refractive volume rendering, as discussed in [22].

As Möller et al point out in [13, 14], the quality of the normals has a greater effect on image quality than the quality of the data itself. They were referring only to image quality in the context of opaque, non-refractive objects, but this observation is even more applicable to transparent, refractive objects. This is because when a ray refracts, its direction after refraction is directly governed by the normal at the point of refraction. Consequently, an error in the normal estimation process affects not only the shading at that point (as is the case where refraction is not involved), but also the location of every subsequent sample point. A means of obtaining high quality normals is therefore critical to good image quality with refractive volume rendering.

Errors in the normals may be attributed to three main sources. Reconstruction errors occur when a dataset is under-sampled (pre-aliasing), or when the reconstruction method used is inadequate (post-aliasing). Additionally, the scanning process typically introduces noise into the data. Post-aliasing may be reduced to acceptable levels by use of higher order filters; this problem has been thoroughly addressed in the literature [12, 26] and will not be discussed further here. Pre-aliasing and noise in the dataset, however, have been dealt with in general less satisfactorily, and not at all in the specific context of refraction.

In addition to issues caused by pre-aliasing errors and noise in the data, an excess of fine detail in the dataset can also reduce the effectiveness of refraction in delivering depth cues and other information about the shape of the volume object to the viewer, with similar results to noisy data. We demonstrate these problems with Fig. 2, where a CT head dataset is visualized in conjunction with a brain dataset. In Fig. 2(a) which

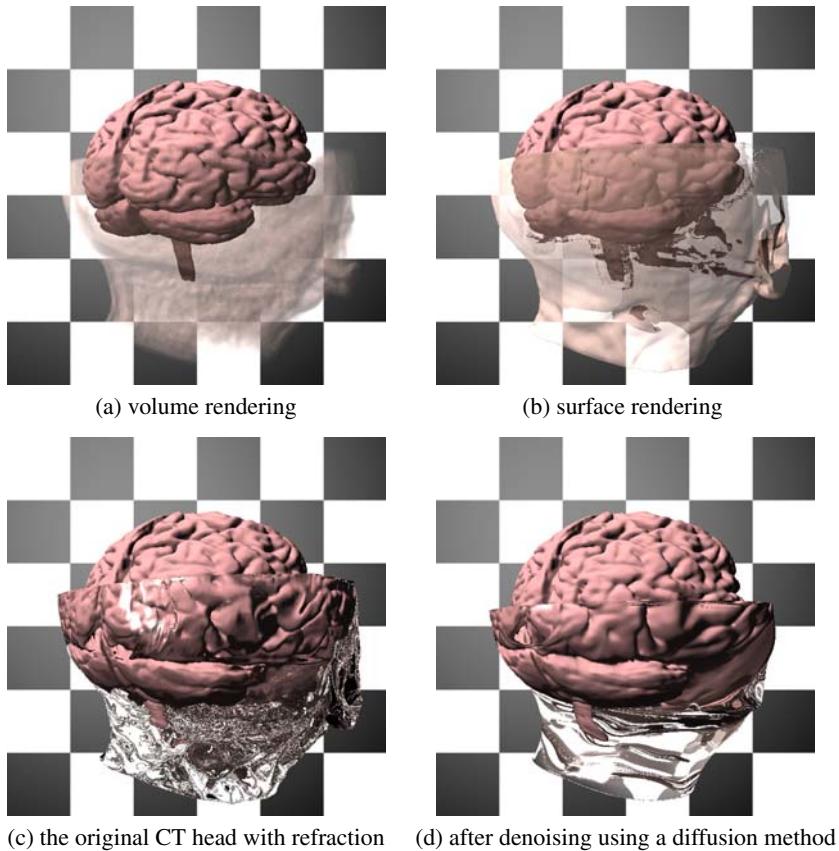


Fig. 2. Combinational visualization of a CT head dataset and a brain dataset. (a) and (b): the original CT head is rendered without refraction. (c): the original CT head is rendered with refraction. (d): the CT head is rendered after denoising

was rendered using volume ray casting without refraction, the CT head has its top part removed and is shown as a translucent object. Whilst it clearly displays the internal structure of the brain, it offers limited depth cues, and poor information about the spatial relationship of the brain and the skull. Figure 2(b), which results from direct surface rendering, provides slightly better depth cues, but incorrect visual representation of the spatial relationship.

Figure 2(c), also rendered using volume ray casting, introduces refraction into the visualization, which permits much better depth perception and appreciation of geometrical structure. However, it introduces at the same time several artefacts, including scattered, noisy specular highlights, large regions of noise attributable to incoherent refraction, and a cracking effect directly above the brain stem, all attributable to ‘noisy’ refraction. In Fig. 2(d), a denoising filter is applied to the CT dataset prior to the rendering; as a result, the smoothed head dataset facilitates a much clearer

visualization of internal structure (i.e., the brain). Moreover, it gives a considerably less noisy impression of the neck area, it conveys a sense of curvature via the refractive patterns in the ‘glass’ which is not present in Fig. 2(c), and it correctly shows the brain stem as an unbroken section of tissue. Artefacts such as scattered, noisy specular highlights present in the unprocessed data are also eliminated in Fig. 2(d).

In order to address these problems, we begin by demonstrating their effects in Sect. 3, after a brief review of related work. In Sect. 4 we propose a number of different methods for ‘improving’ normals; specifically convolution and non-linear diffusion, and in Sect. 5 we develop a means of measuring the effects of any dataset-processing technique on both image quality and accuracy. The denoising methods are tested and evaluated according to our metrics in Sect. 6; and finally, we make some concluding remarks in Sect. 7.

2 Related Work

There are a diversity of modalities, such as CT, MRI and ultrasound, for the acquisition of volume datasets. The aliasing and noise introduced during digitization is of concern to many algorithms for processing and rendering such datasets. Much effort has been placed on the correct reconstruction of normals; a number of normal estimation methods have been studied and compared by Yagel et al [32] and Möller et al [13]. Recent work by Möller et al [15], Neumann et al [17], Persoon et al [21] and Rössl et al [23] represents attempts to reconstruct accurately the scalar function associated with a volume dataset and its gradient.

In image processing and computer vision, we commonly see a different approach to the problem of noise, and a variety of filters were designed for smoothing or denoising images [10, 11, 20, 28]. In recent years, some of these filters have been successfully generalized for smoothing surfaces. For example, Peng et al employed Wiener filtering for smoothing noisy triangular meshes [19]. Desbrun et al employed anisotropic diffusion for denoising height functions and bivariate data [6]. Tasdizen et al used an anisotropic diffusion method for smoothing surfaces via normal maps [25]. In addition, there was much other effort in surface fairing, smoothing and denoising [4, 5, 8, 16, 18, 24, 29, 30].

In terms of volume visualization, surface denoising can be applied to a particular iso-surface contained in a volume data set, but is not suitable for direct volume rendering. There have been some attempts in direct volume smoothing. Hilton et al proposed two wavelet-based noise removal algorithms for denoising MRI data [9], and Angelin et al gave a brief description of a 3D implementation of brushlet basis functions for Fourier domain denoising [2]. The former was a study based on statistical analysis and the latter was an investigation in the context of segmentation. Recently Bertram proposed an iterative fairing method for smoothing contours inside volume datasets using bicubic parametric functions [3].

It is necessary to investigate the effectiveness of denoising methods in the context of direct volume rendering. The successful deployment of nonlinear diffusion methods in surface denoising offers a good starting point for volume denoising, and

the demand for smooth gradient in visualizing refraction [22] makes it a suitable case study for evaluating the effectiveness of volume denoising methods.

3 Motivation

We demonstrate the importance of having good quality normals for refraction by means of the following test case. Figure 3(a, b) shows an opaque sphere rendered with analytically derived normals, and with randomly perturbed normals¹. The same experiment is repeated with a translucent sphere in Fig. 3(c, d).

It can be seen by visual inspection that image quality is better preserved in (b), in the absence of refraction, than in (d); this observation is endorsed by a comparison of the root mean square (RMS) difference between the opaque image pair, and the transparent image pair, which have RMS difference equal to 39.1% and 99.0% respectively.

Improving dataset normals does not necessarily connote greater accuracy; rather, our goal is to manipulate the dataset normals in such a way that we may produce refractive visualizations of the dataset which use refraction to convey three dimensional cues such as depth perception in a meaningful way, without overwhelming the viewer with noisy, over-complicated images. In some senses, we wish to simplify the dataset, and smooth the appearance of the object in question somewhat, in order to improve the subjective image quality.

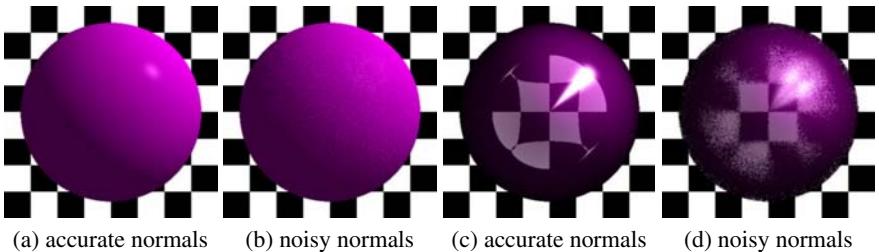


Fig. 3. An opaque sphere (a, b) and a translucent sphere (c, d) rendered, respectively, with (a, c) accurate normals, and (b, d) randomly perturbed normals

4 Denoising Methods

In this section we discuss a number of methods for denoising datasets. We consider the dataset to have resolution $R = (R_1, R_2, R_3) \in \mathbb{N}^3$. We thus denote the unprocessed dataset by $F(\mathbf{x})$ where $\mathbf{x} = (x_1, x_2, x_3) \in \mathbb{N}^3$, such that $0 \leq x_i < R_i$ for $i = 1, 2, 3$, and

¹Each normal is perturbed by a random angle θ , selected such that θ has an average value of 20° and a normal (Gaussian) distribution.

the processed dataset derived from F by $F'(\mathbf{x})$. We also define X to be the set of all such \mathbf{x} .

4.1 Convolution with a Gaussian Filter

A simple method for removing high frequencies in the dataset is to convolve with an ideal low pass filter (the sinc function); that is, to remove all frequencies above a certain threshold. This method is not realizable in practice, since the sinc function is of infinite width; it follows that convolving the sinc function with a dataset of finite width is equivalent to convolving with a truncated sinc function, which gives poor performance in the stop band, and leads to ringing effects (Gibb's phenomenon) [7].

For this reason, we do not discuss use of an ideal low pass filter further, but instead examine a low pass filter which has better performance in the stop band.

By using a filter which has better stop-band performance, we may reduce the artefacts noted above. The natural choice here is the Gaussian filter, $K_\sigma(\mathbf{x})$, defined as

$$k_\sigma(y) = \frac{1}{2\pi\sigma^2} \cdot e^{\left(-\frac{y^2}{2\sigma^2}\right)} \quad (1)$$

$$K_\sigma(\mathbf{x}) = k_\sigma(x_1) \cdot k_\sigma(x_2) \cdot k_\sigma(x_3) \quad (2)$$

where σ denotes the standard deviation of the Gaussian filter; higher values cause increased reduction of high frequencies. Whilst in theory this function has infinite width, thus rendering it susceptible to the artefacts previously observed with the ideal filter, in practice it closely approaches zero sufficiently quickly for this issue not to arise. We thus define $F'(\mathbf{x})$ by convolution with K_σ :

$$F'(\mathbf{x}) = (K_\sigma * F)(\mathbf{x}) . \quad (3)$$

By selecting different values for σ , we may control the degree of smoothing, as demonstrated in the first row of Figures 4 and 5. These images, and in particular the opaque visualizations in Fig. 4(a)–(d), show that the *Gaussian convolution* approach is unable to preserve fine detail. This is not surprising, since this operation has no ability to locally adapt to the dataset. Consequently, there is some degradation of the overall shape of the head, and fine structures such as the nose are especially adversely affected.

Comparing this degradation to the improved, ‘cleaner’ appearance of the refractive visualizations in Fig. 5(a)–(d) clearly shows that this method, whilst certainly effective at removing noise from the dataset, offers a tradeoff between noise reduction and shape preservation.

These results appear to confirm that the source of the noise in the refractive visualization is indeed mostly high frequency noise in the original dataset, since application of a low-pass Gaussian filter is able to reduce these visualization artefacts. However, the shape degradation present when high frequencies are filtered out indicates that this approach has significant drawbacks.

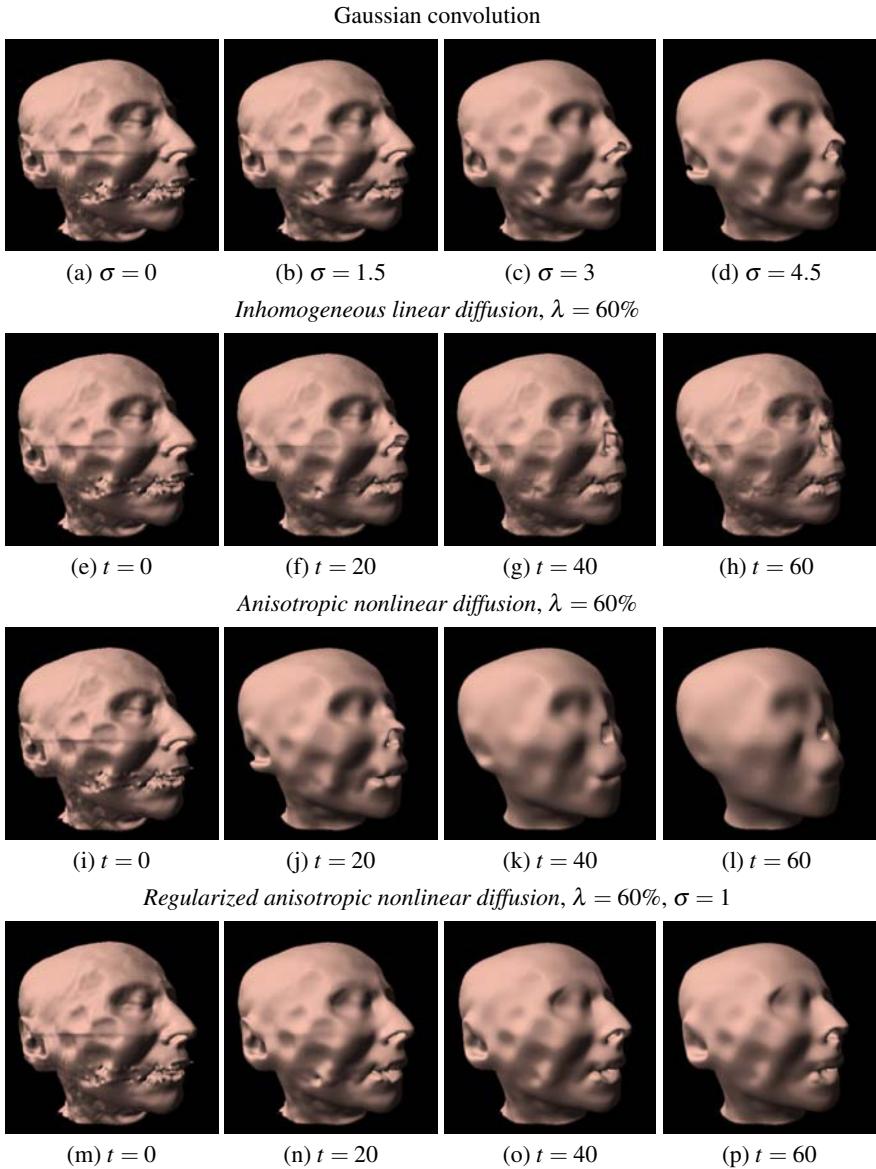


Fig. 4. Results of applying different denoising methods to an opaque object

4.2 Inhomogeneous Linear Diffusion

A significant drawback of convolution with the Gaussian filter is that this method does not permit any local adaption to the dataset; it is not possible to inhibit smoothing in non-boundary regions, or to control the orientation or degree of smoothing at boundaries. *Inhomogeneous linear diffusion* [27] offers more control in this respect.

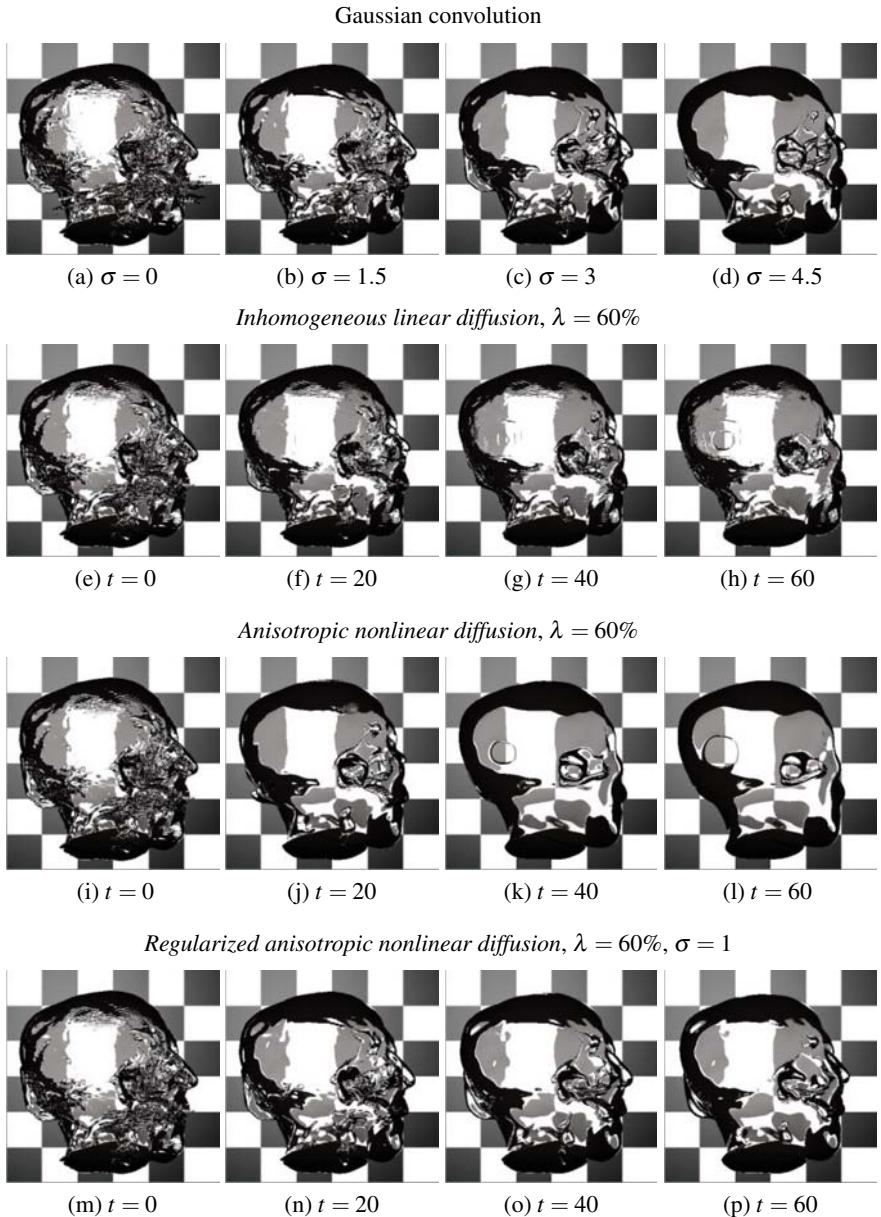


Fig. 5. Results of applying different denoising methods to a translucent object

Firstly, we define the *scale-space* as $U(\mathbf{x}, t)$, where t is the time. We consider the scale-space to be evolving such that at time $t = 0$, $U(\mathbf{x}, t) = F(\mathbf{x})$, and so that as t increases, U represents a more heavily processed version of F .

We begin by defining² the general nonlinear diffusion equation in terms of the rate of change of $U(\mathbf{x}, t)$ with respect to t :

$$\frac{dU}{dt} = \nabla \bullet (D \cdot \nabla U). \quad (4)$$

Here, $D \cdot \nabla U$ is the *flux*, which is equal to the *diffusion tensor* D , a positive definite symmetric matrix, multiplied by the normal of U . This flux can be thought of as representing the flow of ‘particles’ from a region of high density to a region of low density.

When D is a constant, this process is said to be linear and homogeneous, and is in fact equivalent to convolution with a Gaussian filter, as discussed in Sect. 4.1. In the inhomogeneous case, however, D is a function of \mathbf{x} ; typically, of $F(\mathbf{x})$.

Here we discuss isotropic *inhomogeneous linear diffusion*; that is, where the flux is always parallel to ∇F , and we therefore replace the diffusion tensor D with a scalar-valued *diffusivity* function, g . There are many possible choices for g ; here we use the Perona-Malik diffusivity function,

$$g(|\nabla F|^2) = \frac{1}{\sqrt{1 + |\nabla F|^2/\lambda^2}} \quad (\lambda \neq 0). \quad (5)$$

This results in the following diffusion equation:

$$\frac{dU}{dt} = \nabla \bullet (g(|\nabla F|^2) \cdot \nabla U). \quad (6)$$

That is, at any given point \mathbf{x} in the dataset, there is a flow of density along the direction of the original dataset normal, ∇F , the magnitude of which is proportional to the diffusivity, $g(|\nabla F|^2)$.

This diffusivity function inhibits diffusion as $|\nabla F|^2$ increases. The effect is to better maintain well-defined surfaces (by inhibiting smoothing for large values of $|\nabla F|^2$), whilst permitting smoothing in inner regions.

Some example results of applying *inhomogeneous linear diffusion* to the CT head are given in the second row of Figures 4 and 5. The effectiveness of such a process is of course strongly dependant on the value chosen for λ . Selecting suitable values for λ is not straightforward, and is discussed in detail in Sect. 4.5.

4.3 Anisotropic Nonlinear Diffusion

In order to preserve boundaries whilst permitting smoothing in these regions, it is necessary to use an anisotropic method [28]. This method derives its non-linearity from the fact that g is a function of $U(\mathbf{x}, t)$. We therefore have the diffusion equation

²In a small abuse of notation, ∇U refers to the gradient of the scalar field U , whereas $\nabla \bullet V$ refers to the divergence of the vector V .

$$\frac{dU}{dt} = \nabla \bullet (g(|\nabla U|^2) \cdot \nabla U) . \quad (7)$$

This has the effect of allowing the flow to increase for large values of t at the location of small structures that have been smoothed at lower values of t . We might therefore expect more smoothing at the location of these small structures for sufficiently large values of t .

Some results of applying *anisotropic nonlinear diffusion* to the CT head are given in the third row of Figures 4 and 5. In a similar manner to *inhomogeneous linear diffusion*, this method preserves surfaces by inhibiting smoothing at boundaries. However, for the purposes of refraction, it is the normals at the boundaries that are most critical, and where smoothing is required most; whilst we might expect these two methods to preserve fine detail and overall shape better than *Gaussian convolution*, the theoretical basis for improving the smoothness of normals in boundary regions is weak.

4.4 Regularized Anisotropic Nonlinear Diffusion

A significant problem with *anisotropic nonlinear diffusion* is that it gives rise to so-called *staircasing artefacts*, as shown in Fig. 6. These artefacts manifest themselves as small, regularly spaced discontinuities in the processed dataset. Whilst they do not significantly affect the overall shape of the processed dataset, they have a substantial effect on the normals of the processed dataset, which are critical to obtaining high quality refractive images.

In order to improve the performance of *anisotropic nonlinear diffusion*, we make use of a *regularization step* [1] to control the diffusion process, giving rise to the following diffusion equation:



Fig. 6. Staircasing effect with *anisotropic nonlinear diffusion*

$$\frac{dU}{dt} = \nabla \bullet (g(|\nabla U_\sigma|^2) \cdot \nabla U) \quad (8)$$

where we replace U in (7) with U_σ , which is the convolution of U with a Gaussian filter with fixed $\sigma > 0$ as discussed in Sect. 4.1. In other words, we have $U_\sigma(\mathbf{x}, t) = (U * K_\sigma)(\mathbf{x}, t)$.

The effect of this *regularization step* is to improve the stability of the diffusion process. In particular it improves the performance of the diffusion process in the presence of noise, and dramatically reduces staircasing artefacts. It is therefore well suited to applications such as refractive visualization, where the smoothness of the normals is critical. Some results of applying *regularized anisotropic nonlinear diffusion* to the CT head are given in the fourth row of Figs. 4 and 5.

4.5 Determining a Value for Lambda

The choice of a value for λ strongly determines the effects of the diffusivity function. However, fine-tuning values for λ by hand is a time consuming trial and error process. Specifically, very low values for λ result in smoothing being suppressed everywhere, and the resulting images do not convey the benefits of having been smoothed. In the case where λ is set too high, smoothing is not sufficiently inhibited at surfaces, and the result is that the surfaces, as well as the internals of the dataset, become blurred; distortion is therefore increased.

A useful approach for structuring the choice of λ somewhat is to compute all the normals in the dataset, sort them (in ascending order) by magnitude, discarding all zero-length normals, and then set λ equal to the magnitude of the normal at the n th percentile, where n is controlled by the user³. Selecting a suitable percentile for λ is discussed further in Sect. 6.

5 Metrics for Measuring Distortion and Smoothing Effects

In order to properly compare these different methods for improving refractive image quality, it is necessary to devise an objective scheme for quantifying the ways in which both the data and the images are affected. Since in each method, the smoothing process represents a tradeoff between smoothing and volume shape degradation, we separate this task into two separate problems: evaluating the degree to which the overall shape of a volume is affected, and quantifying the improvement in image quality. We address these two problems in the following two sections, and proceed to discuss how these schemes may be used to interpret the efficacy of the various methods in Sect. 4.

³We make use of this method in other parts of this paper where we refer to $\lambda = n\%$ indicating lambda equal to the magnitude of the normal at the n 'th percentile.

5.1 Distortion Metric

Geometric distortion present in a processed dataset $F'(\mathbf{x})$ is encapsulated in the changes of its voxel values, i.e., *data distortion*, in relation to the original dataset $F(\mathbf{x})$. One simple measurement of data distortion is to sum the voxel differences between $F(\mathbf{x})$ and $F'(\mathbf{x})$. However, the level of data distortion does not necessarily provide an informative representation of geometric distortion, especially in the context of refraction. Such a measurement, for instance, would include data distortion in regions of a homogenous refraction index.

In this work, we focus on geometric distortion in the context of refraction, and hence are more interested in *significant* changes in the dataset by exploiting the a priori knowledge of the specification of the refraction indices, by considering a *transfer function* τ , in the form of a lookup table. In principle, this transfer function defines all the regions in the dataset where the refraction index varies. In practice, it often corresponds to the transfer function responsible for extracting the data of interest during visualization.

Using such a transfer function enables us to ignore small changes in the data which do not affect our interpretation of it; for example, if $F(\mathbf{x}) \neq F'(\mathbf{x})$, but both $F(\mathbf{x})$ and $F'(\mathbf{x})$ lie in the range taken to represent air, so that $\tau(F(\mathbf{x})) = \tau(F'(\mathbf{x}))$, no distortion is considered to have occurred at \mathbf{x} . This allows the distortion metric to take advantage of the a priori knowledge of the dataset encapsulated in τ .

We then define P , the set of points at which $F(\mathbf{x})$ and $F'(\mathbf{x})$ are considered to represent different features, by

$$P = \left\{ \forall \mathbf{x} \in X \mid \tau(F(\mathbf{x})) \neq \tau(F'(\mathbf{x})) \right\}. \quad (9)$$

We may then define m_d as the ratio of the number of points in P to the total number of points occupied by $F(\mathbf{x})$:

$$m_d = \frac{|P|}{|X|} = \frac{|P|}{R_1 \cdot R_2 \cdot R_3}. \quad (10)$$

It follows that low values of m_d indicate lower distortion, and that $0 \leq m_d \leq 1$ (since $P \subseteq X$). The ideal smoothing method will have m_d close to zero, indicating no distortion.

An alternative measure of distortion might be to weight the metric, imposing a proportionally higher penalty in regions where a point on an iso-surface of interest in F lies further from the corresponding point in F' ; that is, in areas where the processed dataset differs strongly from the original. A suitable metric might be based on the following equation for m_d :

$$m_d = \sqrt{\sum \psi_{F,F'}(\mathbf{x})^2} \quad (11)$$

$\forall \mathbf{x} \in X$ such that \mathbf{x} lies on the iso-surface of interest in F , and where $\psi_{F,F'}(\mathbf{x})$ represents the distance from \mathbf{x} to the corresponding point in F' .

However, there are some practical issues with this metric, not least the problem of establishing the location of the corresponding point for \mathbf{x} . Whilst nonlinear diffusion

theory guarantees continuity in the succession of datasets $U(\mathbf{x}, 0)$ to $U(\mathbf{x}, t)$ (that is, features smoothly and continuously evolve into their final state at time t), and that in principle features may be tracked, in practice implementing this would be considerably more complex than the previous approach. Moreover, it is not clear that such a weighted metric is desirable: there is no obvious order of preference between a processed dataset with a moderate level of shape degradation everywhere and one with low degradation generally and a few areas of high degradation, given an equal score on the first proposed metric. For these reasons we use the metric proposed in (10) to evaluate distortion.

5.2 Coherence Improving Metric

It is harder to design a coherence metric, since the goal of this work is to some extent subjective. However, proceeding from the stated goal that we wish to produce refractive images which are free from noise, it follows that an object-space metric might be somewhat contrived, on the grounds that this metric aims to measure the improvements made in the image domain. (The previous metric for distortion measures shape degradation, which occurs in the volume domain, and is therefore more appropriately measured in this domain). We will therefore consider image-space methods of evaluating improvement.

Since object shape degradation is considered in the preceding metric, we ignore this here and focus only on coherence. Comparing, for example, the images in Fig. 3, we see that the images rendered with unperturbed normals contains less energy at high frequencies than their counterparts with perturbed normals. This observation has been observed to hold for a range of images. We may measure improvements, therefore, by considering the frequency distribution of the image; images with a high proportion of coherent rays (that is, rays which travel along similar paths to adjacent rays) will tend to contain less high frequency energy.

We consider the frequency distribution by applying the discrete Fourier transform to the images. The result is a two dimensional complex array; in order to reduce this to a single value which can be easily compared for different images, we multiply the magnitude of each element in the array, which corresponds to the amount of energy present at a certain frequency, by the frequency represented by that element, and take the average of the resulting array. The result is a number which gives an indication of frequency distribution for the entire image, with values close to zero indicating few high frequencies, and large values indicating a large quantity of high frequency energy. We express this coherence metric m_c mathematically in the following way, given an image $i : \mathbb{N}^2 \rightarrow \mathbb{N}$ of resolution M, N :

$$I(u, v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} i(x, y) e^{-2\pi j(ux/M + vy/N)} \quad (12)$$

$$m_c = \frac{4}{MN} \sum_{u=0}^{\frac{M-1}{2}} \sum_{v=0}^{\frac{N-1}{2}} |I(u, v)| \cdot \sqrt{\frac{u^2}{M^2} + \frac{v^2}{N^2}}. \quad (13)$$

According to the periodicity property of the discrete Fourier transform (given by (12), the discrete Fourier transform of $i(x, y)$, denoted here by $I(u, v)$, is symmetric about the origin, and repeats horizontally and vertically respectively, at distances $\frac{M}{2}$ and $\frac{N}{2}$ from the origin. For this reason, in order to analyze the frequencies present in the discrete Fourier transform we need only consider the values in the range $(0, 0)$ to $(\frac{M}{2}, \frac{N}{2})$; therefore we sum only these frequencies, and multiply the result by $\frac{4}{MN}$, which is the reciprocal of the number of values considered. Note that in computing the frequency represented by some point in $I(u, v)$ – that is, in the last term of (13) – we scale u, v by M, N respectively in order to ensure that the value produced by this metric is (approximately) invariant with respect to image size and resolution, modulo changes in precision attributable to change in resolution.

This metric measures the high frequency content of the image, and is sensitive to edges as well as noise. In other words, it measures the smoothness of the silhouette of the rendered volume object, as well as the level of noise, which is a noticeable but undesirable feature in rendering volume datasets with refraction. In general, after denoising, the number of edge pixels in the image (i.e., the silhouette of the volume object) does not change significantly, whereas the level of noise does decrease in a noticeable manner. Thus, the numerical difference given by this metric can reliably be attributed to the changing levels of noise.

As will be seen in Sect. 6, this metric results in values which tend to decrease as λ or t increase. This decrease is not exactly monotonic, since a few data points do not fit this pattern; however, the trend is otherwise significant enough to suggest that this metric has some correspondence with increased ray coherence in the rendered images. Whilst it is not meaningful to use this metric to compare unrelated images, it is nonetheless useful in comparing similar images from a series of datasets processed with different parameters but rendered using identical parameters.

We demonstrate the validity of this metric in Fig. 7. Here, the CT Head dataset is processed using *anisotropic nonlinear diffusion* (described in Sect. 4.3) with varying parameters. The parameters are selected such that m_c has a similar value in each case; it can be seen that the resulting images exhibit comparable levels of coherence, especially when contrasted with the images in Fig. 5, which exhibit substantial diversity.

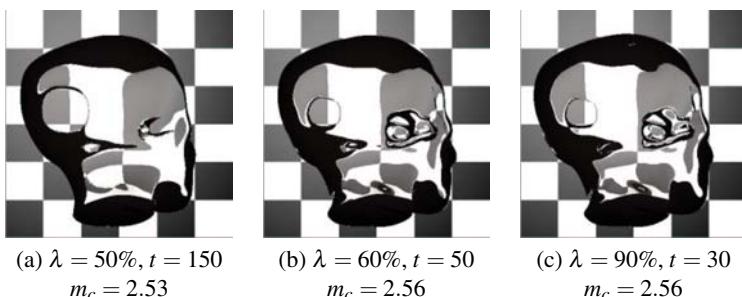


Fig. 7. Images with similar values for m_c

6 Results and Remarks

Figures 4 and 5 show the results of applying the methods discussed in Sect. 4 to the CT Head dataset, and for each method, examples of both opaque and refractive rendering are given. The methods vary widely in both the degree to which image quality is improved, and in the level of distortion introduced. In the following sections, we discuss these images subjectively, and with respect to the metrics introduced in Sect. 5.

The CT Head dataset is a demanding test for refractive visualization, in that it features significant levels of noise in all regions of the dataset, as a consequence of the scanning process, and particularly high levels of noise around the teeth, attributed to metal fillings in the teeth interfering with the original scanning process. Additionally, the pockets of air in the nasal cavities cause frequent, incoherent refraction in this region. There are some pre-aliasing artefacts present at the top of the head, visible as curved bands in the images in the first row of Fig. 5. The presence of fine structures, such as the nose and the ear pose additional problems for smoothing algorithms which aim to preserve the overall shape.

6.1 Qualitative Analysis of Results

The opaque images in Fig. 4 demonstrate that both *inhomogeneous linear diffusion* and *anisotropic nonlinear diffusion* are poor at preventing distortion. *Inhomogeneous linear diffusion* performs particularly badly with respect to fine structures, as can be seen in the erosion of the nose. Moreover, it fails to smooth the noise around the teeth. *Anisotropic nonlinear diffusion* distorts heavily everywhere; large features such as the dimples in the side of the face are smoothed away. *Gaussian convolution* and *regularized anisotropic nonlinear diffusion* both perform well, although the images in the fourth column indicate that *Gaussian convolution* performs less well at preserving thin structures in the data (such as the nose or the ears). For both of these methods, the overall shape of the head is well preserved; both deal well with the noise around the teeth.

Improvements in coherence can be observed in Fig. 5. *Gaussian convolution* performs well; as a consequence of filtering out all high frequencies, noise is reduced, and adjacent rays tend to travel along more closely matched paths, leading to significant overall improvement in image quality. It is also effective at reducing pre-aliasing artefacts. *Inhomogeneous linear diffusion* shows some improvements; noise in all parts of the dataset has been significantly reduced. Some new artefacts have been introduced, however; these may be observed in the region above the right ear. This method also fails to smooth the pre-aliasing artefacts. *Anisotropic nonlinear diffusion* is extremely effective at smoothing both types of artefact, even for low values of t ; however, this method also introduces the same artefacts as *inhomogeneous linear diffusion*. *Regularized anisotropic nonlinear diffusion* performs well, smoothing both pre-aliasing artefacts and noise, without introducing new artefacts.

6.2 Hybrid Rendering Method

A possible approach for improving image quality without affecting distortion is to use the smoothed dataset only for computation of normals. This is demonstrated in Fig. 8, using *regularized anisotropic nonlinear diffusion* ($\lambda = 60\%$, $t = 50$). Figure 8(a) shows the original (un-smoothed) dataset; Fig. 8(b) shows the hybrid approach, and Fig. 8(c) both dataset values and normals computed from the smoothed dataset.

The benefits of this approach are clear, in that it offers a means of improving image quality without distorting the original data at all. However, the smoothed dataset may not always have appropriate normals in the location of boundaries in the original data; this problem prevents it from being as effective in increasing coherence as the conventional method of rendering; and in fact as F' diverges further from F , artefacts are introduced, as may be seen in Fig. 8(c).

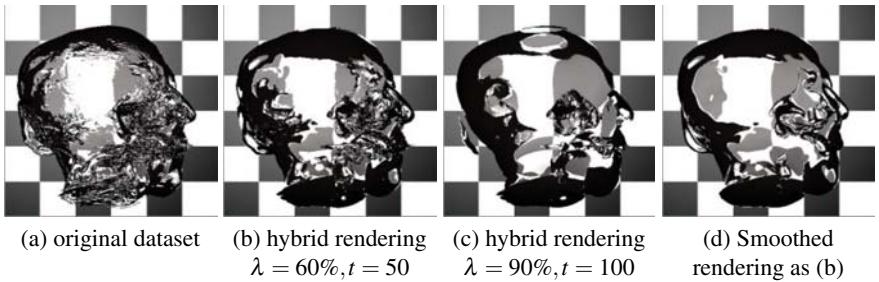


Fig. 8. Hybrid rendering method

6.3 Quantitative Analysis of Results

The data in Table 1, which is obtained from the metrics discussed in Sect. 5, suggests that *regularized anisotropic nonlinear diffusion* performs better than all other methods. Direct comparisons are difficult, since values for both m_d and m_c do not exactly correspond.

We therefore select values for t, λ and σ such that m_d remains approximately constant in Table 2, so that we may directly compare the effectiveness of different methods in terms of m_c . When m_d is fixed at around 0.015 giving similar level of distortion for all methods, the *regularized anisotropic nonlinear diffusion* method (and, to a lesser extent, the *anisotropic nonlinear diffusion* method) offers better improvement of smoothness as measured by the coherence metric.

Similarly, in Table 3, we select parameter values such that m_c remains approximately constant to enable us examine results in terms of m_d . It is clear that when m_d is fixed at around (for instance) 15.1, *regularized anisotropic nonlinear diffusion* results in much less distortion than any of the other three methods.

Table 1. Results of applying distortion and coherence metrics to datasets and refractive images from Fig. 5

Gaussian Filter	σ	0.0	1.5	3.0	4.5
<i>Gaussian convolution</i>	m_d	0	0.0027	0.0108	0.0287
	m_c	23.19	19.40	16.34	13.88
Diffusion Methods with $\lambda = 60\%$	t	0	20	40	60
<i>Inhomogeneous linear diffusion</i>	m_d	0	0.0040	0.0195	0.0365
	m_c	23.19	18.36	16.55	15.96
<i>Anisotropic nonlinear diffusion</i>	m_d	0	0.0122	0.0398	0.0617
	m_c	23.19	14.12	10.80	10.44
<i>Regularized anisotropic nonlinear diffusion</i>	m_d	0	0.0024	0.0066	0.0102
	m_c	23.19	17.22	14.70	13.33

Table 2. Comparison of results for different methods in a condition where parameters selected to give similar values for m_d . Methods are listed in the descending order of the noise level (i.e., incoherence) indicated by m_c .

Method	m_d	m_c
<i>Inhomogeneous linear diffusion</i> ($t = 35, \lambda = 60\%$)	0.0147	16.88
<i>Gaussian convolution</i> ($\sigma = 3.18$)	0.0151	16.09
<i>Anisotropic nonlinear diffusion</i> ($t = 15, \lambda = 70\%$)	0.0153	13.64
<i>Regularized anisotropic nonlinear diffusion</i> ($t = 95, \lambda = 60\%, \sigma = 1$)	0.0152	12.11

Table 3. Comparison of results for different methods in a condition where parameters are selected to give similar values for m_c . Methods are listed in the descending order of the amount of distortion indicated by m_d

Method	m_d	m_c
<i>Inhomogeneous linear diffusion</i> ($t = 30, \lambda = 70\%$)	0.0261	15.12
<i>Gaussian convolution</i> ($\sigma = 3.75$)	0.0253	15.08
<i>Anisotropic nonlinear diffusion</i> ($t = 75, \lambda = 40\%$)	0.0204	15.13
<i>Regularized anisotropic nonlinear diffusion</i> ($t = 135, \lambda = 40\%, \sigma = 1$)	0.0067	15.06

This confirms the outcome of qualitative analysis in Sect. 6.1 based on the visual results in Figures 4 and 5. *Regularized anisotropic nonlinear diffusion* is clearly most effective at improving image quality whilst preserving the shape of the original data, whilst preserving the shape of the original data. *Anisotropic nonlinear diffusion* performs less well, and *Gaussian convolution* and *inhomogeneous linear diffusion* both perform relatively poorly.

Considering the overall ability to improve ray coherence with minimal cost of distortion, Fig. 9 gives an intriguing illustration of the relative performance of these methods, each depicted by a line graph with m_d as its x -axis. *Gaussian convolution* was applied to the CT datasets with a series of σ values, $\sigma = 0, 1, 2, 3, \dots$ marked by the corresponding data points. All three diffusion methods are computed with $\lambda = 40\%$ and $t = 0, 20, 40, \dots, 140$. $\sigma = 4$ was chosen for computing the line graph

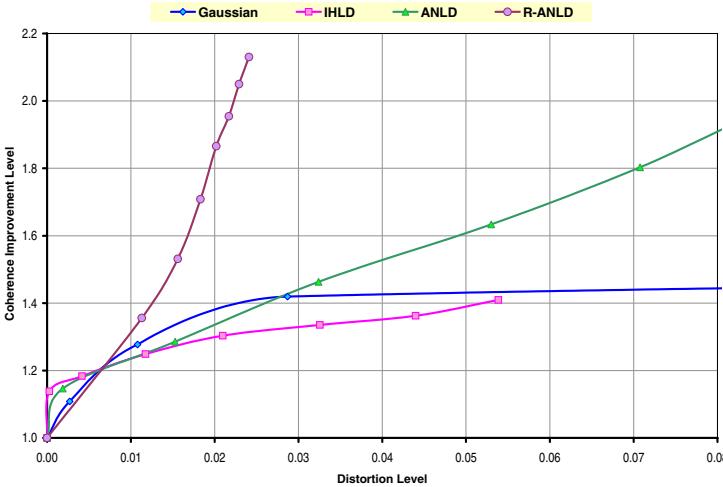


Fig. 9. Relative performance of four different denoising methods. The measurement of coherence improvement is the result of dividing the m_c value obtained from the unprocessed dataset by the m_c value obtained from the processed dataset

of *regularized anisotropic nonlinear diffusion*. The y-axis indicates the relative improvement of coherence m_c in the context of a rendered refractive image with a view similar to those in Fig. 5. For example, we compute $m_c(\sigma = 0)/m_c(\sigma)$ for the *Gaussian convolution*, and $m_c(t = 0)/m_c(t)$ for the diffusion methods. Figure 9 supports the conclusions drawn above; in particular, the limitations of Gaussian convolution are revealed, and the strong performance of *regularized anisotropic nonlinear diffusion* is made apparent.

Some more detailed observations about the performance of *regularized anisotropic nonlinear diffusion* may be derived from the graphs in Fig. 10. For values of λ below roughly 50%, the rate of growth of m_d is low. The rate of reduction of m_c , however, with respect to λ , is fairly uniform in the range 30% – 70%. For these reasons, it seems reasonable to suggest that suitable choices for λ typically lie around the 50th percentile. This is borne out informally by experiences using these methods, although this cannot be proven without further experimentation on a range of datasets.

7 Conclusions and Future Work

We have shown that it is possible to achieve good results when using refraction to render datasets which contain noise, pre-aliasing and small features. Figure 2, which depicts the CT head dataset both (a) unprocessed and (b) after application of *regularized anisotropic nonlinear diffusion*, gives a compelling example of deploying such techniques in medical visualization.

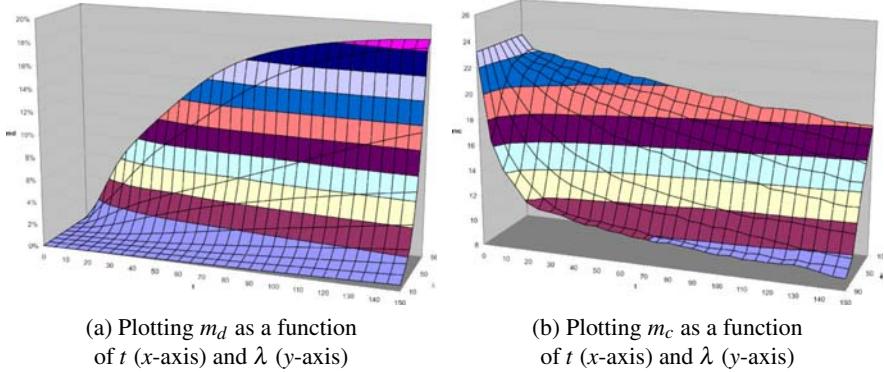


Fig. 10. Metrics for *regularized anisotropic nonlinear diffusion*

By introducing metrics which allow us to quantify distortion and improvements in image quality, we are able to make objective comparisons between a number of different methods for smoothing datasets. *Regularized anisotropic nonlinear diffusion* appears to be the most effective of the methods tested, both from a theoretical standpoint, and in practice. We believe by using these methods for refractive visualizations that depth perception can be improved, and that significant improvements in image quality may be attained, as Fig. 11 shows.

It is possible to eliminate distortion by means of the hybrid rendering method; however, this method cannot achieve the same subjective levels of quality as directly rendering smoothed datasets.

There exists a clear tradeoff between image quality and distortion. However, given the large parameter space and the complex relationship between these parameters and the resulting image quality (with respect to both coherence and distortion), it is difficult to present a ‘best’ approach that is suitable for all applications.

The success of *regularized anisotropic nonlinear diffusion* can be attributed to the fact that it is rather more sophisticated than the other methods considered. To summarize the differences between the methods used, Gaussian filtering provides a homogenous smoothing process which cannot be locally controlled. Thus, geometric details of the volume object, such as the CT head in our examples, are lost. The *inhomogeneous linear diffusion* method provides a simple control mechanism (in that flow is scaled by $g(|\nabla F|^2)$), but the controls are ‘fixed’ according to the original dataset; thus, as smoothing progresses, the relevance of these controls diminishes. As Fig. 9 illustrates, *inhomogeneous linear diffusion* is the only non-linear method to experience a tapering off in performance. *Anisotropic nonlinear diffusion* addresses this weakness by having the controls incorporate feedback from the evolving dataset, and as a result, is able to perform better than Gaussian convolution. *Regularized anisotropic nonlinear diffusion* refines the *anisotropic nonlinear diffusion* method further, by incorporating a regularization step. This makes the controls themselves less sensitive to noise. In the limit, as σ approaches infinity, the control term approaches a constant, and thus the *regularized anisotropic nonlinear diffusion*

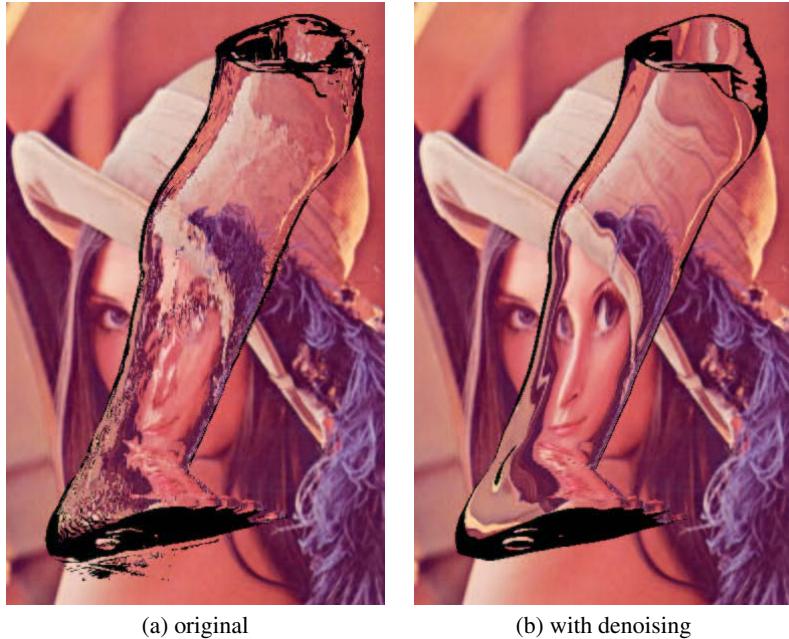


Fig. 11. A statue leg rendered **(a)** without and **(b)** with application of *regularized anisotropic nonlinear diffusion*

process can be steered between *anisotropic nonlinear diffusion* and Gaussian convolution by selecting a suitable value for σ . The effect is to dampen the feedback mechanism somewhat, improving the stability of the process and eliminating stair-casing artefacts.

We aim to continue this work by focusing more strongly on *regularized anisotropic nonlinear diffusion*; in particular, by examining different diffusion functions, and considering the effects of varying σ in more detail. In addition, other methods for denoising may be able to improve on these results.

Acknowledgments

The first author is grateful for his PhD studentship from University of Wales Swansea for the period 1999-2003. The CT Head dataset is the courtesy of University of North Carolina, Chapel Hill, the statue leg dataset courtesy of German Federal Institution for Material Research and Testing, Berlin, Germany. All images in this paper are rendered using *vlib* [31].

References

1. L. Alvarez, P.L. Lions, and J.M. Morel. Image selective smoothing and edge detection by nonlinear diffusion. *SIAM J. Numer. Anal.*, 29:845–866, 1992.
2. E. Angelini, A. Laine, S. Takuma, and S. Homma. Directional representations of 4D echocardiography for temporal quantification of LV volume. In *Proc. MICCAI'99*, pages 430–440, Cambridge, 1999.
3. M. Bertram. Fairing scalar fields by variational modeling of contours. In *Proc. IEEE Visualization 2003*, pages 387–392, Seattle, Washington, 2003.
4. U. Clarenz, U. Diewald, and M. Rumpf. Anisotropic geometric diffusion in surface processing. In *Proc. IEEE Visualization 2000*, pages 397–405, Salt Lake City, Utah, 2000.
5. M. Desbrun, M. Meyer, P. Schröder, and A. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proc. SIGGRAPH 1999*, pages 317–324, 1999.
6. M. Desbrun, M. Meyer, P. Schröder, and A. Barr. Anisotropic feature-preserving denoising of height fields and bivariate data. In *Proc. Graphics Interface 2000*, pp. 145–152, 2000.
7. Josiah W. Gibbs. Fourier series. *Nature*, 59, 1899.
8. I. Guskov and Z. Wood. Topological noise removal. In *Proc. Graphics Interface 2001*, pp. 19–26, 2001.
9. M. Hilton, T. Ogden, D. Hattery, G. F. Eden, and B. Jawerth. Wavelet denoising of functional MRI data. In A. Aldroubi and M. Unser, editors, *Wavelets in Biology and Medicine*, pp. 93–112. CRC Press, 1996.
10. D. T. Kuan, A. A. Sawchuk, T. C. Strand, and P. Chavel. Adaptive noise smoothing filter for images with signal-dependent noise. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-7:165–177, 1985.
11. J. S. Lee. Digital image enhancement and noise filtering by use of local statistics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2:165–168, 1980.
12. Steve Marschner and Richard Lobb. An evaluation of reconstruction filters for volume rendering. In R. D. Bergeron and Arie Kaufman, editors, *Proc. IEEE Visualization '94*, pp. 100–107. IEEE Computer Society Press, October 1994.
13. Torsten Möller, Raghu Machiraju, Klaus Mueller, and Roni Yagel. A comparison of normal estimation schemes. In *Proc. IEEE Visualization '97*, pp. 19–26, Phoenix, AZ, November 1997.
14. Torsten Möller, Raghu Machiraju, Klaus Mueller, and Roni Yagel. Evaluation and design of filters using a Taylor series expansion. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):184–199, April - June 1997. ISSN 1077-2626.
15. Torsten Möller, Klaus Mueller, Yair Kurzion, Raghu Machiraju, and Roni Yagel. Design of accurate and smooth filters for function and derivative reconstruction. In *Proc. IEEE Symposium on Volume Visualization*, pp. 143–151, October 1998.
16. H. P. Moreton and C. H. Sequin. Functional optimization for fair surface design. In *Proc. SIGGRAPH '92*, pp. 167–176, 1992.
17. L. Neumann, B. Csébfalvi, A. König, and E. Gröller. Gradient estimation in volume data using 4D linear regression. *Computer Graphics Forum*, 19(3):C351–C357, 2000.
18. Y. Ohtake, A. G. Belyaev, and I. A. Bogaevski. Polyhedral surface smoothing with simultaneous mesh regularization. In *Proc. Geometric Modeling and Processing 2000*, pp. 229–237, Hong Kong, 2000.
19. J. Peng, V. Strela, and D. Zorin. A simple algorithm for surface denoising. In *Proc. IEEE Visualization 2001*, pp. 107–112, San Diego, California, 2001.

20. Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, July 1990.
21. M. P. Persoon, I. W. O. Serlie, F. H. Post, and F. M. Vos. Visualization of noisy and biased volume data using first and second order derivative techniques. In *Proc. IEEE Visualization 2003*, pp. 379–385, Seattle, Washington, 2003.
22. David Rodgman and Min Chen. Refraction in discrete raytracing. In Klaus Mueller and Arie Kaufman, editors, *Proc. Volume Graphics 2001*, New York, 2001. Springer. ISBN 3-211-83737-X.
23. C. Rössl, F. Zeilfelder, G. Nürnberger, and H.-P. Seidel. Visualization of volume data with quadratic super splines. In *Proc. IEEE Visualization 2003*, pp. 393–400, Seattle, Washington, 2003.
24. G. Sapiro. *Geometric Partial Differential Equations and Image Analysis*. Cambridge University Press, 2001.
25. T. Tasdizen, R. Whitaker, P. Burchard, and S. Osher. Geometric surface smoothing via anisotropic diffusion of normals. In *Proc. IEEE Visualization 2002*, pp. 125–132, Boston, Massachusetts, 2002.
26. Thomas Theußl. *Sampling and Reconstruction in Volume Visualization*. PhD thesis, Vienna University of Technology, 2000.
27. Joachim Weickert. A review of nonlinear diffusion filtering. In *Scale-Space Theories in Computer Vision*, pp. 3–28, 1997.
28. Joachim Weickert. *Anisotropic Diffusion in Image Processing*. ECMI Series, Teubner, Stuttgart, 1998.
29. W. Welch and A. Witkin. Free-form shape design using triangulated surfaces. In *Proc. SIGGRAPH '94*, pp. 247–256, 1994.
30. R. T. Whitaker. Volumetric deformable models: active blobs. In R. A. Robb, editor, *Proc. Visualization in Biomedical Computing, SPIE*, 1994.
31. Andrew S. Winter and Min Chen. vlib: A volume graphics API. In *Proc. Volume Graphics 2001*, pp. 133–147, New York, 2001. Springer.
32. Roni Yagel, Daniel Cohen, and Arie Kaufman. Normal estimation in 3D discrete space. *The Visual Computer*, pp. 278–291, 1992.

Emphasizing Isosurface Embeddings in Direct Volume Rendering

Shigeo Takahashi¹, Yuriko Takeshima², Issei Fujishiro³, and
Gregory M. Nielson⁴

¹ The University of Tokyo, Tokyo, 153-8902 Japan
`takahashis@acm.org`

² Japan Atomic Energy Research Institute, Tokyo, 110-0015 Japan
`takesima@koma.jaeri.go.jp`

³ Ochanomizu University, Tokyo, 112-8610 Japan
`fiji@is.ocha.ac.jp`

⁴ Arizona State University, Tempe AZ, 85287 USA
`nielson@asu.edu`

1 Introduction

Providing clear insight into the inner structures involved in volume dataset has been a challenging task in the field of volume visualization. However, due to the recent progress in the performance of computing/measurement environments, objective volume datasets have become larger and more complicated. Such volume datasets ubiquitously involve nested structures of 3D scalar fields, regardless of how they are acquired. Some typical examples include biological structures, chemical interactions, and mechanical shapes.

A volume dataset contains an infinite number of disjointed isosurfaces at different target scalar field values, while its field structure is often characterized by spatial configurations of a finite number of feature isosurfaces that segment the volume dataset into several important components. In the volume dataset which involves nested inner structures, the connected components of the feature isosurface commonly have inclusion relationships in 3D space. Such isosurface spatial configurations are referred to as *isosurface embeddings* in this paper. This type of spatial configuration reveals some meaningful structures of the objective volume, and differs from the simple isosurface occlusions [4] in that the isosurface embeddings are independent of the viewing direction.

Although direct volume rendering is one of the popular techniques for semi-automatically visualizing the entire volumetric inner structures at once, it still requires time-consuming human interactions for tweaking visualization parameters to obtain comprehensible rendering results. In particular, conventional transfer functions cannot fully clarify such inner structures, especially when the connected components of an isosurface are intricately nested. This is because the conventional

transfer functions cannot assign different optical properties to voxels of the same scalar field value while the nested isosurfaces may also have the same scalar field value.

Figure 11 shows such an example. This dataset is obtained by simulating two-body distribution probability of a nucleon [20], and actually involves complicated inclusion relationships between isosurfaces at some scalar field values. Figure 11(b) is obtained using naive transfer functions with linear hue and flat opacity, Fig. 11(c) is obtained using conventional transfer functions that only accentuate some feature isosurfaces [7, 31, 33], and Fig. 11(d) is obtained using our new framework that also emphasizes nested structures of the feature isosurfaces. Unfortunately, the first result conveys nothing about the inner structures involved in the dataset. While the second result accentuates the feature isosurfaces individually, the corresponding method still misses their associated nested structures. For example, the small sphere-like isosurface on the top clearly appears in Fig. 11(d) while it is missing in Fig. 11(c).

This paper presents a new framework for identifying and visualizing such isosurface embeddings involved in the volume datasets. The contributions of this paper are twofold. The first contribution is an algorithm for extracting the isosurface embeddings by tracing the topological volume skeleton delineated from the given volume dataset. As presented in Fig. 11(a), the topological volume skeleton is a level-set graph that represents topological transitions of isosurfaces with respect to the scalar field, and allows us to find feature isosurfaces that have significant meaning in the given dataset. The second is an algorithm for emphasizing such nested inner structures using multi-dimensional transfer functions so that we can render each voxel separately according to its relative geometric position as well as its scalar field value. The second algorithm calculates an additional dimension for the multi-dimensional transfer functions to visualize the nested structures of the volume datasets.

This paper is organized as follows: The next section describes previous work related to our approach. In Sect. 3, we introduce the new algorithm that identifies isosurface embeddings through the topological volume skeletonization. Sect. 4 describes how to design the multi-dimensional transfer functions in order to emphasize the isosurface embeddings. Sect. 5 demonstrates the feasibility of our framework with application to several datasets. After discussing the validity of our new framework in Sect. 6, we conclude this paper and refer to our future work in Sect. 7.

2 Related Work

The present framework is related to several research categories, which can be classified into the following four groups:

Level-Set Analysis

Our algorithm seeks inclusion relationships between isosurfaces by analyzing a level-set graph that symbolizes topological isosurface transitions according to the change of the scalar field. The contour tree is one of the common graphs that effectively represent the level-set of a given volume. Van Kreveld et al. [32] developed an

excellent algorithm for calculating contour trees with minimal computational complexity, and Bajaj et al. [1] used it to explore feature isosurface transitions for visualization purposes. Carr et al. [5] extended this algorithm to handle objects of higher dimensions. Recently, Pascucci et al. [22] developed an algorithm that identifies the topology (i.e. genus) of a connected component of an isosurface at any point on the contour tree.

Our level-set graph, called the volume skeleton tree [31], differs in that it can capture the transitions of isosurface spatial configurations as well as the changes in the number of connected isosurface components and their associated topologies (i.e., genera). This is enabled by identifying each type of topological isosurface transition correctly and then traversing the entire level-set graph, as described in Sect. 3.4.

Occlusion Culling

To our knowledge, the present algorithm is the first to extract view-independent nested structures, i.e., isosurface inclusion relationships, without multi-directional ray intersection tests. On the other hand, several excellent algorithms have been proposed to detect view-dependent features such as object occlusions. For example, Schaufler et al. [24] proposed a method of calculating conservative volumetric visibility, while Kłosowski et al. [17] presented an approximate visibility culling technique. Refer to an excellent survey of the visibility problems by Cohen-Or et al. [4] for details.

Transfer Function Design

The appropriate design of transfer functions is crucial to direct volume rendering in visualizing inner structures of the given volume datasets. In recent years, various methods have been proposed to design such transfer functions that take into account existing volumetric features.

Castro et al. [2] formalized transfer functions specific to medical applications as the linear combinations of basis functions assigned to different tissues such as bones, skins and muscles. Kindlmann et al. [14] proposed a method that emphasizes boundaries between different materials in a volume using histogram volume, which captures relationships between the scalar field and its first and second directional derivatives throughout the volume.

Recently, multi-dimensional transfer functions, which originate from the pioneering work by Levoy [18], have received much attention because they can distinguish between voxels according to some variables in addition to the scalar field value. For example, Kniss et al. [15, 16] designed 3D transfer functions based on the method of Kindlmann et al. [14]. Moreover, Hladuvka et al. [12] proposed a different approach where they used volume curvature as an additional variable for the multi-dimensional transfer functions.

While these methods work well, they sometimes suffer from gentle gradients of the scalar field especially when handling simulated datasets. This is because they only take into account local shape features such as derivatives and curvatures in designing transfer functions, and often miss the underlying global structures in the

input datasets. Nevertheless, our approach to transfer function design [31] makes it possible to emphasize such global structures effectively by extracting the topological transitions of isosurfaces from input volumes. While Fujishiro et al. [6, 7, 11] and Weber et al. [33] presented methods for locating topological changes of isosurfaces for this purpose, they still considered local features around the critical points only and ignored their associated global connectivity.

Non-Photorealistic Volume Rendering

Another approach to illuminating such nested structures is non-photorealistic techniques in volume rendering. Interrante [13] developed a method of mapping LIC-based textures in order to identify each connected components in feature isosurfaces. Treavett et al. [27] applied pen-and-ink strokes to isosurface rendering to enhance the conventional volume rendering pipeline. While these two methods succeeded in generating intuitive visualization results, they are rather oriented to isosurface rendering. On the other hand, Rheingans et al. [23] succeeded in incorporating non-photorealistic techniques into direct volume rendering. Csébfalvi et al. [3] then achieved interactive rate of such non-photorealistic volume rendering for fast exploration of involved volume structures. Recently, Lu et al. [19] introduced stippling techniques in such non-photorealistic volume rendering to accentuate object boundaries and silhouettes. Although these non-photorealistic rendering methods are compelling, they still miss some significant features because they extract volumetric features without referring to the global structures of the input volume datasets.

3 Extracting Isosurface Embeddings

This section describes the first contribution of this paper, an algorithm for extracting isosurface embeddings involved in a volume dataset. The second contribution will be described in Sect. 4.

Actually, the algorithm has been implemented by enhancing our previous approach called *topological volume skeletonization* [31], which analyzes topological transitions of isosurfaces with respect to the scalar field. The topological volume skeletonization is originally developed for finding important scalar field values to be emphasized in the design of transfer functions, which helps us find appropriate visualization parameters for comprehensible volume rendering.

The key idea of our new algorithm here is to distinguish the specific type of topological transition in isosurfaces that yields inclusion relationships between their connected components. In the analysis, a level-set graph called a *volume skeleton tree* (VST) plays a central role in extracting such isosurface transitions. Figure 1 shows a nested isosurface structure and its corresponding VST of a volume dataset, which is calculated from the following analytic volume function:

$$f(x, y, z) = (x^2 + y^2 + z^2)(x^2 + y^2 + z^2 - a) - bx, \\ \text{where } a > 0, \quad b > 0, \quad 8a^3 > 27b^2. \quad (1)$$

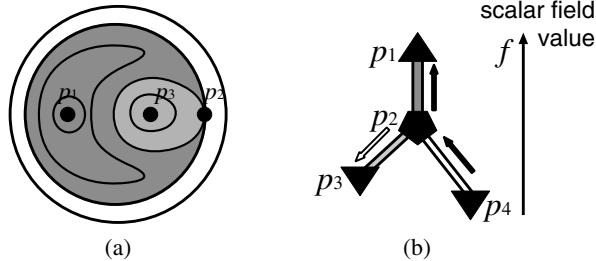


Fig. 1. An example of a nested structure: (a) isosurface transitions and (b) its VST

Here, the VST node corresponds to a critical point where a topological change occurs in isosurfaces, and its link represents an isolated interval volume [9, 10] confined by isosurfaces associated with the two end critical points. Note that, throughout this paper, we arrange the VST nodes p_i from top to bottom according to their corresponding scalar field values $f(p_i)$, as shown in Fig. 1(b). In this example, the node p_2 gives rise to the specific type of isosurface transition that results in the inclusion relationships between isosurfaces within the scalar field interval $[f(p_3), f(p_2)]$. The remainder of this section describes how to capture such inclusion relationships together with the volume skeletonization process.

3.1 Assumptions on Isosurface Transitions

First of all, we make some assumptions on topological transitions of isosurfaces. In general, the input volume dataset can be represented as discrete grid samples of a 3D single-valued function:

$$w = f(x, y, z), \quad (2)$$

where x , y , and z represent ordinary 3D coordinates and w represents its corresponding scalar field value. This lets us classify evolving isosurfaces into two categories: *solid isosurfaces* where their interior samples are larger in the scalar field than those on the corresponding isosurfaces, and *hollow isosurfaces* where the interior is smaller. We can also orient the w -axis in such a way that the outermost isosurfaces become solid and thus expand as the scalar field value decreases. This implies, under the assumption of single-valued volume functions, that solid isosurfaces always expand as the scalar field value decreases while hollow isosurfaces always shrink.

3.2 Critical Points

Critical points are crucial to our algorithm for finding inclusion relationships between feature isosurfaces because such a relationship inevitably occurs at a specific type of critical point. Roughly speaking, a critical point is defined to be a point that activates a topological change in isosurface evolution such as creation, merging, splitting, and

deletion of isosurfaces. The field value associated with a critical point is called a *critical scalar field value* in this paper. Figure 1(a) shows three critical points, where p_1 invokes isosurface creation, p_2 joins two surface regions, and p_3 deletes an existing isosurface. Actually, p_2 is one of the specific types of critical points that generate nested structures of isosurfaces. Note that in Fig. 1(b) a critical point p_4 is added to the VST. This is called the *virtual minimum* in this paper, which is introduced to assure the correctness of the extracted critical points, and will also serve as a clue to our algorithm for extracting inclusion relationships between feature isosurfaces for later use.

More formally, a critical point p of the function (2) is defined to be a point that satisfies

$$\frac{\partial f}{\partial x}(p) = \frac{\partial f}{\partial y}(p) = \frac{\partial f}{\partial z}(p) = 0. \quad (3)$$

According to the Morse lemma [8], an infinitesimal neighborhood around a critical point of the function (2) has a local coordinate system where f has either of the following quadratic forms:

$$f = \begin{cases} -x^2 - y^2 - z^2 & \text{maximum (index 3)} \\ -x^2 - y^2 + z^2 & \text{saddle (index 2)} \\ -x^2 + y^2 + z^2 & \text{saddle (index 1)} \\ x^2 + y^2 + z^2 & \text{minimum (index 0).} \end{cases} \quad (4)$$

Here, the index represents the number of negative eigenvalues of the Hessian matrix at the critical point. This leads us to the fact that the critical points of the function (2) are classified into four types: a maximum (index 3), a saddle (index 2), a saddle (index 1), and a minimum (index 0). In the following, we use the symbols C_3 , C_2 , C_1 , and C_0 to represent the above critical points, where each subscript represents the index of the corresponding critical point.

Figure 2 shows isosurface transitions around a maximum (C_3) and a minimum (C_0). At a maximum, a new topological sphere appears in 3D space, while an existing sphere disappears at a minimum.

At a saddle of index 2 (C_2) two isosurface regions are merged, while an existing isosurface region is split into two at a saddle of index 1 (C_1). Thus, the corresponding topological changes become more complicated than those at maxima and minima. Note here that some specific types of saddles definitely introduce isosurface inclusion relationships which we are going to emphasize in our rendering process. This requires rigorous classification of isosurface transitions at the saddles by taking



Fig. 2. Isosurface transitions at a maximum and a minimum

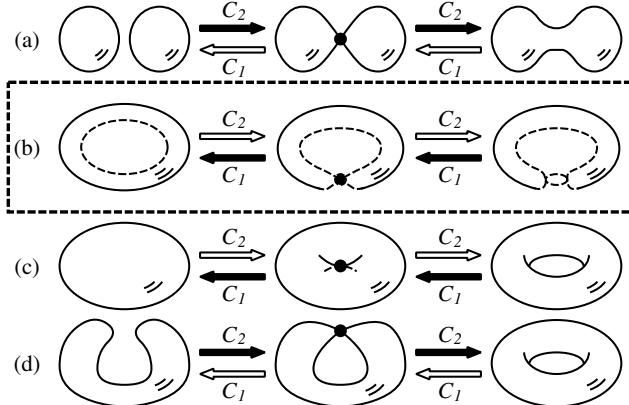


Fig. 3. Classification of isosurface transitions at saddles depending on the embedding in 3D space. Thick arrows represent transitions of solid isosurfaces while outlined arrows represent those of hollow isosurfaces. Isosurface inclusion relationships occur only in the transition path (b) (outlined by broken lines)

into account their embeddings into 3D space. Figure 3 shows such a classification result where each type of saddle possesses four different topological transitions. Furthermore, the previous assumption in Sect. 3.1 restricts such isosurface transitions at saddles as indicated by thick and outlined arrows in the figure. Here, solid isosurfaces can follow only the two paths of C_2 and the two paths of C_1 indicated by the thick arrows because they expand as the scalar field value decreases. On the other hand, hollow isosurfaces follow only the two paths of C_2 and two paths of C_1 indicated by the outlined arrows.

We can now derive that an inclusion relationship occurs through only one transition path shown in Fig. 3(b) out of four, for each type of saddle point (i.e. C_2 and C_1). This enables us to locate inclusion relationships between isosurfaces at any scalar field values by identifying transitions at saddles with their embeddings. This can be achieved by tracing the links in the VST, which will be described later in Sect. 3.4.

As described previously, our algorithm accounts for the virtual minimum, which is artificially added to the volume function (2) so that we can think of an input dataset as a topological 3D sphere S^3 . This setting enables us to check the mathematical consistency of the extracted critical points by consulting the Euler formula [31]:

$$\#\{C_3\} - \#\{C_2\} + \#\{C_1\} - \#\{C_0\} = 0, \quad (5)$$

where $\#\{C_i\}$ represents the number of critical points of index i .

3.3 Revised Algorithm for VST Construction

For constructing a VST automatically from a given volume dataset, we use our previous algorithm described in [31], which originates from the surface analysis frame-

work [28] used in GIS applications. However, we have inserted a new step and made some modifications to the existing steps in the algorithm in order to extract spatial configurations of isosurfaces. Consequently, the algorithm has the following five steps:

Step 1: Extract Critical Points

This step begins with linear interpolation of the volume dataset through tetrahedralization, in order to uniquely determine the isosurface evolution as the scalar field value decreases. Here, the tetrahedralization is performed in such a way that it can simulate the isosurface evolution extracted by using the asymptotic decider [21] (See [31] for the details). In the tetrahedralization, each voxel has its neighboring voxels that constitute a triangulated sphere surrounding the target voxel itself. By comparing the scalar field values of the neighboring voxels on the sphere with that of the target, the algorithm partitions the surrounding sphere into two types of connected regions: plus regions that include points having larger scalar field values than the target, and minus regions having smaller scalar field values. By identifying the configuration of these plus and minus regions on the sphere, we can extract critical points together with their types.

Step 2: Construct a VFN

Before constructing a VST from the volume dataset, our algorithm constructs a graph called a *voxel flow network* (VFN) that works as an intermediate representation in the overall process. Constructing the VFN begins with tracing voxel flows which emanate from each saddle critical point. From each saddle, we traverse voxel flows from its neighbors having the larger (smaller) scalar field value up to maxima (down to minima) along the steepest ascent (descent) direction. The VFN is then introduced to capture the scalar field configuration between the critical points, where its node represents the critical point and its link represents the voxel flow.

Step 3: Convert the VFN into a VST

In this step, the constructed VFN is converted to a VST by fixing the links in the VST from its ends to center. This is possible because the VST is constructed by assembling the parts as shown in Fig. 4 where each part involves one critical point. Recall that the VST node signifies the critical point and its link represents one isolated interval volume swept by an evolving isosurface component. Details of the conversion process can be found in [31]. Note that the resultant VST becomes a tree because the input volume dataset satisfies the assumption of single-valued functions. (See Sect. 3.1.)

Step 4: Mapping between Voxels and the VST Links

This step is newly introduced here in order to establish a mapping from each voxel onto a VST link, which will help us generate an additional dimension for multi-dimensional transfer functions that emphasize nested inner structures systematically.

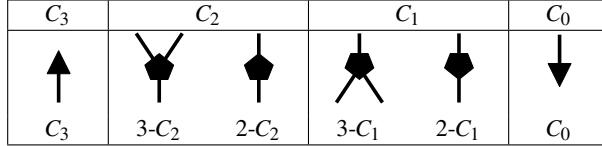


Fig. 4. VST parts around critical points

The mapping is defined to match a voxel to the link in such a way that the interval volume corresponding to the link involves the voxel. For example, voxels involved in the interval volumes shaded in dark gray, light gray, and white in Fig. 1(a) are mapped onto the links $\overline{p_1 p_2}$ (in dark gray), $\overline{p_2 p_3}$ (in light gray), and $\overline{p_2 p_4}$ (in white) in Fig. 1(b), respectively. In our implementation, each voxel has a pointer to its corresponding link while the link also has a pointer to the voxel.

The algorithm constructs the mapping without explicitly extracting isosurfaces at every scalar field value. We can match the voxel to the corresponding link by finding its neighboring voxels that have already been mapped to the same link of the VST in most cases. If this is not the case, we find the local maximum and minimum reachable from the voxel using the tracing process similar to that in Step 2. In addition, we extract candidate links that go across the scalar field value of the current voxel, and trace the VST from each candidate link constantly with respect to the scalar field to collect a set of end maxima and minima. By comparing these sets with the maximum and minimum reachable from the current voxel, the corresponding link is identified.

Suppose that we want to map a voxel that lies in the interval volume corresponding the link $\overline{p_2 p_3}$ in Fig. 1. In this case, the local maximum and minimum reachable from the target voxel are p_1 and p_3 , respectively. On the other hand, the candidate links are $\overline{p_2 p_3}$ and $\overline{p_2 p_4}$ because they include the scalar field value of the target voxel. Note here that the link $\overline{p_2 p_3}$ has the corresponding maximum p_1 and minimum p_3 while the link $\overline{p_2 p_4}$ has the maximum p_1 and minimum p_4 . This concludes that the target voxel corresponds to the link $\overline{p_2 p_3}$ because the maximum and minimum reachable from the voxel is included in the sets of maxima and minima that can be reached from the link $\overline{p_2 p_3}$ on the VST. Since the VST becomes a tree under our assumption in Sect. 3.1, this process is fully justified.

Step 5: Simplify the VST

Although the VST effectively captures the topological skeleton of a volume dataset, it often suffers from high-frequency noise that produces a large number of minor critical points. This is why our algorithm simplifies the constructed VST by removing minor critical points in order to distinguish the important global skeleton from the unknown volume. Figure 5 presents three VST patterns to be removed in this simplification process, where the last pattern may have other critical points between the two nodes. In the simplification process, our algorithm computes the sum of weight values assigned to the links between the two nodes for each pattern, and then selects one pattern to be removed by finding the link having the smallest value. As

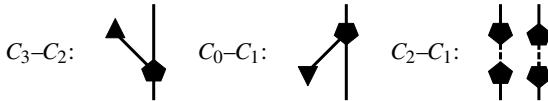


Fig. 5. VST patterns to be eliminated in the simplification process

the weight values to be assigned, we employ the volume swept by the evolving iso-surfaces that corresponds to the links, multiplied by a difference between the end scalar field values [29]. Here, the swept volume is estimated by counting the number of voxels assigned to the corresponding link. Note that this is possible because we have already established the mapping between voxels and the links of the VST in the previous step. When eliminating a link, we transfer voxels assigned to the link to one of its adjacent links so that the voxels still survive in the existing VST. The number of simplification steps is controlled by a threshold that limits the acceptable weight values to the links. Due to this simplification step, our algorithm will extract nested volumetric structures robustly even from a dataset having a large amount of high-frequency noise.

3.4 Extracting Isosurface Inclusions

We are now ready to introduce our new algorithm that extracts isosurface embeddings in 3D space from the VST. As mentioned in Sect. 3.2, an isosurface inclusion occurs in the transition path at a saddle shown in Fig. 3(b). Here, a new inclusion relationship appears when the saddle is C_1 (from right to left), and an existing inclusion relationship dissolves when the saddle is C_2 (from left to right) when lowering the corresponding scalar field value. This implies that, in order to extract the inclusion relationships, we have to identify each VST part around a saddle point in Fig. 4, with a topological transition of an isosurface in Fig. 3.

The assumption made in Sect. 3.1 restricts topological transitions of solid and hollow isosurfaces at saddles as shown in Fig. 3. This enables a one-to-one correspondence between the VST components (Fig. 4) and isosurface transitions (Fig. 3), once we can identify the incident links with solid or hollow isosurface components. Figure 6 illustrates VST parts that correspond to the four paths of isosurface transitions in Fig. 3 row by row for each of C_2 and C_1 , where each link incident to a saddle is recognized as solid (indicated by a thick arrow) or hollow (indicated by an outlined arrow). This allows us to identify all the links in the VST with solid or hollow isosurfaces by consulting Fig. 6.

Our algorithm finds each link in the VST to be solid or hollow, by traversing the links in the VST. In practice, this tracing process starts from the virtual minimum because the link incident to the virtual minimum is known to represent an outermost solid isosurface under the assumption in Sect. 3.1. Furthermore, the virtual minimum can easily be distinguished from other critical points since it is artificially introduced to the volume dataset.

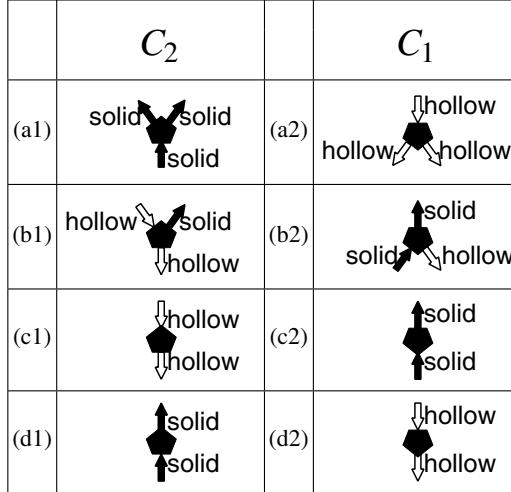


Fig. 6. Isosurface embeddings around saddles: Each row corresponds to the isosurface transition in Fig. 3, and the arrows represent the direction of tracing

In our implementation, the virtual minimum serves as a starting point for tracing solid links of the VST in the upward direction, especially to check how outermost isosurfaces evolve as the scalar field value increases. During the tracing process, each critical point on the way is identified with a VST part in Fig. 6 according to the attribute (solid or hollow) of the incoming link, and other unvisited outgoing links are also matched with solid or hollow by consulting the corresponding part in Fig. 6. The unvisited links are then registered to a queue Q_s in the algorithm if they are solid, and registered to another queue Q_h if hollow. Note that these two queues are FIFOs. After having examined the current critical point, the algorithm removes one solid link from the queue Q_s to process it next. Here, the queue Q_s stores unvisited solid links that are examined in the ongoing upward tracing process while the queue Q_h stores hollow links to be handled in the next downward tracing process. Consequently, this upward tracing process ends when the queue Q_s becomes empty. After that, we examine the queue Q_h and begin the tracing process in the opposite (downward) direction for hollow links this time. The overall tracing process continues until all the links in the VST are processed.

When identifying a saddle point in the VST with a topological transition, we will encounter the four types of the parts around saddles as shown in Fig. 4: 3- C_2 , 2- C_2 , 3- C_1 , and 2- C_1 . The arrows in Fig. 6 show how each of the four parts is traced in the upward and downward tracing processes depending on the attributes of its incident links. When tracing solid links in the upward direction, the four types of parts 3- C_2 , 2- C_2 , 3- C_1 , and 2- C_1 in Fig. 4 correspond to those in Fig. 6(a1), (d1), (b2), and (c2), respectively. On the other hand, these four types are matched to the parts in Figs. 6(b1), (c1), (a2), and (d2) when tracing hollow links. Note that a new

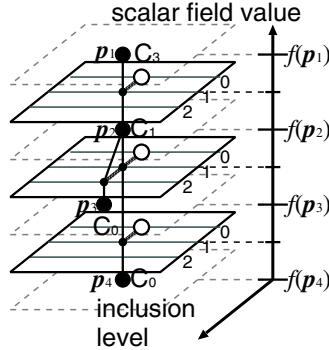


Fig. 7. VST and its isosurface inclusion trees of the analytic function (1)

inclusion relationship only occurs in the part of Fig. 6(b2) and an existing inclusion relationship disappears in the part of Fig. 6(b1), when reducing the scalar field value. Figure 1(b) shows how to trace the VST links for the analytic function (1) with the arrows indicating the tracing directions. In this case, the node p_2 is identified with the part in Fig. 6(b2), which implies the existence of isosurface inclusion relationships.

Our method stores such inclusion relationships between isosurfaces for each interval between two adjacent critical scalar field values because the inclusion relationships do not change within that interval. In our implementation, we assign a tree data representation to each interval so that the tree can represent inclusion relationships between any pairs of connected components of an isosurface. Here, the inclusion relationship is represented by the link of the tree where its parent and child nodes correspond to the outer and inner isosurfaces, respectively. Figure 7 shows such isosurface inclusion trees for the analytic function (1). Note that each of the trees introduces an artificial root node (illustrated as outlined circles) because we assume that the virtual isosurface includes all the existing isosurfaces within the interval. This representation scheme comes from the previous work on designing 3D surfaces by using their topological skeleton [25, 30]. In our framework, this data representation plays an important role in emphasizing nested inner structures using multi-dimensional transfer functions.

4 Emphasizing Isosurface Embeddings

While the aforementioned algorithm successfully extracts nested structures of feature isosurfaces, it is still a challenging task to effectively emphasize such nested structures in the visualization stage because the conventional transfer functions depend only on the scalar field. This means that the conventional transfer functions cannot distinguish between nested isosurfaces if they are at the same scalar field value. Thus, emphasizing inner and outer isosurfaces individually requires more

sophisticated transfer functions that depend on relative geometric positions in addition to the scalar field.

To this end, we developed a new algorithm that effectively accentuates such nested structures using multi-dimensional transfer functions. The concept of multi-dimensional transfer functions was originally introduced by Levoy [18], where he employed a 2D opacity transfer function that uses gradient magnitude for the second dimension. In our framework, we also formulate a 2D opacity transfer function while we employ a different attribute value called an *inclusion level* for the second dimension.

In the remainder of this section, we begin by designing conventional 1D transfer functions by referring to the VST. We then define 2D transfer functions that accept the inclusion levels as the second dimension in order to emphasize the nested structures of isosurfaces.

4.1 Design Principles of 1D Transfer Functions

Since the VST robustly extracts the global structure of a given volume dataset, it allows us to easily identify feature isosurfaces to be emphasized in the visualization stage. As the feature isosurfaces, our rendering framework selects significant isosurface components from each scalar field interval bound by adjacent critical scalar field values. In practice, we define a *representative scalar field value* as the midpoint of each scalar field interval and extract the corresponding isosurface as *representative isosurfaces* to illuminate the distinctive isosurface transition in the volume. These representative isosurfaces will be also suitable for expressing inner structures of the given volume if we consider the nested structures there.

In our implementation, the scalar field values are mapped affinely onto an 8-bit range [0, 255], and c_1, c_2, \dots, c_m represent a sequence of critical scalar field values in descending order, where c_m is the scalar field value of the virtual minimum and set to be 0. Although there may be multiple candidates for design principles of 1D transfer functions, we use the hue and opacity transfer functions shown in Figs. 8(a) and (b) as the common principles.

The hue transfer function maps the scalar field onto the range $[0, 2\pi]$ on the top of the HSV hexcone. Furthermore, in our design principle, each scalar field interval $[c_{i+1}, c_i]$ ($i = 1, \dots, m - 1$) has a hue range of the same interval, and it has linear descent slope of the hue value as shown in Fig. 8(a). Note that the hue transfer function is 1D and depends only on the scalar field value throughout this paper, while the opacity transfer function may be 1D or 2D as described later.

The 1D opacity transfer function is designed to be zero except for hat-like elevations around representative scalar field values r_1, r_2, \dots, r_{m-1} where $r_i = (c_i + c_{i+1})/2$ ($i = 1, \dots, m - 1$), so that it can accentuate the representative isosurfaces individually [31]. Here, the height of the elevation decreases by a constant amount from r_1 to r_{m-1} because the outermost solid isosurfaces expand as the scalar field value decreases according to the assumption made in Sect. 3.1. Empirically, this principle adequately reduces the influence of the outermost isosurfaces as they expand.

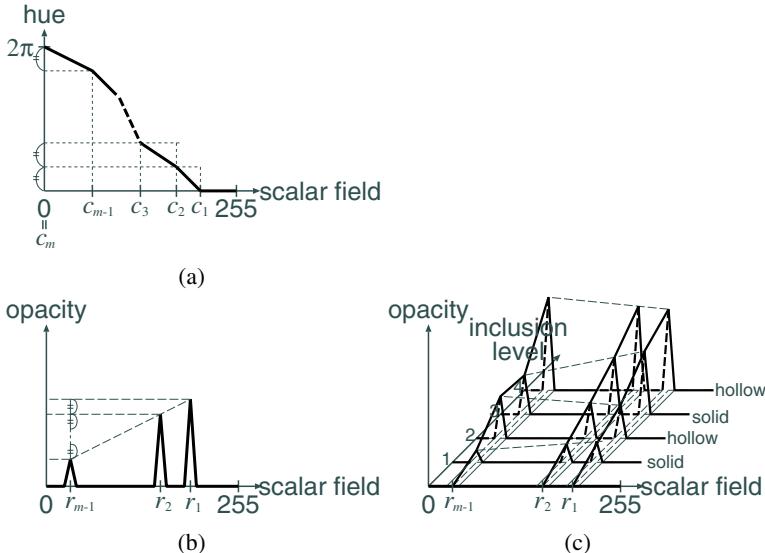


Fig. 8. Transfer function design principles: (a) hue, (b) 1D opacity and (c) 2D opacity

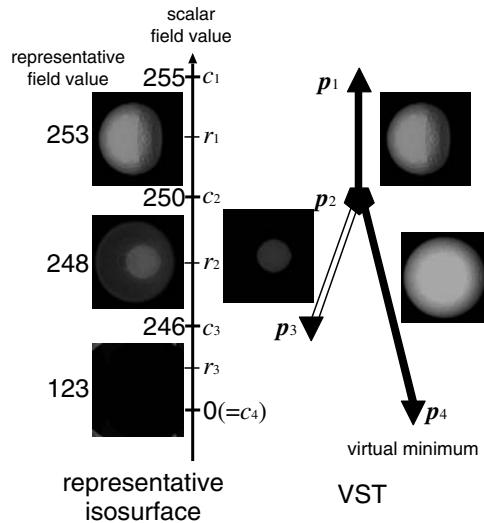


Fig. 9. Feature isosurfaces and VST of a nested sphere volume

As an example, we are going to apply our design principles to the volume dataset generated by sampling the function (1) when $a = 0.8$ and $b = 0.1$, where its resolution is $64 \times 64 \times 64$. Figure 9 shows representative isosurfaces (on the left) and the extracted VST (on the right). Here, each link of the VST is accompanied by a

disjoint component of the corresponding representative isosurface. In this case, the links $\overline{p_1 p_2}$ and $\overline{p_2 p_4}$ are solid while the link $\overline{p_2 p_3}$ is hollow. We learn from this figure that the volume has three representative scalar field values $r_1 = (f(p_1) + f(p_2))/2$, $r_2 = (f(p_2) + f(p_3))/2$, and $r_3 = (f(p_3) + f(p_4))/2$, where the scalar field value of the virtual minimum $f(p_4)$ is assumed to be 0. It also turns out that any isosurface lying within the scalar field interval $[f(p_3), f(p_2)]$ has two connected components that have an inclusion relationship, which implies that the representative isosurface components at r_2 are nested.

Figure 10(a) shows a visualization result of this dataset obtained by using the 1D transfer functions in Figs. 8(a) and (b). Although the result seems to be effective, it still obscures the isosurface inclusion at the representative scalar field value r_2 . This motivates us to incorporate an additional attribute for the multi-dimensional transfer functions as described below.

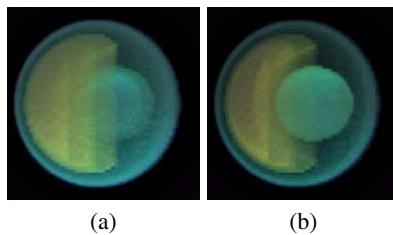


Fig. 10. Emphasizing isosurface embeddings of a nested sphere volume: **(a)** with 1D transfer functions, and **(b)** with 2D transfer functions

4.2 Design Principles of 2D Transfer Functions

An inclusion level is the second variable for our multi-dimensional transfer functions, and represents the depth of the corresponding isosurface component in a nested tree structure as shown in Fig. 7. Thus, it allows us to systematically locate the relative geometric position of each voxel within such a nested volumetric structure. Moreover, the inclusion level also differs from the conventional attributes for multi-dimensional transfer functions in that it is a topological attribute and comes from the rigorous analysis of the global volumetric structures, while all the conventional attributes represent local volumetric properties such as derivatives [14] and curvatures [12].

Actually, our algorithm will calculate this attribute value for each voxel by consulting the data structure as shown in Fig. 7. Here, the VST together with its associated trees for isosurface inclusions is embedded into 3D space. In the figure, the vertical and depth axes represent the scalar field and inclusion level, respectively, and the lateral lines on each horizontal plane represent the scales of the inclusion level for each isosurface inclusion tree.

Since we have already established the mapping between the voxels and the VST links in Sect. 3.3, we can easily calculate the inclusion levels for all the voxels by handling the VST links one by one. Suppose we are now looking at the link $\overline{p_2 p_4}$ in Fig. 9. From the mapping, our algorithm retrieves the array of voxels that belong to the link. Each voxel is then distributed to the corresponding scalar field interval according to its scalar field value. For example, in this case, if the voxel has a larger scalar field value than p_3 , it will be assigned to the interval $[f(p_3), f(p_2)]$. Otherwise, it will be delivered to the interval $[f(p_4), f(p_3)]$. The inclusion level can be obtained by evaluating the depth of its VST link in the corresponding isosurface inclusion tree, because the tree represents isosurface spatial configurations within the scalar field interval to which the voxel is distributed.

It should be noted that the inclusion level is intrinsically discrete and may have gaps at some critical scalar field values because the inclusion relationships between isosurface components suddenly change at the values. These gaps could result in unexpected artifacts in our final visualization results. We thus avoid these artifacts by linearly interpolating the gap so that the inclusion level becomes continuous. Figure 7 shows such an example, where the inclusion level of the link $\overline{p_2 p_3}$ linearly rises from 1 to 2 while the scalar field value reduces from $f(p_2)$ to $(f(p_2) + f(p_3))/2$. In practice, our algorithm assigns the intermediate inclusion levels to the voxels according to their scalar field values.

Now we are ready to extend the previous 1D opacity transfer function to 2D so that it effectively emphasizes the nested isosurfaces with the help of the inclusion levels. As described previously, the height of the hat-like elevation for the 1D opacity transfer function reduces by a constant amount at each representative scalar field value as the scalar field value decreases as shown in Fig. 8(b). This is because solid isosurfaces expand while the corresponding scalar field value decreases as described in Sect. 3.1. However, hollow isosurfaces have the reverse behavior, i.e., they shrink as the scalar field value decreases. This lets us raise the height of hat-like elevation for hollow isosurfaces to clarify their topological changes in deeper positions.

By taking these points into account, we design the 2D opacity transfer function as shown in Fig. 8(c). Here, at each discrete inclusion level, the hat-like elevation decreases if the corresponding isosurface is solid and increases if hollow as the scalar field value goes down. We then define the opacity for intermediate non-integer inclusion levels by interpolating between those on integer inclusion levels. This means that the opacity transfer function around the representative scalar field values has a form of hyperbolic paraboloid. Note that the opacity transfer function is defined to increase as the inclusion level becomes larger. This is achieved by ensuring that the maximum opacity value at the inclusion level l does not exceed the minimum opacity of the inclusion level $l + 1$.

Our experiments show that this design principle makes the inner isosurface components clearly visible through the outer ones, and avoids obscuring the nested inner structures. Figure 10(b) shows such an example where the nested structure of the isosurface components at $(f(p_2) + f(p_3))/2$ is effectively emphasized. The previous 1D color transfer function is used for these visualization results.

5 Application to Real Datasets

In this section, we apply the present framework to two datasets in order to demonstrate its applicability to real datasets. In all of these datasets, the scalar field values were normalized to an 8-bit range [0, 255]. In our experiments, the present algorithms are implemented on an ordinary PC environment (CPU: Pentium IV, 2.4GHz, RAM: 1GB).

5.1 Nucleon

The first example is a “nucleon” dataset presented in Fig. 11. This dataset is obtained by simulating two-body distribution probability of a nucleon in the atomic nucleus ^{16}O [20], and its resolution here is $41 \times 41 \times 41$. From the dataset, our algorithm extracted a VST as shown in Fig. 11(a), which shows that the dataset actually involves isosurface inclusions within the scalar field interval [0, 161]. Note that a link for a solid isosurface is in blue while that for a hollow isosurface is in orange in this figure.

As described in Sect. 1, Fig. 11(b) obscures the inner structures and provides no useful information because it is generated using naive transfer functions. While Fig. 11(c) accentuates representative isosurfaces individually using the conventional 1D transfer functions [7, 31, 33], it still misses the nested inner structures which we are interested in, unfortunately. On the other hand, Fig. 11(d) effectively emphasizes

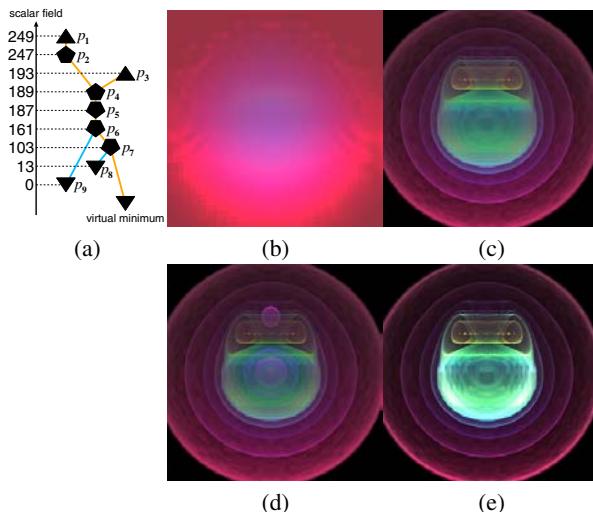


Fig. 11. Inner structure of the “nucleon” dataset [20]: (a) The VST. Visualization results (b) with naive (linear hue and flat opacity) 1D transfer functions, (c) with topologically-accentuated 1D transfer functions, (d) with 2D transfer functions, and (e) generated using emission-only optical model

the inner isosurface components due to the use of 2D transfer functions with the inclusion levels.

5.2 Antiproton-Hydrogen Atom Collision

As the second example, we consider the antiproton-hydrogen atom collision at intermediate collision energy below 50keV. In the antiproton-hydrogen collision system, a single antiproton comes into collision with a single hydrogen atom. The details of the formulation and established numerical schemes can be found in [26].

Here, we applied our framework to the dataset with a resolution of $64 \times 64 \times 64$. Figure 12(a) shows the VST extracted from this dataset. When exploring isosurface transitions as the scalar field value reduces, we can see that the isosurface component of the link $\overline{p_2 p_5}$ splits into nested isosurface components of the links $\overline{p_5 p_7}$ and $\overline{p_5 p_{10}}$ at the scalar field value 118, and simultaneously encloses another isosurface component of the link $\overline{p_4 p_6}$. After that, the enclosed isosurface of the link $\overline{p_4 p_6}$ also divides itself into the nested surface components of the links $\overline{p_6 p_7}$ and $\overline{p_6 p_8}$ at 112. This means that we have fourfold nested structures in the scalar field interval [98, 112].

Figures 12(b) and (c) show rendered images of the antiproton-hydrogen atom collision volume dataset. Figure 12(b) shows a rendered image generated using the conventional 1D transfer functions, where representative isosurfaces are accentuated while ignoring their accompanying nested structures. As shown in the figure, the nested structure lying in the scalar field interval [98, 112] was densely occluded by the outermost isosurface component. On the other hand, our new algorithm clearly emphasizes the nested structures as shown in Fig. 12(c) where 2D opacity transfer function is applied. Note here that our algorithm controls voxel opacities by taking into account isosurface embeddings using the multi-dimensional transfer functions. These results suggest that our present framework can explicitly extract significant volumetric structures that may be missed by using the conventional methods.

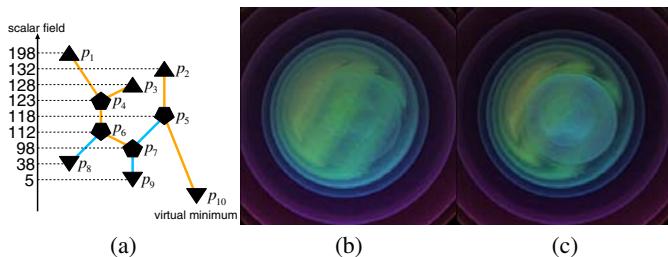


Fig. 12. Antiproton-hydrogen atom collision volume dataset: (a) the VST. Visualization results (b) with 1D transfer functions, and (c) with 2D transfer functions

6 Discussion

This section discusses the validity of the present framework for emphasizing isosurface embeddings in direct volume rendering.

6.1 Comparison with an Emission-Only Optical Model

The present framework effectively emphasizes isosurface embeddings in volume rendering because it can extract inclusion relationships between isosurfaces by tracing their global transitions in a given volume. On the other hand, another promising approach may be an emission-only optical model where all the feature isosurface components equally contribute to the final image and none is attenuated by the surrounding components. Figure 11(e) is an image generated by using this model. However, when compared with Fig. 11(d) that is generated by our new framework, this model cannot suppress the opacity of outer isosurfaces while emphasizing inner ones because it lacks the ability to distinguish between outer and inner isosurfaces. This shows that the present framework provides more sophisticated visualization techniques than the simple emission-only optical model.

6.2 Computational Cost

The computational cost is another issue to be considered since the present framework requires more computational time for the rigorous analysis of a given volume. In practice, our algorithm extracted nested isosurface structures in 4 minutes for the “nucleon” dataset (Fig. 11(d)) and 30 minutes for the antiproton-hydrogen atom collision dataset (Fig. 12(c)). However, these computational costs can be reduced remarkably by introducing adaptive tetrahedralization for the interpolation of the given scalar field without sacrificing the accuracy of the analysis [29]. According to the results in [29], it took only 25 seconds for the “nucleon” dataset, and 60 seconds even for the high-resolution version ($129 \times 129 \times 129$) of the antiproton-hydrogen atom collision dataset. This concludes that the present framework will be a more promising approach for comprehensible volume rendering as the computational performance becomes better in future.

7 Conclusion

This paper has presented a new rendering framework that clearly emphasizes nested isosurface structures embedded in the given volume datasets. The key to the present framework is the use of multi-dimensional transfer functions that assign different optical properties to each voxel. This is achieved by using the inclusion level as well as the conventional scalar field. In order to calculate such an additional attribute, we developed a new algorithm that extracts inclusion relationships between feature isosurfaces by tracing the topological skeleton delineated from the given dataset. Several experimental results demonstrated the feasibility of the present method.

As a future research theme, we plan to take into account psychological factors of color science so that we can accommodate the inclusion level as a new dimension when designing color transfer functions. Furthermore, our framework may also provide the basis for future research on incorporating textures and lighting properties because such properties may provide significant landmarks on the visualization results.

Acknowledgements

We wish to acknowledge the support of the Japan Society of the Promotion of Science under Grants-in-Aid for Young Scientists (B) (No. 14780189), the Okawa Foundation for Information and Telecommunications, the Office of Naval Research (N00014-02-1-0287), the National Science Foundation (NSF IIS-9980166 & ACI-0083609), and DARPA (MDA972-00-1-0027).

References

1. C. L. Bajaj, V. Pascucci, and D. R. Schikore. The contour spectrum. In *Proc. of IEEE Visualization '97*, pp. 167–173, 1997.
2. S. Castro, A. König, H. Löfleemann, and E. Gröller. Transfer function specification for the visualization of medical data. Technical Report TR-186-2-98-12, Vienna University of Technology, 1998. [<http://www.cg.tuwien.ac.at/research/TR/98/TR-186-2-98-12Abstract.html>].
3. B. Csébfalvi, L. Mroz, H. Hauser, A. König, and E. Gröller. Fast visualization of object contours by non-photorealistic volume rendering. *Computer Graphics Forum*, 20(3):452–460, 2001.
4. D. Cohen-Or, Y. Chrysanthou, C. Silva, and G. Drettakis. Visibility, problems, techniques and applications. *Siggraph '00 Course Notes*, 2000.
5. H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry*, 24(2):75–94, 2003.
6. I. Fujishiro, T. Azuma, and Y. Takeshima. Automating transfer function design for comprehensive rendering based on 3D field topology analysis. In *Proc. of IEEE Visualization '99*, pp. 467–470, 563, 1999.
7. I. Fujishiro, T. Azuma, Y. Takeshima, and S. Takahashi. Volume data mining using 3D field topology analysis. *IEEE Computer Graphics & Applications*, 20(5):46–51, 2000.
8. A. T. Fomenko and T. L. Kunii. *Topological Modeling for Visualization*, chapter 6, pp. 105–125. Springer-Verlag, 1997.
9. I. Fujishiro, Y. Maeda, and H. Sato. Interval volume: A solid fitting technique for volumetric data display and analysis. In *Proc. of IEEE Visualization '95*, pp. 151–158, CP-18, 1995.
10. I. Fujishiro, Y. Maeda, H. Sato, and Y. Takeshima. Volumetric data exploration using interval volume. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):144–155, 1996.
11. I. Fujishiro, Y. Takeshima, S. Takahashi, and Y. Yamaguchi. Topologically-accentuated volume rendering. In F. H. Post, G. M. Nielson, and G.-P. Bonneau, editors, *Data Visualization: The State of the Art*, pp. 95–108. Kluwer Academic Publishes, 2002.

12. J. Hladívka, A. König, and E. Gröller. Curvature-based transfer functions for direct volume rendering. In *Proc. of Spring Conference on Computer Graphics 2000*, pp. 58–65, 2000.
13. V. L. Interrante. Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution. In *Computer Graphics (Proc. of Siggraph '97)*, pp. 109–116, 1997.
14. G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proc. of IEEE Symposium on Volume Visualization*, pp. 79–86, 1998.
15. J. Kniss, G. Kindlmann, and C. Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proc. of IEEE Visualization 2001*, pp. 255–262, 2001.
16. J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, 2002.
17. J. T. Klosowski and C. T. Silva. The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):108–123, 2000.
18. M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8(5):29–27, 1988.
19. A. Lu, C. J. Morris, D. S. Ebert, P. Rheingans, and C. Hansen. Non-photorealistic volume rendering using stippling techniques. In *Proc. of IEEE Visualization 2002*, pp. 211–218, 2002.
20. M. Meißner. Web Page [<http://www.volvis.org/>].
21. G. M. Nielson and B. Hamann. The asymptotic decider: Removing the ambiguity in marching cubes. In *Proc. of IEEE Visualization '91*, pp. 83–91, 1991.
22. V. Pascucci and K. Cole-McLaughlin. Efficient computation of the topology of level sets. In *Proc. of IEEE Visualization 2002*, pp. 187–194, 2002.
23. P. Rheingans and D. Ebert. Volume illustration: Nonphotorealistic rendering of volume models. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):253–264, 2001.
24. G. Schaufler, J. Dorsey, X. Decoret, and F. X. Sillion. Conservative volumetric visibility with occluder fusion. In *Computer Graphics (Proc. of Siggraph '00)*, pp. 229–238, 2000.
25. Y. Shinagawa, Y. L. Kergosien, and T. L. Kunii. Surface coding based on morse theory. *IEEE Computer Graphics & Applications*, 11(5):66–78, 1991.
26. R. Suzuki, H. Sato, and M. Kimura. Antiproton-Hydrogen atom collision at intermediate energy. *IEEE Computing in Science and Engineering*, 4(6):24–33, 2002.
27. S. Treavatt and M. Chen. Pen-and-ink rendering in volume visualization. In *Proc. of IEEE Visualization 2000*, pp. 203–210, 2000.
28. S. Takahashi, T. Ikeda, Y. Shinagawa, T. L. Kunii, and M. Ueda. Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data. *Computer Graphics Forum*, 14(3):181–192, 1995.
29. S. Takahashi, G. M. Nielson, Y. Takeshima, and I. Fujishiro. Topological volume skeletonization using adaptive tetrahedralization. In *Proc. of Geometric Modeling and Processing 2004*, pp. 227–236, 2004.
30. S. Takahashi, Y. Shinagawa, and T. L. Kunii. A feature-based approach for smooth surfaces. In *Proc. of the ACM 4th Symposium on Solid Modeling and Applications*, pp. 97–110, 1997.
31. S. Takahashi, Y. Takeshima, and I. Fujishiro. Topological volume skeletonization and its application to transfer function design. *Graphical Models*, 66(1):24–49, 2004.

32. M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *Proc. of the 13th ACM Symposium on Computational Geometry*, pp. 212–220, 1997.
33. G. H. Weber, G. Scheuermann, H. Hagen, and B. Hamann. Exploring scalar fields using critical isovalues. In *Proc. of IEEE Visualization 2002*, pp. 171–178, 2002.

Diagnostic Relevant Visualization of Vascular Structures

Armin Kanitsar¹, Dominik Fleischmann², Rainer Wegenkittl³, and
Meister Eduard Gröller¹

¹ Institute of Computer Graphics and Algorithms,
Vienna University of Technology, Vienna, Austria,
{kanitsar,meister}@cg.tuwien.ac.at

² Department of Radiology, Stanford University, US,
dominik.fleischmann@univie.ac.at

³ TIANI Medgraph, Austria,
rainer.wegenkittl@tiani.com

Summary. Traditional volume visualization techniques sometimes provide incomplete clinical information needed for applications in medical visualization. In the area of vascular visualization important features such as the lumen of a diseased vessel segment may not be visible. One way to display vascular structures for diagnostic purposes is to generate longitudinal cross-sections in order to show their lumen, wall, and surrounding tissue in a curved plane. Curved planar reformation (CPR) has proven to be an acceptable practical solution. We discuss four different methods to generate CPR images from single vessel segments: Projected CPR, stretched CPR, straightened CPR, and helical CPR. Furthermore we investigate three different methods for displaying vascular trees: Multi-path projected CPR, multi-path stretched CPR, and untangled CPR. The principle concept of each method is discussed and detailed information for the realization is given. In addition the properties, advantages and disadvantages of each method are summarized.

1 Introduction

Non-invasive imaging of the vascular system with computed tomography (CT) and magnetic resonance imaging (MRI) has become a well established alternative to invasive intraarterial angiography. CT and MRI provide high-resolution volumetric data sets of the human body. These data, however, may contain many objects of less or no diagnostic interest. This makes volume-rendering (i.e., maximum intensity projection (MIP), ray casting, shaded surface display) without preprocessing often impossible or inaccurate. In addition to that pathological features may superimpose diagnostically relevant information. In the case of a circular vessel wall calcification the true vessel lumen can not be determined by conventional volume rendering.

CPR - Curved Planar Reformation is a way to visualize vascular structures with small diameters. High level information as the vessel's centerline is used to re-sample

and visualize the data. By this technique the entire tubular structure is displayed within a single image. Vascular abnormalities, i.e., stenoses, occlusions, aneurysms and vessel wall calcifications, are then investigated by physicians. This process is sometimes referred to as *Multi Planar Reformation* (MPR). However the term MPR is not precise enough, as it is commonly used for planar cross-sections re-sampled from volumetric data. Another known synonym for curved planar reformation is *medial axis reformation* (MAR).

Even though CPR is an established technique in the medical community, the visual properties, advantages, and disadvantages of different types of CPRs have not been specifically addressed in the literature. Understanding this features is essential for the accurate interpretation of the resulting images. One issue of this work is the generation and discussion of properties of different CPR methods.

Traditional CPR techniques allow the investigation of the vessel lumen in a longitudinal section through the central axis of the vessel. However, vascular abnormalities might not be touched by this plane and therefore they do not appear in the generated image. One way to overcome this problem is to rotate the re-sampled plane around the central axis. This results in a set of images to be interpreted by the radiologist. A more comprehensive display of the entire vascular lumen in one representative image is highly desirable. A new visualization method was developed to overcome this limitation.

Another important aspect in *computed tomography angiography* (CTA) is the efficient visualization of treelike vascular structures using CPR display techniques. Multi-path CPR techniques based on a projective combination of vessel segments provide a spatially coherent display of the vascular anatomy. However, depending on the intersecting plane, parts of the arteries might be superimposed by other arteries. For a detailed inspection of the entire vascular tree different sections through the vessel's central axis have to be re-sampled. In order to have diagnostically valuable results the vessel lumen should be visible within each image. Thus a new technique for unobstructed displaying of an arterial tree is proposed.

Section 2 describes related work in this area. In Sect. 3 four methods for visualizing vascular segments are presented. Techniques for displaying entire vascular trees are presented in Sect. 4. Finally the proposed methods are summarized and compared in Sect. 5.

2 Related Work

The most important prerequisite for CPR visualization is an appropriate estimation of the vessel centerline. Latest CT technology, such as multiple detector-array CT, provide high resolution volumetric data sets. Due to the large size of these data sets (up to 1500 transverse cross-sectional images of the abdomen and entire legs), the manual definition of the vessel centerline is no longer an option. In this respect several algorithms [8, 13, 14] have been developed with different properties concerning reliability, speed and accuracy.

Avants and Williams [3] present a vessel tracking method consisting of two parts. From user defined seed points a surface expansion is computed based on the eikonal partial differential equation. A minimal cost path is calculated from these regions. From this path a cross-sectional area/radius profile is generated.

He et al. [6] proposed a path extraction method based on a two-dimensional region-growing algorithm with a subsequent shortest path algorithm. The resulting path is refined using the multi-scale medial response. The vascular tree is flattened in a semiautomatic method called *Medial Axis Reformation*.

Some authors propose to take the central axis as an input for the generation of an abstract vessel model. Abstract vessel models allow fast rendering, as polygonal meshes of low complexity are generated [4]. Furthermore non-photorealistic rendering provides the possibility to emphasize global properties of the vascular tree [5].

Kanitsar et al. [7] compared three methods for CPR generation: Projected CPR, stretched CPR and straightened CPR with respect to three extensions. These extensions have been proposed to overcome the most relevant clinical limitations, i.e., thick CPR, rotating CPR and multi-path CPR.

A comparison of this technique with conventional volume visualization techniques is not the topic of this paper, as such comparisons are already available [2]. Further information about the clinical relevance of the CPR visualization technique can be found in [1, 11, 12].

3 Single Vessel CPR Methods

The goal of CPR visualization is to make a tubular structure visible in its entire length within one single image. To accomplish this goal a-priori information about the tubular structure, notably the object's central axis, is required. Without loss of generality the object's central axis is assumed to be a sequence of points at sub voxel resolution.

In general the spatial position and shape of the central axis determines which parts of 3D space are visualized. On the left side of Fig. 1 the central axis is shown. The re-sampled surface is shown on the right side of Fig. 1. As the surface is not well defined by just one curve in 3D, an additional vector vi (*vector-of-interest*) is introduced. Together with a point from the central axis, the vector-of-interest defines a straight line li (*line-of-interest*). All voxels touched by this line are taken to re-sample the volume along the line-of-interest.

Figure 2 illustrates the different CPR methods. The horizontal plane represents the image and the image y-axis as horizontal blue arrow. Corresponding to this axis, the curve in the volumetric data set is sketched by the vertical blue arrow.

3.1 Projected CPR

The projected CPR can be seen as a parallel projection of a data set, taking into account only a thin slice of voxels (see Fig. 2a). This slice is defined by the central

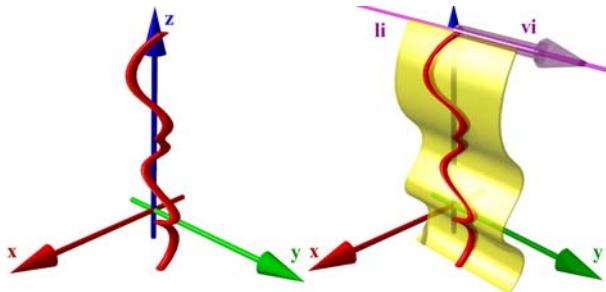


Fig. 1. Principle of the CPR visualization: The vector-of-interest (*vi*) and the line-of-interest (*li*) define the re-sampling plane

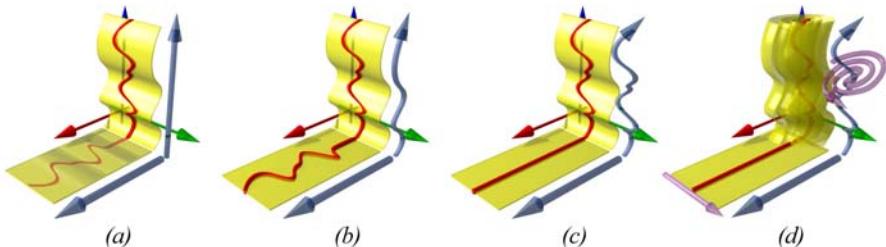


Fig. 2. CPR generation methods: **(a)** projected, **(b)** stretched, **(c)** straightened, **(d)** helical CPR

axis of the tubular structure and the vector-of-interest. We assume the vector-of-interest to be colinear with the y-axis and apply a parallel projection to a free-form surface along the y-axis.

In particular, for each point of the central axis the line-of-interest is projected to the corresponding line of the image. This relationship is defined by the camera's coordinate system (i.e., the up-vector). If the up-vector of the camera is parallel to the z-axis, the z-coordinate of the line-of-interest is mapped directly to the image. If multiple lines-of-interest project onto the same image area compositing is done using *maximum intensity projection* (MIP), *minimum intensity projection* (MinIP), or *averaging* (AVG). Due to parallel projection the spatial relations are maintained by this method. This helps the observer to perceive the spatial arrangement of the vessels.

The first disadvantage of this method is that structures of higher intensity (i.e., bony structures) still may obscure the structures of interest (i.e., vessels). This situation arises, if parts of the line-of-interest associated with a certain point of the central axis contains bony structures and these parts are projected to an image region containing vascular structures from another line-of-interest. The occurrence of such situations heavily depends on the application area. In the case of peripheral vascular

structures this case hardly ever arises. However, the visualization of the carotid artery at the level of the skull-base often leads to such situations.

Another disadvantage of the projected CPR method is the distortion of the central axis' length due to parallel projection. Therefore isometry is not preserved.

3.2 Stretched CPR

The surface defined by the vessel central axis and the vector-of-interest is curved in one dimension and planar in the other one. Stretching the curved dimension results in a plane showing the tubular structure in its entirety without overlapping (see Fig. 2b). This type of CPR is referred to as *stretched CPR*.

Processing all points of the central axis successively, the corresponding lines-of-interest are mapped to the image. This is done by rotating the consecutive point of the central axis around the current line-of-interest. The point is rotated in a way that the resulting plane is coplanar to the viewing plane. Isometry is maintained by this operation as the distance between the two consecutive points is preserved in image space.

Especially we are only interested in the image y-coordinates of the lines-of-interest. Let's assume point P_i to be the last processed point and point P_{i+1} the currently processed point of the central axis. The vector $\mathbf{d}_i = \overrightarrow{P_i P_{i+1}}$ represents the path direction at position i . Furthermore \mathbf{l} is the normalized direction of the line-of-interest. According to formula (1) the offset Δ_i in image space is.

$$\Delta_i = \sqrt{|\mathbf{d}_i|^2 - (\mathbf{l} \cdot \mathbf{d}_i)^2} \quad (1)$$

The image position y_{i+1} of the line-of-interest related to point P_{i+1} is given by $y_{i+1} = y_i + \Delta_i$ where $y_0 = 0$.

The central axis is assumed to be sampled with sub-voxel resolution. Therefore all rows of the image are filled. Introducing a zooming capability requires to interpolate between the lines-of-interest, if necessary.

The generation process of a stretched CPR ensures that other objects do not cover vascular structures. This is one of the key requirements in vessel visualization. The curvature of the tubular structure is still largely maintained by this kind of visualization, thus spatial orientation is still possible for the user.

The main advantage of this CPR type is the preserved isometry. This is important for accurate preoperative planning of endovascular stent-graft treatment of aortic aneurysms. The lengths of normal and abnormal vascular segments need to be determined accurately for sizing an endovascular prosthesis.

3.3 Straightened CPR

The third type of curved planar reformation fully straightens the tubular structure (see Fig. 2c). This CPR method generates a linear representation of the vessel with varying diameter.

At each point P_i of the central axis the tangent vector t_i is calculated. The plane ε_i (*cross-section*) is defined by P_i and t_i . A local coordinate system is defined by two generating vectors of the plane ε_i : \vec{u}_i and \vec{v}_i whereby $\vec{u}_i \perp \vec{v}_i$. The line-of-interest is defined within the plane ε_i by an angle within the unit circle: the angle-of-interest φ .

As either \vec{u} or \vec{v} is mapped to the local coordinate system's x-axis, excessive rotation along the central axis may cause undesired artifacts. Methods exist to minimize this effect [10].

In particular it is not necessary to re-sample the entire cross-section from the data set. It is more efficient to do a transformation from the local coordinate system to the global coordinate system. The direction of the line-of-interest \mathbf{l}_i is given by formula (2):

$$\mathbf{l}_i = \cos \varphi \cdot \mathbf{u}_i + \sin \varphi \cdot \mathbf{v}_i \quad (2)$$

The image offset Δ_i for the line-of-interest corresponding to point P_{i+1} equals to the distance from point P_i to P_{i+1} :

$$\Delta_i = |\overrightarrow{P_i P_{i+1}}| \quad (3)$$

The most obvious disadvantage is the lack of spatial orientation. Only short segments of visible side branches of the parent vessels indicate the topographic position of a given arterial segment.

One advantage of this method is the preserved isometry. Furthermore the direct relation between image height and central axis length makes it easy to create linked displays. Whenever the user points to a certain position of the image, the corresponding cross-section is displayed in a separate view. This feature improves on the lack of spatial orientation.

Another advantage is the easy perception of diameter variations. Due to the elimination of the curvature of the central axis the only varying property along the central axis is the diameter.

3.4 Helical CPR

The basic idea of helical CPR visualization is to display the volumetric interior of a vessel within one image. To accomplish this, a re-sampling strategy different from the previously discussed CPR methods is introduced. All previous mentioned methods re-sample the data in a linear way defined by the vector v_i . The vector-of-interest describes the re-sampling direction which is orthogonal to the viewing direction.

The helical CPR method is based on a non-linear re-sampling of the data. The vector-of-interest as generating element for the surface is replaced by a *spiral-of-interest* (*si*) (see Fig. 2d). This results in a convoluted surface around the central axis. With a sufficiently small distance between each winding the vessel is intersected several times. Stenoses, calcifications, and occlusions are included in the computed surface. The helical surface is flattened and displayed.

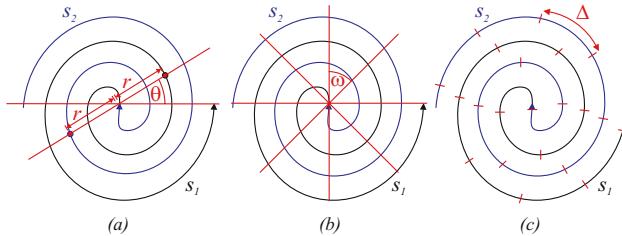


Fig. 3. Spiral-of-interest (a), constant angle sampling (b), constant arc-length sampling (c)

Method Description

Along the central axis of the vessel cross-sections are calculated at an appropriate sampling distance. Within each section a local 2D coordinate system is defined. The center of the cross-section represents the estimated center of the vessel lumen at the corresponding centerline position. Starting from this center point two interleaved spirals s_1 and s_2 are computed (see Fig. 3a). In order to maintain a uniform sampling of the vessel cross-sections a spiral with constant inter-winding distance is selected. This requirement is satisfied by the Archimedean spiral which can be expressed in a simple equation using polar coordinates r and θ :

$$r = a\theta \quad (4)$$

The transformation of points on the spiral into Cartesian coordinates is straightforward. Thus the computation of a point X_{s_1} on s_1 and a point X_{s_2} on s_2 is performed as follows:

$$X_{s_1} = a\theta \begin{pmatrix} \cos \theta \\ \sin \theta \end{pmatrix} \quad X_{s_2} = a\theta \begin{pmatrix} \cos \theta' \\ \sin \theta' \end{pmatrix} \quad \text{where } \theta' = \theta + \pi \quad (5)$$

For an appropriate sampling of the vessel lumen the parameter a was set to $1/\pi$. This assures a constant distance of one between the windings of the two interleaved spirals. The computed points X_{s_1} and X_{s_2} on the spirals are transformed back into volume space and re-sampled.

The center of each scanline in the final CPR image corresponds to the center of the vessel cross-section. Starting from this reference point the image space to the left is filled with data re-sampled by s_1 and to the right with data from s_2 .

Sampling Strategy

The current implementation of the helical CPR technique supports two sampling strategies for computing points from the spirals. In the case of *constant angle* sampling (see Fig. 3b) the angle θ is increased by a constant angle ω for each point. If *constant arc-length* sampling is applied (see Fig. 3c) for each sampling step a constant distance Δ on the arc-length of the spiral is covered.

Constant Angle

For each point re-sampled from the spiral the generating angle θ is incremented by a fixed angle ω . Each winding is rendered into an equal sized area in the final image. Therefore the comparably dense sampled area in close vicinity of the vessel center is amplified in image space. The resulting fish-eye zooming effect is achieved at the cost of an increased distortion.

Constant Arc-Length

Given a fixed sampling distance Δ between two adjacent points on the spiral the increment ω of the angle θ is approximated. The increment ω is defined by the ratio of Δ and the circumference calculated from the most recent radius. As usually small sampling distances are used the error introduced by this approximation is negligible. The extent of the vessel in the CPR image is directly proportional to the volume of the vessel lumen. Thus large vessels occupy superproportional large parts of the image space.

4 Vessel Tree CPR Methods

One substantial disadvantage of CPR visualization for diagnostic purposes is the restriction to only one tubular element. Most clinically relevant “tubes” are part of a branching, anatomic structure. For instance the peripheral arterial tree begins at the abdominal aorta and branches into the left and right common iliac artery which again branches into internal and external iliac arteries and so on. A comprehensive display of all clinically relevant vessels in one image is highly desirable.

The straight forward approach of simply compositing all calculated CPRs does not produce the desired result. No matter which re-sampling strategy is applied artifacts are always introduced in the generated image. Vascular structures are obscured by bones from other layers. Therefore we propose a new image space driven method for compositing projected or stretched CPRs from multiple central axes.

4.1 Multi-Path CPR

The tree of central axes and the volumetric data set is taken as input for the algorithm (Fig. 4a). An enhanced z-buffer b of the same size as the image provides space for information entries containing a reference to a path, a reference to a point of this path (*point*), and depth information. Furthermore each entry contains information about an associated span *begin* and *end*. A span represents the part of an image scanline belonging to a certain path segment.

Tree Projection

The tree (see Fig. 4b) is mapped to the buffer b according to the applied CPR method. Figure 4c shows the entries of the buffer after the CPR projection process. If points of different paths are mapped to the same image position, the one spatially closer to the observer is taken.

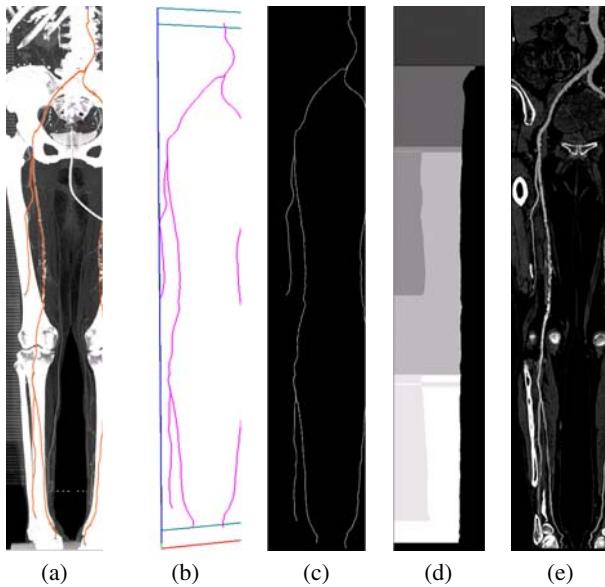


Fig. 4. Generation of multi-path CPRs

Buffer Traversal

In a second step two buffer traversals are needed to determine the length of a span. A span is computed so that the space between neighboring paths within a line is equally divided. The scanline portions at the border of the image are assigned to the leftmost or rightmost path segment respectively (see Fig. 4d).

Data Re-Sampling

In the final step the buffer is traversed again. Each filled entry $b[x,y]$ is processed so that the image line y is filled from position $b[x,y].begin$ to $b[x,y].end$ with the data values associated with point $b[x,y].point$.

This results in a composed CPR through multiple vessel centerlines without overlapping structures, as each tree segment is drawn in a separate image region (see Fig. 4e).

A projected multi-path CPR is generated by using the projected CPR method for each vessel segment in the tree projection step. The properties of the projected CPR method also hold true for the projected multi-path CPR.

Similar to the previously mentioned multi-path CPR method the **stretched multi-path CPR** method is generated by using the stretched CPR method for tree projection. In the recursive tree traversal the image position y_0 of the vessel segment's first line-of-interest has to be handled correctly.

4.2 Untangled CPR

The aim of untangled CPR visualization is to display a vascular tree without overlapping arteries [9]. In order to accomplish this requirement the spatial relations of the projected vessels have to be relaxed. Branching points of the vessel tree are used as pivot points. Rotating the corresponding vessels around these pivot points in image space eliminates vessel overlaps. Keeping the introduced distortions small maintains fast perception and reduces the impact of re-sampling artifacts in the final image. Thus the applied transformations are restricted to the branching points (bifurcations) of the arterial tree only. Additionally these transformations should be appropriate in terms of changing the tree layout and appearance without violating the non-intersection criterion. The *non-intersection criterion* is defined in a way that two vessel hulls must not intersect at any time.

The input of the algorithm is a tree graph representing the topology of the vascular structure. For each vessel segment the centerline of the vessel is stored as a set of adjacent points at an appropriate sampling distance. In practice it turned out that diameter estimations of vessels are not reliable enough in certain cases. Therefore for the purpose of generality the algorithm does not take diameter information into account. However, the adaptation to this additional information would be straightforward.

Method Outline

The untangled CPR method consists of four main steps. As all untangling calculations are performed in image space the tree graph is first mapped to image space using a stretched CPR projection. In a consecutive step a rotation of each subtree with respect to the non-intersection criterion is performed. Afterwards the image space is partitioned in a way that each vessel obtains those parts of the image space which are closest in scan-line direction. Finally the image is rendered.

Tree Projection

The vascular tree is mapped to a projection plane parallel to the viewing plane. For two successive points on a vessel path the subsequent point is rotated around an axis defined by the previous point and the vector-of-interest. The rotation is carried out for each point starting from the root of the vascular tree. The result is a stretched vascular tree.

Untangling Operation

From the projected tree graph in image space the necessary transformations for each node are calculated. This is done by recursively circumscribing the subtrees with vessel hulls. The first pass is bottom up maintaining only the correct transformation of the largest enclosing vessel hull. In a second pass the final transformation for each vessel hull is accumulated top down.

Image Space Partitioning

Before rendering the final image the extent of each vessel segment is cropped in a way that no overlapping image areas remain. This process determines the starting point and the end point of each scanline for rendering.

Rendering

Each vessel segment is rendered separately. Conceptually the vessel strip is first extracted from the dataset using a stretched CPR mapping. Afterwards the strip is transformed to the position defined by the untangling process. In a further step the strip is clipped according to the space partitioning information. Finally each cropped scanline is rendered into the image.

The Vessel Hull Primitive

The *vessel hull* is the basic primitive for further intersection tests. It encloses the vessel's centerline in image space as shown in Fig. 5. The centerline is given as a set of points $P = \{P_0..P_{n-1}\}$. A vessel hull is a sector of a circle. The root of a vascular subtree defines the center point H_c . A matrix \mathfrak{R}_{H_c} associated with each center point describes a rotational transformation of the subsequent tree. The points H'_r and H'_l result from a conservative estimation of the leftmost and rightmost extent of the enclosing subtree seen from the center point.

The vessel hull encloses the centerline of the vessel thus two neighboring vessels touching each other is not prevented by this primitive. To overcome this situation the vessel hull is enlarged by a small angle ε as depicted in Fig. 5. Depending on the size of the inspected vessels this ε may be adjusted by the user on the fly. However an $\varepsilon \leq 2^\circ$ was found to be appropriate for most tested datasets. If the vessel diameter is known at the extremal points P_i and P_j then ε can be easily calculated more accurately. The points comprising the ε -tolerance are referred to as H_r and H_l .

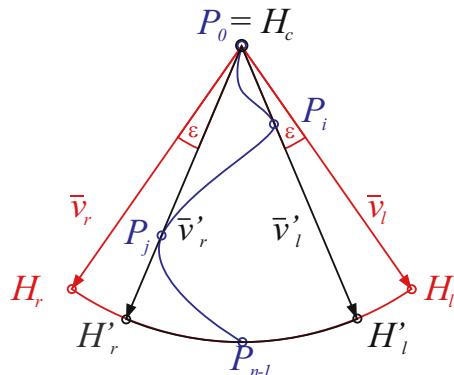


Fig. 5. The vessel hull primitive

Each vessel hull primitive can be described as a tuple of $V = \{H_c, H_l, H_r, \mathfrak{R}_{H_c}\}$ where $\mathbf{v}_l = \overrightarrow{H_c H_l}$ and $\mathbf{v}_r = \overrightarrow{H_c H_r}$.

Putting Things Together

A rule based approach is applied for combining vessel hulls from various parts of the vascular tree (see Fig. 6). The projected vessel tree is approximated by an enclosing hierarchy of vessel hulls. Constructing the vessel hull hierarchy involves several cases which are discussed in the following. A vessel hull created from the vessel centerline is based on *case 1*. Neighboring vessel hulls are combined according to *case 2*. An enclosing vessel hull from two consecutive vessel hulls is created in *case 3*. The assembling process is done bottom up. This results in a binary tree where each node is represented by a vessel hull circumscribing all subjacent vessel hulls.

Case 1

The center point H_c is defined by the first point P_0 of the vessel segment (Fig. 6a). The first point on the convex hull of the vessel segment in clockwise orientation is denominated as point P_i and in counterclockwise direction as point P_j . Because of the stretched CPR mapping from volume to image space, point P_{n-1} is the point with maximum distance from P_0 . Thus the radius of the vessel hull is computed as $|P_{n-1} - P_0|$. The vectors \mathbf{v}_l and \mathbf{v}_r represent the directions from H_c to P_i and P_j respectively. These vectors are scaled according to the radius of the vessel hull. The tolerance angle ϵ is incorporated by a rotation with \mathfrak{R}_ϵ and $\mathfrak{R}_{-\epsilon}$. Finally H_r and H_l are computed.

Case 2

For the case of two adjacent vessel hull primitives V^1 and V^2 (see Fig. 6b) the ordering has to be determined. The left vessel hull with respect to $H_c = H_c^1 = H_c^2$ is denominated as V^l and the right one as V^r . If the vessel hull primitives overlap, an untangling angle γ is computed from \mathbf{v}_l^r and \mathbf{v}_r^l (see Fig. 9). From this angle the rotational matrices $\mathfrak{R}_{H_c^r}$ and $\mathfrak{R}_{H_c^l}$ are calculated. These matrices define a transformation of the vessel hull primitives V^r and V^l in a way that the primitives do not overlap anymore. This implies a transformation of the associated vascular subtree (see Fig. 9).

The vectors \mathbf{v}_l and \mathbf{v}_r of the combined vessel hull are computed from the transformed vectors \mathbf{v}_l^r and \mathbf{v}_r^l . The radius of the enclosing vessel hull is defined by the maximum radius of V^1 and V^2 . From this information H_l and H_r is computed. The newly generated vessel hull encloses the non-overlapping underlying vessel hulls.

Case 3

The combination of two successive vessel hulls is straightforward. V^1 is considered to be the predecessor of V^2 as depicted in Fig. 6c. H_c^1 is taken as the new center point

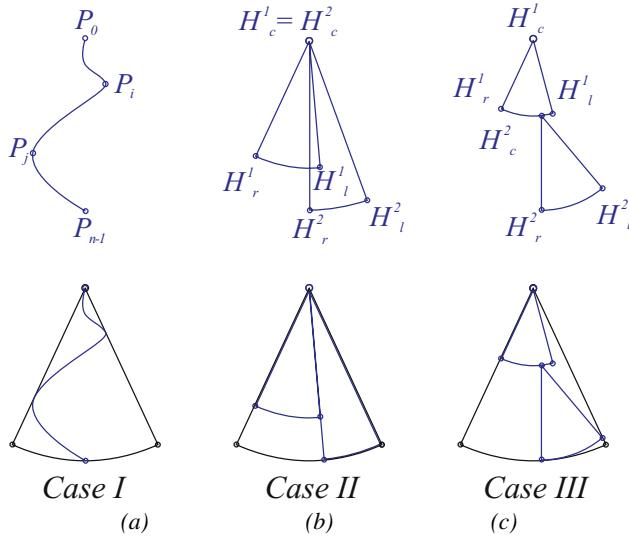


Fig. 6. The three vessel hull combination cases

H_c of the enclosing vessel hull. The direction to the rightmost point of H_r^1 and H_r^2 with respect to H_c is considered to be \mathbf{v}_r . Vector \mathbf{v}_l is calculated likewise. The radius of the new vessel hull V is defined by the maximum distance from H_c to H_r^1, H_l^1, H_r^2 , and H_l^2 .

All treelike vascular structures can be processed using this set of rules. The recursive algorithm finishes with a hierarchy of enclosing vessel hulls where the root hull contains the entire vessel tree. A detailed description of each assembling case in abstract notation is given in Fig. 7.

Layout Definition

The decision of the vessel hull ordering in *case 2* has a significant impact on the layout of the displayed vascular tree. Two different approaches have been investigated (see Fig. 8). The *adaptive layout* is a left-to-right ordering based on the spatial relations of the vascular tree according to the currently used viewing direction. This results in less distortion at the cost of discontinuities during rotational interaction. In contrast to that a *fixed layout* is independent of the viewing direction. For clinical routine a standardized ordering of the blood vessels might be a reasonable solution.

Image Space Partitioning

Before rendering the final image the image space has to be partitioned into fragments for each vascular structure. A principle drawing direction is associated with

Case 1:

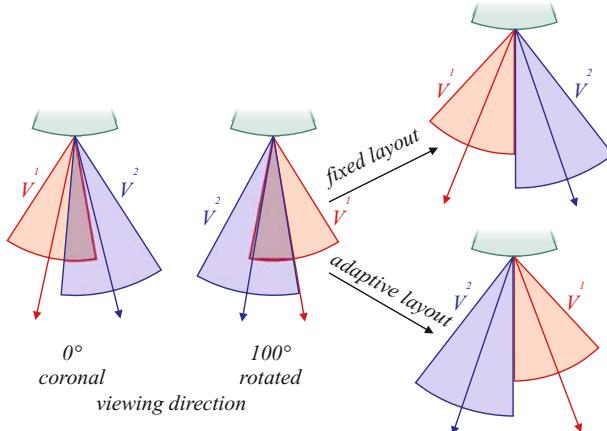
H_c	$\leftarrow P_0$
\mathbf{v}_r	$\leftarrow P_j - H_c, \{P_j P_j, \ell \in P \wedge \forall \ell (\ell \text{ left of } \overrightarrow{H_c P_j})\}$
\mathbf{v}_l	$\leftarrow P_i - H_c, \{P_i P_i, \ell \in P \wedge \forall \ell (\ell \text{ right of } \overrightarrow{H_c P_i})\}$
rad	$\leftarrow P_{n-1} - P_0 $
H_r	$\leftarrow H_c + rad \cdot (\mathfrak{R}_\epsilon \star \mathbf{v}_r / \mathbf{v}_r)$
H_l	$\leftarrow H_c + rad \cdot (\mathfrak{R}_{-\epsilon} \star \mathbf{v}_l / \mathbf{v}_l)$

Case 2:

H_c	$\leftarrow H_c^1$
(V^l, V^r)	$\leftarrow \text{if}(\text{changeOrder}) \text{ then } (V^2, V^1) \text{ else } (V^1, V^2)$
$\mathfrak{R}_{H_c^r}$	$\leftarrow \text{RotationMatrix}(H_c, -0.5 \max(\triangle(\mathbf{v}_l^r, \mathbf{v}_r^l), 0))$
$\mathfrak{R}_{H_c^l}$	$\leftarrow \text{RotationMatrix}(H_c, +0.5 \max(\triangle(\mathbf{v}_l^r, \mathbf{v}_r^l), 0))$
\mathbf{v}_r	$\leftarrow \mathbf{v}_r^r$
\mathbf{v}_l	$\leftarrow \mathbf{v}_l^l$
rad	$\leftarrow \max(rad^l, rad^r)$
H_r	$\leftarrow H_c + rad \cdot (\mathfrak{R}_{H_r} \star \mathbf{v}_r / \mathbf{v}_r)$
H_l	$\leftarrow H_c + rad \cdot (\mathfrak{R}_{H_{left}} \star \mathbf{v}_{left} / \mathbf{v}_l)$

Case 3:

H_c	$\leftarrow H_c^1$
\mathbf{v}_r	$\leftarrow \text{if}(H_r^2 \text{ left of } \overrightarrow{H_c H_r^1}) \text{ then } H_r^1 - H_c \text{ else } H_r^2 - H_c$
\mathbf{v}_l	$\leftarrow \text{if}(H_l^2 \text{ right of } \overrightarrow{H_c H_l^1}) \text{ then } H_l^1 - H_c \text{ else } H_l^2 - H_c$
rad	$\leftarrow \max(rad^1, H_c - H_r^2 , H_c - H_l^2)$
H_r	$\leftarrow H_c + rad \cdot \mathbf{v}_r / \mathbf{v}_r $
H_l	$\leftarrow H_c + rad \cdot \mathbf{v}_l / \mathbf{v}_l $

Fig. 7. Assembling of vessel hulls**Fig. 8.** Different layout definition

each vessel segment. Each point of the projected vessel centerline maintains a scanline deduced from this principal drawing direction. These scanlines are transformed according to the rotational matrix \mathfrak{R}_{H_c} (see Fig. 9). The scanlines represent those parts of the image into which the corresponding re-sampled data from the dataset are rendered.

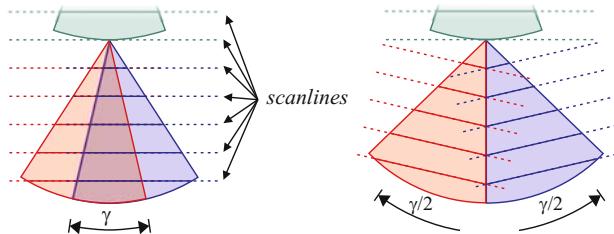


Fig. 9. Image space partitioning

In order to avoid overlapping areas an appropriate start and endpoint for each scanline has to be determined. A directional distance map is defined by the projected centerlines of the vessel tree and its scanlines. In contrast to the traditional distance map the distance metric is not defined by an Euclidean distance but by a distance along scanlines. The result of this operation is a fragmented image space where each vessel segment is assigned a maximal image area.

5 Results

Table 1 summarizes the introduced methods: projected CPR (*Proj.*), stretched CPR (*Stre.*), straightened CPR (*Stra.*), and helical CPR (*Helical.*), multi-path projected CPR (*M-Proj.*), multi-path stretched CPR (*M-Stre.*), and untangled CPR (*Untang.*). The methods are grouped according to whether displaying a whole vessel tree is possible or not (*Vessel tree*). The criterion *Spatial Perception* indicates how radiologists judge the spatial expressiveness (i.e. the easiness to map positions within the CPR to locations in the volumetric dataset without computational aid). Whether the method preserves true distances in close vicinity of the computed centerline is depicted by the field *Isometry*. The possible occurrence of bones superimposing the tracked vessel is expressed by *Occlusions (bone)*. The criterion *Occlusions (artery)* shows whether arteries may cross in image space and consequently overlap each other. Finally *Rotation needed* indicates if multiple viewing directions are needed in order to investigate the total vessel lumen.

Table 1. Comparison of CPR methods

	Proj.	Stre.	Stra.	Helical	M-Proj.	M-Stre.	Untang.
Vessel tree	no	no	no	no	yes	yes	yes
Spatial Perception	high	med	low	low	high	med	med
Isometry	no	yes	yes	yes	no	yes	yes
Occlusions of bones	poss.	no	no	no	poss.	no	no
Occlusions of arteries	poss.	no	no	no	poss.	poss.	no
Rotation needed	yes	yes	yes	no	yes	yes	yes

5.1 Single Vessel CPR Methods

A summary of the four different CPR methods displaying single vessel segments is presented in this Section. The comparison is performed on a dataset of the abdominal area (see Fig. 10) as well as from peripheral arteries (see Fig. 11 and Fig. 12).

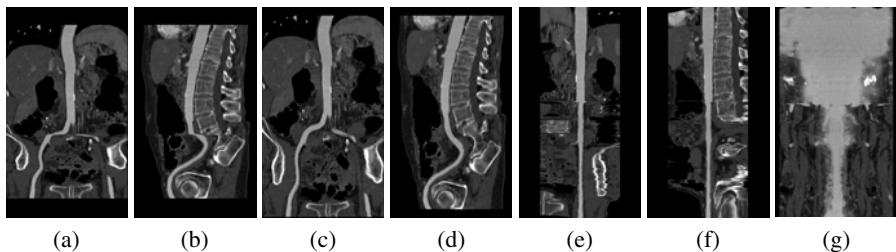


Fig. 10. Comparison of CPR methods: A coronal and sagittal display of a projected CPR (a,b), stretched CPR (c,d), and straightened CPR (e,f). A helical CPR (g)

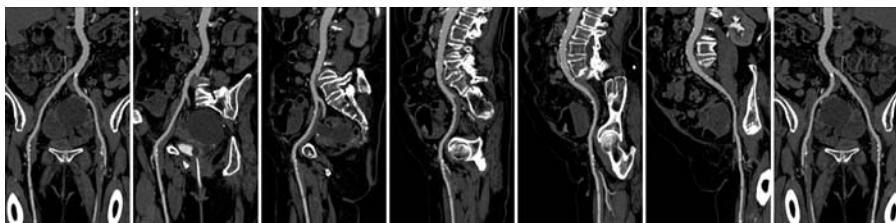


Fig. 11. Rotating stretched CPR: 0° , 30° , 60° , 90° , 120° , 150° , 180°

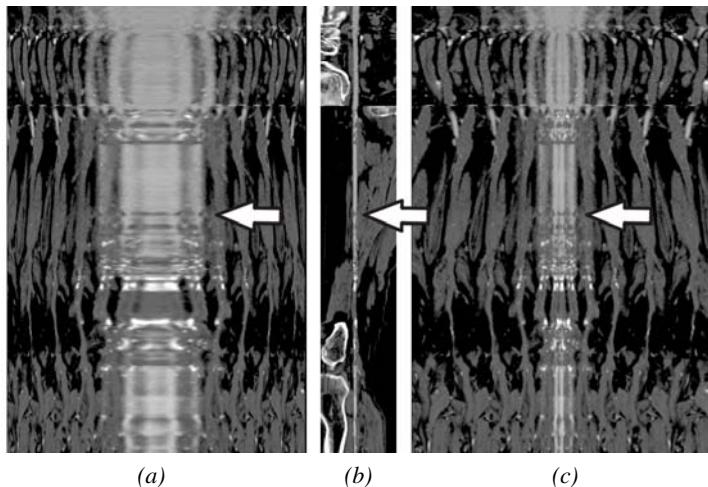


Fig. 12. A helical CPR with constant angle sampling (**a**), a straightened CPR (**b**), and constant arc-length sampled helical CPR (**c**)

Projected CPR

In Fig. 10a,b a projected CPR from coronal and sagittal view is displayed. Projection artifacts arise whenever the vessel is in plane with the viewing direction.

Stretched CPR

Accordingly Fig. 10c,d presents a stretched CPR. The vessel's isometry is preserved in the image.

Straightened CPR

In the case of a straightened CPR the vessel is straightened to a line centered in the image space (see Figure 10e,f).

Helical CPR

The application of a helical CPR technique is presented in Fig. 10. A dataset with more anatomical features is presented in Fig. 12. A constant angle and constant arc-length sampled helical CPR is compared to a straightened CPR image. The white arrows illustrate an example where the helical CPR outperforms a traditional CPR. The small flow-channel of the stenosis is not touched by the displayed longitudinal section of the straightened CPR and therefore not visible. However in both helical images this flow-channel is displayed. As eccentric lesions cause repetitive patterns in image space, the attention of the observer is immediately drawn to those areas even if a lesion is not visible in a standard CPR display.

5.2 Vessel Tree CPR Methods

This section presents the application of the introduced CPR methods. Vessel trees from a real world data set with a scanned resolution of $512 \times 512 \times 988$ are visualized.

Projected Multi-Path CPR

A projected multi-path CPR is presented in Fig. 13. The projected multi-path CPR on the left provides an overview of the processed data set. A close-up of the upper image region is shown from different viewing directions.

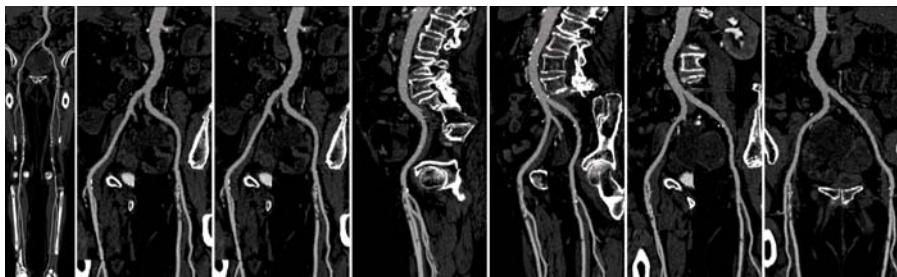


Fig. 13. Overview of a projected multi-path CPR at 0° . Enlargement of the projected multi-path CPRs at $30^\circ, 60^\circ, 90^\circ, 120^\circ, 150^\circ, 180^\circ$

Stretched Multi-Path CPR

If the vessel tree is mapped into the image space according to the stretched CPR method a combination of isometry preserving vessel segments is displayed. In Fig. 14 the corresponding area was processed using the stretched multi-path CPR method.

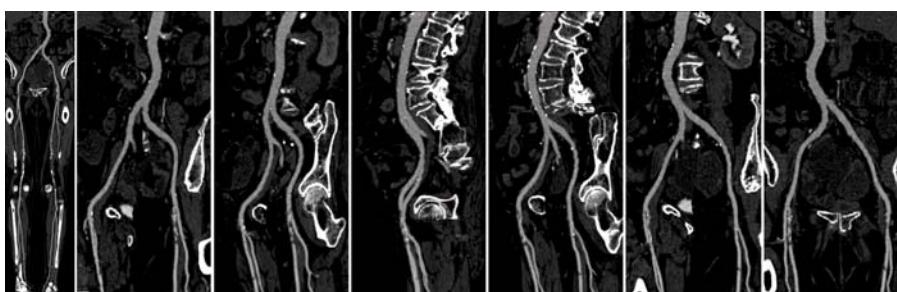


Fig. 14. Overview of a stretched multi-path CPR at 0° . Enlargements of the stretched multi-path CPRs at $30^\circ, 60^\circ, 90^\circ, 120^\circ, 150^\circ, 180^\circ$

Untangled CPR

A comparison of a stretched multi-path CPR and an untangled CPR is presented in Fig. 15. In the case of a lateral view the multi-path CPR display provides hardly any diagnostically relevant information. Many superimposed arteries obscure each other. In comparison to that the untangled CPR still provides an unobstructed view of the entire vascular tree. Each vessel segment is displayed in diagnostic quality.

The examination is intended to be done on just a small set of pre-computed images, the performance of the algorithm is acceptable for applications in the clinical workflow. The displayed untangled CPR image in Fig. 15 was calculated with an original image size of 1164×1097 pixels. The average rendering time per image of the current Java based implementation on a PC workstation with an Intel PIII 1GHz main processor takes 2.3 seconds.

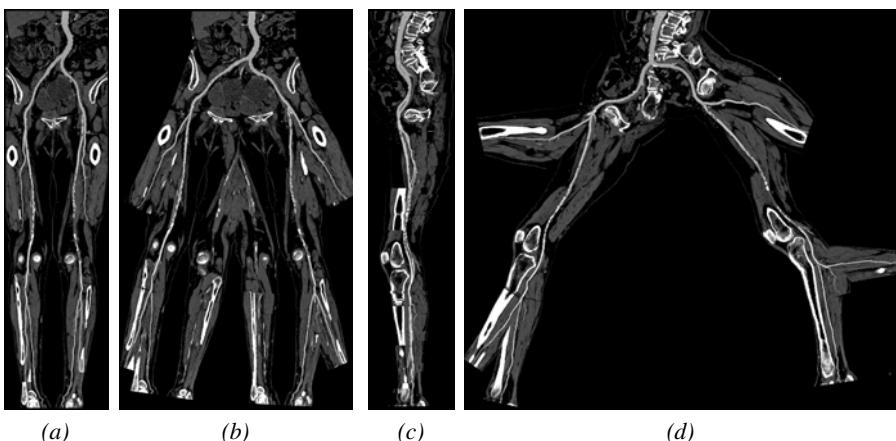


Fig. 15. A peripheral CTA dataset rendered from coronal and sagittal view using stretched multi-path CPR (**a, c**) and untangled CPR (**b, d**) respectively (fixed layout)

Figure 16 presents a sequence of untangled CPR images from an abdominal CT angiographic dataset. A fixed layout was used. A 1D transfer function was applied to the re-sampled data approximating the tissue color of the anatomical structures.

6 Conclusions

In this paper methods for the generation of *Curved Planar Reformation* (CPR) images have been presented. This method allows the visualization of entire tubular structures with minimal modification of the original data. The main application of this visualization method is *Computed Tomography Angiography* (CTA). With volume rendering even mild vessel wall calcifications may obscure the true vessel lumen

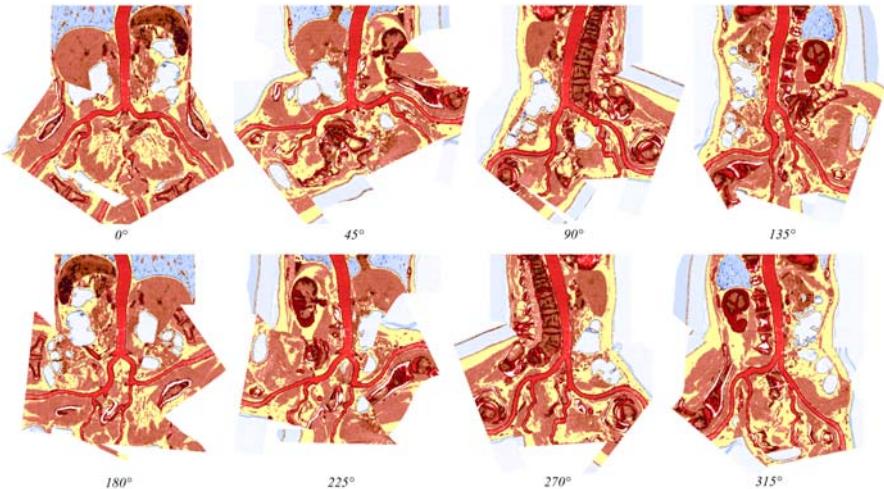


Fig. 16. A colored sequence of untangled CPR images from different viewing directions

(flow channel), which is the clinically relevant information. CPR displays the vessel lumen also in the presence of vessel wall calcifications.

Four different methods (i.e., projected, stretched, straightened, and helical CPR) have been demonstrated. A comparison of the three methods with respect to spatial perception, isometry, and possible occlusions has shown that the stretched CPR is the preferred method for many applications. With the helical CPR the dataset is re-sampled in a spiral manner making the entire vessel visible. The motivation for this visualization technique was not an imitation of the natural appearance of the object but the revelation of diseased vessel segments. An explorative study has shown that the detection of vessel abnormalities is possible from such visualizations which literally peel-off the vessel volume.

As fewer images have to be generated and as the spatial relationships of the vascular tree are well preserved, we believe that multi-path CPRs are not only very well suited for diagnostic purposes, but also for the documentation and for communicating the extent of diseases to the treating physician. However, this method suffers from arteries possibly crossing each other in image space and thus reduce the diagnostic value of the image.

Small distortions of the vascular tree prevent self-intersections in the case of the untangled CPR. The untangled CPR has significant advantages over existing multi-path CPR techniques. This new technique produces an unobscured display of a vascular tree, independent of the viewing direction. Small rotations around the branching points of a vessel tree eliminate occlusions. Therefore the size of the introduced distortion is kept small.

Even though the use of image space is not optimal, the main requirement of unobscured display of vessels from any viewing direction is fulfilled. In addition,

untangled CPR preserves isometry which is an important requirement for vascular lesion assessment. The potential for clinical application of this technique is obvious. The untangled CPR provides a more efficient way to assess any complex arterial trees for the presence and extent of vascular disease. Clinical validity and applicability to other vascular territories are currently investigated.

Acknowledgements

The work presented in this publication has been funded by the ADAPT project (FFF-804544). ADAPT is supported by *Tiani Medgraph*, Vienna (<http://www.tiani.com>), and the *Forschungsförderungsfonds für die gewerbliche Wirtschaft*, Austria. See <http://www.cg.tuwien.ac.at/research/vis/adapt> for further information on this project. qu

References

1. S. Achenbach, W. Moshage, D. Ropers, and K. Bachmann. Curved Multiplanar Reconstructions for the Evaluation of Contrast-Enhanced Electron-Beam CT of the Coronary Arteries. In *Am. J. Roentgenol.*, pp. 895–899, 1998.
2. K. Addis, K. Hopper, T. Iyriboz, Y. Liu, S. Wise, C. Kasales, J. Blebea, and D. Mauger. CT Angiography: In Vitro Comparison of Five Reconstruction Methods. In *Am. J. Roentgenol.*, pp. 177:1171–1176, 2001.
3. B. Avants and J. Williams. An Adaptive Minimal Path Generation Technique for Vessel Tracking in CTA/CE-MRA Volume Images. In *MICCAI 2001*, pp. 707–716, 2000.
4. P. Felkel, A. Fuhrmann, A. Kanitsar, and R. Wegenkittl. Surface Reconstruction Of The Branching Vessels For Augmented Reality Aided Surgery. In *BIO SIGNAL 2002*, pp. 252–254, June 2002.
5. H. Hahn, B. Preim, D. Selle, and H.-O. Peitgen. Visualization and Interaction Techniques for the Exploration of Vascular Structures. In *IEEE Visualization 2001*, pp. 395–402. ACM, October 2001.
6. S. He, R. Dai, B. Lu, C. Cao, H. Bai, and B. Jing. Medial Axis Reformation: A New Visualization Method for CT Angiography. *Academic Radiology*, 8:726–733, 2001.
7. A. Kanitsar, D. Fleischmann, R. Wegenkittl, P. Felkel, and M. E. Gröller. CPR - Curved Planar Reformation. In *IEEE Visualization 2002*, pp. 37–44, October 2002.
8. A. Kanitsar, R. Wegenkittl, P. Felkel, D. Fleischmann, D. Sandner, and E. Gröller. Computed Tomography Angiography: A Case Study of Peripheral Vessel Investigation. In *IEEE Visualization 2001*, pp. 477–480, October 2001.
9. A. Kanitsar, R. Wegenkittl, D. Fleischmann, and M. E. Gröller. Advanced Curved Planar Reformation: Flattening of Vascular Structures. In *IEEE Visualization 2003*, pp. 43–50, October 2003.
10. F. Klok. Two Moving Coordinate Frames for Sweeping along a 3D Trajectory. In *Computer Aided Geometry Design*, pp. 3:217–229, 1986.
11. A. Koechl, A. Kanitsar, F. Lomoschitz, E. Groeller, and D. Fleischmann. Comprehensive Assessment of Peripheral Arteries using Multi-path Curved Planar Reformation of CTA Datasets. In *Europ. Rad.*, volume 13, pp. 268–269, 2003.

12. G. D. Rubin, A. Schmidt, L. Logan, and M. Sofilos. Multi-Detector Row CT Angiography of Lower Extremity Arterial Inflow and Runoff: Initial Experience. In *Radiology 2001*, pp. 146–158, 2001.
13. O. Wink, W. Niessen, and M. Viergever. Fast Delineation and Visualization of Vessels in 3-d Angiographic Images. *IEEE Transactions on Medical Imaging*, 19:337–346, 2000.
14. C. Zahlten, H. Juergens, and H.-O. Peitgen. Reconstruction of Branching Blood Vessels from CT-Data. In *Eurographics Workshop on Visualization in Scientific Computing*, pp. 161–168, 1994.

Part III

Vector Field Visualization

Clifford Convolution and Pattern Matching on Irregular Grids

Julia Ebliing and Gerik Scheuermann

Department of Computer Science
University of Leipzig
PF 920
D-04109 Leipzig
{ebling|scheuer}@informatik.uni-leipzig.de

Summary. Flow features are the essence of fluid flow data and their extraction and analysis is a major goal of most flow visualizations. Unfortunately, most techniques are sensitive to noise and limited to a certain class of features like vortices. Excellent general feature detection methods for scalar fields can be found in image processing. Many of these methods use convolution filters. In an earlier paper, we showed that the convolution operator can be extended to vector fields using Clifford algebra, but the approach is limited to uniform grids. In this article, we extend this approach to irregular grids by examining three different methods. Results on several CFD data sets clearly favor a local resampling of the flow field.

1 Introduction and Related Work

Most flow simulations and measurements want to study overall structure and specific features, i.e. pattern of streamlines with conspicuous behavior. Flow visualization intends to help the user to find and analyze features and structures. Direct visualization methods like hedgehogs do not reveal features like vortices, sinks, sources, separation and attachment. Even streamline based methods may lead to missing features, especially without knowing the right starting points. Texture based methods like LIC [1, 12] do a quite good job in 2D, but a convincing solution in 3D is still missing. Topology [5, 15, 16], on the other hand, is directed to the overall structure since not all features are easily connected to it. Furthermore, the presentation of 3D topology produces visibility problems.

Thus, an automatic approach for feature extraction and visualization is needed, allowing the detection of every kind of important pattern in flows. Streamlines can be used in a second step to study the features. Earlier attempts usually try to give an analytic model of a feature and create an algorithm for feature detection from there. Besides the limitations of the model, most approaches have severe robustness problems.

In an earlier paper [2], we proposed to transfer image processing methods, especially convolution filters, to flow feature detection. Unfortunately, the method was

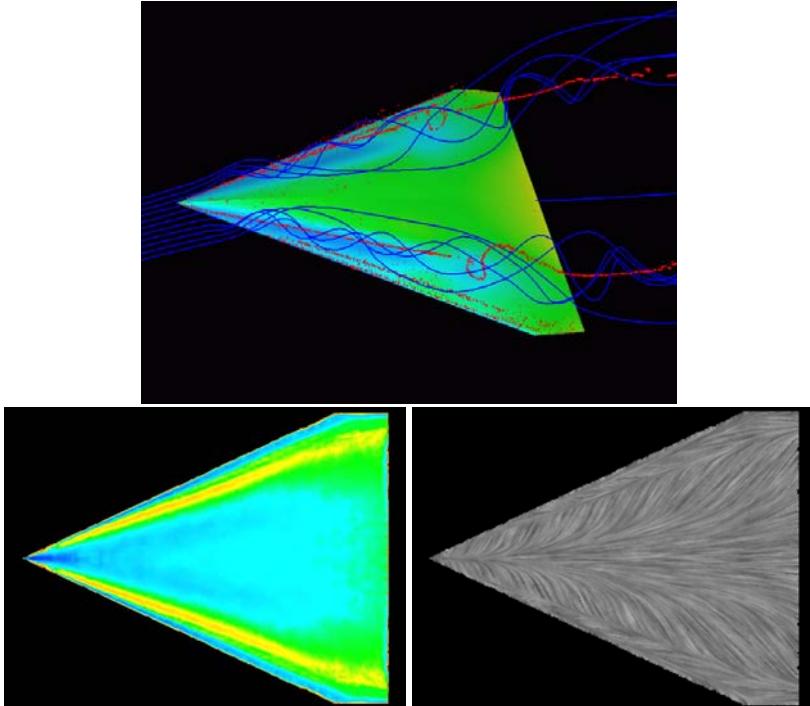


Fig. 1. (*Top*) The 3D delta wing data set. This is a study of vortex breakdown. The pressure on the surface is color coded. The results of the algorithm of Sujudi-Haines basically depict two vortices. Some streamlines are drawn to enhance the understanding of the flow. (*Bottom*) The flattened surface of the delta wing with the wall shear stress vectors. (*Bottom left*) Pattern matching of a 2D vector field with a 5×5 divergence filter mask. Adaptive color coding of the results. Light areas corresponds to the highest similarity and to convergence and dark areas to high negative similarity values and divergence. (*Bottom right*) LIC of the original vector field

limited to uniform grids. In practice, many data sets are defined on unstructured grids. Thus the grid can be dense in interesting regions where the flow behavior is complex and sparse in regions where the flow is more simple. Thus the size of the data sets is reduced considerably in comparison to those on regular or uniform grids while still fulfilling the sampling theorem which states how often a certain frequency has to be sampled in order to allow a complete reconstruction of the signal.

2 Related Work

Pagendarm and Walters [11] and Walsum et al. [17] were the first to shape the term “features” for flow fields. A feature is simply a region of interest in the dataset. It can be extracted using a feature criterion evaluation function. This function can be

a logical combination of several scalar thresholds using boolean algebra. Often, the scalar thresholds are not only applied to the data values but to derived values as well.

An example for model based feature extraction is due to Kenwright [10], for separation and attachment line detection. These lines show where the flow attaches itself to or separates from a surface. Kenwright gives two algorithms based on eigenvector analysis of the velocity gradient tensor to extract separation and attachment lines.

The most prominent feature class in flows are vortices. There are region and line based vortex definitions and corresponding algorithms in the literature. A good overview can be found in the thesis of Roth [13].

Image processing [8, 9] provides a lot of robust and useful tools for feature detection. It seems to be a good idea to transfer them to flow visualization. The first idea regarding image processing on vector fields is to simply treat a vector field as several scalar fields. Thus, the Fourier transformation can be used. Granlund and Knutson [3] have investigated this approach in 2D. They define lines and edges by a simple neighborhood. This means that the neighborhood can be modeled by a function that varies only in one direction. This function is called simple function. When the points are in a simple neighborhood, they can estimate local orientation, symmetries and curvature. They use these methods to extract texture borders that can be described as a sudden change in a feature vector descriptor field.

Another approach is to define a multiplication of vectors and thus convey the convolution to vector fields. Heiberg et al. [4] define convolution on vector fields with the scalar product of two vectors:

$$(\mathbf{h} * \mathbf{f})_s(\mathbf{x}) = \int_{\mathbb{R}^d} \langle \mathbf{h}(\mathbf{x}'), \mathbf{f}(\mathbf{x} - \mathbf{x}') \rangle d\mathbf{x}'$$

where s_n is the filter response, \mathbf{f} is the normalized vector field and \mathbf{h}_n the filter mask with direction n . This convolution is referred to as scalar convolution in the following. As the scalar product is used, it gives an approximation of the cosine of the angle between the structure in the vector field and the direction of the filter mask. Thus it results in a similarity measure.

Furthermore, Heiberg et al. [4] give an algorithm to compute a similarity measure in 3D independent of the direction of the filters. The vector field is normalized and the filter mask weighted with a rotational symmetric function. The filter mask is rotated in six directions evenly distributed over a hemisphere. The six rotated filters form a filter set. Next, the convolutions of the six filters and the field are computed. Then, with help of a tensor, direction and similarity are calculated out of the squared filter responses and the directions of the six filter masks. The drawback of this algorithm is that it assumes the masks to describe a simple neighborhood. Furthermore, the filter directions are only within one hemisphere. Due to annihilation effects, directed patterns are only recognized well when they are also directed in this hemisphere. The algorithm works well with straight vortices but a structure like the one in Fig. 2 is not recognized when rotated disadvantageous.

Clifford convolution is an extension of the classical convolution from image processing to multivector fields. A multivector in 3D consists of the sum of a scalar, a 3D vector, a 3D bivector and a trivector. Scalar and vector are as usual. Bivectors

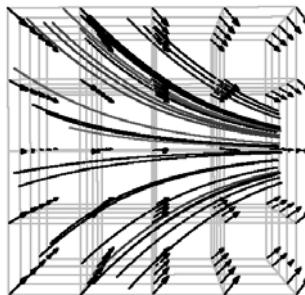


Fig. 2. A convergent flow. The algorithm of Heiberg et al. [4] does not recognize this structure if it is rotated disadvantageous

can be identified with a planar direction and a limited oriented area. The unit trivector gives the volume spanned by three orthogonal unit vectors building a right hand system (Fig. 3). Thus, Clifford convolution is a unified notation for the convolution of scalar, vector and multivector valued field and filter. That means scalar filters like gradient or smoothing filters from image processing can be applied just like vector filters for pattern matching.

Convolution and correlation are closely related. One can be computed from the other by just permuting the mask accordingly. Thus, correlation between vector fields can be computed for pattern matching. As Clifford convolution is based on the Clifford product, convolution of vector valued field and mask results in an approximation of the relative geometric position between the structures in the filter mask and the vector field. This is a property which will be used for pattern matching. The idea is to rotate the mask in the direction of the structure in the field which is computed from the results of a Clifford convolution. A scalar convolution of the rotated mask and the vector field is computed as a similarity measure. In practice, the algorithm is a little bit more difficult.

So far the Clifford convolution and the other approaches for transferring image processing to vector fields have only been defined for continuous or uniform grids and can not be directly transferred to unstructured grids. In this paper, we discuss and compare several approaches for the extension of Clifford convolution to irregular

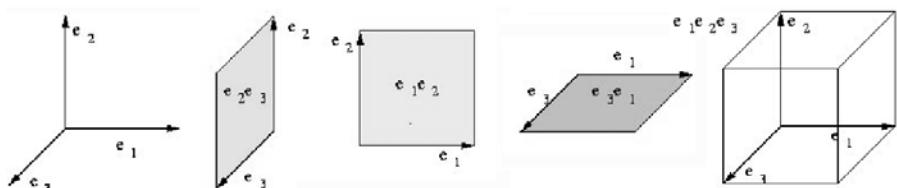


Fig. 3. Unit vectors, bivectors and trivector in G_3

grids. Thus, the pattern matching algorithm can be applied to irregular grids, too. An example is pattern matching on a flattened surface of the delta wing in Fig. 1.

The extensions to irregular grids are all based on some sort of local resampling. First, the mask is aligned to a grid point and scaled accordingly to the local grid. The first idea is to resample the vector field at the nodes of the aligned mask. For acceleration, we also discuss some different methods of resampling the mask. We resample at all points of the vector field which lie in the box defined by the aligned and scaled mask. Another approach which has been tried is to use only those points which are in a n -neighborhood of the point where the convolution is computed.

3 Clifford Algebra

Clifford algebra [6, 7, 15] extends the classical description of an Euclidean n -space, which is a real n -dimensional vector space with scalar product, to a real algebra. Thus, a multiplication of vectors is defined. Furthermore, a geometric interpretation of the product of two vectors is given. This product contains sine and cosine of the angle between the two vectors and the plane in which the angle is measured. Rotation of vectors can be easily described and calculated within Clifford algebra, too [6]. The Clifford product is used for the definition of the Clifford convolution.

3.1 Clifford Algebra in 3D

For the 3-dimensional Euclidean vector space E^3 , we get a 8-dimensional \mathbb{R} -algebra G^3 with the basis $1, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_1\mathbf{e}_2, \mathbf{e}_2\mathbf{e}_3, \mathbf{e}_3\mathbf{e}_1, \mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$ as a real vector space. The elements of the algebra are called multivectors. The multiplication of multivectors is defined as associative, bilinear and by the equations

$$\begin{aligned} 1\mathbf{e}_j &= \mathbf{e}_j, & j &= 1, 2, 3 \\ \mathbf{e}_j\mathbf{e}_j &= 1, & j &= 1, 2, 3 \\ \mathbf{e}_j\mathbf{e}_k &= -\mathbf{e}_k\mathbf{e}_j, & j, k &= 1, 2, 3, j \neq k \end{aligned}$$

Thus, a multiplication of vectors is described, too. The usual vectors $(x, y, z) \in \mathbb{R}^3$ are identified with

$$x\mathbf{e}_1 + y\mathbf{e}_2 + z\mathbf{e}_3 \in E^3 \subset G^3.$$

An arbitrary multivector \mathbf{A} can be written as

$$\mathbf{A} = \alpha + \mathbf{a} + I_3(\mathbf{b} + \beta)$$

with $\alpha, \beta \in \mathbb{R}$, $\mathbf{a}, \mathbf{b} \in E^3$, $I_3 = \mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$, $(I_3)^2 = -1$. α is the scalar part, \mathbf{a} the vector, $I_3\mathbf{b}$ the bivector and $I_3\beta$ the trivector.

The grade projectors $\langle \rangle_j: G^3 \rightarrow G^3$ are the maps

$$\langle \mathbf{A} \rangle_0 = \alpha, \quad \langle \mathbf{A} \rangle_1 = \mathbf{a},$$

$$\langle \mathbf{A} \rangle_2 = I_3 \mathbf{b}, \quad \langle \mathbf{A} \rangle_3 = I_3 \beta.$$

The Clifford multiplication of two vectors $\mathbf{a}, \mathbf{b} \in E^3$ results in

$$\mathbf{ab} = \langle \mathbf{a}, \mathbf{b} \rangle + \mathbf{a} \wedge \mathbf{b},$$

where $\langle \cdot, \cdot \rangle$ is the inner product and \wedge the outer product. Furthermore, we have

$$\langle \mathbf{ab} \rangle_0 = \langle \mathbf{a}, \mathbf{b} \rangle = \|\mathbf{a}\| \|\mathbf{b}\| \cos \omega$$

$$\|\langle \mathbf{ab} \rangle_2\| = \|\mathbf{a} \wedge \mathbf{b}\| = \|\mathbf{a}\| \|\mathbf{b}\| \sin \omega$$

where ω is the angle between \mathbf{a} and \mathbf{b} . $\langle \mathbf{ab} \rangle_2$ corresponds to the plane through \mathbf{a} and \mathbf{b} as it is the corresponding bivector.

The 2D Clifford algebra is defined analog, see [6, 7, 15].

3.2 Integral and Derivative

Let \mathbf{F} be a multivector valued function of a vector variable \mathbf{x} defined on some region of the Euclidean space \mathbb{R}^d . This function can also be called field. If the function is only scalar or vector valued, we will call it scalar or vector field, but we will always regard it as a special multivector valued function within Clifford algebra.

The Riemann integral of a multivector valued function \mathbf{F} is defined as

$$\int_{\mathbb{R}^d} \mathbf{F}(x) |d\mathbf{x}| = \lim_{\substack{|\Delta \mathbf{x}_j| \rightarrow 0 \\ n \rightarrow \infty}} \sum_{i=1}^n \mathbf{F}(\mathbf{x}_j) |\Delta \mathbf{x}_j|.$$

This integral is grade preserving and can be discretized into sums using quadrature formulas. The directional derivative of \mathbf{F} in direction \mathbf{r} is

$$\mathbf{F}_r(\mathbf{x}) = \lim_{s \rightarrow 0} \frac{[\mathbf{F}(\mathbf{x} + s\mathbf{r}) - \mathbf{F}(\mathbf{x})]}{h}$$

with $s \in \mathbb{R}$. With the gradient

$$\nabla = \sum_{j=1}^d \mathbf{e}_j \frac{\partial}{\partial \mathbf{x}_j},$$

the total derivative of \mathbf{F} can be defined as

$$\nabla \mathbf{F}(\mathbf{x}) = \sum_{j=1}^d \mathbf{e}_j \mathbf{F}_{\mathbf{e}_j}(\mathbf{x}).$$

Curl and divergence of a vector valued function \mathbf{f} can be computed as:

$$\text{curl } \mathbf{f} = \nabla \wedge \mathbf{f} = \frac{(\nabla \mathbf{f} - \mathbf{f} \nabla)}{2}$$

$$\text{div } \mathbf{f} = \langle \nabla, \mathbf{f} \rangle = \frac{(\nabla \mathbf{f} + \mathbf{f} \nabla)}{2}$$

4 Clifford Convolution

Clifford convolution gives a unified notation for the convolution of scalar, vector and multivector valued field and mask. It is an extension of the classical convolution on scalar fields.

Clifford multiplication can be regarded as a correlation of a point in the vector field with a 1×1 filter mask. Thus, Clifford correlation with larger masks is an averaging of the geometric relations of the single vectors. The direction of a structure in the field can be computed out of this correlation and the direction of the filter mask. This property is used for the pattern matching presented in this paper. As every correlation can be described by a convolution with a suitably adjusted filter mask, we will often interchange convolution and correlation. Figure 4 presents some typical vector masks.

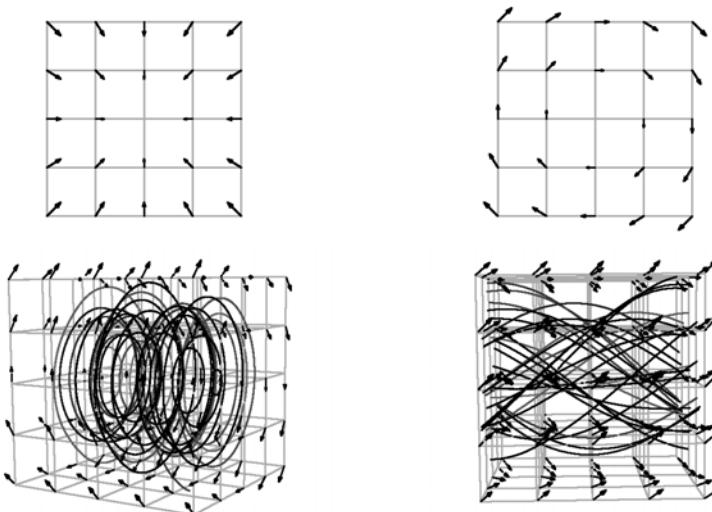


Fig. 4. Some filter masks. (Top left) 2D convergence; (top right) 2D rotation; (bottom left) rotation in one coordinate plane in 3D; (bottom right) potential vortex in 3D

4.1 Convolution in Scalar Fields

For a continuous signal $f : \mathbb{R}^d \rightarrow C$, the convolution with the filter $h : \mathbb{R}^d \rightarrow C$ is defined by

$$(h * f)(\mathbf{x}) = \int_{\mathbb{R}^d} h(\mathbf{x}') f(\mathbf{x} - \mathbf{x}') d\mathbf{x}'.$$

The spatial correlation is defined by

$$(h \star f)(\mathbf{x}) = \int_{\mathbb{R}^d} h(\mathbf{x}') f(\mathbf{x} + \mathbf{x}') d\mathbf{x}'.$$

Thus, it is just a convolution with a filter that has been reflected at its center.

Every linear and shift invariant filter (LSI filter) can be described by a convolution with a filter mask. A lot of filters for smoothing images and for edge detection are LSI filter. Thus, convolution is an important operation in image processing [8, 9].

4.2 Convolution in Vector Fields

Now let \mathbf{F} be a multivector field and \mathbf{H} a multivector valued filter. The Clifford convolution is defined as

$$(\mathbf{H} * \mathbf{F})(\mathbf{x}) = \int_{\mathbb{R}^d} \mathbf{H}(\mathbf{x}') \mathbf{F}(\mathbf{x} - \mathbf{x}') |\mathrm{d}\mathbf{x}'|$$

using the Clifford product of multivectors. The spatial Clifford correlation is defined analog:

$$(\mathbf{H} \star \mathbf{F})(\mathbf{x}) = \int_{\mathbb{R}^d} \mathbf{H}(\mathbf{x}') \mathbf{F}(\mathbf{x} + \mathbf{x}') |\mathrm{d}\mathbf{x}'|$$

For discrete multivector fields, convolution and correlation have to be discretized. For 3D uniform grids, the discretizations are

$$(\mathbf{H} * \mathbf{F})_{j,k,l} = \sum_{s=-r}^r \sum_{t=-r}^r \sum_{u=-r}^r \mathbf{H}_{s,t,u} \mathbf{F}_{j-s,k-t,l-u}$$

and

$$(\mathbf{H} \star \mathbf{F})_{j,k,l} = \sum_{s=-r}^r \sum_{t=-r}^r \sum_{u=-r}^r \mathbf{H}_{s,t,u} \mathbf{F}_{j+s,k+t,l+u}$$

with $j, k, l, s, t, u \in \mathbb{Z}$. r^3 is the dimension of the grid of the filter mask and the (j, k, l) are grid nodes. The Clifford convolution is an extension of the scalar convolution of Heiberg et al. [4] as

$$(\mathbf{h} * \mathbf{f})_s = < (\mathbf{h} * \mathbf{f}) >_0$$

for vector valued \mathbf{h} and \mathbf{f} .

The convolution has to be computed at every point of the grid. But at the border of the vector field, the convolution needs values outside the vector field. So, similar to image processing, there is the problem of boundary values. The solutions for this problem are the same as in image processing with all their advantages and disadvantages [8, 9]. The values can be chosen as follows:

1. Zero. Thus, artificial edges at the border are created.
2. Extrapolated. At the simplest case, one can take the values at the boundary. All extrapolations lay too much stress on the border values.
3. Cyclic convolution. This is much dependent on the chosen display window as most images and vector fields are not periodic as assumed here.
4. Window function. The values are gradually reduced to zero near the boundary. Some values at the border are lost but otherwise this is the preferred approach in image processing.

4.3 Vector Derivative as Convolution

In image processing, it is well known that the derivative operation is a convolution. The vector derivative ∂ as described in Sect. 3.2 can be discretized using many different approaches. One example are central differences. This is discussed now in relation to convolution and correlation.

Discretizing the derivative using central differences yields

$$\partial \mathbf{f} = \sum_{j=1}^d \mathbf{e}_j \mathbf{f}_{\mathbf{e}_j}(\mathbf{x}) = \sum_{j=1}^d \mathbf{e}_j \frac{\mathbf{f}(\mathbf{x} + s\mathbf{e}_j) - \mathbf{f}(\mathbf{x} - s\mathbf{e}_j)}{2h}.$$

When \mathbf{f} is defined on a uniform 2D grid, it can be written as $\mathbf{f}(x) = \mathbf{f}_{m,n}$ at the grid nodes. Setting $s = 1$ results in

$$\partial \mathbf{f} = \frac{\mathbf{e}_1 \mathbf{f}_{m+1,n} - \mathbf{e}_1 \mathbf{f}_{m-1,n} + \mathbf{e}_2 \mathbf{f}_{m,n+1} - \mathbf{e}_2 \mathbf{f}_{m,n-1}}{2}.$$

Now the masks for the derivative operation using central differences can be computed. They are shown in Fig. 5 for convolution in 2D and 3D and correlation in 2D. As

$$\text{curl } \mathbf{f} = \nabla \wedge \mathbf{f}$$

$$\text{div } \mathbf{f} = < \nabla, \mathbf{f} >,$$

we can extract curl and divergence out of the results of the computation of the derivative using Clifford convolution. We get:

$$\text{curl } \mathbf{f} = < (\nabla * \mathbf{f}) >_2$$

$$\text{div } \mathbf{f} = < (\nabla * \mathbf{f}) >_0$$

Thus, the divergence is the scalar part and the curl the bivector part of the result of the derivative computation. This *curl* operator gives the bivector describing the plane of strongest rotation. The classical *curl* operator gives the corresponding normal vector. The computation of the divergence becomes clear when looking at the central difference derivative masks as they depict divergence of local flow for correlation and convolution, respectively.

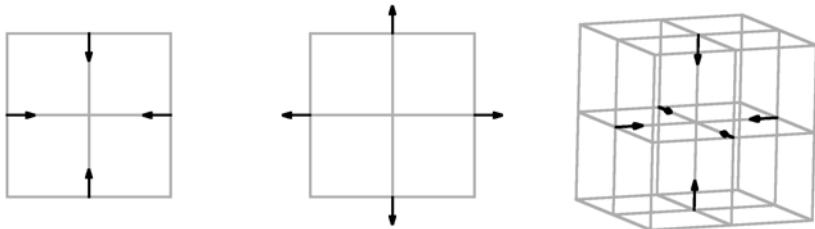


Fig. 5. Central difference derivative masks. (*Left*) mask for 2D convolution; (*middle*) mask for 2D correlation; (*right*) mask for 3D convolution

4.4 Pattern Matching in Vector Fields

In flow visualization, it is important to find vortices as they use a lot of energy. Sometimes, many vortices are desired if for example two gases shall be mixed or if lift of an airplane is to be supported. Sometimes, vortices are not welcome as they slow the flow and put a lot of stress on the surrounding material. Other interesting features are shock waves, separation lines and attachment lines. Regions with divergence and convergence in the flow are of interest, too. Features like these can be described by small filter masks. Thus, they can be found with pattern matching based on Clifford correlation as described here. Some 3D vector filter masks are given in Fig. 2 and 4.

The similarity measure should be independent of the direction of the structure within the vector field and the mask. Otherwise, one has to rotate the filter mask many times and compute the similarities for all the rotated masks. In a last step, it would be necessary to compute the maximum similarity and take the corresponding direction as the direction of the structure. The Clifford correlation gives the direction of the structure in the field directly.

In a first step, the vector field is normalized. As streamlines are everywhere tangent to the vector field, one can regard pattern matching of a normalized vector field as pattern matching of streamlines. The normalization is not necessary for the algorithm itself, it just makes the computation a little easier to understand. If field and mask are not normalized, one has to regard the different vector length and account for them in the computation of the angles. Furthermore, the length of the vectors can be dominant in the correlation and thus pattern matching becomes more difficult.

Clifford correlation gives an approximation of the relative geometric position of the structures in field and mask. Field and mask being normalized, we get:

1. (2D)
 - a) $\langle (\mathbf{h} * \mathbf{f})(\mathbf{x}) \rangle_0 \approx \gamma \cos \alpha_x$
 - b) $\langle (\mathbf{h} * \mathbf{f})(\mathbf{x}) \rangle_2 \approx \gamma \sin \alpha_x$
2. (3D)
 - a) $\langle (\mathbf{h} * \mathbf{f})(\mathbf{x}) \rangle_0 \approx \gamma \cos \alpha_x$
 - b) $\| \langle (\mathbf{h} * \mathbf{f})(\mathbf{x}) \rangle_2 \| \approx \gamma \sin \alpha_x$
 - c) $\langle (\mathbf{h} * \mathbf{f})(\mathbf{x}) \rangle_2$ is the normal vector of the plane of the angle α_x

α_x is the angle between filter mask \mathbf{h} and structure in field \mathbf{f} at point \mathbf{x} . γ is the absolute value of the mask and gained by summing the absolute values of all entries of the mask. Now the direction of the structure can be computed. The mask is then rotated in this direction and one scalar convolution is computed for the similarity. When filter mask and structure are equal, the similarity is 1.

The equations are only approximations. First, sine and cosine of the angle are summed in the convolution instead of the angle itself. The approximation of the angle between the directions of mask and structure gets more imprecise when the angle is bigger. Then, in the correlation the approximations for the angle are summed and thus they can annihilate each other. This is demonstrated in Fig. 6. Thus it is not enough to compute one Clifford convolution for the approximation of the direction of the structure.

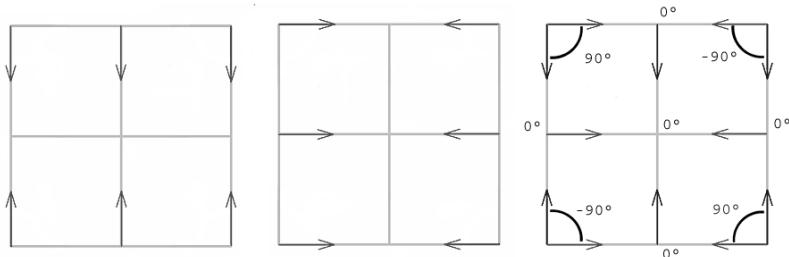


Fig. 6. Correlation of the mask (*left*) and a rotated copy (*middle*) results in a zero multivector as the approximations for the angles annihilate each other (*right*)

So, additional masks with different directions have to be used. We decided to use 3 mask directions in 2D and 6 in 3D in order to get a stable and robust pattern matching algorithm. Further detail of the algorithm can be found in [2].

5 Convolution on Irregular Grids

Most often data sets from flow visualization are defined on irregular grids. The cell sizes differ greatly in size as they are very small in regions of interest and pretty large in regions where the flow is mostly homogeneous. This is illustrated in Fig. 7. The Clifford convolution described so far only works on vector fields defined on uniform grids. Simple regridding of irregular grids results in a high number of grid points and oversampling in most parts of the vector field.

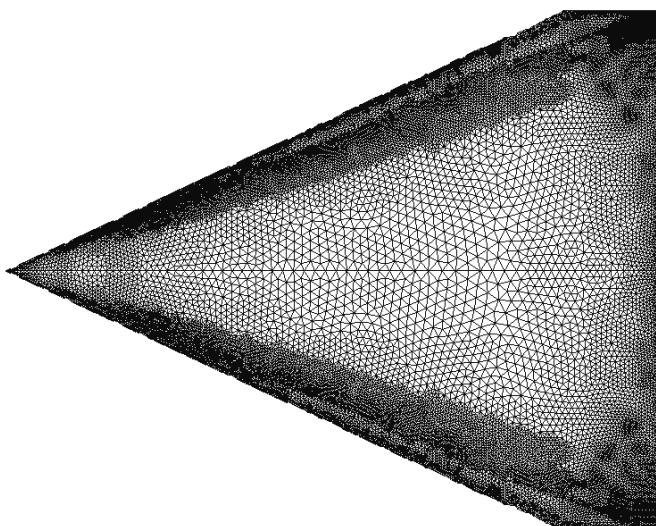


Fig. 7. Grid of the flattened surface of the delta wing

Here we discuss several approaches of local resampling of field and mask in order to extend Clifford convolution and correlation to irregular grids.

5.1 Scaling the Mask

As the cells differ greatly in size, the mask is scaled according to some measure of the cell sizes of the cells surrounding grid point P . Coping with all kinds of different cell types like tetrahedron, prism, cube and hexahedron, we decided to take the longest edge connected to P for the scaling. Let s be the length of this edge. Then the mask defined on a uniform grid is scaled with s . That means that every edge of the mask has length s now. Then the mask can be used for the convolution at P .

The techniques for Clifford convolution on irregular grids based on local resampling of field or mask all use this scaling. Therefore we will not mention it every time. From now on we assume that the mask is scaled properly. Some vector field might require another scaling measure like the shortest edge length or an averaged length. All these measures will have some degenerated fields where they will not work well.

Before we discuss the actual sampling strategies, let's have a look at how this scaling affects the pattern matching. As the mask is scaled differently at grid points with different maximal edge length s , features of different scales are found with the same mask (Fig. 8).

If this is not intended, some multiscale approach has to be applied to the field first. When the cells all have a similar size it assures that the mask is scaled only in a certain range and thus only detects features of this scale. A multiscale approach results in a set of vector fields of different scales. Using this approach means that only a small mask has to be used on the fields to get features of the corresponding scales. This is advantageous as it is not computationally efficient to use large masks. Furthermore, an equal mask size can be chosen once for all positions, accelerating the computation and easing the interpretation. Mostly the scaling poses no disadvantage as the cells are already scaled to the size of the feature that is expected.

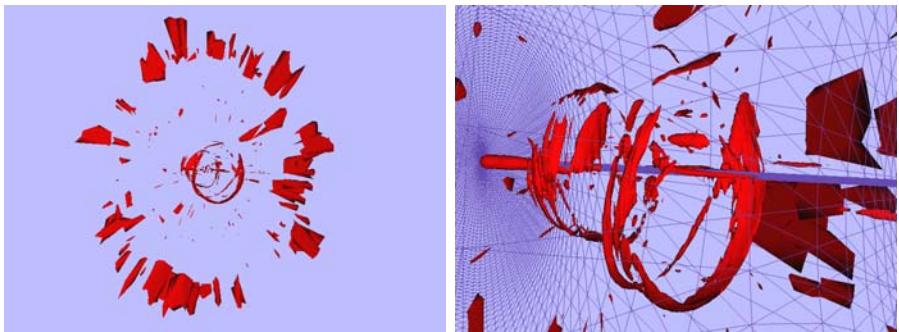


Fig. 8. Swirling jet entering a fluid at rest, 3D simulation. Pattern matching with a $5 \times 5 \times 5$ rotational mask. As the cells differ greatly in size, features of different scales are detected

5.2 Local Resampling

Local Resampling of the Field

The first idea is to “lay” the mask down on the field, the center of the mask aligned with the grid point P to be convolved. Then the field is sampled on those points where the grid points of the mask “lie” on the field. These values are then multiplied with the corresponding values of the mask. This approach is analog for correlation.

Local Resampling of the Mask I

Most of the computational cost of the previous approach comes from point location for the sampling. This leads to the next idea. First, the mask is permuted according to the conversion from convolution to correlation. Then the field is “laid” on the mask. The center of the mask is again aligned with the grid point P to be convolved. The vector field is cut off at the border of the mask. Then the mask is sampled on those points where the grid points of the vector field “lie” on the mask. The sampled values are then multiplied with the values of the vector field that “lie” on the same spot.

This time the computation is even more expensive, as all grid points of the vector field in a bounding box have to be found.

Local Resampling of the Mask II

To avoid this overhead, we tried to use only the points in the n -neighborhood where $(2n + 1)^2$ or $(2n + 1)^3$ is the size of the mask. Again the mask is permuted first. Then, the n -neighborhood of P is computed, that is the set of all grid points which are connected to P by n edges at most. All points of the n -neighborhood are projected on the mask, the mask is sampled there and the sampled values are multiplied with the values of the field.

Another possibility is to use only the 1-neighborhood. This corresponds to convolution of a mask with size 3^2 or 3^3 .

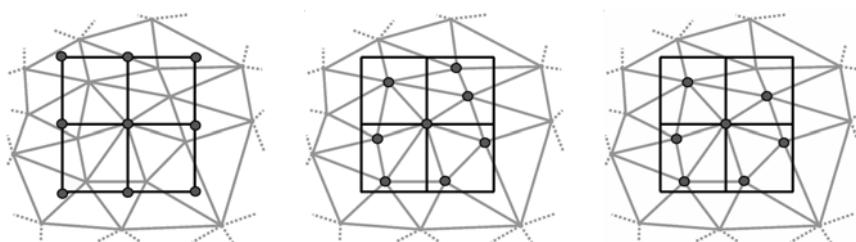


Fig. 9. (Left) Local resampling of the field; (middle) local resampling of the mask; (right) local resampling of the mask, only 1-neighbourhood

5.3 Pattern Matching on Arbitrary Surfaces

The algorithms described so far do not work for arbitrary surfaces. The 2D algorithm assumes a planar grid and the 3D algorithm only works for grids describing volumes. When the surface can be projected onto a planar grid like the delta wing in Fig. 1, the 2D algorithm can be applied. It is also possible to project the surface only locally onto a planar grid, which is a lesser constrain. Another approach is to use geodesics on the surface to determine the resampling locations of the mask. But geodesics are not always distinct and a small error can result in quite a different position. However, regardless of the approach, the mask has to be assumed small enough and the field well behaved. Otherwise, the mask can be distorted so much at different positions that the interpretation of the matching results is lost.

6 Results

We have chosen two test data sets from real applications. The first data set is a turbulent swirling jet entering a fluid at rest (Fig. 8). The simulation considers a cylinder. Since a lot of small and large scale vortices are present in the flow, a discrete numerical simulation (DNS) using a higher order finite difference scheme is used to solve the incompressible Navier-Stokes equations. A planar cut along the axis of the cylinder is used as domain and is discretized by a 124×101 rectilinear grid with smaller rectangles towards the axis of the cylinder. Results of using the different approaches of extending Clifford convolution to irregular grids on this dataset are shown in Fig. 10. This is only an academic example as the grid is regular. The effects of the described scaling method are shown for the original irregular 3D data set in Fig. 8. There, matching based on Clifford convolution with a $5 \times 5 \times 5$ rotational mask (Fig. 4, bottom left) is computed and the results are visualized using marching cubes. The different cell sizes result in the detection of features of different sizes when matching.

The next data set is the delta wing (Fig. 1, 11). This is a vortex break down study. We took the surface of the delta wing and flattened it to get a planar 2D data set. This flattened delta wing is defined on an irregular grid with 25800 grid points and 49898 cells.

On uniform grids, all approaches reduce to the Clifford convolution as defined before. On irregular grids, the approach using the n -neighborhood is the fastest one. Local resampling of the field corresponds to using zero values outside the vector field in the convolution. Using resampling of the mask seems to result in no boundary problem as no values outside the field have to be used. But it corresponds to extending the vector field in an ideal way. Pattern matching thus assumes that the pattern are ideal outside the field and the similarity values near the border are often too high. This can be seen in Fig. 10.

Looking at Fig. 11, the images of the results of resampling the mask seem to be splotchy. This has a couple of reasons. First, the data set has some cells of size zero which distorts the results of the convolution. This is an extreme example of distortion which different cell sizes introduce into the convolution. Using n -neighborhood

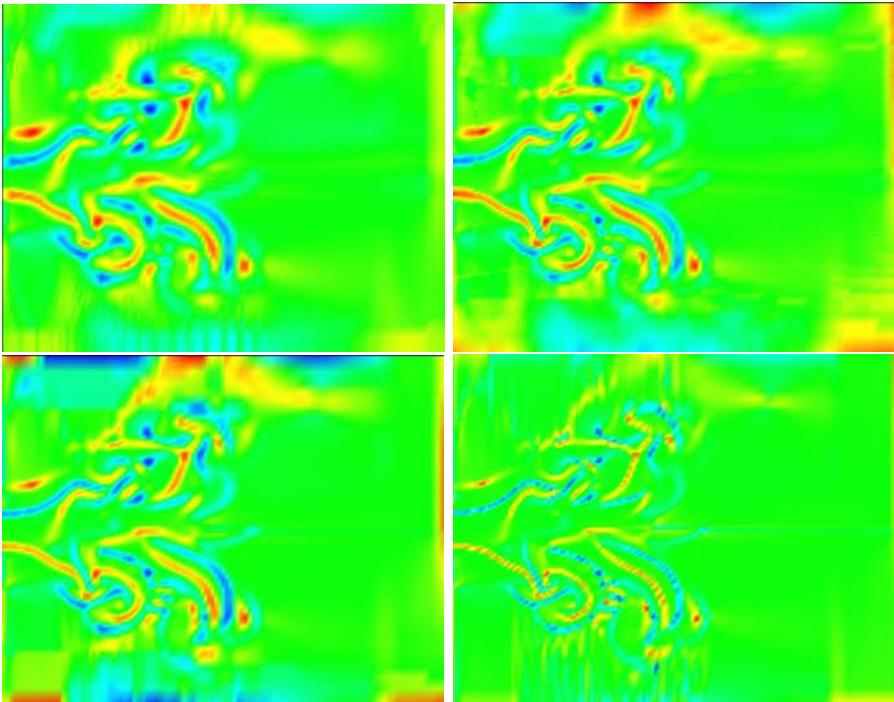


Fig. 10. Pattern matching of a 2D vector field with a 5×5 rotation filter mask. The grid is regular. The similarity values are normalized. Light areas corresponds to the highest similarity and to a righthanded rotation in the field and dark areas to a lefthanded rotation, as a righthanded rotation mask is changed into a lefthanded by multiplication with -1. (*Top left*) local resampling of the field; (*top right*) local resampling of the mask; (*bottom left*) n -neighborhood; (*bottom right*) 1-neighborhood

or 1-neighborhood, the distortions get worse. Cells which have large aspect ratios cause the same problem. Another reason for the differences is the resampling process which is based on interpolation of the grid points. Thus the first two approaches show different results although in the continuous case they would have exactly the same results. Note that using the convergence mask corresponds to computing a derivative. The errors seen in Fig. 11 are typical for a discrete derivative computation.

The approach based on the n -neighborhood is sensitive in terms of noise. We recommend the first approach of local resampling of the field. This is not the fastest but the most robust approach. Further analysis of the resampling strategies should be done in frequency domain with the help of a Fourier transformation on multivector fields. Then, the resampling could also be done in frequency domain.

In Tables 1 and 2, we give the timings of Clifford convolution and pattern matching on two different data sets. The swirling jet is defined on a regular grid, thus local resampling of the field is faster than local resampling of the mask as the point

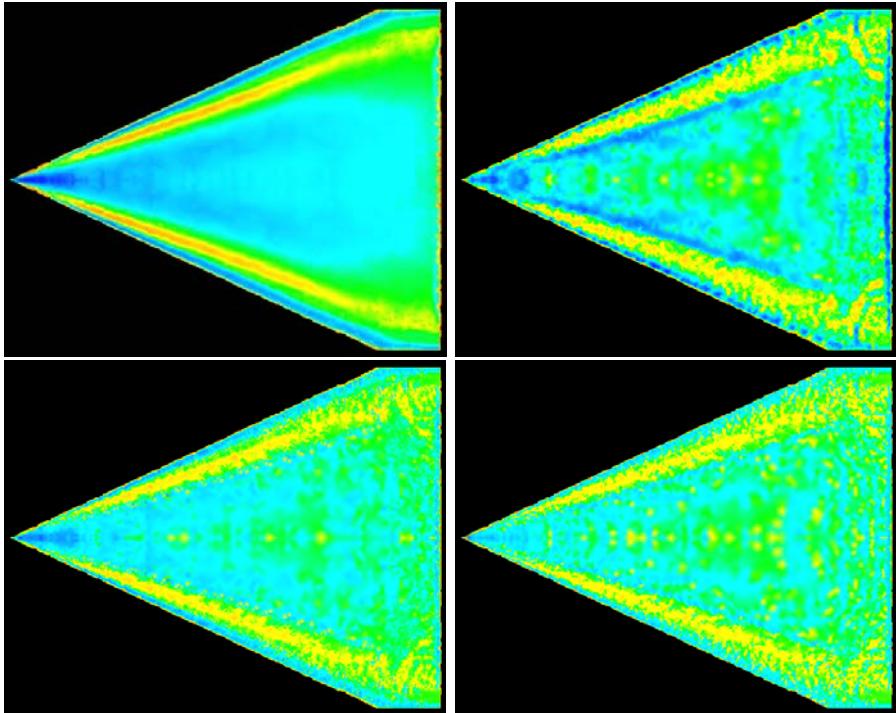


Fig. 11. Regions of convergence (light areas) and divergence (dark areas) on the wing. Adaptive color coding of the results of pattern matching with a 5×5 convergence mask. The grid is irregular. (*Top left*) local resampling of the field; (*top right*) local resampling of the mask; (*bottom left*) n -neighborhood; (*bottom right*) 1-neighborhood

Table 1. Timings for computing one convolution on two data sets on a 1,3 Ghz computer

dataset	size of mask	resampling of field	resampling of mask	n -neigh- borhood
swirl. jet	3×3	4 s	75 s	5 s
(12524 p)	5×5	6 s	314 s	24 s
delta wing	3×3	33 s	88 s	10 s
(25800 p)	5×5	90 s	390 s	40 s

location is not as expensive as on irregular grids. The delta wing is defined on an irregular grid. Both are 2D datasets.

Table 2. Timings for complete pattern matching on the same data sets as in Table 1

dataset	size of mask	resampling of field	resampling of mask	n -neighborhood
swirl. jet	3×3	8 s	88 s	10 s
(12524 p)	5×5	16 s	352 s	40 s
delta wing	3×3	65 s	103 s	21 s
(25800 p)	5×5	183 s	441 s	66 s

7 Conclusion and Future Work

We have presented several approaches for local resampling of field or mask in order to extend Clifford convolution and pattern matching to datasets defined on irregular grids. The approach based on local resampling of the field is robust and works well.

The other approaches of accelerating the computation do not work well on irregular grids. Especially the results of the fastest algorithm resampling the mask at points of the field in a n -neighborhood were quite distorted. This approach was sensitive to noise, too. Further analysis will need the definition of a Fourier transformation on multivector fields. There, new strategies of extending the convolution may become obvious, too.

Future work will therefore include the definition of a Fourier transformation on multivector fields. Then, the local resampling will be evaluated again in frequency domain. Approaches for resampling with the help of a Fourier transformation will be discussed, too.

Scale spaces and multiresolution pyramids will be investigated in more detail with regard to convolution and pattern matching operations. There, the definition of a Fourier transformation would also be an asset. The acceleration of the presented method is clearly another topic for further work, too.

Acknowledgments

First of all, we like to thank the members of the FAnToM development team at University of Kaiserslautern, especially Tom Bobach, Christoph Garth, David Gruys, Kai Hergenröther, Max Langbein and Xavier Tricoche, for their help with programming and production of the pictures. We further acknowledge the fruitful discussions in the whole computer graphics group at Kaiserslautern. We thank Prof. Kollmann, MAE department, University of California at Davis, for producing the swirling jet data set. Further thanks go to Markus Rütten, DLR Göttingen, for providing the delta wing data set.

References

1. Cabral, B., Leedom, L.C.: Imaging Vector Fields Using Line Integral Convolution. Proceedings of SIGGRAPH '93. New York, 263–270 (1993).

2. Ebling, J., Scheuermann, G.: Clifford Convolution And Pattern Matching On Vector Fields. Proceedings of IEEE Visualization '03. IEEE Computer Society, Los Alamitos, CA, 193–200 (2003).
3. Granlund G.H., Knutsson, H.: Signal Processing For Computer Vision. Kluwer Academic Publishers, Dordrecht, The Netherlands (1995)
4. Heiberg, E.B., Ebbers, T., Wigstroem, L., Karlsson, M.: Three Dimensional Flow Characterization Using Vector Pattern Matching. IEEE Transactions on Visualization and Computer Graphics, **9 (3)**, IEEE Computer Society Press, Los Alamitos CA, 313–319 (2003)
5. Helman, J.L., Hesslink, L.: Visualizing vector field topology in fluid flows. IEEE Computer Graphics and Applications, **11 (3)**, 36–46 (1991)
6. Hestenes, D.: New Foundations For Classical Mechanics. Kluwer Academic Publishers, Dordrecht, The Netherlands (1986)
7. Hestenes, D.: Clifford Algebra to Geometric Calculus. Kluwer Academic Publishers, Dordrecht, The Netherlands (1997)
8. Jain, A.K.: Fundamentals Of Digital Image Processing. Prentice Hall, Englewoods Cliffs, NJ, USA (1989)
9. Jaehne, B.: Digital Image Processing. Springer Verlag, Berlin, Germany (2002)
10. Kenwright, D.N., Henze, C., Levit, C.: Feature Extraction Of Separation And Attachment Lines. IEEE Transactions on Visualization and Computer Graphics, **5 (2)**, IEEE Computer Society Press, Los Alamitos CA, 151–158 (1999)
11. Pagendarm, H.G., Walter, B.: Feature Detection from Vector Quantities in a Numerically Simulated Hypersonic Flow Field in Combination with Experimental Flow Visualization. Proceedings of IEEE Visualization '94. IEEE Computer Society Press, Los Alamitos, CA, 117–123 (1994)
12. Post, F.H., Vrolijk, B., Hauser, H., Laramee, R.S., Doleisch, H.: Feature Extraction And Visualization Of Flow Fields. Eurographics 2002 State of the Art Reports, The Eurographics Association, Saarbrücken, Germany, 69–100 (2002)
13. Roth, M.: Automatic Extraction Of Vortex Core Lines And Other Line Type Features For Scientific Visualization. PhD Thesis, ETH, Hartung-Gorre Verlag Konstanz (2000)
14. Scheuermann, G., Hagen, H., Krueger, H., Menzel, M., Rockwood, A.: Visualization of Higher Order Singularities in Vector Fields. Proceedings of IEEE Visualization, IEEE Computer Society Press, Los Alamitos, CA, 67–74 (1997)
15. Scheuermann, G.: Topological Vector Field Visualization With Clifford Algebra. PhD Thesis, University of Kaiserslautern, Germany (1999)
16. Tricoche, X., Scheuermann, G. Hagen, H.: A Topology Simplification Method For 2D Vector Fields. Proceedings of IEEE Visualization 2000. IEEE Computer Society Press, Los Alamitos CA, 359–366 (2000)
17. Van Walsum, T. Post, F.H., Silver, D., Post, F.J.: Feature Extraction And Iconic Visualization. IEEE Transactions on Visualization and Computer Graphics, **2 (2)**. IEEE Computer Society Press, Los Alamitos CA, 151–158 (1996)

Fast and Robust Extraction of Separation Line Features

Xavier Tricoche¹, Christoph Garth², and Gerik Scheuermann³

¹ Scientific Computing and Imaging Institute, University of Utah
tricoche@sci.utah.edu

² Department of Computer Science, University of Kaiserslautern
garth@rhrk.uni-kl.de

³ Institute of Computer Science, University of Leipzig
scheuermann@informatik.uni-leipzig.de

The visualization of a three-dimensional viscous flow around an embedded object is typically based on the analysis of its wall shear stress. This vector field defined over the object body exhibits structures that are key to the qualitative evaluation of the surrounding flow. Open separation and attachment lines are of essential interest in aerodynamics due to their adverse effects on the object motion and their implication in vortex genesis. The paper presents a new method for the efficient analysis and visualization of separation and attachment lines on polyhedral surfaces in three-space. It combines local prediction and global feature extraction to yield a scheme that is both efficient and accurate. In particular, it does not suffer from the restrictions induced by assumptions of local linearity and is able to detect features where existing techniques fail. The algorithm is built upon an efficient streamline integration scheme on polyhedral surfaces. The latter is also employed to develop a variation of the LIC scheme. Results are proposed on CFD data sets that demonstrate the ability of the new technique to precisely identify and depict interesting structures in practical applications.

1 Introduction

Modern numerical simulations in Computational Fluid Dynamics (CFD) generate large scale datasets that must undergo qualitative and quantitative evaluation for interpretation. Typically, the analysis relies on the extraction and identification of structures of interest that are used to gain insight into essential properties of the flow for the considered application. In the field of aircraft design in particular, huge amounts of flow data are computed and processed to better understand the properties of design prototypes or to look for optimal configurations, especially during critical flight situations. The usual approach to this analysis is to study the interaction between the three-dimensional air flow around the body and the so-called shear stress vector

field. The latter is tangential to the surface and induces the oil-flow patterns traditionally observed during wind tunnel experiments. Separation and attachment lines are features of key interest in this context. They correspond to one-dimensional loci where the flow leaves or converges toward the body. For aeronautical design this phenomenon is accompanied by adverse effects on lift and drag behavior, in particular during takeoff and landing phases. In automotive engineering flow separation results in a drop in pressure that has negative impact on driving stability. More generally, separation and attachment lines are essential structural features involved in flow partition and vortex genesis. Their automatic extraction and depiction is therefore an important and challenging task for scientific visualization.

In general, the intrinsic limitation of most feature extraction methods is their attempt to extract global structures by means of local analysis. This approach is induced by the need to efficiently address the visualization of very large datasets. Hence, the analysis is aimed at identifying a similarity with some predefined model of the structure of interest. The principal contribution so far to the visualization of separation and attachment lines is the work by Kenwright et al. [5, 6]. Based on considerations inspired by the study of linear vector fields, these authors came up with a simple criterion for local feature identification. Unfortunately, their simple method shows several strong shortcomings, especially in the processing of CFD data sets defined over unstructured grids.

The paper presents a new method for the efficient extraction and visualization of separation and attachment lines in two-dimensional flows defined over arbitrary surfaces in 3D space. The basic idea behind this scheme is to combine local flow probes and global structural information to drive feature search and obtain accurate results fast, even for very large datasets. As a matter of fact, the detection of global features requires the analysis to take global information into account. Since starting a dense set of streamlines over the whole surface to observe and measure their convergence would require a huge computational effort, it is infeasible on typical datasets. However, streamlines are the most natural way to characterize separation and attachment lines. As these lack a formal definition, streamlines are constitutive elements of their empirical characterization. In practice one monitors the flow convergence (resp. divergence) within regions of interest. Concerning our implementation, these regions are characterized by large values of the point-wise divergence operator and are then abstracted to a skeleton of one-dimensional edges by ridge and valley line extraction. The lines obtained can then serve as start positions for streamline integration. To make streamline computation efficient on polygonal surfaces we consider cell-wise constant vector values, resulting in a stable integration scheme that emphasizes attachment and separation behavior.

The paper is organized as follows. Related work is presented in Sect. 2. The technique used for fast streamline integration is introduced in Sect. 3. In particular we discuss integration through so-called singular edges. Additionally, we consider the application of our scheme for fast LIC computation over simplicial surfaces. As mentioned previously, our local feature predictor is the point-wise value of the divergence operator. Its computation is explained in Sect. 4. Section 5 describes the simple though robust algorithm we use to extract ridge and valley lines from the

resulting scalar field. This provides the starting locations required for streamline integration which permits to monitor flow convergence as shown in Sect. 6. Finally, we show some results on two CFD data sets from aerodynamics and comment on the application of our technique.

2 Related Work

In flow visualization, the extraction and visualization of line type features has received much attention in recent years. Besides vortex cores (see [13] for a bibliography), researchers have tried to detect and show separation and attachment lines on bodies immersed in three-dimensional flow. Kenwright [5] made the first major contribution in this area. He proposed a simple and fast method that is suitable for large data sets. A triangular grid and a linear field in each triangle are assumed. The basic idea is that separation and attachment lines can be found in two linear patterns, namely saddle points and proper nodes, where they are aligned with an eigenvector of the Jacobian. Therefore, his method works cell-wise and looks in the corresponding piece-wise linear vector field for the intersection of such lines with the grid cells. The discontinuity of the Jacobian results in disconnected line segments in general. However, inspired by the *Parallel Operator* of Peikert and Roth [10], Kenwright proposed a modified version of his algorithm [6]. It is based on the point-wise evaluation of streamline curvature at the grid vertices. It follows that the extraction of separation and attachment lines reduces to the computation of zero-isolines of the curvature field. As a result one usually obtains connected segments. They must be filtered in a post-processing step to discard false positives [13]. Moreover, dependence on the point-wise computation of the Jacobian introduces a problematic high sensitivity to noise, especially in the case of unstructured data sets. Another approach was used by Okada and Kao [9]. They improve on the classical *Line Integral Convolution technique* (LIC) [1, 2] by first applying a second LIC iteration to sharpen the paths of individual streamlines, and then using histogram equalization to increase contrast. Additionally, they color-code the flow direction which they use to highlight the converging/diverging behavior observed along separation and attachment lines. This method is computationally intensive due to the required LIC processing. Furthermore it does not provide the exact geometry of the feature lines but rather puts emphasis on regions where they are likely to be found. Nevertheless, our method shows in some extent similarities to the ideas used by these authors.

Another aspect directly related to our method is the computation of streamlines constrained to the surface of an object in three-space. This is a classical problem in visualization and many approaches can be found in the literature. Globus et al. [4] mention a simple scheme to keep the streamlines close to the wall along their path. This is done by starting streamlines close to the wall and re-projecting the successive streamline points obtained by integration in 3D onto the object. Max et al. [7] use an Euler method with projection to integrate streamlines on implicit surfaces. Since the surfaces are defined by an implicit function, they can use the function's gradient to define the surface for the projection. Forssell [3] applies LIC to curvilinear

surfaces and gives a corresponding integration scheme. For this, she uses the global parameterization of curvilinear surfaces for her calculation. Battke et al. [1] carry out integration directly on arbitrary surfaces by combining Runge-Kutta with adaptive step-size and linear extrapolation over each triangular cell. Nielson and Jung [8] proposed a computational framework for streamline integration over simplicial grids. Their work is based on the existence of a closed formula for streamlines in linear vector fields. Applied on a cell-wise basis, this permits an exact computation in each cell, reconnected to curves over the whole grid. Unfortunately this technique is quite slow for very large grids. Another mathematical treatment of streamlines on simplicial surfaces is given by Polthier and Schmies [12]. Their method is based on geodesics in accordance with concepts from differential geometry, leading to an adaptation of various numerical integration schemes of varying order for smooth surfaces. We use this technique in the course of our method to obtain smooth and accurate feature lines. Nevertheless, for efficiency reason we adopt an alternative streamline computation scheme to monitor flow convergence as described next.

3 Wall Streamlines over a Simplicial Surface

Before discussing the integration scheme used in our implementation, we briefly introduce the shear stress vector field that is the basic setting in further computations.

3.1 Shear Stress Vector Field

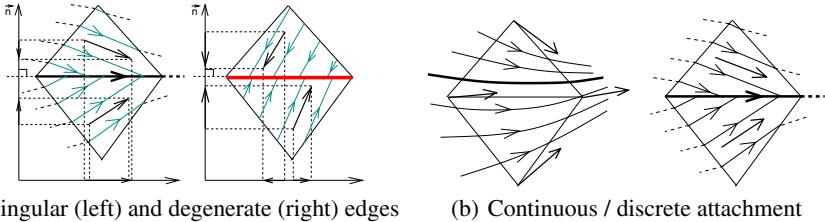
We are concerned with three-dimensional flows that have so-called no-slip boundary condition. This condition is encountered in CFD simulations of viscous flows. It forces the velocity to zero as the body of an embedded object is approached along the surface normal. Therefore, flow analysis around the object deals with the structure of its *shear stress* field [6]. It is a tangential vector field defined over the surface and corresponds to the derivative of the velocity vector field normal to the surface: $\mathbf{v} = J\mathbf{n}$, where J is the 3×3 Jacobian matrix of the velocity field and \mathbf{n} is the local surface normal. Hence it describes how the three-dimensional flow behaves close to the body. Streamlines in the shear stress field are called *wall streamlines*. This definition implies that the integration of wall streamlines takes place in the tangent bundle of the surface. In practice, object boundaries are defined as polygonal surfaces. They are not smooth manifolds since the tangent plane is piecewise constant in each cell but discontinuities occur across edges. Therefore, efficient and reliable techniques to handle numerical streamline integration are needed in this case. In the following we do not assume a particular structure or global parameterization for the surface. Consequently, integration cannot be carried out in a two-dimensional computational space and the results mapped back onto the surface. All things considered the problem to solve is that of streamline integration directly on the surface regardless of its embedding. Furthermore we require computation to be fast enough to be efficiently included in the method presented in Sect. 6. These requirements motivate the scheme discussed next.

3.2 Streamline Integration in Piecewise Constant Vector Fields

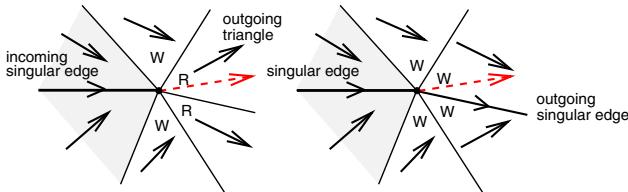
Practically, we transform the original shear stress field with vertex-based 3D vector values into a cell-wise constant vector field in which the vectors lie in the tangential plane of the corresponding cells (i.e. resampling to the dual grid and projection into the tangent planes). This choice of procedure comes along with several desirable properties for our purpose. First, we are no longer concerned with the problem induced by the tangent plane indeterminacy at each grid vertex: point-wise 3D vector values lead to different 2D projections onto the tangent planes of incident cells, whereas the tangent plane is well defined for any particular cell. Second, a first-order Euler integration provides the exact solution of the corresponding differential equation. Although a loss of accuracy seems inevitable, the resolution of typical grids provided by CFD simulations allows us to satisfactorily approximate the real path of streamlines in this way. Remark that potential integration instability caused by flow divergence is properly handled in our technique as explained in Sect. 6. The usual requirements on smoothness of a vector field to ensure existence and uniqueness of integral curves are not fulfilled here but separation and attachment patterns extend in this particular setting, as we show next. Moreover, since the path of streamlines is independent of the norm of the underlying vector field (e.g. normalizing a vector field corresponds to a re-parameterization of streamlines by arc length) we normalize the cell-wise vectors for numerical stability concerns in further processing. Now, the integral curve in each cell is a straight line segment connecting two edges. Hence, integrating a curve over the surface corresponds to a cell-wise line clipping, directed by the corresponding vector value. This way of computing streamline was already used in the original LIC technique [2] where the vector information is constant over each rectangular pixel. We consider here a more general problem since we process arbitrary triangles. Nevertheless, provided the connectivity information of the grid, the implementation can be made very efficient.

3.3 The Role of Singular Edges

Since streamlines remain parallel inside each cell, all structural properties of the flow are observed either on the edges or vertices of the grid. We call an edge *singular* if the vector values associated with the cells lying on both sides have opposite normal components with respect to the edge. This means that streamlines reach the edge from both sides but cannot pass through it. If, furthermore, the vectors' components parallel to the edge have opposite directions, no consistent orientation can be decided for further integration and the edge is called *degenerate*. It plays the role of a 1D-singularity since streamlines end there. Refer to Fig. 1(a). If the directions parallel to the edge are consistent, integration can proceed along the edge. This quality of cell-based streamlines is illustrated in comparison to the equivalent situation in the continuous case. The edge corresponds to a contraction (resp. dilation) of the flow (cf. Fig. 1(b)). This property enables streamline integration over piecewise constant vector fields to characterize both separation and attachment in particular cases. As a matter of fact, convergence and divergence are no longer restricted to asymptotic

**Fig. 1.** Discrete vector field and singular edges

behavior, but correspond also to the “interception” of streamlines by singular edges, leading in further integration to a one-dimensional flow of all intercepted streamlines from the vertex reached. Of course, the occurrence of this configuration depends on the relative orientation of flow and grid edges. Nevertheless, this interesting attribute plays an important role in the method presented in Sect. 6. Now, once a vertex has been reached along a singular edge, integration must proceed from this position in one of the triangles in its one-neighborhood. Obviously, there is a direction indeterminacy. We first exclude both triangles sharing the considered singular edge. Now, each of the remaining triangles incident to the vertex is a potential candidate to proceed if its vector value lies within the angular domain bounded by both of its edges incident to the vertex, see left configuration in Fig. 2. Both triangles marked *W* are

**Fig. 2.** Streamline integration through a vertex

discarded because their vector values lie outside the angular domain as opposed to the triangles marked “*R*” that allow for further integration. Practically, we solve the remaining indeterminacy problem by always taking in first place the sector pointed by the vector value originally assigned to the vertex (dashed arrow in Fig. 2). If this sector does not satisfy the angle criterion, we select one of the satisfying neighbors closest to this direction. If no such neighbor exists, we have either reached a singular vertex (in which case integration is terminated) or integration has to proceed along an additional singular edge which is, as well, selected closest to current direction. Refer to right configuration in Fig. 2.

3.4 An Alternative Approach to Line Integral Convolution on Simplicial Surfaces

The streamline integrator on simplicial surfaces derived above is straightforward and computationally cheap. It can thus serve as a building block for the adaption of streamline-based visualization schemes that are usually applied to two-dimensional planar fields. As an example, we consider Line Integral Convolution. Since this technique requires the computation of a great number of streamlines, a fast scheme for streamline integration is mandatory.

In the following, we describe an application of the basic LIC idea [2] to the wall shear stress vector fields, with some modifications. In the common variants of LIC algorithms that deal with non-planar surface grids, a texture is mapped onto the grid to achieve a resolution high enough to recreate the visual impression of oil droplet smearing. In most cases, the grid resolution is not sufficient to convey this effect. However, texture mapping on surface grids is only simple in the case of curvilinear grids. There have been approaches that divide arbitrary surfaces into rectangular parameter domains, e.g. [14], but they are tedious and do not work in all cases. Furthermore, computational effort is large.

We propose a simple yet effective adaption of the original LIC algorithm in the form of three modifications:

- We abandon the idea of mapping a texture on the grid and instead employ a cell-based scalar field over the original grid that carries the scalar values used for convolution.
- Convolution is carried out directly on this field using our streamline integration scheme. Since the visited cells are naturally obtained from the algorithm, it is straightforward to implement the convolution. Furthermore, arbitrary grids are tractable.
- By subsequent subdivision of the cells the grid is refined until the visual representation of the triangles is small enough to give a sufficient resolution. By imposing an upper bound on the area of triangles, the subdivision is adaptive and results in nearly uniform resolution over the whole surface. This constraint can be chosen in such a way that individual triangles encompass only a small number of pixels (ideally one pixel), so that the rendering of the refined scalar field over the surface results in a LIC-like image.

As usual, the initial scalar field is seeded with white noise. The common improvements (e.g. the Festals technique [15]) are easily applied in this context. Although it might seem a disadvantage to create surface triangulations of large size, in practice the size of individual triangles is limited by the fact that the maximum resolution required is somewhat lower than the screen resolution of a typical monitor. Thus the number of triangles is on the order of 10^6 , a number well within reach of any algorithm and modern graphics hardware. Remark that streamline integration is not slowed down much by the increased grid size, since no cell location is performed and connectivity information is used instead.

4 Local Predictor

As mentioned previously, the basic idea behind our method is to build feature line extraction on top of a convergence monitoring of the flow. Now, for such an approach to be feasible at all, we need a way to restrict computation to regions of interest, i.e. those regions that lie close to the separation and attachment lines contained in the data set. Hence we need a reliable local predictor that indicates where streamlines might show converging behavior and decreases complexity by several orders of magnitude. More precisely, by predictor we imply a scalar field defined over the whole domain that indicates (either for separation or attachment) which regions are most likely to lie close to line features.

In fact, implicit in the definition of separation or attachment behavior is the notion of contraction and dilation of the flow that occurs normal to the flow direction. This effect is responsible for the convergence of neighboring streamlines. A standard operator to measure flow contraction (resp. dilation) is the divergence. For continuous vector fields, it is defined at each position P as the amount of flow generated in an infinitesimal region around P . In a Cartesian basis of the plane, it is given by the expression

$$(\text{div } \mathbf{v})(P) = \frac{\partial}{\partial x} v_x + \frac{\partial}{\partial y} v_y$$

To express divergence in the neighborhood of each vertex of a simplicial surface S , the local geometry around the point must be taken into account. In our implementation we use the formula proposed by Polthier et al. [11] that is expressed as follows for a given position p_i :

$$(\text{div}_S \mathbf{v})(p_i) = \frac{1}{2} \sum_{e_j \in \partial^*(p_i)} \int_{e_j} < \mathbf{v}, \mathbf{n}_j > ds,$$

where $\partial^*(p_i)$ is the oriented set of edges e_j opposite to p_i in its incident triangles, and \mathbf{n}_j is the outward pointing normal of edge e_j . Since vector values are provided cell-wise, the computation is straightforward.

5 Ridge and Valley Lines Extraction

Divergence computation results in a scalar distribution over the grid. We saw previously that these values are related to the converging (resp. diverging) behavior of the flow. Now we need to deduce from this scalar field which regions are most interesting for further processing. Since we want to lower the complexity of our convergence monitoring, we choose to extract the so-called ridge and valley lines and to focus on them in the following.

For a scalar field interpreted as a height field, a ridge or valley line is defined as the set of points where the slope is locally minimal compared to points of the same elevation. Ridge and valley line extraction is a classical task in image processing. They are interpreted as edges in a scalar picture. The existing method in that context

are however of little help for our problem since they typically assume structured grids and require first and second order derivative computation [13]. This cannot be achieved numerically in a satisfactory way on a scalar field obtained itself by local estimation of a derivative. For this reason, we adopted an alternative, much easier approach that is fast and gives satisfying results.

We reformulate the definition of ridge and valley lines as follows. Ridge (resp. valley) lines are curves through the domain of definition of a scalar field. They start at local maxima (resp. minima) and minimize descent (resp. ascent) along the curve. In our discrete setting over triangular grids, the corresponding algorithm starts at vertices corresponding to local maxima (resp. minima) and proceeds the ridge (resp. valley) line extraction towards the direct neighbor with maximum (resp. minimum) value. The resulting line connects vertices via the edge segments of the given triangulation.

Since the data at hand is typically noisy (see above), some improvements are necessary to use this method in practice.

- First, we want to restrict ridge line extraction to major features and discard minor ones, hence we must not take into account local extrema due to high frequency oscillations. In practice, we restrict the starting points of line extraction to vertices that are extrema within a large neighborhood surrounding them.
- Second, we want the extracted feature lines to be as straight as possible and to avoid u-turns and self-intersections. Therefore we impose an angle criterion for the acceptance of new segments, given by the mean direction followed during the last few steps. Moreover, we exclude from further processing every vertex that is a direct neighbor of a vertex selected previously.

Modified in this way, the algorithm is very fast, straightforward to implement and robust to noise.

6 Accumulation Monitoring

Once regions of interest have been determined, streamlines are started there. Our scheme then monitors 1D flow convergence resp. divergence through streamlet integration. What is meant here corresponds to the usual empirical characterization of separation and attachment lines as asymptotic limits of streamline accumulation represented by the paths of limit streamlines. Our method consists of two successive steps, as described next.

Cell-wise Accumulation

Practically, we assume that the ridge and valley lines of the divergence provide a coarse approximation of the actual feature lines. Hence, a correction that provides the actual line position is needed. Motivated by the previous remark we choose to measure the flow convergence from the ridge and valley lines on a cell-wise basis.

Observe that ridge lines (positive values of the divergence) are associated with backward convergence (attachment) whereas valley lines (negative values) are related to forward convergence (separation). Practically a cell-wise scalar field accounts in each cell for the number of streamlines that were integrated through it. Each hit corresponds to a '+1' or '-1' value, depending on the sign of the divergence at the starting position. Since we expect the starting locations to lie close to the search region, we only integrate streamlines along a short arc length to highlight the converging behavior. The cell-wise scalar field obtained is converted to a point-wise field using a basic mean value computation.

Feature Line Extraction

A simple idea to obtain the feature lines from the resulting accumulation scalar field would be to apply again the algorithm for ridge and valley line extraction previously discussed in Sect. 5. However, this choice of procedure has two shortcomings. The first one is that the lines obtained in that way provide no guarantee to follow the flow direction as required by our definition. The second one is a direct consequence of our definition of ridge and valley lines: they are constrained to follow the edges of the triangulation which is a coarse approximation of a streamline.

Flow-driven Ridge and Valley Lines

We solve the first problem with a slight modification of the scheme of Sect. 5. We obtain a flow-driven ridge/valley line extraction by processing as follows. Starting at local maxima of the point-wise accumulation field we iteratively move along the grid edges by taking both the values of the 1-neighbors (like before) and the local flow direction into account. The principle is explained in Fig. 3. To obtain a balance

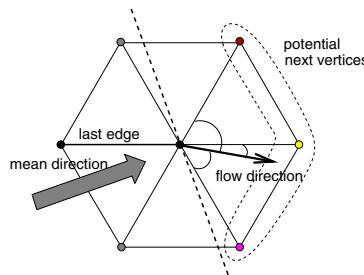


Fig. 3. Flow-driven ridge/valley line extraction

of both scalar value and flow parallelism we perform a simple angular weighting by multiplying the scalar values defined at the vertices of the 1-neighborhood by the cosine of the angles between corresponding edges and local flow direction. Remark that the mean direction of the last few steps is still used to discard vertices inducing a u-turn as shown by the dashed line.

Final Extraction

Given the polyline description of the flow-driven ridge and valley lines, we obtain the actual path of separation and attachment lines by integrating streamlines starting at the upstream (separation) resp. downstream (attachment) end of the ridge and valley lines, and directed toward the converging direction. Integration is terminated when the streamline reaches the vicinity of the other end. This is valid since, according to previous processing, ridge and valley lines are expected to lie at most one cell away from the actual feature line position. Remark that numerical integration in this case is carried out using the geodesic-based technique of Polthier and Schmies [12]. In that way final results are both smooth and very accurate, due to the underlying fourth-order Runge-Kutta scheme with adaptive step size control.

7 Results

To demonstrate the ability of our method to properly extract separation and attachment lines for practical applications we consider in the following two CFD data sets. In both cases we propose a comparison of our results with those obtained using Kenwright's method.

7.1 Delta Wing

The first data set is a steady simulation of airflow around a delta wing at 25 degrees angle of attack. The grid consists of 1.9 million unstructured points forming 6.3 million unstructured elements, 3.9 million tetrahedra and 2.4 million prisms. The delta wing itself is made up of about 81k triangles. We focus on the shear stress vector field defined over the wing that we compute according to the formula mentioned in Sect. 3. Applying Kenwright's method, we get the results shown in Fig. 8, left picture. Observe that a strong pre-smoothing step was necessary to permit a satisfying Jacobian computation. However the results have poor quality due to disconnected segments, shifted features and numerous false positives. The successive steps of our method are shown in Fig. 4. The upper left picture illustrates ridge and valley line extraction from the divergence scalar field. It can be seen that the local divergence computation leads to noisy values. This induces zigzag paths for the lines obtained. However their global aspect provides a satisfying approximation of the features' position as shown in the upper right picture: streamline integration is carried out (either forward or backward) starting along ridge and valley lines until convergence is reached. The resulting cell-wise scalar field accounts for streamline accumulation and is next submitted to flow-driven ridge line extraction, see lower left picture. Smooth streamline integration along the corresponding ridge and valley lines finally gives the exact position of separation and attachment lines. These results are shown again in Fig. 6 together with a LIC texture of the shear stress computed with the technique presented in Sect. 3.4.

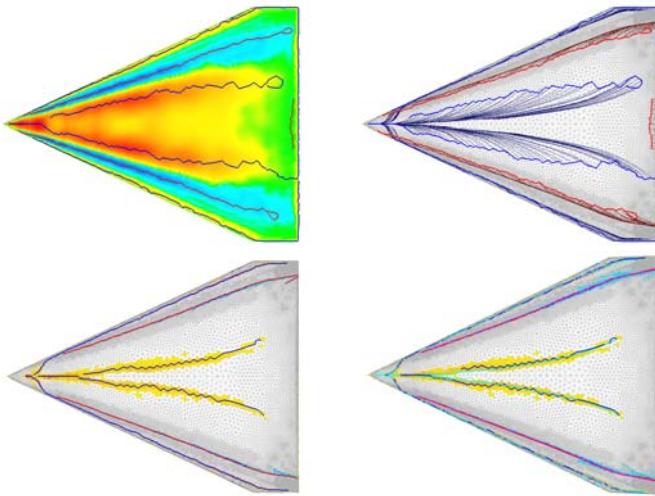


Fig. 4. Delta wing dataset. *Upper left*: Colormap of flow divergence and corresponding ridge lines. *Upper right*: Divergence ridge lines and streamlines started from ridge line points. *Lower left*: Colormap of streamline accumulation scalar field (only cells with hits drawn) and corresponding flow driven ridge lines. *Lower right*: Streamline accumulation and flow driven ridge lines together with final results

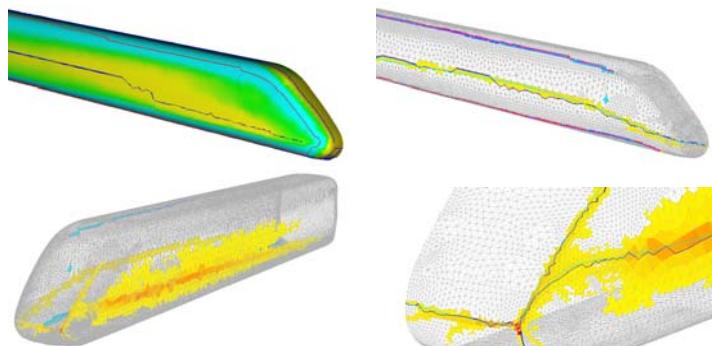


Fig. 5. ICE train dataset. *Upper left*: Colormap of flow divergence and corresponding ridge lines *Upper right*: Colormap of streamline accumulation (only cells with hits drawn), flow driven ridge lines. *Lower left*: Colormap of streamline accumulation. Note the widely spread accumulation on the train side indicating slow convergence and hence a weak feature. *Lower right*: Zoom of train nose, streamline accumulation, flow driven ridge lines and resulting feature lines

7.2 High Speed Train

The second data set corresponds to a single time step of an unsteady simulation of the German train ICE. In this case, the train travels at a velocity of about 250 km/h with wind blowing from the side at an angle of 30 degrees. The wind causes vortices to form on the lee side of the train, creating a drop in pressure that has adverse effects on the train's track holding. The original grid consists of 2.6 million elements. We restrict our considerations to the front wagon that contains 53k triangles. For comparison, the results of Kenwright's method are shown in Fig. 8, right picture. Previous remarks related to pre-smoothing apply here too. In this case, the results are even worse than for the delta wing. We obtain a lot of disconnected segments and most of them would have been filtered out by a simple check on flow parallelism. Moreover the feature line lying on the nose of the train is significantly shifted toward the middle. Again, the successive steps of our method can be seen in Fig. 5. The main difference with previous data set is the presence of weak features on both sides of the wagon that correspond to slow flow divergence. Practically we obtain for these features ridge and valley lines of the divergence that lie fairly far away from their actual position. This implies that the correction step associated with flow monitoring must follow streamlines along a longer path to detect the converging behavior. Another consequence is the fuzzy resulting accumulation scalar field around the features. Nevertheless, our flow driven ridge line extraction is able to properly track their path as illustrated in the upper and lower right pictures. Final results are shown along with a LIC texture in Fig. 7.

8 Conclusion

We have presented a new method for efficient and robust extraction of separation and attachment lines on arbitrary simplicial surfaces embedded in three-dimensional

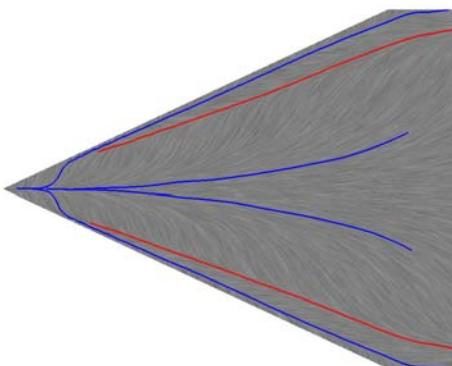


Fig. 6. Resulting features on the delta wing over LIC texture

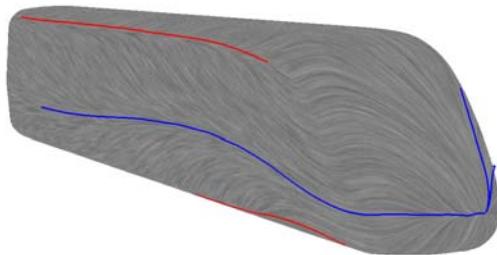


Fig. 7. Resulting features on the ICE train over LIC texture

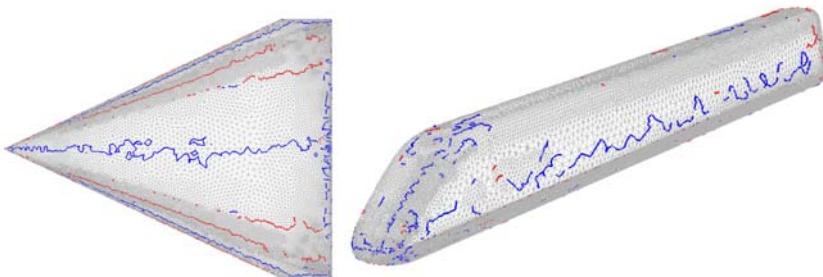


Fig. 8. Results of Kenwright's method, delta wing (left) and ICE train (right).

space. Our approach combines a local predictor and a global correction step. We use fast streamline integration to efficiently monitor flow convergence resp. divergence. The original point-wise vector field is transformed into a cell-wise one which both speeds up computation and permits to characterize separation and attachment behavior locally. We show how to use this technique to produce fast LIC textures on arbitrary surfaces. Applied to two realistic CFD data sets, our new method proved able to precisely detect interesting features, even in cases where flow convergence occurs weakly, on large scales. Such structures are missed by existing techniques, as shown by our comparison with Kenwright's standard scheme, which has difficulties on unstructured grids. This is most likely due to the more difficult Jacobian computation in physical space. Apparently this limitation was not foreseen in the original presentation of the algorithm. Overall, the added robustness makes our method a convenient tool for the structural exploration of large, practical CFD data sets.

Acknowledgments

The authors wish to thank Markus Rütten from German Aerospace Center in Göttingen for providing the delta wing and ICE train datasets. Further we thank the members of the FAnToM team at the University of Kaiserslautern and the University of Leipzig for their implementation effort.

References

1. H. Battke, D. Stalling, and H.-C. Hege. Fast line integral convolution for arbitrary surfaces in 3d. In Springer Berlin, editor, *Visualization and Mathematics*, pp. 181–195, 1997.
2. B. Cabral and L. Leedom. Imaging vector fields using line integral convolution. *Computer Graphics (SIGGRAPH '93 Proceedings)*, 27(4):263–272, 1993.
3. I. Forsell. Visualizing flow over curvilinear grid surfaces using line integral convolution. In IEEE Computer Society Press, editor, *IEEE Visualization Proceedings*, pp. 240–247, Los Alamitos, CA, 1994.
4. A. Globus, C. Levit, and T. Lasinski. A tool for visualizing the topology if three-dimensional vector fields. In *IEEE Visualization Proceedings*, pp. 33 – 40, October 1991.
5. D. N. Kenwright. Automatic detection of open and closed separation and attachment lines. In IEEE Computer Society Press, editor, *IEEE Visualization Proceedings*, pp. 151–158, Los Alamitos, CA, 1998.
6. D. N. Kenwright, C. Henze, and C. Levit. Features extraction of separation and attachment lines. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):135–144, 1999.
7. N. Max, R. Crawfis, and C. Grant. Visualizing 3d velocity fields near contour surfaces. In IEEE Computer Society Press, editor, *IEEE Visualization Proceedings*, Los Alamitos, CA, 1994.
8. G. M. Nielson and I.-H. Jung. Tools for computing tangent curves for linearly varying vector fields over tetrahedral domains. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):360–372, 1999.
9. A. Okada and D. L. Kao. Enhanced line integral convolution with flow feature detection. In *Proceedings of IS&T/SPIE Electronic Imaging*, 1997.
10. R. Peikert and M. Roth. The "parallel vectors" operator - a vector field visualization primitive. In *IEEE Visualization Proceedings '00*, pp. 263 – 270, 2000.
11. K. Polthier and E. Preuss. Variational approach to vector field decomposition. In *Eurographics Workshop on Scientific Visualization - Preprint No. 448 TU-Berlin, SFB 288*, 2000.
12. K. Polthier and M. Schmies. Straightest geodesics on polyhedral surfaces. pp. 391–408, 1998.
13. M. Roth. *Automatic Extraction of Vortex Core Lines and Other Line-Type Features for Scientific Visualization*. PhD thesis, ETH Zürich, 2000.
14. A. Sheffer and J. C. Hart. Seamster: Inconspicuous low-distortion texture seam layout. In IEEE Computer Society Press, editor, *IEEE Visualization Proceedings*, pp. 291–298, Los Alamitos, CA, 2002.
15. D. Stalling and H.-C. Hege. Fast and resolution independent line integral convolution. In ACM SIGGRAPH, editor, *Proceedings of SIGGRAPH, Computer Graphics Annual Conference Series*, pp. 249–256, 1995.

Fast Vortex Axis Calculation Using Vortex Features and Identification Algorithms

Markus Rütten¹ and Hans-Georg Pagendarm²

¹ German Aerospace Center (DLR), Institute of Aerodynamics and Flow Technology,
Bunsenstraße 10, 37073 Göttingen, Germany
markus.ruetten@dlr.de

² German Dutch Windtunnel (DNW), Business Unit GuK, Bunsenstrasse 10, 37073
Göttingen, Germany
pagendarm@dnw-germany.aero

Summary. Knowledge about the vortical flow over delta wings, its structure and behavior is an essential issue for the development of highly maneuverable aircraft and high angle of attack flight. Vortex breakdown is one of the limiting factors of extreme maneuvers in flight and poses a difficulty for flight control. The demand of simulating the vortical flow behavior has pushed the limits of current CFD Codes, but also the requirements for advanced post processing tools. From the analysis point of view, an important requirement is the capability to identify vortical flow patterns such as the vortex axis lines, vortex hull surfaces, and vortex-vortex interactions. This information is significant for a deeper physical understanding of various vortical flow phenomena and for flight control of aircrafts.

One goal of this paper is to compare various typical vortex identification methods, applied to the vortical flow over a generic 65° swept delta wing. Another goal is to apply those methods as elements of a construction kit, which defines a fast algorithm to detect and calculate vortex axes.

1 Introduction

The flow field of a symmetric delta wing is dominated by large scale vortical structures on its lee side. Typically, there are two dominant primary vortices and smaller secondary, tertiary as well as other subtype vortices. Although the term vortex is often used by the fluid dynamic scientists in a common sense, a precise mathematical description is difficult and not complete in the sense of describing all features of such a rotational fluid. Lugt [10] mentioned the dilemma of defining a vortex, but nevertheless proposed the following definition of a vortex: *A vortex is the rotating motion of a magnitude of material particles around a common center.* This intuitive definition, which describes the visual observations in nature, is incomplete because it is not Galilean invariant to moving reference frames. Robinson [13] extended the definition introducing a concrete observation time and the general moving reference frame: *A vortex exists when instantaneous streamlines mapped onto a plane normal*

to the vortex core exhibit a roughly circular or spiral pattern, when viewed from a reference frame moving with the center of the vortex core. In a mathematical sense this introduces a new problem, namely, that the vortex core and its motion, which is one of the major patterns of the vortex itself, has to be known beforehand. From a post processing point of view, i.e. analyzing CFD data sets, this definition has two further shortcomings: First, there is not a precise definition of the vortex core. Is it a region or is it a line? Second this definition is not a local one. Here, local means analyzing the properties of “atomic” fluid elements. In contrary to a local approach this definition needs a set of seeding points for particle tracing or streamline calculation, whereby the start positions are selected while referring to a known common center.

Although there is no general vortex definition, various vortex identification methods have been developed and are rather successful in practical use. Some definitions and methods are presented here to cover the relevant issues of vortices without trying to provide a complete coverage of existing methods. At first some clarifications about used terms will be given, because different definitions and detection methods often have a diverging interpretation of similar terms. Therefore, we will give a short overview about analytical vortices and their features, followed by more general vortex definitions. Later we will consider well known vortex axis detection algorithms. At last our approach will be presented and examples will be shown.

2 Analytical Vortices and Their Features

Any motion of a Newtonian fluid can be described by the Navier-Stokes equations. In a coordinate invariant form without considering external forces and concentrating on incompressible effects one gets:

$$\nabla \vec{v} = 0 . \quad (1)$$

$$\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla) \vec{v} = -\frac{1}{\rho} \nabla p + v \nabla^2 \vec{v} , \quad (2)$$

where v is the velocity, p the pressure and ν the dynamic viscosity. With regard to the various intuitive impressions and definitions of a vortex, the main feature is the rotation around a common center, thus, suggesting to use cylindrical coordinates as natural coordinate system, leads the thought to the description of such motions and related forces. Therefore, it is common use to express the underlying Navier-Stokes equations in cylindrical form (r, φ, z) . Without an axial velocity component the vortex is reduced to a motion in the two-dimensional (r, φ) -plane. In most cases analytical solutions of this type of Navier-Stokes equations are describing a vortical motion of fluid particles. Therefore, these solutions are called analytical vortices.

If axial symmetry is the only simplification, then radial and axial flow will be possible. Furthermore the solution of the simplified Navier-Stokes equations is symmetric. Imposing more problem specific simplifications, one distinguishes between two types of vortices, the universal solutions and similarity solutions. Often these vortices are of a very special type and purely academic, but they show the basic

common features of vortices. Therefore, a brief look to these solutions of the Navier-Stokes equations will illustrate the specific features of a vortex.

The universal solutions have to fulfill the following boundary condition:

$$\frac{\partial \vec{\omega}}{\partial t} + \operatorname{curl} (\vec{\omega} \times \vec{v}) = 0 \quad (3)$$

$$\operatorname{curl} \operatorname{curl} \vec{\omega} = 0 , \quad (4)$$

with ω as vorticity. This means that the viscous parts are vanishing without the condition that the viscosity ν itself has to be identical to zero. A further simplification is to excluded time dependency. This leads to an unchangeable vortex and in this sense the vortex is universal.

For a planar flow the axial velocity vanishes, and the radial velocity is set to zero. Then the Navier-Stokes equation reduces to following simple form:

$$\frac{\partial^2 v_\phi}{\partial r^2} + \frac{1}{r} \frac{\partial v_\phi}{\partial r} - \frac{v_\phi}{r^2} = 0 , \quad (5)$$

$$\frac{v_\phi^2}{r} = \frac{1}{\rho} \frac{\partial p}{\partial r} \quad (6)$$

To solve these equations, the following expression can be derived for the primitive variables:

$$\frac{\partial \vec{v}}{\partial t} = (\vec{v} \cdot \nabla) \vec{v} - \frac{1}{\rho} \nabla p = \nu \nabla^2 \vec{v} \equiv 0 \quad (7)$$

The general solution for the velocity is:

$$v_\phi = \frac{a}{r} + br , \quad (8)$$

whereby the integration constants a and b are fixed by the boundary conditions. The pressure can be calculated using following equation:

$$p - p_c = \rho \left[\frac{a^2}{2} \cdot \left(\frac{1}{r_c^2} - \frac{1}{r^2} \right) + 2ab \cdot \log \left(\frac{r}{r_c} \right) + \frac{b^2}{2} (r^2 - r_c^2) \right] . \quad (9)$$

The subscript c denotes the vortex core. This class of solutions contains simple vortices like the potential vortex, the Couette flow, the rigid body rotation or the Rankine vortex.

Equation (6) describes the equilibrium between the pressure gradient and the centripetal acceleration on the circular flow, hence the centrifugal force, which is the cause for the minimum of static pressure in the center of the vortex and thus at the vortex axis. Therefore, (6) describes a main vortex feature, which is often used for a vortex detection and visualization.

To study the chronological development the time derivative of the velocity has to be considered as well. To do this, the equation is extended by this term. With a vortex axis still fixed in time and place, the solutions of following equation and (6) are known as similarity solutions.

$$\frac{\partial v_\varphi}{\partial t} = v \left(\frac{\partial^2 v_\varphi}{\partial r^2} + \frac{1}{r} \frac{\partial v_\varphi}{\partial r} - \frac{v_\varphi}{r^2} \right) = \frac{\partial \omega}{\partial r} \quad (10)$$

In their simplest form similarity solutions are describing the diffusive transport of vorticity:

$$\frac{\partial v_\varphi}{\partial t} = v \frac{\partial \omega_z}{\partial r}, \quad (11)$$

which is a main feature of instationary vortical flow. The circumferential velocity may be solved using a separation approach:

$$v_\varphi = f(r) h(t), \quad (12)$$

which leads to:

$$v_\varphi = (c_1 J_1(\lambda r) + c_2 Y_2(\lambda r)) \cdot e^{-\lambda^2 vt}. \quad (13)$$

J_1 and Y_1 are first order Bessel and Weber functions, c_1 and c_2 are integration constants defined by adequate boundary conditions. A well known example of this type of vortex is the Hamel-Oseen respectively Lamb-Oseen vortex, which can be destinated by using the similarity variable:

$$\lambda = \frac{r^2}{4vt}. \quad (14)$$

This leads to following solution:

$$v_\varphi = \frac{\kappa_0}{r} \left(1 - e^{-\frac{r^2}{4vt}} \right), \quad v_r = v_z = 0. \quad (15)$$

Here κ_0 denotes the vortex strength. The interpretation shows the next interesting main feature of complex analytical vortices: There is a maximum of circumferential velocity, which decreases with time and moves radially outwards away from the vortex axis. Considering these strict relations a vortex core is a region and can not be a line. This term describes a region in the center of a vortex, where the absolute value of circumferential velocity grows from the vortex axis to the limiting hull surface of the vortex core. Within the outer part of the vortex outside the core the circumferential velocity decreases again. The formal definition is given in Lugt [11]. Therefore, a vortex core can not be the same as the vortex axis. The latter one is a line feature. This maximum of the circumferential velocity can be used to visualize the vortex core as the region with the surface of maximal circumferential velocity as a hull. The terms vortex core and vortex axis will be used in this sense. Another term is the vortex core line and is still for discussion. In our opinion, the term vortex core line should be used if a physical variable other than the velocity is used to calculate the common center line of rotating fluid particles. For example this line is described by a pressure minimum or a maximum of vorticity feature. Thus, it is used in a more general sense than the vortex axis term.

Back to the Lamb-Oseen vortex: the overall behavior of the vorticity, which is indeed maximal at the vortex axis but also decreases exponentially in time, is similar to the velocity:

$$\omega_z = \frac{\kappa_0}{2vt} e^{-\frac{r^2}{4vt}}, \omega_r = \omega_\phi = 0 \quad (16)$$

Therewith a next extremal feature of an idealized vortex, the maximum of vorticity at the vortex axis, is identified, which is also a point of vortex core line detection.

From this it may be derived that for a small radius next to the vortex axis the vortex behaves like a rigid body rotation with the angular velocity Ω . A series expansion of (15) at $r = 0$ leads to following approximation:

$$v_\phi = \frac{\kappa_0}{4vt} r = \Omega \cdot r. \quad (17)$$

And this is an interesting feature as well as a crucial problem for vortex axis detection, as we will show later.

One main flow feature, not contained in the idealized vortices considered before, is convection. To describe more realistic flows the equations have to be extended, allowing for axial and radial velocity components. From the first simplifications, only axial symmetry is left. Now the Navier-Stokes equation looks like:

$$\frac{\partial v_r}{\partial r} + \frac{v}{r} + \frac{\partial v_z}{\partial z} = 0 \quad (18)$$

$$\frac{\partial v_r}{\partial t} + v_r \frac{\partial v_r}{\partial r} - \frac{v_\phi^2}{r} + v_z \frac{\partial v_r}{\partial z} = -\frac{1}{\rho} \frac{\partial p}{\partial r} + v \left(\frac{\partial^2 v_r}{\partial r^2} + \frac{1}{r} \frac{\partial v_r}{\partial r} - \frac{v_r}{r^2} + \frac{\partial^2 v_r}{\partial z^2} \right), \quad (19)$$

$$\frac{\partial v_\phi}{\partial t} + v_r \frac{\partial v_\phi}{\partial r} + \frac{v_r v_\phi}{r} + v_z \frac{\partial v_\phi}{\partial z} = v \left(\frac{\partial^2 v_\phi}{\partial r^2} + \frac{1}{r} \frac{\partial v_\phi}{\partial r} - \frac{v_\phi}{r^2} + \frac{\partial^2 v_\phi}{\partial z^2} \right), \quad (20)$$

$$\frac{\partial v_z}{\partial t} + v_r \frac{\partial v_z}{\partial r} + v_z \frac{\partial v_z}{\partial z} = -\frac{1}{\rho} \frac{\partial p}{\partial z} + v \left(\frac{\partial^2 v_z}{\partial z^2} + \frac{1}{r} \frac{\partial v_z}{\partial r} - \frac{v_\phi}{r^2} + \frac{\partial^2 v_z}{\partial z^2} \right). \quad (21)$$

One of the most famous solutions is the Burgers vortex:

$$v_\phi = \frac{\kappa_0}{r} \left(1 - e^{-\frac{ar^2}{2v}} \right), a > 0 \quad (22)$$

$$v_r = -ar, v_z = 2az, a > 0 \quad (23)$$

which is often used in analytical studies of vortex breakdown. The vorticity equation holds

$$\omega_z = \frac{a\kappa_0}{v} e^{-\frac{ar^2}{2v}}, \omega_r = \omega_\phi = 0, \quad (24)$$

for any arbitrary constant a. The so called Q-vortex has the same structure, whose circumferential velocity is:

$$v_\phi = q |\Delta v_z| \frac{\delta}{r} \left(1 - e^{-\frac{r^2}{\delta^2}} \right) \quad (25)$$

and its axial velocity is

$$v_z = \Delta v_z e^{-\frac{r^2}{\delta^2}}, \quad (26)$$

with $q = \frac{\Gamma_\infty}{2\pi\delta\Delta v_z}$ and δ as vortex core radius and Δv_z as difference of the velocity magnitude at the vortex axis to the velocity magnitude at the core border. Here Γ_∞ is the circulation. This difference acts like a scaling function and makes the vortex Galilean invariant.

The last two vortices are time-independent, but to allow for such a quasi-constant solution there must be an equilibrium between diffusion and convection of the angular momentum. If this balance is destroyed, for example if the diffusion is bigger than the convection, the axial velocity has to be decreased. This has a crucial impact on the pressure on the axis. The gradient and therefore the pressure will increase. This is an important point of vortex axis detection, because in this case a pressure gradient method will become more sensitive against overlaying pressure changing.

3 Regions Containing Vortices

Here in this paper a vortex is understood as a limited regional flow pattern of a more realistic three-dimensional flow beyond mentioned simplifications. Due to this it is possible to define a hull surface limiting this region in the sense of including a vortex or vortical structure. In contrary analytical vortices with circumferential velocity equations can have an exponential character gradually fading away. Some definitions will be presented promoting this conception and from the visualization point of view they will be used to encapsulate and show a vortex and vortical structures. This may well be more than one vortex, i.e. vortex – vortex interactions or a spiraling vortex breakdown flow.

An obvious feature of a free vortex is the pressure minimum. As mentioned, it is needed as radial force to provide the centripetal acceleration that keeps a particle rotating around an axis.

$$a_r = \frac{v_\phi^2}{r} = \frac{\partial p}{\partial r} \quad (27)$$

Therefore, a pressure minimum can be used to identify a region containing a vortex. Robinson [13] used this method thereby accepting that a pressure minimum does not contain any information about the sense of rotation and it is not robust against other flow patterns with inherent pressure minima. Accordingly an a priori knowledge about the flow is necessary to use a certain pressure value as an indicator for a vortex. Furthermore, from the visualization point of view a user defined iso-surface value of a scalar field weakens the mathematically objectivity and elegance and has the great disadvantage not to consider the gradual decrease of the vortex strength, which is accompanied by a decreasing pressure minimum. Thus a specified iso-value could suppress weak vortices. But in practical use a pressure minimum criterion can be very successful, to get a quick overview about the main flow structures, in particular once the underlying geometrical configuration, i.e. a delta wing, is simple enough to allow for use of such rough criterion.

An extension of the pure pressure criterion is the total pressure loss criterion, which sums up the loss of total pressure by viscous forces. This has the same disadvantages in principle but it is more successful especially for delta wing flows or vortex wake flows.

$$\Delta p_{loss} = 1 - \frac{p_i + \frac{1}{2}\rho_i |\vec{v}_i|^2}{p_\infty + \frac{1}{2}\rho_\infty |\vec{v}_\infty|^2} \quad (28)$$

Similar problems as with a pressure or total pressure loss criterion are occurring when considering another main feature of a vortex, a growing vorticity value inside a vortex region until it reaches the maximum value of vorticity at the vortex axis. It can be used to identify the vortex axis just as a vortex region itself. Again, this is not a robust criterion because the vorticity is strongly coupled with planar shear as well as to rotational shear. So it may be happen that higher values of vorticity occur in a boundary layer than in a free vortex.

A validation of both approaches, using pressure or vorticity, showed that they are not valid enough to give the right vortical information for further analysis.

To overcome these shortcomings Levy et al. [9] introduced the normalized helicity, often called stream vorticity:

$$H_n = \frac{\vec{v} \cdot \vec{\omega}}{|\vec{v}| \cdot |\vec{\omega}|}. \quad (29)$$

Geometrically interpreted this represents the cosine between the velocity vector and the vorticity vector. A vortex can be defined as a region with cosine values exceeding a certain limit. This approach has two further advantages: first the rotation sense of a vortex is determined by the sign of the helicity, so it is possible to differentiate between counter-rotating vortices, especially to separate primary from secondary vortices and so on. The second important point is, that the vortex axis can be found at the extremal helicity values. This can be used to set a start point for a segment wise streamline integration, following the extremum of helicity. The algorithm is fast and easy to implement, but Roth [14] showed that the resulting line considerably differs from the vortex axis, which is analytically known. Roth mentioned that the curvature of a vortex is the main reason for such failure.

As mentioned before this methods all have one problem in common. Analogously to analytical vortices they do not deliver a sharp limiting border, between where a vortex exists and where not. This is achieved by more general vortex definitions and identification schemes presented now.

Looking for a pointwise local vortex criterion one main feature of a rotational fluid is the complexity of eigenvalues of the velocity gradient tensor. This means that any fluid element with a self rotation has such eigenvalues, which can be used to define a vortex. Dallmann and Vollmers [3, 18] described this, calculating the discriminant D of the velocity gradient tensor.

$$D = 27R^2 + (4P^3 - 18PQ)R + (4Q^3 - P^2Q^2) > 0, \quad (30)$$

with the three invariants of the velocity gradient tensor P, Q, R. A positive value of the discriminant denotes a vortex, so an epsilon above zero can be used as a satisfactory

value to visualize the vortex's limiting iso-surface. But this approach has the shortcoming to deliver signals in the immediate neighborhood of curved walls. Therefore, in technical applications a wall distance parameter is often accounted for excluding these false signals.

One extension of the idea is suggested by Dallmann [2] and Hunt et al. [4]. Instead of using the discriminant the second invariant of the velocity gradient tensor, the invariant Q, has to be calculated. The invariant Q balances the influence of pure shear against the influence of fluid element rotation:

$$Q = \frac{1}{2} (v_{ii}^2 - v_{ij}v_{ji}) = -\frac{1}{2}v_{ij}v_{ji} = \frac{1}{2} \left(\|\tilde{\Omega}\|^2 - \|\tilde{S}\|^2 \right) > 0 \quad (31)$$

In this case an epsilon above zero is also used for visualization, but here it is important to distinguish to regions without flow. Because similar problems to the discriminant criterion are occurring at curved walls Hunt additionally introduced a limiting pressure to cut off signals with the help of this user defined value. This has nearly the same effect as using a certain wall distance.

The balancing between the rotational and the shear part of the fluid element motion of the invariant Q method will be obvious considering the kinematic vorticity number N introduced by Truesdell [17] in comparison. The condition for a point belonging to a vortex region is, that the rotational part out balances the shear part:

$$N = \frac{\|\tilde{\Omega}\|}{\|\tilde{S}\|} > 1 \quad (32)$$

Here $\tilde{\Omega}$ denotes the rotational part, \tilde{S} the deformation part of the velocity gradient tensor. This equation can be deduced from the invariant Q equation:

$$2Q = \Omega_{ij}\Omega_{ji} - S_{ij}S_{ji} \quad (33)$$

and

$$1 + \frac{2Q}{S_{ij}S_{ji}} = \frac{\Omega_{ij}\Omega_{ji}}{S_{ij}S_{ji}} = N_k^2. \quad (34)$$

Here the indices are used to show the multiplication of the components. Looking to (34) is is obvious that for an invariant Q larger than zero the kinematic vorticity number will be larger than one. Therefore these are two expressions of the same contents and the balancing character becomes clear.

One of the most famous vortex detection methods is the lambda2 criterion, introduced by Jeong and Hussain [5]. They define a vortex based on the symmetric deformation tensor S and the antisymmetric spin tensor Ω . They derive the following equation:

$$\Omega_{ik}\Omega_{ki} + S_{ik}S_{ki} = \left(-\frac{1}{\rho} p_{,ij} \right). \quad (35)$$

This can be interpreted in a way, that a vortex exists, where only the tensor $\Omega_{ik}\Omega_{ki} + S_{ik}S_{ki}$ would produce a pressure minimum. The sum of the norm of the

deformation and the norm of the rotation tensor is symmetric and has three real eigenvalues. In order to fulfill the condition of a pressure minimum two eigenvalues have to be less than zero. This is used as a definition equation for a vortex:

$$\lambda_2 < 0 \wedge \lambda_2 \in \{\lambda_i(\Omega_{ik}\Omega_{ki} + S_{ik}S_{ki})\}. \quad (36)$$

Equation (36) means that a point is contained within a vortex, if the second largest eigenvalue, called lambda2, is negative. After this, the vortex hull surface can be visualized by an isosurface with an iso-value of lambda2, an epsilon less than zero.

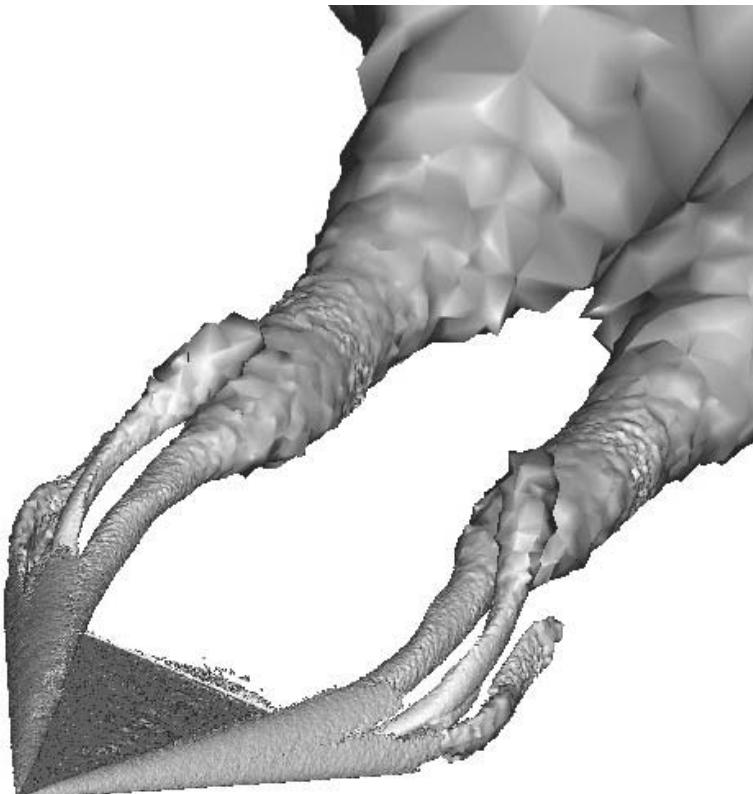


Fig. 1. Lambda2 isosurface as vortex hull

Furthermore, Jeong and Hussain showed that a stronger vortical region is distinguished by a more negative value of lambda2, which means less pressure as well. This makes the strength of two vortices comparable.

Unfortunately, there is again the fundamental problem of this approach, which is the loss of information about the rotational sense due to the norming calculation process. Another problem of the last methods is, that in the immediate neighborhood

of the vortex axis the vortex can behave like a rotating rigid body. In this case the rotation tensor will become constant and the shear vanishes. Then the distinguishing features becomes less precise.

4 The Problem of Defining a Vortex Axis

The interesting feature of a vortex is the vortex axis or in other term vortex core line. There are as many definitions and detection algorithms as vortex definitions.

Banks and Singer [1] proposed a so called predictor-corrector method, using a vector field as predictor field and a scalar field as corrector field. They demonstrated it as a vorticity-predictor, pressure-corrector implementation to find the vortex axis. Based on the observation, that a high vorticity magnitude indicates a vortex and a strong pressure gradient towards the pressure minimum at the vortex axis is given, they suggest the following algorithm: Starting at a minimum pressure grid point with highest vorticity magnitude one step of a vorticity line integration has to be done to get the next point. At this predicted point a plane with the local vorticity vector as normal vector is build. On this plane the point with minimum pressure will be calculated. If this new point is inside a threshold radius of the predicted point, the line segment will be corrected with the new plane point. Then the integration proceeds. One problem of this approach is to find the right starting points. Banks and Singer pointed out, that a seed point may lay outside of a vortex region for instance considering accelerated shear flow. Furthermore using grid points can lead to more than one calculated skeleton for the same vortex core line. But in many cases Banks and Singer's approach shows good results.

Sujudi and Haimes [16] developed a vortex axis detection algorithm based on the eigenvectors of the velocity gradient tensor. A three dimensional region of complex eigenvalues contains one real eigenvector and a conjugate complex pair of eigenvectors. The real eigenvector represents the rotation axis of the local fluid element. Sujudi and Haimes consider a tetrahedral grid domain, in which the velocity gradient tensor is calculated for all tetrahedrons. This tensor is considered to be constant within the complete tetrahedron. Then, inside the region of complex eigenvalues, they destinate the real eigenvector for all tetrahedrons. Now for each corner point of a tetrahedron a new reduced velocity vector is calculated by subtracting the component of the velocity vector projected onto the real eigenvector. After this it will look for points with a vanishing reduced velocity on each face of the tetrahedron. This point has the feature, that the velocity is parallel to the real eigenvector, so the reduced velocity vanishes. Or in other words the local vortical axis has the same direction as the velocity vector. In the case in which the vortex axis crosses this volume element, there are two faces with such points. Following Sujudi and Haimes's approach the connection between these points builds a line segment of the vortex axis. In ideal cases the neighboring elements have such vortex axis line segments as well. So a complete vortex axis can be reconstructed. Because line segments are calculated element wise only, numerical errors or a bad grid resolution leads to gaps

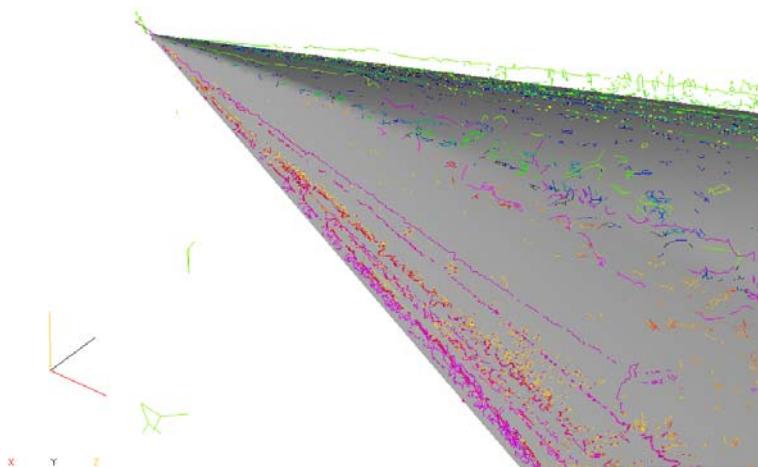


Fig. 2. Unfiltered line segments calculated by applying the parallel operator

or jumps in the vortex axis. This needs to involve filter functions to fill the gaps and to suppress false signals, see Fig. 2.

But for many technical applications it is a very successful algorithm showing impressive results (Fig. 3).

In the sense of a local analysis this cell averaged approach has the disadvantage of the eigenvector not being calculated at the points of the velocity information, here

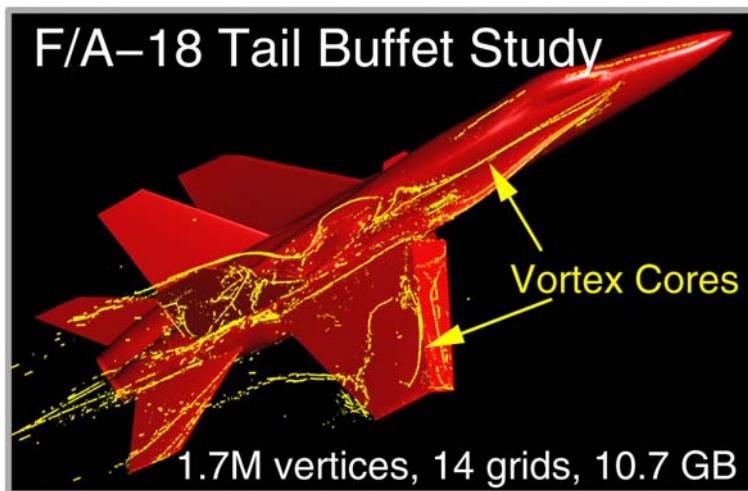


Fig. 3. Vortex axis identification by Kenwright and Haimes [7]

the corner points of the tetrahedron. Only under the condition of the grid cell being treated as a fluid element this analysis can be considered as a local one.

An answer to these shortcomings is the parallel operator by Roth and Peickert [12, 14]. They improved Sujudi and Haimes approach introducing a cell face based algorithm. At all corner point of a cell face the real eigenvectors are calculated instead of considering one eigenvector for the complete cell. Then on the face they again look for points, at which the velocity vector is parallel to the real eigenvector. Both vectors have to be interpolated by the original vector at the corner points. Again the connection of two such cell face points builds a line segment as part of the vortex axis. One advantage of this cell face approach is that neighboring cells have a common cell face, therefore jumps at cell borders are avoided.

Furthermore, Roth can show that it is possible to use the acceleration field instead of the real eigenvector field:

$$\mathbf{v} \parallel \mathbf{e}, \text{ and } \tilde{J}\mathbf{e} = \lambda \mathbf{e}. \quad (37)$$

If parallelism is present, the eigenvector equals the velocity vector, which leads to:

$$\tilde{J}\mathbf{v} = \lambda \mathbf{v}. \quad (38)$$

The left side is the acceleration, which leads to:

$$\mathbf{v} \parallel (\nabla \mathbf{v})\mathbf{v} \text{ or } \mathbf{v} \parallel \mathbf{a}. \quad (39)$$

Using the acceleration instead of the eigenvector has a strong positive effect on computational time and calculation accuracy. Again the method has problems with curved vortices. Therefore Roth extended and generalized his parallel operator using the higher derived vector field curvature of the velocity field instead of the acceleration or the real eigenvector. But from CFD point of view, higher derivatives often have not the sufficient accuracy, in particular using modern industrial unstructured methods for complex realistic configurations. However, Roth showed impressive results for hydraulic turbines and draft tube applications using structured grids, see Fig. 4.

But these are comparably simple to modern aircraft applications.

Summing up the cited methods for vortex features, hull surface and vortex axis, one main result is that the local velocity gradient tensor contains all information for a visualization of a steady vortex. Therefore in our approach this information will be used.

5 A Fast Vortex Axis Detection Combination Approach

All presented algorithms have one overall problem for technical applications. They have to deal with large amounts of data. For example modern CFD calculations of delta wing with detached eddy turbulence models require a grid resolution, which leads to grids with more than 10 millions of cells already for such a simple configuration like the prismatic delta wing. In future the requirements will be certainly more

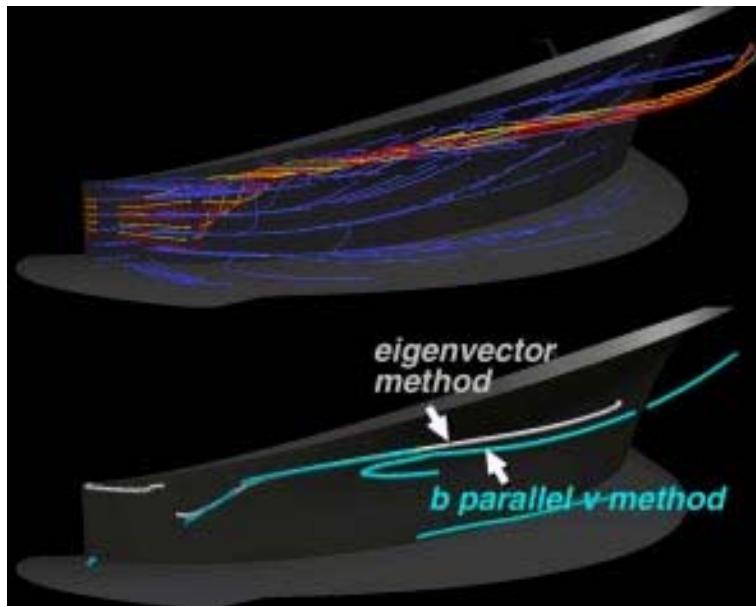


Fig. 4. Application of parallel operator by Roth [14]

than a magnitude higher when taking the step to large eddy simulations (LES). This generates a demand for faster feature and particularly vortex analyzing algorithms. However, fast can not mean to analyze all elements of a given grid. In principle two ways are imaginable: to reduce the region to be analyzed or to find seed points for a streamline respectively vortex line integration.

One approach to find a feasible seed point can be to use knowledge about features of the geometry of the model, which has been calculated before: Looking at the simple prismatic delta wing with its sharp edges and sharp peaks it seems to be easy to use the front tip for starting a streamline integration to get the vortex axes. But especially there, where three edges merge, the discretization error is high and therefore the solution vector has a loss of accuracy. This excludes an integration of a vortex axis and leads to the conclusion that “*a priori*” approaches are not the right answer.

The approach presented here is to combine existing methods to a system, which allows to get a fast impression of the vortical flow behavior even for complex cases:

The first step is to reduce the domain, which has to be considered, by using an advanced vortex criterion. Because pressure or vorticity information alone are not suitable, the complexity of eigenvalues, the invariant Q or the lambda2 criterion can be used to flag the interesting region. For example, if the velocity gradient field and its eigenvalues field are calculated, then only this region with complex eigenvalues will be considered. In Fig. 5 such a vortex hull surface is shown.

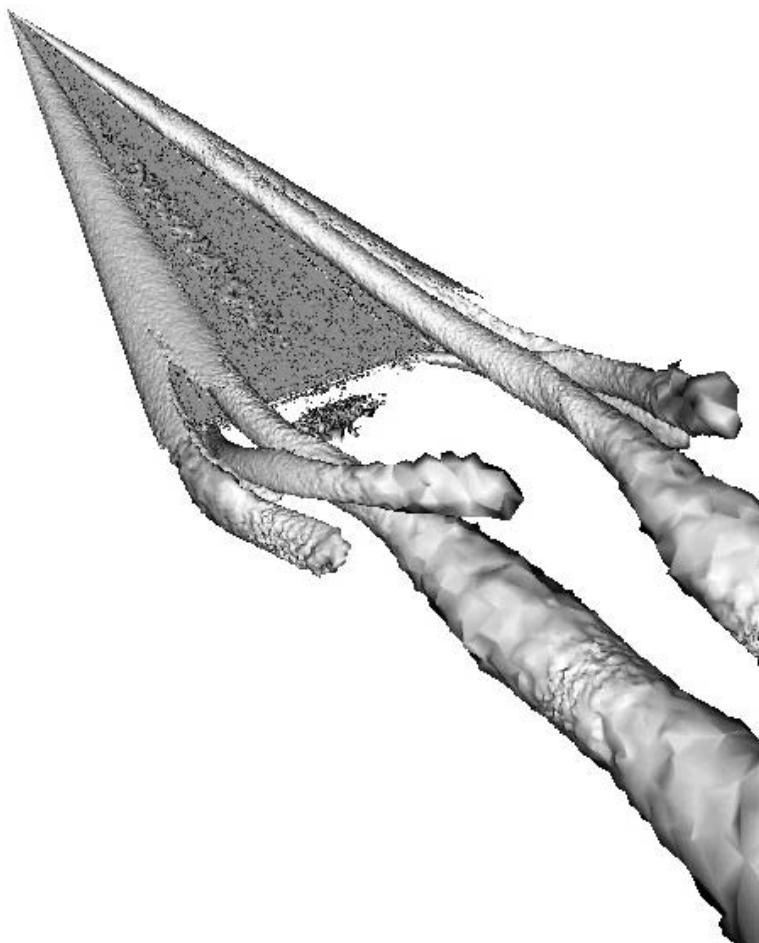


Fig. 5. Invariant Q of velocity gradient tensor with secondary vortices spiraling around primary vortices. Side edge vortices are visible as well.

The next reducing step in the already reduced domain is a search for grid points with a certain extremal vortex feature. In general, this could be a feature such as a minimum of pressure, or maximum of total pressure losses, or a most negative lambda2 value. In other words grid points will be searched for, where one of these features is extremal compared to the same feature at the neighboring points, see Fig. 7. To find fully developed vortices it can be useful to exclude wall or border points. For example the vorticity production in the boundary layer can be a magnitude higher than in the free vortex. Considering that we flag our grid points with a viscous wall distance. That allows us to exclude especially near wall points and reduces false signals.

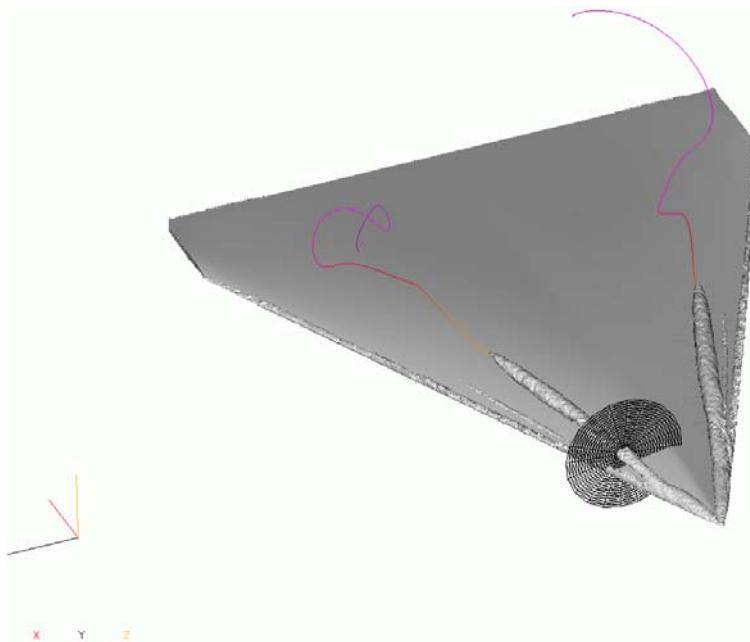


Fig. 6. Vortex axis detection regions and analyzing sheet (upscaled for illustration purpose).

In the third step there is the possibility to flag the neighboring elements of these “extremal” points to create a small region for a next analyzing step. Figure 6 and Fig. 10 are illustrating the results. Another possibility is to introduce a small radial sheet with a plane normal vector in direction of the local velocity vector. This sheet extends radially over the next couples of elements and has only a small number of surface elements. For both approaches only a small number of cell faces have to be considered in the next analyzing step. To prepare this, the real eigenvectors or acceleration vectors have to be calculated at the corner points of the remaining cells. In the case of the analysis sheet the real eigenvectors or acceleration vectors and the velocity vectors have to be mapped onto this sheet.

In a fourth step the parallel operator of Roth is applied to get that cell face point, where the vortex axis passes through. In case of the analysis sheet there should be only one point, the neighbor element approach will almost always deliver several points and the user has to decide, which is the correct one for further proceeding, or he has to implement a checking algorithm.

The fifth step uses these detected points as starting points for a streamline integration to get the vortex axis. There is a discussion about the problem whether a vortex axis is a streamline or not. Therefore, Roth constructs an academic case for which he shows that the vortex axis seems not to be a streamline, but in our opinion his example has a problem: it does not fulfill the basic law of motion, therefore Roth’s argument against vortex axis integration in the sense of streamline integration

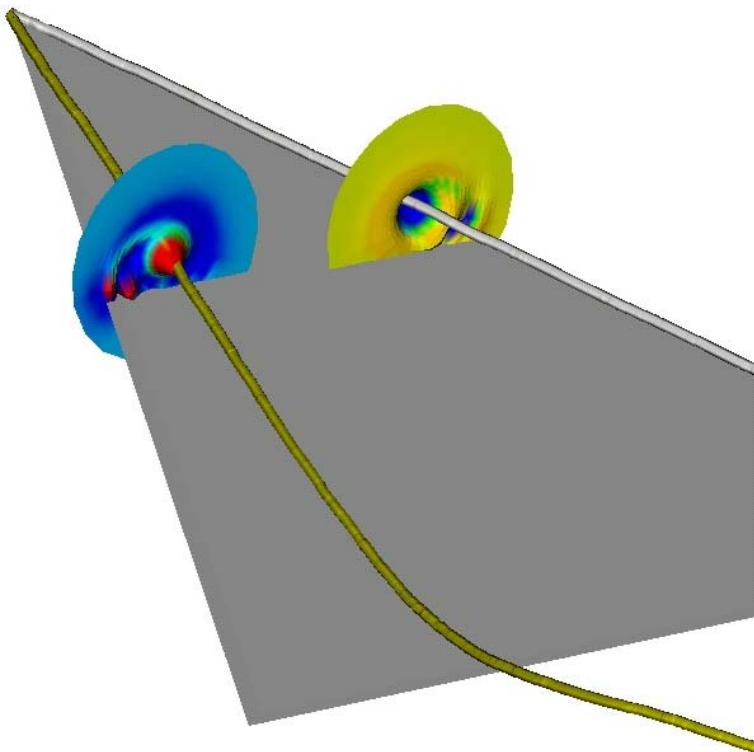


Fig. 7. Primary vortex axes, color coded value of invariant Q of the velocity gradient tensor (left wing side) and color coded value of λ_2 criterion (right wing side respectively)

is weak. For all known solutions of the Navier-Stokes equations the vortex axis is a streamline. Thus it seems to be meaningful to integrate the vortex axis.

To provide a better result, one main idea is to extend the normal line integration algorithm by a self proving part. Again the parallel operator is applied, here each time the integration reaches a new cell face, then the new “parallel” point is the next starting point for the following element. This fulfills the demand of local analysis and prevents accumulation of integration errors. This turns out to be highly advantageous and makes the algorithm robust. Additionally, a smoothing algorithm can be used to get a nicer axis, especially to suppress jumps at cell faces. Note, that the parallel points need to be held fixed. To get a complete vortex axis, the integration has to be done in both directions, upstream and downstream. Both parts will be connected to one line. For comparison see Fig. 8 without correction and Fig. 9 with correction.

The vortex axis integration stops, if the domain border is reached or the streamline reaches a critical point. Furthermore it can happen that the spatial resolution of the grid exceeds a certain limit downstream, or the diffusion of the vortex is so high, that a vortex can not be detected unambiguously. Then the integration has to stop.

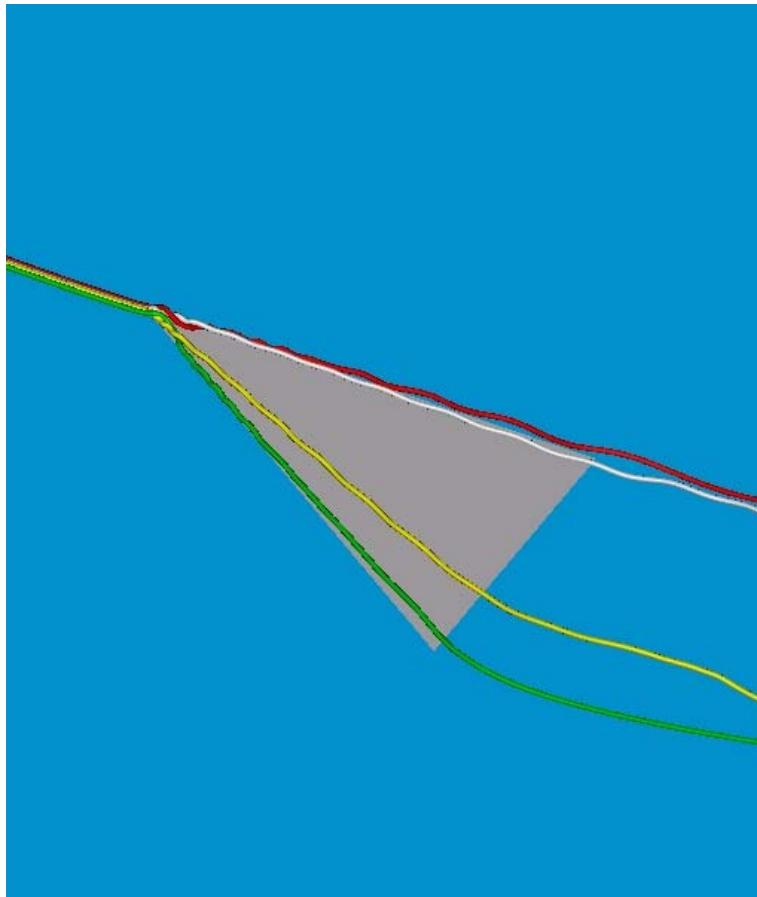


Fig. 8. Integrated vortex axes without parallel operator correction. Note the wiggles of the secondary vortex axes in red. Behind the wing the green vortex axis completely leaves the true path, which should spiral around the yellow vortex axis, see Fig. 1

Our streamline integration extension also delivers a stopping criterion: If the parallel operator has failed within a certain number of cell faces crossed, and these cells are adjacent, then the integration stops and the last steps will be deleted. In this sense the algorithm is self validating.

As mentioned above, more start points are often found than needed or expected. This is a tribute to the accuracy of the data set or in case of the neighbor cell approach it is inherent to the methodology. Therefore, the integration has to stop, if a cell face, crossed by a streamline calculated before, is reached. This newly calculated line segment will not be considered any more to prevent double calculation of one vortex axis. Considering this point the analysis sheet shows the advantage of minimizing such double integration.

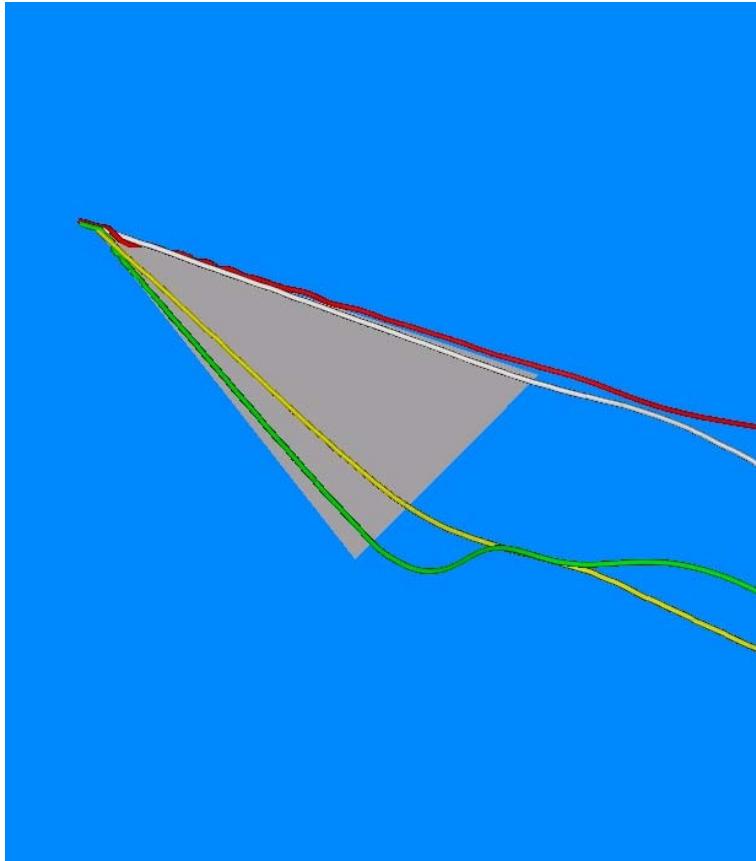


Fig. 9. Integrated vortex axes with parallel operator correction. This improves the accuracy sufficiently. The stopping criterion terminates the vortex line integration upstream of the wing

After all these steps, all vortex axes should be detected and calculated. Furthermore, enough information to visualize hull surfaces of the vortical structure should be present, so it should be possible to get an impression of the overall vortical flow structure.

6 Implementation

The presented algorithm is implemented within a object oriented scientific post-processing code written in C++. Today this code is a brute force implementation and probably has vast potential for optimization. Still it is useful in practise for typical datasets of 400 Gb in size. The graphics make use of the VTK library. The code has been applied to a variety of different dataset. For this paper two delta wing datasets

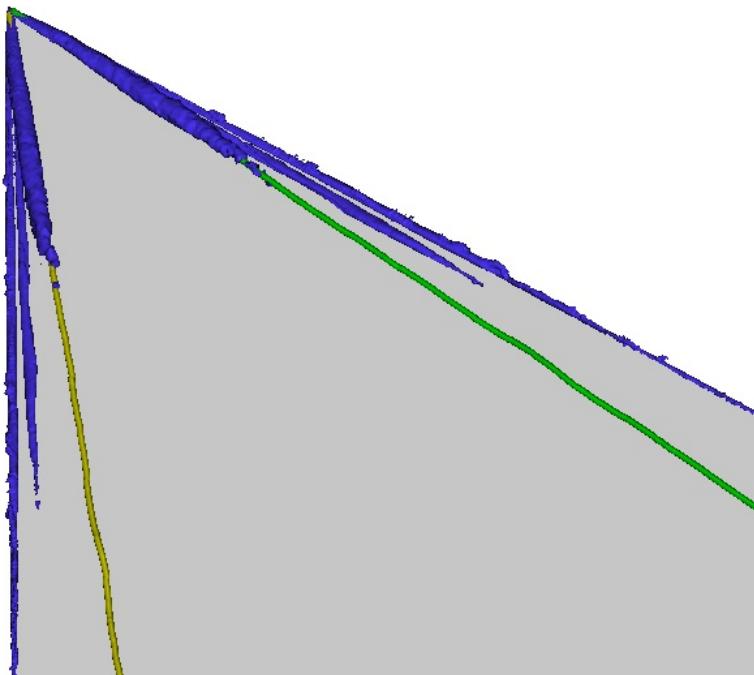


Fig. 10. Detailed view on detected seed point regions (blue) with integrated primary vortex axes (yellow and green)

were used. Each of them has approximately 3 million vertices, 12 million cells of which are 9 million tetrahedrons and 3 million prisms in 20 layers.

7 Conclusions

Although there are a lot of vortex axis detection methods with different advantages and disadvantages, there is no model, that fulfills all requirements for a fast analysis of growing CFD data sets. There is still a need for faster and accurate algorithms.

The approach presented here is designed for reducing the overall computational costs of detection vortex axes. The main idea is to reduce the region to be analyzed using vortex features and identification methods. The complexity of eigenvalues or analogous the lambda2 criterion can be used to shrink the region. Further reduction is done by a local extremum search of a valid scalar flow variable like total pressure losses or lambda2. As result one gets specified grid points inside the vortex with the explicit feature of such extremum. Afterwards an analysis sheet is placed at these locations orientated by the velocity vector as the plane normal vector. After interpolation of velocity and real eigenvector onto the plane the parallel operator of Roth is applied to find valid seed point for a streamline integration, which represents the

vortex axis. An advanced integration control algorithm is implemented to validate the vortex axis while calculation.

Still, the presented algorithm has problems with noisy data, because the parallel operator is sensitive against grid resolution. Therefore, advanced feature or pattern matching algorithm have to be developed. In the near future it will be one of the major challenges in flow pattern detection to develop efficient and fast vortex axes tracking algorithms especially for unsteady flow fields and huge amount of data.

References

1. Banks, D., Singer, B.: Vortex tubes in turbulent flows: Identification, representation, reconstruction. Proceedings of IEEE Visualization '94, Oct. 1994, pp. 132-139
2. Dallmann, U.: Topological Structures of Three-Dimensional Flow Separation. DFVLR-IB 221-82 A 07, 1983
3. Dallmann, U., Vollmers, H., Su, W.H.: Flow Topology and Tomography for Vortex identification in Unsteady and Three-Dimensional Flows. In "Simulation and Identification of Organized Structures in Flow", Proceedings IUTAM Symposium, Technical University of Denmark, Lyngby, May 25-29,1997, J.N. Sorensen (ed.), Kluwer
4. Hunt, J.R.C., Wray, A.A., Moin, P.: Eddies, streams and convergence zones in turbulent flows. In Center of Turbulence Research Report CTR-S88, S.193-208, 1988
5. Jeong, J., Hussain, F.: On the identification of a vortex. J. Fluid Mech. (1995), vol. 285, pp. 69-94
6. Jiang, M., Machiraju, R., Thompson, D.: Detection and Visualization of Vortices, in IEEE Visualization 2002, October 2002
7. Kenwright,D. N., Haimes, R.: Vortex Identification - applications in aerodynamics: A Case study. Proceedings of IEEE Visualization '97, Oct. 1997, pp. 413-416
8. Kenwright, D. N., Henze, C., Levit, C.: Feature Extraction of Separation and Attachment Lines. IEEE Transactions on Visualization and Computer Graphics, vol. 5, no. 2, April 1999
9. Levy, Y., Degani, D., Seginer, A.: Graphical visualization of vortical flows by means of helicity. AIAA Journal, vol. 28, no. 8, Aug. 1990, pp 1347-1352
10. Lutg, H. J.: The dilemma of defining a vortex. In U. Müller, K.G. Roesner, B.Schmidt(eds.), Theoretical and Experimental Fluid Mechanics, pp. 309-321, Springer-Verlag, 1979
11. Lutg, H. J.: Introduction to Vortex Theory. Vortex Flow Press, Inc., Maryland, 1996
12. Peickert, R., Roth, M.: The "parallel vectors" operator - a vector field primitive. Proceedings of IEEE Visualization '99, San Francisco, CA, Oct. 1999
13. Robinson, S. K.: Coherent motions in the turbulent boundary layer. Ann. Rev. Fluid Mech., vol. 23, 1991, pp. 601-639
14. Roth, M.: Automatic Extraction of Vortex Core Lines and Other Line-Type Features for Scientific Visualization. PhD thesis, Swiss Federal Institute of Technology Zürich, 2000
15. Singer, B., Banks, D.: A predictor-Corrector Scheme for Vortex Identification. NASA Contractor Report 194882, ICASE Report No. 94-11, NASA Langley Research Center, Hampton, VA, Mar. 1994
16. Sujudi, D., Haimes, R.: Identification of swirling flow in 3D vector fields. AIAA Paper 95-1715, 12th AIAA CFD Conference, San Diego, CA, Jun. 1995

17. Truesdell, C.: The Kinematics of Vorticity. Indiana University, 1953
18. Vollmers, H.: Separation and vortical-type flow around a prolate spheroid. Evaluation of relevant parameters. AGARD-Symposium on Aerodynamics of Vortical Type Flows in Three Dimensions, Rotterdam, 1983

Topological Features in Vector Fields

Thomas Wischgoll and Joerg Meyer

Electrical Engineering and Computer Science, University of California, Irvine
[twischgo|jmeyer]@uci.edu

Summary. Vector fields occur in many application domains in science and engineering. In combustion processes, for instance, vector fields describe the flow of gases. This process can be enhanced using vector field visualization techniques. Also, wind tunnel experiments can be analyzed. An example is the design of an air wing. The wing can be optimized to create a smoother flow around it. Vector field visualization methods help the engineer to detect critical features of the flow. Consequently, feature detection methods gained great importance during the last years.

Methods based on topological features are often used to visualize vector fields because they clearly depict the structure of the vector field. Most algorithms basically focus on singularities as topological features. But singularities are not the only features that typically occur in vector fields. To integrate other features as well, this paper defines a topological feature for vector fields based upon the asymptotic behavior of the flow. This article discusses techniques that are able to detect this feature.

Key words: Topological analysis, Vector field visualization, Flow visualization, Closed streamlines, Feature detection.

1 Introduction

Many of the problems in natural science and engineering involve vector fields. Fluid flows, electric and magnetic fields are nearly everywhere, therefore measurements and simulations of vector fields are increasing dramatically. As with other data, analysis is much slower and still needs improvement. Mathematical methods together with visualization can provide help in this situation. In most cases, the scientist or engineer is interested in integral curves of the vector field such as streamlines in fluid flows or magnetic field lines. The qualitative nature of these curves can be studied with topological methods developed originally for dynamical systems. Especially in the area of fluid mechanics, topological analysis and visualization have been used successfully [9, 13, 18, 25].

But often, topological methods cover only a few topological features that can occur in vector fields. Basically, only the singularities of a flow are considered in

most algorithms. For instance, a sink is not the only topological feature that is able to attract the surrounding flow. This becomes quite evident when thinking about bifurcation. Consider the Hopf bifurcation as an example. There, a closed streamline may arise from a sink. This closed streamline then has the same properties as the sink it originated from: it attracts the flow in exactly the same way. Based on this motivation, this paper defines a general topological feature that covers singularities such as sources and sinks but also other features depending on their attracting or repelling property.

First, an introduction of existing vector field visualization methods is given including topological techniques. Then, topological features are discussed and a clear definition of a topological feature based on asymptotic behavior of the flow is given. Subsequently, algorithms are described that are able to detect this kind of feature. Finally, results are shown and future work is discussed.

2 Related Works

Several visualization methods for vector fields are available at present. Here, the focus is on describing those methods that are useful in this application area. An overview over the various visualization methods can also be found in other publications [10] and PhD theses [20, 28].

Topological methods depict the structure of the flow by connecting sources, sinks, and saddle singularities with separatrices. Critical points were first investigated by Perry [22], Dallmann [5], Chong [4] and others. The method itself was first introduced in visualization for two-dimensional flows by Helman and Hesselink [12]. Several extensions to this method exist. Scheuermann et al [25] extended this method to work on a bounded region. To get the whole topological skeleton of the vector field, points on the boundary have to be taken into account also. These points are called boundary saddles. To create a time dependent topology for two-dimensional vector fields, Helman and Hesselink [13] use the third coordinate to represent time. This results in surfaces representing the evolution of the separatrices. A similar method is proposed by Tricoche et al. [26, 27] but this work focuses on tracking singularities through time. Although closed streamlines can act in the same way as sources or sinks, they are ignored in the considerations of Helman and Hesselink and others.

To extend this method to three-dimensional vector fields, Globus et al. [9] present a software system that is able to extract and visualize some topological aspects of three-dimensional vector fields. The various critical points are characterized using the eigenvalues of the Jacobian. This technique was also suggested by Helman and Hesselink [13]. But the whole topology of a three-dimensional flow is not yet available. There, stream-surfaces are required to represent separatrices. A few algorithms for computing stream-surfaces exist [16, 24] but are not yet integrated in a topological algorithm.

There are a few algorithms that are capable of finding closed streamlines in dynamical systems that can be found in the numerical literature. Aprille and Trick [2]

propose a so called shooting method. There, the fixed point of the Poincaré map is found using a numerical algorithm like Newton-Raphson. Dellnitz et al. [7] detect almost cyclic behavior. It is a stochastic approach where the Frobenius-Perron operator is discretized. This stochastic measure identifies regions where trajectories stay very long. But these mathematical methods typically depend on continuous dynamical systems where a closed form description of the vector field is available. This is usually not the case in visualization and simulation where the data is given on a grid and interpolated inside the cells. Van Veldhuizen [29] uses the Poincaré map to create a series of polygons approximating an attracting closed streamline. The algorithm starts with a rough approximation of the closed streamline. Every vertex is mapped by the Poincaré map iteratively to get a finer approximation. Then, this series converges to the closed streamline. De Leeuw et al. [6] present a simplification method based on the Poincaré index to simplify two-dimensional vector fields. An example is shown where a closed streamline is simplified to a single critical point. Even though the method might be able to detect closed streamlines in a two-dimensional flow based on the Poincaré index, it is hardly extendable to 3-D.

To get a hierarchical approach for the visualization of invariant sets, and therefore of closed streamlines as well, Bürkle et al. [3] enclose the invariant set by a set of boxes. They start with a box that surrounds the invariant set completely which then is successively bisected in cycling directions. The publication of Guckenheimer [11] gives a detailed overview concerning invariant sets in dynamical systems.

Some publications deal with the analysis of the behavior of dynamical systems. Schematic drawings showing the various kinds of closed streamlines can be found in the books of Abraham and Shaw [1]. Fischel et al. [8] presented a case study where they applied different visualization methods to dynamical systems. In their applications also strange attractors, such as the Lorenz attractor, and closed streamlines occur.

Wegenkittl et al. [30] visualize higher dimensional dynamical systems. To display trajectories, parallel coordinates [17] are used. A trajectory is sampled at various points in time. Then, these points are displayed in the parallel coordinate system and a surface is extruded to connect these points. Hepting et al. [14] study invariant tori in four dimensional dynamical systems by using suitable projections into three dimensions to enable detailed visual analysis of the tori.

Löffelmann [19, 20] uses Poincaré sections to visualize closed streamlines and strange attractors. Poincaré sections define a discrete dynamical system of lower dimension which is easier to understand. The Poincaré section which is transverse to the closed streamline is visualized as a disk. On the disk, spot noise is used to depict the vector field projected onto that disk. Using this method, it can be clearly recognized whether the flow, for instance, spirals around the closed streamline and is attracted or repelled or if it is a rotating saddle. Additionally, streamlines and stream-surfaces show the vector field in the vicinity of the closed streamline.

3 Theory

This chapter introduces the fundamental theory which is needed for the following sections. The description mainly follows the book of Hirsch and Smale [15].

3.1 Data Structures

In most applications in scientific visualization the data is not given as a closed form solution. The same holds for vector fields. Often, a vector field results from a simulation or an experiment where the vectors are measured. In such a case, the vectors are given at only some points of the domain of the Euclidean space. These points are then connected by a grid. A special interpolation computes the vectors inside each cell of that grid. In this paper, we restrict ourselves to a few types of grids, basically triangular and tetrahedral grids. Using barycentric coordinates, vectors inside the cells can be interpolated linearly from the vectors given at the vertices [28, 31].

3.2 Vector Field Features

From a topological point of view critical points are an important part of vector fields. This special feature is described in more detail in this section. We start with the definition of critical points in the general case and then classify different types of singularities.

Definition 3.1 (Critical point)

*Let $v : W \rightarrow \mathbb{R}^n$ be a vector field which is continuously differentiable. Let further $x_0 \in W$ be a point where $v(x) = 0$. Then x_0 is called a **critical point, singularity, singular point, zero, or equilibrium** of the vector field.*

Critical points can be classified using the eigenvalues of the derivation of the vector field. For instance, we can identify *sinks* that purely attract the flow in the vicinity while *sources* repel it purely. A proof for this attracting respectively repelling behavior can be found in [15].

Definition 3.2 (Sink and Source)

*Let v be a vector field which is continuously differentiable and x_0 a critical point of v . Let further $Dv(x_0)$ be the derivation of the vector field v at x_0 . If all eigenvalues of $Dv(x_0)$ have negative real parts, x_0 is called a **sink**. If all eigenvalues of $Dv(x_0)$ have positive real parts, x_0 is called a **source**.*

Streamlines are a very intuitive way to depict the behavior of the flow. But when computing such a streamline it may occur that the streamline computation does not terminate. This mostly is due to closed streamlines where the streamline ends up in a loop that cannot be left. These closed streamlines are introduced and explained in this section. More about the theoretical background can be found in several books [23, 34].

Definition 3.3 (Closed streamline)

Let v be a vector field. A **closed streamline** $\gamma: \mathbb{R} \rightarrow \mathbb{R}^n, t \mapsto \gamma(t)$ is a streamline of a vector field v such that there is a $t_0 \in \mathbb{R}$ with $\gamma(t + nt_0) = \gamma(t) \forall n \in \mathbb{N}$ and γ not constant.

3.3 General Features in Vector Fields

The topological analysis of vector fields considers the asymptotic behavior of streamlines. To describe this asymptotic behavior we have two different kinds of so called limit sets, the origin set or α -limit set of a streamline and the end set or ω -limit set.

Definition 3.4 (α - and ω -limit set)

Let s be a streamline in a given vector field v . Then we define the **α -limit set** as the following set: $\{p \in \mathbb{R}^n | \exists (t_n)_{n=0}^\infty \subset \mathbb{R}, t_n \rightarrow -\infty, \lim_{n \rightarrow \infty} s(t_n) \rightarrow p\}$, while the **ω -limit set** is defined as follows: $\{p \in \mathbb{R}^n | \exists (t_n)_{n=0}^\infty \subset \mathbb{R}, t_n \rightarrow \infty, \lim_{n \rightarrow \infty} s(t_n) \rightarrow p\}$. We speak of an α - or ω -limit set L of v if there exists a streamline s in the vector field v that has L as α - or ω -limit set.

If the α - or ω -limit set of a streamline consists of only one point, this point is a critical point. The most common case of a α - or ω -limit set in a planar vector field containing more than one inner point of the domain is a closed streamline. Figure 1 shows an example for α - and ω -limit sets. There is one critical point and one closed streamline contained in the vector field. Both, the critical point and the closed streamline are their own α - and ω -limit set. For every other streamline the closed streamline is the ω -limit set. If the streamline starts inside the closed streamline, the critical point is the α -limit set. Otherwise the α -limit set is empty. With these explanations, we can give a precise definition of a topological feature of a vector field.

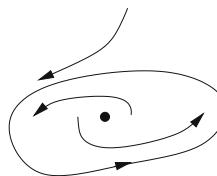


Fig. 1. Example for α - and ω -limit sets.

Definition 3.5 (Topological feature)

Let v be a vector field. Then, a **topological feature** of v is an α - or ω -limit set of the vector field v describing the asymptotic behavior of the flow.

As motivated in the previous example, sources, sinks, and most closed streamlines are considered such a topological feature.

4 Detection in Planar Flows

As can be seen from the definition of sinks and sources, these topological features are relatively easy to determine by calculating the eigenvalues of the flow. Unfortunately, it is not as easy to find the closed streamlines of a flow. Therefore, this chapter describes an algorithm that detects if an arbitrary streamline c converges to a closed curve, also called a limit cycle. This means that c has γ as α - or ω -limit set depending on the orientation of integration. We do not assume any knowledge on the existence or location of the closed curve. We exploit the fact that we use linear interpolation inside the cells for the proof of our algorithm. But the principle of the algorithm works on any piecewise defined planar vector field where one can determine the topology inside the pieces.

4.1 Detection of Closed Streamlines

In a precomputational step every singularity of the vector field is determined. To find all stable closed streamlines we mainly compute the topological skeleton of the vector field. We use an ordinary streamline integrator, such as an ODE solver using Runge-Kutta, as a basis for our algorithm. In addition, this streamline integrator is extended so that it is able to detect closed streamlines. In order to find all closed streamlines that reside inside another closed streamline we have to continue integration after we found a closed streamline inside that region.

The basic idea of our streamline integrator is to determine a region of the vector field that is never left by the streamline. According to the Poincaré-Bendixson-Theorem, a streamline approaches a closed streamline if no singularity exists in that region. To reduce computational cost we first integrate the streamline using a Runge-Kutta-method of fifth order with an adaptive stepsize control. Every cell that is crossed by the streamline is stored during the computation. If a streamline approaches a limit cycle it has to reenter the same cell again. This results in a *cell cycle*.

Definition 4.1 (Cell cycle)

Let s be a streamline in a given vector field v . Further, let G be a set of cells representing an arbitrary rectangular or triangular grid without any holes. Let $C \subset G$ be a finite sequence c_0, \dots, c_n of neighboring cells where each cell is crossed by the streamline s in exactly that order and $c_0 = c_n$. If s crosses every cell in C in this order again while continuing, C is called a *cell cycle*.

This cell cycle identifies the previously mentioned region. To check if this region can be left we could integrate backwards starting at every point on the boundary of the cell cycle. If there is one point converging to the currently investigated streamline we know for sure that the streamline will leave the cell cycle. If not, the currently investigated streamline will never leave the cell cycle. Since there are infinitely many points on the boundary this, of course, results in a non-terminating algorithm. To solve this problem we have to reduce the number of points that need to be checked. Therefore we define *potential exit points*:

Definition 4.2 (Potential exit points)

Let C be a cell cycle in a given grid G as in definition 4.1. Then there are two kinds of **potential exit points**. First, every vertex of the cell cycle C is a **potential exit point**. Second, every point on an edge at the boundary of C where the vector field is tangential to the edge is also a **potential exit point**. Here, only edges that are part of the boundary of the cell cycle are considered. Additionally, only the **potential exit points** in the spiraling direction of the streamline need to be taken into account.

To determine if the streamline leaves the cell cycle, a backward integrated streamline is started to see where a streamline has to enter the cell cycle in order to leave at that exit. We will show later that it is sufficient to only check these potential exit points to test if the streamline can leave the cell cycle.

Definition 4.3 (Real exit points)

Let P be a potential exit point of a given cell cycle C as in definition 4.2. If the backward integrated streamline starting at P does not leave the cell cycle after one full turn through the cell cycle, the potential exit point is called a **real exit point**.

Since a streamline cannot cross itself, the backward integration starting at a real exit point converges to the currently investigated streamline. Consequently, the currently investigated streamline leaves the cell cycle near that real exit point. Figure 2(a) shows an example for such a real exit point.

If on the other hand no real exit point exists we can determine for every potential exit point where there is a region with an inflow that leaves at that potential exit. Consequently, the currently investigated streamline cannot leave near that potential exit point as shown in Fig. 2(b).

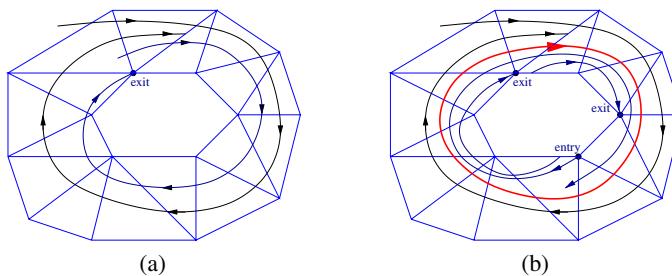


Fig. 2. If a real exit point can be reached, the streamline will leave the cell cycle (a); if no real exit point can be reached, the streamline will approach a limit cycle (b)

With these definitions we can formulate the main theorem for the algorithm:

Theorem 4.4

Let C be a cell cycle with no singularity inside and E the set of potential exit points. If there is no real exit point among the potential exit points E or there are no potential exit points at all then there exists a closed streamline inside the cell cycle.

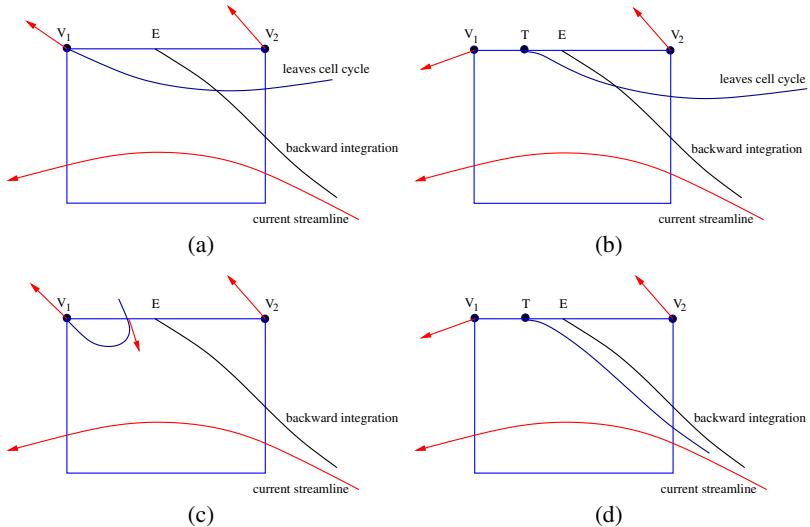


Fig. 3. Different cases of potential exits. (a) and (b) is impossible because streamlines cannot cross each other, (c) contradicts the linear interpolation on an edge, in (d) backward integrations converge to the current streamline so that the point E is a real exit.

Proof: (Sketch)

Let C be the cell cycle. It is obvious that the streamline cannot leave the cell cycle C if all backward integrated streamlines started at every point on the boundary of C leave the cell cycle C . According to the Poincaré-Bendixson-theorem, there exists a closed streamline inside the cell cycle in that case.

We will show now that it is sufficient to only consider the potential exit points. If the backward integrated streamlines starting at all these potential exit points leave the cell cycle the backward integration of any point on an edge will also do.

Figure 3 shows the different configurations of potential exits. Let E be an arbitrary point on an edge between two potential exit points. In part (a) both backward integrated streamlines starting at the vertices V_1 and V_2 leave the cell cycle. Consequently, E cannot be an exit. It would need to cross one of the other backward integrated streamlines which is not possible.

Part (b) of Fig. 3 shows the case where the vector at a point on the edge is tangential to the edge. Obviously, if E lies between V_1 and T the backward integrated streamline will leave the cell cycle immediately. If it lies between T and V_2 and converges to the currently investigated streamline it has to cross the backward integrated streamline started at T . This contradicts the fact that streamlines cannot cross each other. Because of the linear interpolation at the edge, part (c) is also impossible.

We have shown that the currently investigated streamline cannot leave the cell cycle if there are only real exits. Consequently, there exists a closed streamline inside C since there is no singularity inside C . \square

With theorem 4.4 it is possible to describe the algorithm in detail. It mainly consists of three different stages: first a streamline is integrated and one cell change after the other is identified. At each cell the algorithm checks if a cell cycle is completed. In case of a cell cycle it looks for exits by going backwards through the crossed cells and looking for potential exit points. Finally, the exits that were found are validated. Therefore, a streamline starting at the potential exit is integrated backwards through the whole cell cycle. If it is not possible to integrate backwards one full turn throughout the cell cycle for at least one backward integration a closed streamline resides in this cell cycle. Otherwise the forward integration of the original streamline is continued. The algorithm exits if no real exit points are found among all of the potential exit points or if a critical point or the boundary of the vector field is reached. Theorem 4.4 guarantees that the algorithm then detects closed streamlines if every potential exit point [32] is checked.

5 Detection of Features in 3-D Vector Fields

Closed streamlines can be found in three-dimensional vector fields as well. For instance, the *Terrestrial Planet Finder Mission* of NASA [21] deals with stable manifolds where 3-D periodic halo orbits play an important role. These orbits are nothing else than closed streamlines in a three-dimensional vector field.

This section describes how to detect closed streamlines in three-dimensional vector fields. Although the principle to detect closed streamlines in a three-dimensional vector field is similar to the two-dimensional case, there are some differences. The following subsections explain the theoretical and algorithmic differences and similarities.

5.1 Theory

It is assumed that the data is given on a tetrahedral grid, but the principle would work on other cell types as well. The detection of a cell cycle works in the same way as in definition 4.1. Of course, the cells are three-dimensional in this case. To check if we can leave the cell cycle we have to consider every backward integrated streamline starting at an arbitrary point on a face of the boundary of the cell cycle. Looking at the edges of a face we can see directly that it is not sufficient to just integrate streamlines backwards. It is necessary to integrate a stream-surface backwards starting at an edge of the cell cycle. The streamlines starting at the vertices of that edge may leave the cell cycle earlier than the complete surface. In fact, it often occurs that one of these streamlines exit the cell cycle directly while parts of the stream surface itself may stay inside. Consequently, a different definition for exits is required.

Definition 5.1 (Potential Exit Edges)

*Let C be a cell cycle in a given tetrahedral grid G as in Definition 4.1. Then we call every edge at the boundary of the cell cycle a **potential exit edge**. Analogue to the two-dimensional case we define a line on a boundary face where the vector field is tangential to the face as a **potential exit edge** also.*

Due to the fact that we use linear interpolation inside the tetrahedrons it can be shown that there will be at least a straight line on the face where the vector field is tangential to the face or the whole face is tangential to the vector field [33]. Therefore, isolated points on a face where the vector field is tangential to the face do not need to be considered. When dealing with edges as exits, stream-surfaces need to be computed in order to validate these exits. Analogue to definition 4.3 we define *real exit edges*.

Definition 5.2 (Real exit edge)

*Let E be a potential exit edge of a given cell cycle C as in definition 5.1. If the backward integrated stream-surface does not completely leave the cell cycle after one full turn through C then this edge is called a **real exit edge**.*

For the backward integrated stream-surface a simplified version of the stream-surface algorithm introduced by Hultquist [16] is used. Since there is no triangulation of the surface required, only the integration step of that algorithm needs to be executed. Initially, we start the backward integration at the vertices of the edge. If the distance between two neighboring backward integrations is greater than a specific error limit a new backward integration is started in-between. This continues until an approximation of the stream-surface that respects the given error limit has been reached.

The integration stops when the whole stream-surface leaves the cell cycle or when one full turn through the cell cycle is completed. To construct the surface properly it may be necessary to continue a backward integration process across the boundary of the cell cycle. This is due to the fact that parts of the stream-surface are still inside the cell but the backward integrated streamlines have already left it. With these definitions and motivations we can formulate the main theorem for the algorithm:

Theorem 5.3

Let C be a cell cycle as in definition 4.1 with no singularities inside and E the set of potential exit edges. If there is no real exit edge among the potential exit edges E or there are no potential exit edges at all then there exists a closed streamline inside the cell cycle.

Proof: (Sketch)

Let C be a cell cycle with no real exit edge. Every backward integrated stream-surface leaves the cell cycle C completely. As in the 2-D case it is obvious that the cell cycle cannot be left if every backward integration starting at an arbitrary point on a face of the boundary of the cell cycle C leaves that cell cycle. Let Q be an arbitrary point on a face F of the boundary of the cell cycle C . Let us assume that the backward integrated streamline starting at Q converges to the currently investigated streamline. We will show that this is a contradiction.

First case: *The edges of face F are exit edges and there is no point on F where the vector field is tangential to F .*

From a topological point of view the stream-surfaces starting at all edges of F form a tube that leaves the cell cycle. Since the backward integrated streamline starting at Q converges to the currently investigated streamline it does not leave the cell cycle. Consequently, it has to cross the tube formed by the stream-surfaces. This is not possible because streamlines cannot cross each other and therefore a streamline cannot cross a stream-surface either.

Second case: *There is a potential exit edge e on the face F that is not part of the boundary of F .*

Obviously, the potential exit edge e divides the face F into two parts. In one part there is outflow out of the cell cycle C while at the other part there is inflow into C because the flow is tangential at e . We do not need to consider the part with outflow any further because every backward integrated streamline starting at a point of that part immediately leaves the cell cycle C .

The backward integrated surface starting at the potential exit edge e and parts of the backward integrated stream-surfaces starting at the boundary edges of the face F again form a tube from a topological point of view. Consequently, the backward integrated streamline starting at Q has to leave the cell cycle C .

We have shown that the backward integrated streamline starting at the point Q has to leave the cell cycle also. Since there is no backward integrated streamline converging to the currently investigated streamline at all, the streamline will never leave the cell cycle. \square

5.2 Algorithm

With theorem 5.3 it is possible to describe the algorithm in detail. Similar to the two-dimensional case, a streamline is integrated while every cell change is memorized to detect cell cycles. If a cell cycle was found the algorithm looks for potential exits by going backwards through the cell cycle and validating these using backward integrated stream-surfaces. According to theorem 5.3, there exists a closed streamline inside this cell cycle if all backward integrated stream-surfaces leave the cell cycle. In that case, we can find the exact location by continuing the integration process of the streamline that we currently investigate until the difference between two subsequent turns is small enough. This numerical criterion is sufficient in this case since the streamline will never leave the cell cycle.

6 Results

The first example is a simulation of a swirling jet with an inflow into a steady medium. The simulation originally resulted in a three-dimensional vector field but

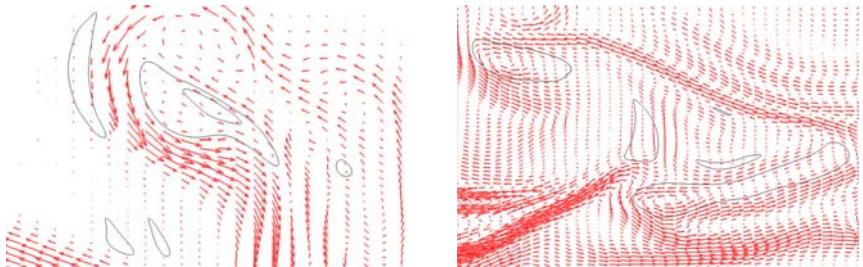


Fig. 4. Vorticity vector field of a turbulent flow – limit cycles.

we used a cutting plane and projected the vectors onto this plane to get a two-dimensional field. In this application one is interested in investigating the turbulence of the vector field and in regions where the fluid stays for a very long time. This is necessary because some chemical reactions need a special amount of time. These regions can be located by finding closed streamlines. Figure 4 shows some of the closed streamlines detected by our algorithm in detail. In addition, a hedgehog representation of the vector field is given. All these limit cycles are located in the upper region of the vector field. Figure 5 shows all closed streamlines of this vector field including the topological skeleton.

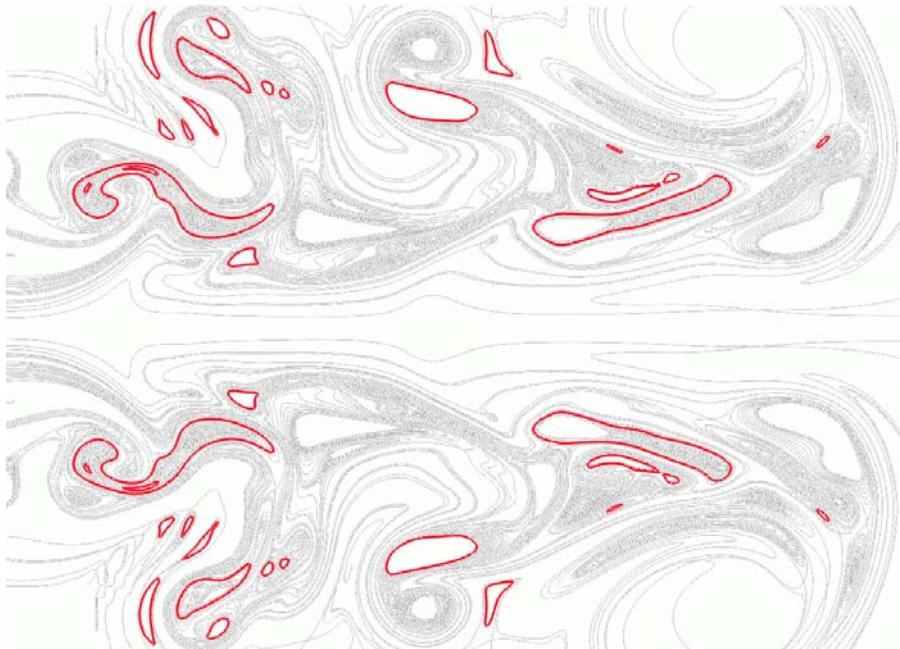


Fig. 5. Vorticity vector field visualized by the topological skeleton including closed streamlines.

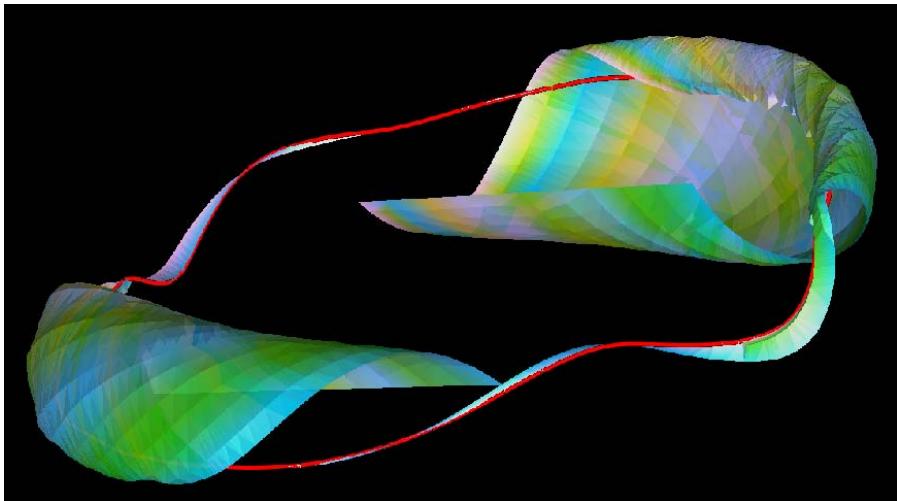


Fig. 6. Limit cycle in a 3-D vector field with stream-surfaces.

To test our 3-D detection, a synthetically created dataset which includes one closed streamline is used. We first created a two-dimensional vector field. The vector field contains a saddle singularity in the center and two symmetrical sinks. To get a three-dimensional flow we rotated the two-dimensional vector field around the axis of symmetry. Due to the symmetrical arrangement of the sinks this vector field includes exactly one closed streamline. Figure 6 shows the result of the algorithm including two stream-surfaces to depict the surrounding flow. Since the closed streamline is attracting, the stream-surfaces approaches the closed streamline. The stream-surface gets smaller and smaller while spiraling around the closed streamline. After a few turns around the closed streamline, it is only slightly wider than a streamline and finally it totally merges with the closed streamline. A random color scheme for the surface is used to enhance the three-dimensional effect. Overall, it is shown that the algorithms discussed in this paper are capable of detecting most of the previously defined features.

7 Impending Challenges

If more than one closed streamline crosses the same cell, the algorithm may fail to detect these closed streamlines, for instance, if there is a structural unstable configuration with one closed streamline inside the other both located in the same cell cycle. One closed streamline acts like a source, lets say the inner one, while the other one behaves like a sink. Therefore, the flow originates at the first one and is attracted by the second one. Since there is an outflow from the cell cycle the algorithm cannot distinguish between a regular outflow and this configuration. A solution for such a situation could be to use a subdivision of the grid for the detection of cell cycles only

to avoid the presence of two closed streamlines in the same cell cycle. In addition, the algorithm for finding closed streamlines in 3-D needs to be applied to more realistic datasets. Also, there exist more closed structures in 3-D such as a torus for instance. Therefore, the algorithm could be extended to find these structures as well.

8 Conclusion

In order to complete the topological analysis of vector fields, this article defined topological features solely based on the asymptotic behavior of the flow. Since singularities are not the only features inside a vector field that can attract or repel the surrounding flow, this is an important extension to topological analyses. Algorithms were presented that are capable of successfully detecting these topological features both in two- and three-dimensional vector fields.

Acknowledgments

We would like to thank the graphics group at the University of Technology at Kaiserslautern, Germany, especially Hans Hagen, Gerik Scheuermann, Xavier Tricoche, and all the students working in this group. Part of this work was funded by DFG (Deutsche Forschungsgemeinschaft) and Land Rheinland-Pfalz, Germany. Wolfgang Kollmann, Mechanical and Aeronautical Engineering Department of the University of California at Davis, provided us with the vorticity dataset. We are very grateful for this and several helpful hints and discussions.

References

1. Ralph H. Abraham and Christopher D. Shaw. *Dynamics – The Geometry of Behaviour: Bifurcation Behaviour*. Aerial Press, Inc., Santa Cruz, 1982.
2. T. J. Aprille and T. N. Trick. A computer algorithm to determine the steady-state response of nonlinear oscillators. *IEEE Transactions on Circuit Theory*, CT-19(4), July 1972.
3. D. Bürkle, M. Dellnitz, O. Junge, M. Rumpf, and M. Spielberg. Visualizing complicated dynamics. In A. Varshney, C. M. Wittenbrink, and H. Hagen, editors, *IEEE Visualization '99 Late Breaking Hot Topics*, pp. 33 – 36, San Francisco, 1999.
4. M. S. Chong, A. E. Perry, and B. J. Cantwell. A General Classification of Three-Dimensional Flow Fields. *Physics of Fluids*, A2(5):765–777, 1990.
5. U. Dallmann. Topological Structures of Three-Dimensional Flow Separations. Technical Report DFVLR-AVA Bericht Nr. 221-82 A 07, Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt e.V., April 1983.
6. W. de Leeuw and R. van Liere. Collapsing flow topology using area metrics. In D. Ebert, M. Gross, and B. Hamann, editors, *IEEE Visualization '99*, pp. 349–354, San Francisco, 1999.
7. M. Dellnitz and O. Junge. On the Approximation of Complicated Dynamical Behavior. *SIAM Journal on Numerical Analysis*, 36(2):491 – 515, 1999.

8. Georg Fischel, Helmut Doleisch, Lukas Mroz, Helwig Löffelmann, and Eduard Gröller. Case study: visualizing various properties of dynamical systems. In *Proceedings of the Sixth International Workshop on Digital Image Processing and Computer Graphics (SPIE DIP-97)*, pp. 146–154, Vienna, Austria, October 1997.
9. A. Globus, C. Levit, and T. Lasinski. A Tool for Visualizing the Topology of Three-Dimensional Vector Fields. In G. M. Nielson and L. Rosenblum, editors, *IEEE Visualization '91*, pp. 33 – 40, San Diego, 1991.
10. Eduard Gröller, Helwig Löffelmann, and Rainer Wegenkittl. Visualization of Analytically Defined Dynamical Systems. In *Proceedings of Dagstuhl '97*, pp. 71–82. IEEE Scientific Visualization, 1997.
11. John Guckenheimer. Numerical analysis of dynamical systems, 2000.
12. J. L. Helman and L. Hesselink. Automated analysis of fluid flow topology. In *Three-Dimensional Visualization and Display Techniques, SPIE Proceedings Vol. 1083*, pp. 144–152, 1989.
13. J. L. Helman and L. Hesselink. Visualizing Vector Field Topology in Fluid Flows. *IEEE Computer Graphics and Applications*, 11(3):36–46, May 1991.
14. D. H. Hepting, G. Derkx, D. Edoh, and R. D. Russel. Qualitative analysis of invariant tori in a dynamical system. In G. M. Nielson and D. Silver, editors, *IEEE Visualization '95*, pp. 342 – 345, Atlanta, GA, 1995.
15. M. W. Hirsch and S. Smale. *Differential Equations, Dynamical Systems and Linear Algebra*. Academic Press, New York, 1974.
16. J. P. M. Hultquist. Constructing Stream Surface in Steady 3D Vector Fields. In *Proceedings IEEE Visualization '92*, pp. 171–177. IEEE Computer Society Press, Los Alamitos CA, 1992.
17. A. Inselberg and B. Dimsdale. Parallel Coordinates: a Tool for Visualizing Multidimensional Geometry. In *IEEE Visualization '90 Proceedings*, pp. 361–378, Los Alamitos, 1990. IEEE Computer Society.
18. D. N. Kenwright. Automatic Detection of Open and Closed Separation and Attachment Lines. In D. Ebert, H. Rushmeier, and H. Hagen, editors, *IEEE Visualization '98*, pp. 151–158, Research Triangle Park, NC, 1998.
19. H. Löffelmann, T. Kučera, and E. Gröller. Visualizing Poincaré Maps Together with the Underlying Flow. In H.-C. Hege and K. Polthier, editors, *Mathematical Visualization, Algorithms, Applications, and Numerics*, pp. 315–328. Springer, 1997.
20. Helwig Löffelmann. *Visualizing Local Properties and Characteristic Structures of Dynamical Systems*. PhD thesis, Technische Universität Wien, 1998.
21. Ken Museth, Alan Barr, and Martin W. Lo. Semi-Immersive Space Mission Design and Visualization: Case Study of the "Terrestrial Planet Finder" Mission. In *Proceedings IEEE Visualization 2001*, pp. 501–504. IEEE Computer Society Press, Los Alamitos CA, 2001.
22. A. E. Perry and B. D. Fairly. Critical Points in Flow Patterns. *Advances in Geophysics*, 18B:299–315, 1974.
23. Robert Roussarie. *Bifurcations of Planar Vector Fields and Hilbert's Sixteenth Problem*. Birkhäuser, Basel, Switzerland, 1998.
24. G. Scheuermann, T. Bobach, H. Hagen, K. Mahrous, B. Hahmann, K. I. Joy, and W. Kollmann. A Tetrahedra-Based Stream Surface Algorithm. In *IEEE Visualization '01 Proceedings*, Los Alamitos, 2001. IEEE Computer Society.
25. G. Scheuermann, B. Hamann, K. I. Joy, and W. Kollmann. Visualizing local Vetor Field Topology. *Journal of Electronic Imaging*, 9(4), 2000.

26. X. Tricoche, G. Scheuermann, and H. Hagen. Topology-Based Visualization of Time-Dependent 2D Vector Fields. In R. Peikert D. Ebert, J. M. Favre, editor, *Proceedings of the Joint Eurographics–IEEE TCVG Symposium on Visualization*, pp. 117–126, Ascona, Switzerland, 2001. Springer.
27. X. Tricoche, T. Wischgoll, G. Scheuermann, and H. Hagen. Topology Tracking for the Visualization of Time-Dependent Two-Dimensional Flows. *Computer & Graphics*, pp. 249–257, 2002.
28. Xavier Tricoche. *Vector and Tensor Field Topology Simplification, Tracking and Visualization*. PhD thesis, University of Kaiserslautern, 2002.
29. M. van Veldhuizen. A New Algorithm for the Numerical Approximation of an Invariant Curve. *SIAM Journal on Scientific and Statistical Computing*, 8(6):951 – 962, 1987.
30. R. Wegenkittl, H. Löffelmann, and E. Gröller. Visualizing the Behavior of Higher Dimensional Dynamical Systems. In R. Yagel and H. Hagen, editors, *IEEE Visualization '97 Proceedings*, pp. 119 – 125, Phoenix, AZ, 1997.
31. Thomas Wischgoll. *Closed Streamlines in Flow Visualization*. PhD thesis, Universität Kaiserslautern, Germany, 2002.
32. Thomas Wischgoll and Gerik Scheuermann. Detection and Visualization of Closed Streamlines in Planar Flows. *IEEE Transactions on Visualization and Computer Graphics*, 7(2), 2001.
33. Thomas Wischgoll and Gerik Scheuermann. Locating Closed Streamlines in 3D Vector Fields. In *Joint Eurographics–IEEE TCVG Symposium on Data Visualization 2002*, pp. 227–232, Barcelona, Spain, 2002.
34. Ye Yan-qian, Cai Sui-lin, Chen Lan-sun, Huang Ke-cheng, Luo Ding-jun, Ma Zhi-en, Wang Er-nian, Wang Ming-shu, and Yang Xin-an. *Theory of Limit Cycles*. American Mathematical Society, Providence - Rhode Island, 1986.

Part IV

Visualization Systems

Generalizing Focus+Context Visualization

Helwig Hauser

VRVis Research Center in Vienna, Austria

<http://www.VRVis.at/>, Hauser@VRVis.at

Focus+context visualization is well-known from information visualization: certain data subsets of special interest are shown in more detail (locally enlarged) whereas the rest of the data is provided as context (in reduced space) to support user orientation and navigation.

The key point of this work is a generalized definition of focus+context visualization which extends its applicability also to scientific visualization. We show how different graphics resources such as space, opacity, color, etc., can be used to visually discriminate between data subsets in focus and their respective context. To furthermore demonstrate its general use, we discuss several quite different examples of focus+context visualization with respect to our generalized definition. Finally, we also discuss the very important interaction aspect of focus+context visualization.

1 Introduction

For a long time already, modern society is greatly influenced by computers. Mainly, computers are used to process data of various kind. Additionally, computers are also used to support the acquisition of data, for example, through measurements or computational simulation. Due to a steadily increasing performance of computers (Moore's law), year by year more data is processed. Since users do not extend their capabilities in data-processing at a comparable rate, there is an increasing need for efficient tools to support the processing of large amounts of data.

One very useful opportunity for accessing large amounts of data is visualization. Data is communicated to the user in a visual form to ease processing. Instead of dealing with loads of numbers, the user accesses the data through pictures and a graphical user interface. This approach is especially useful when the data has at least some spatial form inherently associated with it. In many scientific applications, for example, data is tightly related to concrete parts of our real world, e.g., a 3D computer tomography scan of a human body in a medical application or the 3D simulation of air flow around the computer model of a new aircraft.

The main advantage of visualization is that it uses the great bandwidth of the human visual system for visualization. However, also for visualization the amount

of data to be shown at once is limited. For very large data sets, details cannot be shown for all of the data at the same time. In this case, the user usually is offered the opportunity to either get an overview of the data (no details), or zoom into specific parts of the data and get all of the details there.

While scientific visualization (SciVis, the visualization of scientific data) has been researched for dozens of years already, more recently also the visualization of non-scientific, abstract data (InfoVis, information visualization) such as bank account data or census data has become popular. In InfoVis, an additional step is required in the visualization process, i.e., the mapping of non-spatial data to a visual form. As the user has to learn this additional mapping to effectively use the visualization (and to successfully build up a mental map of the data–form relation), more care is required to support the user with orientation in the visualization. Careless zooming across multiple levels of details can easily cause an effect like being lost in too many details. Thus, advanced solutions have been developed in this field to supply users with both overview and details of the data at the same time.

2 Focus+Context Visualization

In information visualization, an approach called focus+context visualization (F+C visualization) has been developed which realizes the combination of both a general overview as well as a detailed depiction within one view of the data at the same point in time. Traditionally, focus+context visualization refers to an uneven distortion of visualization space such that relatively more space is provided for a certain subset of the data (data in focus). At the same time the rest of the visualization is compressed to still show the rest of the data as a context for improved user orientation.

The idea of using different magnification factors for different parts of the visualization (in one image) to display information in a F+C style already dates back to the '70s of the 20th century [9, 22]. Furnas' work on the fisheye view [10] in 1981 often is accepted as the historical start into computer-based F+C visualization. In this work, Furnas describes how information is selected for display depending on an a-priori importance and the distance of each data item to the current focus of the visualization. Also in the early 1980s, Spence and Apperley presented the bifocal display as a one-dimensional distortion technique [43] to provide a shrinked context on both the left and the right side of an undistorted focal region in the middle of the visualization.

During the 1980s, both approaches have been generalized and extended [11, 32]. In 1992, Sarkar and Brown presented the graphical fisheye view [41], based on Furnas' work, but more focused on the graphical appearance of the F+C visualization (comparable to a real fisheye lens). One year later, Sarkar et al. discussed two techniques (orthogonal and polygonal stretching) for F+C visualization based on the concept of a stretchable rubber sheet [42]. In 1994, Leung and Apperley already presented a review of distortion-oriented F+C visualization, including additional approaches such as the perspective wall [36] (see also later), and providing a respective

taxonomy [33]. They describe techniques of F+C visualization by the characteristics of the magnification function (being the derivative of the transformation function from the undistorted view to the F+C view). Doing so, three classes of techniques are differentiated: (1) approaches with a continuous magnification function (such as the graphical fisheye [41]), (2) techniques with piece-wise constant magnification factors (the bifocal display [43], for example), and (3) others (the perspective wall [36], for example).

The perspective wall, presented by Mackinlay et al. in 1991 [36], is based on the concept of “bending backwards” parts of the display on both the left and the right side of the focus region in the center of the screen (similar to the bifocal display [43]). Perspective projection is used to achieve a variation in magnification factors within this kind of F+C visualization. In 1993, this approach was extended to the so-called document lens [40] – also parts above and below the focal region are used for context visualization.

In the domain of distortion techniques with continuous magnification functions, further extensions have been presented after the first half of the 1990s. In 1995, the three-dimensional pliable surface was presented by Carpendale et al. [3], also using perspective projection to achieve different magnification factors in different parts of the display. Gaussian profiles are used to generate magnification (and thus yield to a continuous magnification) and multiple foci are possible in one view. In 1996, Keahey and Robertson presented non-linear magnification fields as a technique independent from perspective projection and with direct control over the magnification function on every point of a grid over the display [23, 24]. The transformation function is computed in an iterative process, locally optimizing on the difference between the discrete derivative of the transformation field and the input (magnification field).

In 1995, the mapping of hyperbolic space to the plane was used by Lampert et al. [29–31] to achieve F+C visualization, enabling the visualization of infinite space on a limited screen space (at least in principle). In a similar fashion, Kreuseler et al. used the mapping from spherical space to the plane for F+C visualization [28], allowing to move the focal center around the sphere.

The large amount of work on distortion techniques for F+C visualization documents the relevance of this approach, especially in the domain of information visualization. But instead of discussing more details about distortion techniques for F+C visualization or other approaches in this field, we restrict this overview to the above mentioned examples and proceed towards our generalization of F+C visualization. First, however, we briefly discuss how focus is separated from context, an inherently necessary part of every focus+context visualization.

3 Separating Focus from Context

When dealing with focus+context visualization, it is inherently necessary to have a notion of which parts of the data are (at least at one point in time) considered to be “in focus” and what others are not (context part of the data). In the course of this work, we use a so-called *degree-of-interest function* (DOI function), *doi()*, which

describes for every item of the data whether (or not) it belongs to the focus (similar to Furnas' definition [10], but normalized to the unit interval $[0, 1]$):

$$doi_{\text{bin}}(\text{data}[i]) = \begin{cases} 1 & \text{if } \text{data}[i] \text{ is part of the focus} \\ 0 & \text{if } \text{data}[i] \text{ is part of the context} \end{cases}$$

In many application scenarios, a binary discrimination between focus and context (as formulated above) is appropriate, i.e., to assume a sharp boundary between the data items in focus and all the others. In many other cases, however, it is more appropriate to allow a smooth change of $doi()$ -values between the data items in focus and their context (resulting in a *smooth degree of interest* [6]). In other words, the question of whether a data item belongs to the focus (or not) also can be answered by the use of a fuzzy attribute $doi()$:

$$doi(\text{data}[i]) = \begin{cases} 1 & \text{if } \text{data}[i] \text{ is part of the focus} \\ doi \in]0, 1[& \text{if } \text{data}[i] \text{ is part of the smooth} \\ & \text{boundary between focus and context} \\ 0 & \text{if } \text{data}[i] \text{ is part of the context} \end{cases} \quad (1)$$

Accordingly, a fractional value of doi is interpreted as a percentage of being in focus (or interest). A fractional $doi()$ -value can be the result of a multi-valued definition with multiple (still discrete) levels of interest, e.g., $doi \in \{0, 25\%, 50\%, 75\%, 1\}$, a non-sharp definition of what is interesting (e.g., through a definition which is based on continuous spatial distances), or a probabilistic definition (e.g., through a definition incorporating a certain amount of uncertainty).

Usually, the specification of the $doi()$ -function is tightly coupled with the user interface. Different approaches are used to let the user specify which parts of the data (at one point in time) are of special interest (explicit vs. implicit specification, for example). In Sect. 5 we discuss the interaction aspect of F+C visualization in more detail.

In traditional F+C visualization (space-distortion techniques), the degree of interest $doi(\text{data}[i])$ is directly related to the local magnification used to depict a data item $\text{data}[i]$ (this 1:1-relation only holds to a certain extent of accuracy – in general it is not possible to translate every DOI/magnification function into a corresponding transformation function [23, 33]): the larger the $doi()$ -value is, the more screen space is used for visualization, at least locally.

4 Generalized Focus–Context Discrimination

Although the vast majority of research work on F+C visualization has been devoted to space-distortion techniques, the idea of visually discriminating the parts of the data in focus from all the rest, i.e., the context, is more general. In addition to using more space for showing the focus in more detail, other visual dimensions can be used in a similar way. In volume rendering, for example, usually more opacity is used for parts of the data in focus [34], whereas a greater amount of transparency

Table 1. Realizing (generalized) F+C visualization by the uneven use of graphics resources (space, opacity, color, etc.) to discriminate parts of the data in focus from the rest (context) – more details in Sect. 4

graphics resource	approaches	sample technique(s)
space	more space (magnification) for data in focus	graphical fisheye view [41], ...
		F+C process visualization [37]*
opacity	focus rather opaque, context rather transparent	direct volume rendering [34], ...
		RTVR [38]*
color	colored focus in gray context	WEAVE [12], SimVis [5, 6, 8]*
	focus: saturated/light colors	Geospace [35], RTVR [38]*
frequency	sharp focus, blurred context	semantic depth of field [25, 26]*
style	context in reduced style (non-photorealistic rendering)	two-level volume rendering [17, 18]*
		NPR-contours [4]*

*... techniques which are described in more detail in Sect. 4

Table 2. Sample images of five different F+C visualization techniques (from left to right): RTVR-based volume rendering, two-level volume rendering, F+C visualization of simulation data, semantic depth of field, F+C process visualization

sample image					
graphics resource	opacity	style	color	frequency	space
section	4.1	4.2	4.3	4.4	4.5
related paper(s)	[38]	[4, 17, 18]	[5, 6, 8]	[25, 26]	[37]

is used for context visualization. Additionally, also color can be effectively used to visually discriminate different parts of the data. In a system called WEAVE [12], for example, those parts of the data which positively respond to a certain user query (i.e., the current focus) are shown in color, whereas the rest of the data (the context) is shown in gray-scale.

Similarly, other visual dimensions, such as image frequencies, rendering style, etc., can be used to achieve focus–context discrimination (see below for examples). We therefore propose to generalized the definition of focus+context visualization in the following way: *focus+context visualization is the uneven use of graphics resources* (space, opacity, color, etc.) *for visualization* with the purpose to visually discriminate data-parts in focus from their context, i.e., the rest of the data. In Table 1, we give several examples of F+C visualization which are quite different from each other but which all match the above definition and thereby demonstrate its general character. The examples differ from each other with respect to which graphics resource is (unevenly) used to achieve F+C visualization. Below we discuss some of

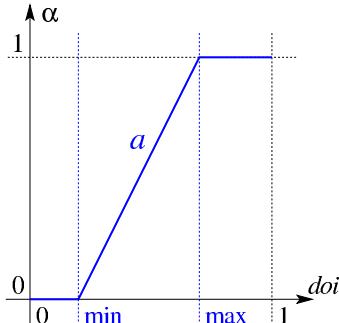


Fig. 1. A simple “window” often is sufficient to map *doi*-values to α -values: *doi*-values up to a certain minimum are mapped to a minimal value of opacity (usually 0), whereas *doi*-values above a certain maximum are mapped to 1 (completely opaque). In between, a linear map from *doi*-values to α -values is used

these examples in more detail (those marked with an asterix in Table 1). In Table 2 we provide a side-by-side comparison of five sample techniques (one sample image each) with pointers to other parts of this document with more detail as well as also to other pieces of related literature.

4.1 More Opacity for Visualization in Focus

One alternative style of F+C visualization (alternative to space-distortion techniques) is identified in a domain where usually other objectives, slightly different from focus–context discrimination, actually govern the development of new techniques. In volume rendering, all from the beginning on [34], a so-called opacity transfer function (OTF) $\alpha()$ is used to deal with the fact that usually not all of the 3D data can be shown simultaneously at full intensity – OTF $\alpha()$ is used to map the data domain to the unit interval ($1 \leftrightarrow$ opaque, $0 \leftrightarrow$ completely transparent). Using an OTF, different values of opacity/transparency are assigned to different parts of the data. This causes that some parts of the data become more prominently visible in the rendered image while others are not (or only hardly) visible.

Originally, the use of an opacity transfer function was not argued with the need to discriminate parts of the data “in focus” from their “context”. However, the goal to visually bring out certain parts of the data in the visualization while reducing the visual appearance of all the rest very well matches the principal idea of F+C visualization. On the basis of a degree-of-interest function, an OTF can be specified by

$$\alpha(\text{data}[i]) = a(\text{doi}(\text{data}[i]))$$

with $a()$ being the identity map ($a(x) = x$), a simple windowing function (see Fig. 1), or any other (potentially simple) monotonic map from $[0, 1]$ to $[0, 1]$. When $\text{doi}()$, for example, is defined on the basis of a scaled distance from a pre-defined isovalue – $\text{doi}(\text{data}[i]) = \max\{1 - s|\text{data}[i] - v_{iso}|, 0\}$ –, then one of Levoy’s OTF is regenerated with $a()$ being the identity map (or a simple window).

From many years of work on the question of how to specify an optimal opacity transfer function [39] we know that one simple data-dependent function $\text{doi}()$ (or $\alpha()$) often is not sufficient to optimally discriminate focus from context in a visualization of 3D data, e.g., 3D medical data or 3D data from computational simulation.

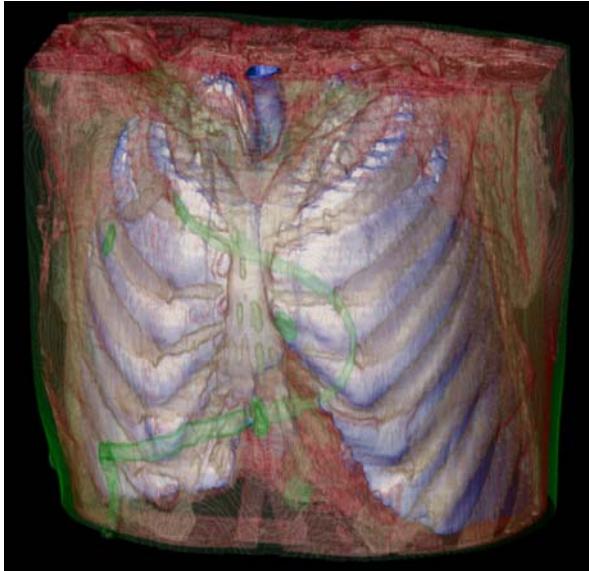


Fig. 2. A segmented CT-dataset of a chest, visualized using two-level volume rendering. Different values of overall opacity have been used for lung (completely opaque), bones (semi-transparent), and skin (very transparent)

Instead, often sophisticated segmentation algorithms are used to do a proper focus–context discrimination before the actual visualization. The result of a segmentation algorithm usually is an n -valued object map $object()$, telling for each and every data item $data[i]$ which object it belongs to.

In two-level volume rendering (2IVR) [13, 17, 18], such an object map is used to improve the F+C visualization of 3D data: instead of directly deriving $doi()$ from the data, the degree of interest is defined on the basis of $object()$, i.e., for all the objects in the data (and not the singular data items) it is determined how interesting they are. This is done, because in many applications the 3D data anyhow is assumed to be composed of objects (in medical applications, for example, a dataset is assumed to be composed of bones, tissue, etc.). Therefore, the user automatically tends to formulate the focus–context discrimination in terms of the data objects (like “I’d like to see the bones and the blood vessels in the context of the skin.”). For rendering, two values of opacity are used in two-level volume rendering: in addition to the $object()$ -based (global) opacity $\alpha_{global} = a_{global}(doi(object))$, which yields the overall opacity for an object (depending on its degree of interest), a local (object-wise specified) OTF $\alpha_{local}(data[i], object(data[i]))$ is used to individually steer the visual appearance of every object.

For example, assuming $\alpha_{local}(., 1)$ to be a relatively sharp Levoy-OTF (comparably large s) and a_{global} to be the identity map, object 1 would be rendered like an iso-surface with its importance $doi(1)$ directly relating to its opacity. Through this separation of α_{global} and α_{local} the task of emphasizing certain parts of the data (semantical question) is separated from the question of how to render the different parts of the data (syntactical question). Accordingly, the parameterization of two-level volume rendering (adjustment of opacities) is much more intuitive (when compared to

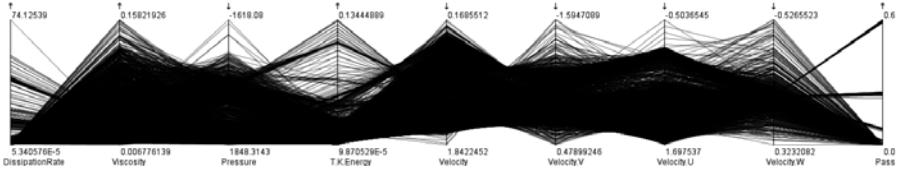


Fig. 3. 9-dimensional data from computational flow simulation (values from 5400 cells of a T-junction grid), visualized with parallel coordinates

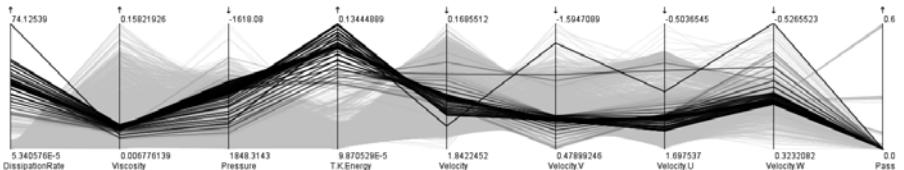


Fig. 4. DOI-based opacity used to visually separate some parts of the data “in focus” (characterized through rather large values of “T.K.Energy”) from all the rest (context)

the use of a standard OTF only) and thus it is possible to achieve better results in shorter time. See Fig. 2 for a sample visualization of segmented 3D chest data with the focus on the lung-object.

In addition to opacity variations, two-level volume rendering also offers alternative ways to achieve a visual focus–context discrimination, for example, by varying the rendering style. But before we furthermore discuss 2IVR, another example for opacity-based F+C visualization is briefly described, which comes from the field of information visualization. Parallel coordinates [19–21] are a well-established technique for the visualization of high-dimensional data. Every n -dimensional data item is plotted as a polyline across n parallel axes in screen space such that a data item’s polyline intersects the axes exactly at those points which relate to the data item’s n attributes (see Fig. 3 for a sample image).

When many data items have to be shown simultaneously (tens of thousands or more), problems with overdraw easily occur: many pixels are covered by several (or even many) polylines. The resulting effect is that the visualization loses effectiveness due to visual clutter – a classical scenario where F+C visualization can help. Using a DOI-based opacity to draw semi-transparent polylines over each other [15], an improved display is gained which allows for interactive analysis of the n -dimensional data (see Fig. 4). Note that the ability to interactively focus in such a F+C application is essential here to effectively exploit the visual superiority of this kind of visualization.

4.2 Reduced Style for Context Visualization

Another option of visually distinguishing between objects in focus and their context is to use different rendering styles. In two-level volume rendering, for example, it is possible to use different rendering techniques for different objects in the data. On the global level, the different representations of the data objects are combined using

Table 3. Visualization properties of different rendering styles for 3D visualization together with a rough assessment of how they can be used for F+C visualization. Depending on whether the focus is inside the context (or outside), or if the context is of complex shape (or a rather coherent object), different combinations of rendering styles yield good results for F+C visualization (details in Sect. 4.2)

rendering style	visualization properties
α -compositing	conveys appearance of semi-transparent 3D medium (F), opacity difficult to control
shaded surface display	well conveys 3D form (F), good transparency control (C)
max. intensity proj.	good for complex forms (F), limited 3D appearance, good transparency control (C)
x-ray rendering	good for overview (C), complex opacity distribution
contour rendering	reduced appearance (C), little problems with occlusion
	F ... good for focus visualization, C ... good for context visualization

standard compositing (α -blending) to achieve the final image. In addition to standard volume rendering, shaded surface rendering, maximum intensity projection (MIP), x-ray rendering, and non-photorealistic contour rendering can be used to depict an object. In Table 3 some visualization properties are listed for different rendering styles in 3D visualization. A good opacity control, for example, favors the visualization as part of the context, because occlusion is easier controlled. The ability to visualize 3D form well, as another example, favors the visualization of data parts in focus. In the following we discuss several useful combinations of different rendering styles for F+C visualization.

Shaded surface display very well acts as visualization of objects in focus, especially if the object(s) in focus are inside the context and, consequently, an opaque surface is used for visualization. This way, usually a strong and sharp appearance of the objects in focus is possible with a good communication of 3D shape. For context visualization, in such a case, the use of contour rendering and/or MIP is very interesting. Contour rendering works fine, because of its reduced appearance (lots of object parts are left away whereas only their contours are shown) and the fact that usually the middle parts of the visualization (where the objects in focus are shown) is rarely occluded (see Fig. 5, right image). Additionally, also MIP usually is useful for context visualization because of its easy-to-control opacity – only one data value per viewing ray is chosen for display, all object representatives share the same opacity (see Fig. 5, left image).

In case of context which is inside the objects in focus, like the bones acting as context to blood vessels (as the objects of interest in angiography), for example, shaded surfaces are doing a good job of focus visualization. The surfaces, however, need to be rendered semi-transparent (at least to some extent) to allow the user to peer inside and get visual access to the otherwise occluded context. MIP again is useful for the depiction of the context objects (good transparency control) – see Fig. 6 (left image) for a sample rendering of such a situation. Similarly, an x-ray simulation

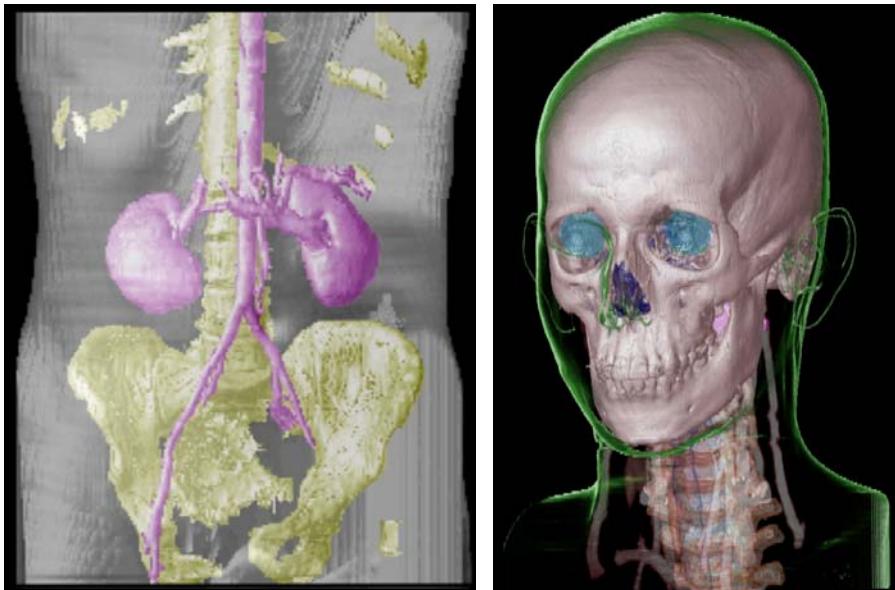


Fig. 5. MIP is useful for context visualization (skin on left side) because of its easy-to-trim opacity. Contour rendering works very well for context visualization (skin on the right) because of its reduced appearance (little problems with occlusion)

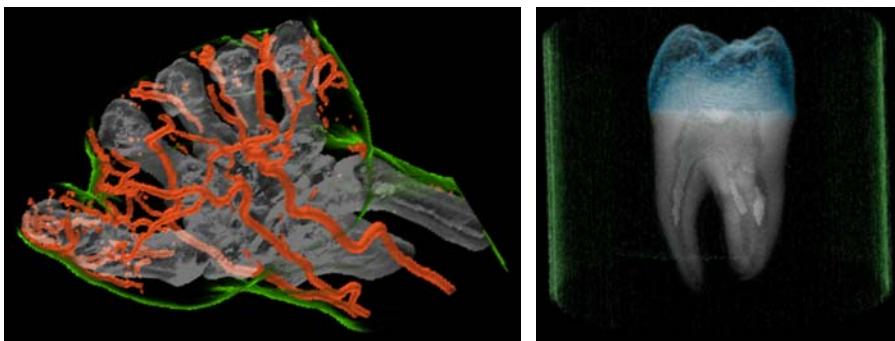


Fig. 6. F+C visualization of CT data of a human hand (left side): the objects of interest (blood vessels) are drawn as semi-transparent surfaces, whereas the bones are rendered using MIP. Contour rendering has been used to depict the skin. An x-ray simulation has been used to depict the dentine of the tooth on the right side (semi-transparent surface rendering of the adamantine and contour rendering of surrounding material)

sometimes is useful for context visualization within objects of interest (see Fig. 6, right image, for an example).

In addition to context rendering, MIP is also useful for depicting objects in focus, especially if they are of complex shape (like an entire system of blood vessels or a chaotic attractor in a dynamical system [1]). In Fig. 7 two examples of such a visu-

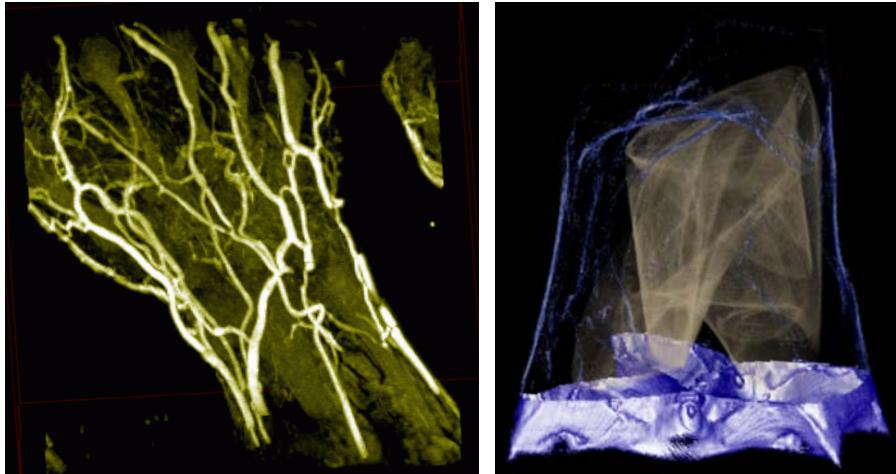


Fig. 7. Two examples of using MIP for complex objects in focus: the system of blood vessels in the CT hand data (left side) and a chaotic attractor within its basin of attraction on the right side (parts of the basin are shown as shaded surface whereas the rest of the basin is shown using contour rendering to minimize occlusion)

alization are given. On the left side MIP is used to show the blood vessels within the CT hand data. On the right side a complex attractor with fractal shape is visualized using MIP. The context (the basin of attraction, in this case) is shown in two ways: whereas the lower parts are shown as a shaded surface, the upper parts are provided using contour rendering only (to reduce problems with occlusion).

4.3 Eye-catching Colors for Focus Visualization

In addition to opacity and style as discussed in the previous two sections, also color is effectively used to focus within a visualization. From perceptual research on preattentive vision [44, 45] we know, for example, that human observers can very quickly “find” colored parts in a visualization of otherwise gray-scale representations – the “search” succeeds even before the observer actually starts searching in an active manner, i.e., in a time usually shorter than 200 ms from stimulus. Accordingly, coloring some parts of a visualization (which are in focus) and showing all the rest in a gray-scale way, also works fine as a F+C visualization technique.

Gresh et al. presented a system called WEAVE [12] which uses this style of F+C visualization for the display of complex simulation data of a beating human heart. Different views of different types of visualization (a scatterplot, a 3D view, etc.) are used to depict and analyse the multi-dimensional simulation data. To assess the large amount of data, the user is able to select certain data subsets of special interest. These parts of the data are then drawn in color whereas all the rest is displayed in gray-scale style. First of all, the colored parts of the visualization immediately stand out of every view where this kind of focus–context discrimination is used. Secondly,

the coloring is done consistently across all the views, so visual linking is established between the views. The same color always indicates the same selection of the data (focus), just visualized differently according to the different views (thereby different characteristics/dimensions of the same data are visualized in the different views). In information visualization this approach is called *linking and brushing* (L&B) – “brushing”, because the process of selecting a data subset of interest usually is done directly on one of the linked views, similar as in a drawing program.

In a system called SimVis [5–8], we use this approach to visualize data from computational simulation of processes in the automotive industry. An extended brushing technique called *smooth brushing* [6] allows for a gradual transition of the *doi*-function from the subset of interest (focus, $doi=1$) to the rest (context, $doi=0$). For visualization, a gradual reduction of color saturation is used to reflect the continuously diminishing degree of interest. See Fig. 8 for a sample result of this kind of visualization, where a data subset of high pressure and high velocity was selected using smooth brushing in the scatterplot on the right. On the left, a visually linked 3D view shows where those areas of high pressure and high velocity lie in the 3D flow domain (a model of a catalytic converter). In addition to the DOI-based variations of color saturation also the glyph size is varied according to the data item’s degree of interest (the more interesting the bigger the glyphs used).

In Fig. 9 volume rendering on the basis of α -compositing [16] was used to depict a subset of a flow through an extended T-junction (characterized through values of high temperature and high turbulent kinetic energy, see scatterplot on the right). In Fig. 10 another sample snapshot from an interactive visual analysis session (using SimVis) is shown [8]. A scatterplot (on the lower left) and a histogram (in the middle) are used to focus on the oxidation front within a diesel particle filter (characterized by lots of carbon oxides, i.e., oxidation products, and high temperatures). The linked 3D view shows the spatial location of the oxidation front at a certain point in time (35 secs. after the simulation start). In the upper left a tree view is visible which provides direct access to the focussing information, i.e., the *doi* attributions of the data as related to the current analysis step.

In a system called GeoSpace [35], user queries are answered visually through high-lighting the data parts in focus, i.e., those data items which positively respond to the user query. High-lighting is done, for example, by increasing the color lightness. Thereby the selected data subsets visually stand out from the rest of the depiction. In two-level volume rendering, this approach is used to provide feedback to the user during object selection in the 3D domain. For a short time after the selection of an object in the scene, the selected object is shown with a different transfer function (increased color lightness, increased opacity). Thereby a clear visual linking between an object’s name or ID and its visual representation as part of the visualization is established (see Fig. 11).

This is especially useful, when volume visualization is performed in a virtual environment. In this case, especially when 3D objects have to be selected directly through 3D user interaction (for example, by the use of a 3D pointing device), object high-lighting greatly supports the interactive placement of the 3D pointing device. While moving the pointing device, the user immediately gets feedback on which

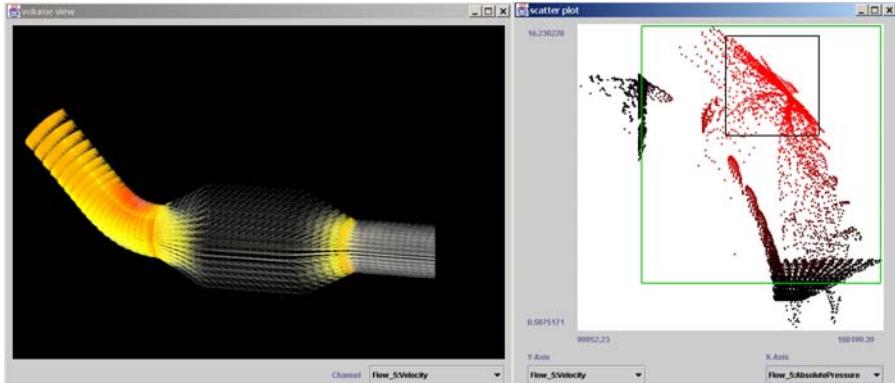


Fig. 8. F+C visualization of CFD data (flow through a catalytic converter). A data subset, represented by values of high pressure and high velocity, has been selected by smooth brushing on the scatterplot on the right. Gradual changes of color saturation on the left (in the 3D view) represent the smooth degree of interest



Fig. 9. Visualization of flow through a T-junction. The visualization focuses on a flow subset which is characterized by high temperature and high turbulent kinetic energy. The junction-geometry is added as context (contour rendering)

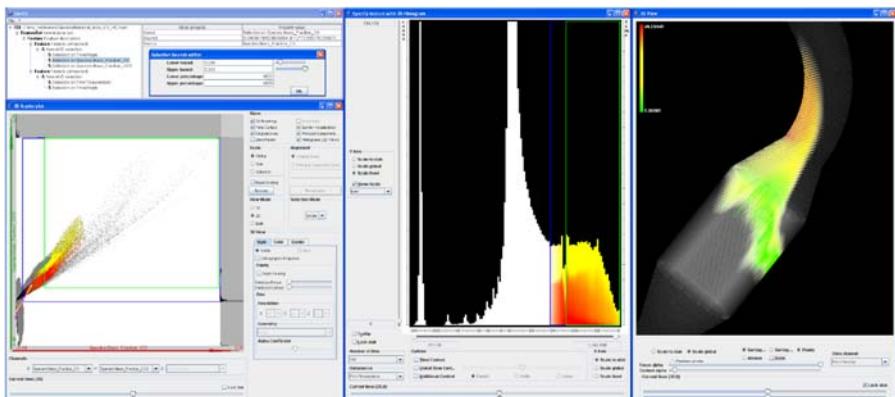


Fig. 10. Flow through a diesel particle filter: a scatterplot and a histogram are used to focus on hot flow which also exhibits large amounts of carbon-oxides (oxidation products CO & CO₂). The 3D view shows the spatial location of the oxidation front at time 35 secs. after the simulation start (color shows velocity magnitudes)

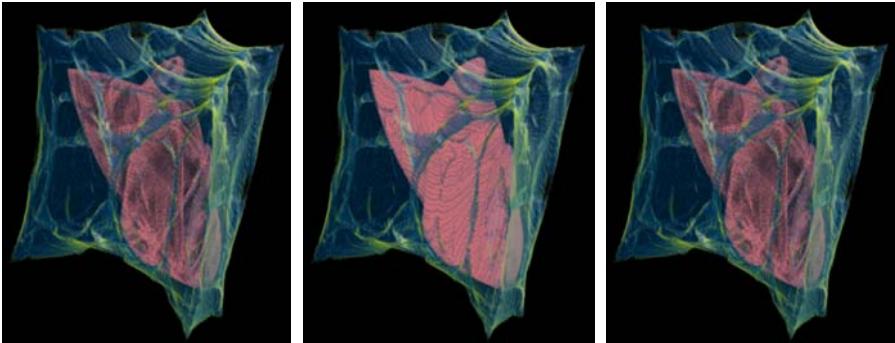


Fig. 11. Object high-lighting in the course of object selection: before the selection ($t = t_0$, left image), right after the selection of the chaotic attractor ($t = t_0 + \approx \frac{1}{2}$ sec., middle image), and a little later after high-lighting ($t = t_0 + \approx 1$ sec., right image)



Fig. 12. Several snapshots from a video which was taken through a session where the user moved a 3D pointing device across a 3D dataset of a human chest in a virtual environment. Visual object high-lighting reflects the current 3D position of the 3D pointing device which is very useful to efficiently position the device in 3D space during object selection

object the pointer currently is pointing towards. Thereby, the user is easily able to efficiently select the one object of special interest without a lot of trial and error (which otherwise is quite normal for 3D direct selection). Figure 12 gives a number of snapshots of a video which was taken during a session where the user moved a 3D pointing device around a 3D dataset of a human chest with different segmented parts of the data. Whenever the 3D pointing device enters another object in the scene, the respective object is rendered in a high-highlighted fashion according to the above mentioned transfer function alternation.

4.4 Band-limited Context

Before we come back to the traditional way of F+C visualization (Sect. 4.5), we furthermore describe one additional way of visually discriminate the visualization of data parts in focus from all the rest (context). Again (as compared to the use of eye-catching colors for focus visualization, see Sect. 4.3) it is an argument from perceptual psychology which motivates this alternative approach: the difference between a sharp and blurred object depiction efficiently can be used for visual focus–context discrimination [25, 26], a technique we call *semantic depth of field* (SDOF). In a user study we could prove that the perceptual identification of sharp objects among

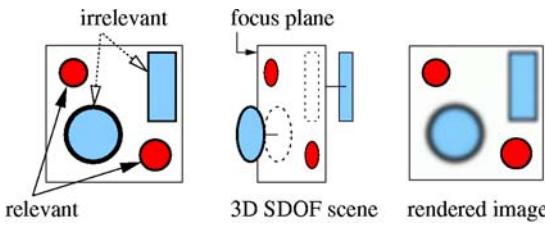


Fig. 13. The basic idea of SDOF for 2D visualization: assuming a lens-based camera model for rendering, the visualization objects are virtually moved back or forth along the viewing direction to achieve a blurred and sharp depiction for irrelevant and relevant data items, respectively

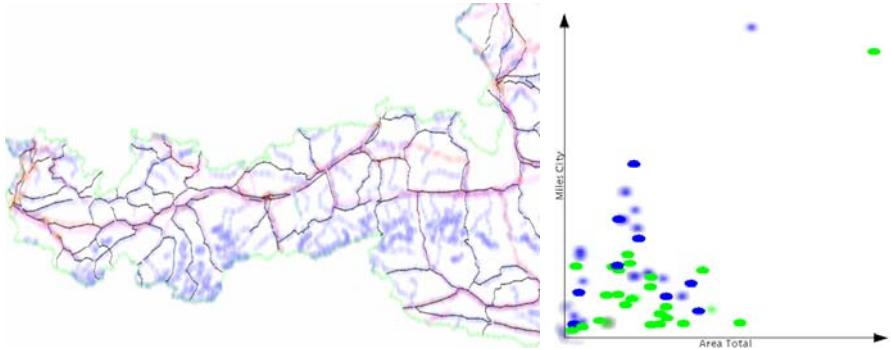


Fig. 14. Two examples of an SDOF visualization: streets standing out of an SDOF map visualization on the left (other parts of the map blurred) and a scatterplot with SDOF effect on the right

blurred others indeed is preattentive [27], i.e., is performed by the human perceptual system within a very short time ($<\approx 200$ ms).

In 2D, the basic idea of SDOF (semantic depth of field) is (a) to assume a camera model with a depth-of-field effect in rendering and (b) to virtually displace parts of the visualization along the viewing axis to achieve a blurred or sharp depiction of irrelevant and relevant parts of the data, respectively (see Fig. 13). With a lens-based camera, objects are only displayed sharply if they are located at the focal distance from the lens. Objects which are displaced along the viewing axis are blurred according to their distance from the lens. Therefore, the displacement in the depth direction is done according to the degree-of-interest values which are associated to all the data elements (and not as a spatial function of the data as it is in real-world photography). For 3D visualization, a similar SDOF model exists [25, 26].

Confronted with the result of such a SDOF visualization (see Fig. 14), the user can immediately identify the data subsets in focus (similar to photography where sharpness also directly correlates to the fact of being in focus). Therefore, this kind of F+C visualization becomes especially useful when the DOI assignment is done implicitly, e.g., through brushing of invisible dimensions (with a range slider, for example) or through defining the DOI value by how well a data item matches a certain user query [2]. In all these cases the first task a user usually performs is

to identify which data items actually have been assigned a high DOI value (and which not). With SDOF this is easily possible as the sharp parts of the visualization, representing the relevant data items, stand out of the depiction automatically.

4.5 More Space for Details

After discussing four alternative ways of realizing focus–context discrimination in visualization (based on the variation of opacities, styles, colors, and frequencies), we come back to the traditional way of F+C visualization, i.e., to the variation of magnification factors within a single image. This kind of completes the picture of our generalization. In Sect. 2 we already discussed the extensive block of literature on this kind of F+C visualization. In our case, we have applied this classic principle to process visualization [37] where this has not been done before.

In process visualization, data which is streaming in from a larger number of processes has to be presented to a user such that process surveillance as well as interactive analysis is possible. In analogy to traditional process visualization, where processes are visualized with analog instruments like gauges or other display devices, programs for process visualization (at least partially) mimic this kind of visualization with virtual instruments. One disadvantage of virtual instruments is that they take up quite a lot of screen space. When multiple streams of process data have to be shown simultaneously, not enough screen space is available to show all the data with regularly sized instruments. In such a situation, distortion-based F+C visualization becomes useful.

To achieve F+C visualization of process data, several levels of detail have been designed for the different virtual instruments in use. The different levels of detail use different amounts of screen space, ranging from a small lamp, color-coding the process data, up to a fully fletched virtual instrument, using a hundred times the amount of screen space as compared to the lamp. If not all of the data can be shown at the highest level of detail simultaneously (due to lack of screen space), different levels of detail can be combined according to DOI values of the different data items. See Fig. 15 for an example, where three streams of process data are visualized. DOI values inversely correlate to the distance between the respective virtual instrument and the pointer which is interactively moved by the user (from left to right). Thereby, those virtual instruments which are nearest to the pointer are displayed at the highest level of detail whereas with increasing distance from the pointer lower levels of detail are used.

In process visualization, data usually originates at concrete 3D locations like a sensor at a certain place or a simulation output with a specific 3D position. Accordingly, the visualization of process data can be organized on the screen such that this relation between the virtual instruments and the related 3D model becomes obvious. In a prototype implementation of F+C process visualization, we first draw the underlying 3D model as a wire-frame rendering. Then, the virtual instruments are shown on top of the wire-frame model at those screen coordinates which correlate to the screen-projection of the corresponding 3D locations of the data sources (see Fig. 16).

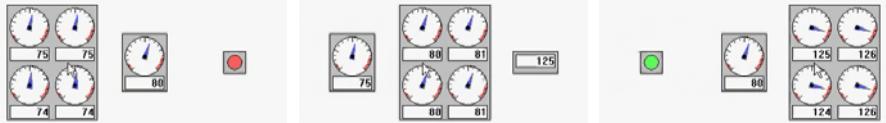


Fig. 15. F+C process visualization: depending on where the user points, the virtual instruments are drawn at a smaller or larger level of detail (from left to right: the pointer is moved from left to right)

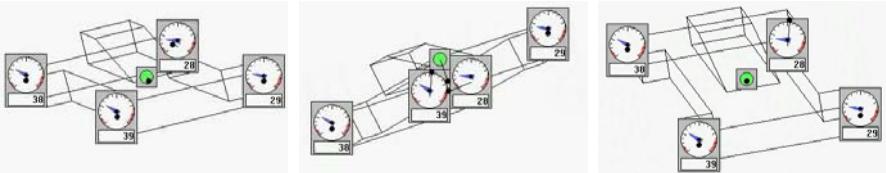


Fig. 16. 3D anchoring and collision avoidance in F+C process visualization: virtual instruments are placed at the screen-projection of that 3D point which is related to the data origin, for example, a sensor (3D anchoring); to avoid cluttering due to overlapping dials a physically-based spring model is used to relocate instruments such that they do not overlap (collision avoidance)

With such a layout strategy (called *3D anchoring* – the virtual instruments are “anchored” at their respective 3D source locations), it can easily happen that screen projections of sensor locations lie near each other such that a naïve implementation of 3D anchoring would cause overlapping virtual instruments. In our prototype implementation we therefore use a physically-based spring model to resolve for non-overlapping instruments (collision avoidance). See Fig. 16 for three snapshots of this prototype which were taken while the user rotated the 3D model (the black dots, which are connected to the centers of the instruments with black lines, mark the screen-projections of the 3D anchors, i.e., the 2D locations where in the optimal case the virtual instrument should be displayed).

5 Interaction

Focus+context visualization requires interaction. Most important, the user needs to have interactive means to focus in a F+C visualization, i.e., he or she needs to steer which parts of the data have to be shown in focus. Accordingly, focussing also includes interactive means to navigate in the visualization, i.e., to change from the visualization of one part of the data (in focus) to another. For applications of F+C visualization, different approaches to focussing are available (see Table 4 for an overview of some of them), which can be classified with respect to several different aspects. One question is of whether focussing is done directly on the visualization (or not). Another question is of whether focussing is done explicitly, i.e., by either directly brushing the data items of interest or naming them explicitly. Thirdly, the question of whether the user actively performs the focussing (or the system does it for the user) also classifies the different approaches to focussing.

Table 4. Different approaches to focussing – techniques can be classified according to whether they act directly on the view (or not), their definition is explicit (or implicit), or whether they are triggered by the user (or not). This differentiation is discussed in more detail in Sect. 5

focussing	action	selection	user	sample applications
brushing	on the view	explicit	active	SimVis [5–8], parallel coordinates [15]
pointing				RTVR [38], process visualization [37]
selection	off-view			SDOF [25, 26], RTVR [38]
range slider		implicit		SimVis [5–8], SDOF [25, 26]
querying				SimVis [5–8]
plot-based		both	passive	SDOF [25, 26]
alerting				process visualization [37]

Most intuitive, *explicit selection* of especially interesting data subsets *directly on the view* results in a (new) specification of the current focus. Prominent examples of this kind of focussing are *brushing* on the one side (as used, for example, in the previously described SimVis system) and *pointing* on the other side (used in F+C process visualization as well as in 3D visualization using RTVR). Similarly, the user can *explicitly focus* by selecting objects through an *off-view list of objects* (as used in volume visualization using RTVR, for example, and the SDOF-visualized map viewer where layers can be selected off-view).

More complex, and a little less intuitive, *implicit selection* also serves for focussing. In the simpler case, selections on invisible axes can be used to describe what currently is most interesting (as also used in SimVis, for example). Alternatively, also complex queries can be used to achieve implicit focussing. Again SimVis is an example: a so-called feature definition language has been developed for the purpose of formally describing what actually is of greatest interest to the user [5].

In addition to methods where the user actively steers which parts of the data are to be visualized in focus, there are other cases, where the system has this role. In a tutoring system, for example, a predefined plot describes which parts of the visualization are in focus at which point in time. This kind of focussing was used in a chess tutoring system with the purpose of showing historic competitions to moderately experienced users (see Fig. 17). In F+C process visualization it is possible to let the system assign DOI values according to whether (or not) the values of a certain sensor lie inside (or outside) a certain safety interval. In case of an alert (value out of range) the user immediately is confronted with a F+C display where most visual emphasis is put on the values in question.

6 Summary and Conclusions

Taking a step back, we can try to round up the matters discussed up to now and to summarize the most important points addressed. In the beginning we started out with a discussion of the well-established approach of *focus+context visualization*

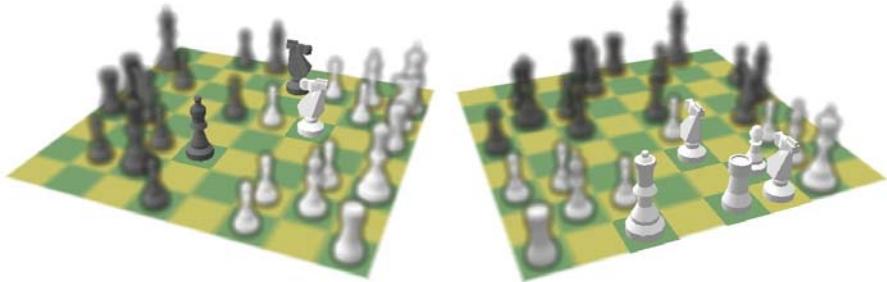


Fig. 17. SDOF-visualized chess tutoring system: through selective sharpness the system shows which pieces threaten (left image) or cover (right image) the white horse on E3

(F+C visualization) as known from information visualization. It is usually associated with the process of providing more space in a visualization for the detailed depiction of some selected parts of the data (those in focus) while still showing the rest of the data in reduced size to provide context information for better orientation and navigation.

This idea of integrating data subsets in focus with their respective context within one visualization also can be found in other fields, especially in scientific visualization. There, however, usually other means than space distortion are used to achieve F+C visualization. In scientific visualization the spatial arrangement of a visualization is tightly coupled with the spatial arrangement of the data origin, e.g., the 3D layout of patients in medical applications or the 3D setup of a flow simulation, and therefore usually resists uneven distortions. In volume visualization, for example, the use of opacity is varied to achieve F+C visualization of 3D data. In the 3D visualization of segmented data (two-level volume rendering, 2IVR), different styles are used to graphically distinguish between objects in focus and their context. Non-photorealistic contour rendering, for example, is very useful for context visualization. In the visualization of data from computational simulation (WEAVE, SimVis), the use of eye-catching colors (within a gray-scale context) also very well serves for F+C visualization. Similarly, the differentiation between a sharp and blurred depiction can yield to F+C visualization (SDOF). All this variety of possible realizations of focus+context visualization yields to a more general definition of F+C visualization: *focus+context visualization is the uneven use of graphics resources*, such as space, opacity, color, frequencies, and style, *for visualization* with the purpose to visually discriminate those parts of the data in focus from all the rest.

A discussion of several concrete examples of different types of F+C visualization shows that often several graphics resources are used to do the focus–context discrimination. In F+C volume visualization by the use of RTVR and 2IVR, for example, in some cases all three of opacity, rendering styles, and coloring are varied to achieve F+C visualization (see Fig. 5, left side, for a sample image). In F+C visualization of 3D data from computational flow visualization (SimVis), coloring, opacity, and glyph size are adjusted according to the DOI values of the data to achieve the desired

visual discrimination (see Fig. 8, left side, for a sample image). Looking through the glasses of our generalized definition of focus+context visualization at the very broad field of applications shows how useful this approach of graphically integrating data subsets in focus and their respective context within a visualization actually is and how general its applicability is.

In addition to the discussion about different ways to graphically discriminate focus from context, also the interactive aspect of F+C visualization is discussed. Once, focus+context visualization is established, it immediately becomes essential to provide sufficient interactive means for focussing, i.e., to select which parts of the data actually are to be drawn in focus or to navigate through a F+C display. Different options of how to categorize focussing with respect to how it is done (on the view vs. off-view focussing; explicit vs. implicit selection; active/passive user) help to give an overview about available strategies. Another way of looking at focussing, however, is to differentiate user goals: whereas in one case the user wants to see more (details) of certain data subsets (→ space distortion, style variations), in other cases the user just wants to visually emphasize the graphical depiction of certain parts (→ opacity, color variations). In again other cases, the visualization goal is to visually attract the user towards a certain subset of the visualization (→ SDOF, coloring, space distortion). Sometimes, these goals do overlap in an application or are followed upon each other during analysis (first the user needs to be attracted, for example, to a sensor out of range, then the user wants to investigate this sensor data in more detail).

Despite the main result that focus+context visualization indeed is in general applicable and useful (almost regardless of the application field), another conclusion of this work is that scientific visualization and information visualization do not lie far apart from each other, but can mutually support each other. There are very good ideas on both sides and visualization systems which integrate approaches from both fields can gain superb advantages over pure SciVis- or InfoVis-solutions [14].

Acknowledgments

This work is based on a lot of related work which would have been impossible without the great contributions of many. To name just a few of them, grateful thanks go to Lukas Mroz, Csébfalvi Balázs, Gian-Italo Bischi, Berk Özer, Anton Fuhrmann, Helmut Doleisch, Martin Gasser, Matej Mlejnek, Markus Hadwiger, Florian Ledermann, Robert Kosara, Silvia Miksch, Krešimir Matković, Wolfgang Rieger, Wolfgang Meyer, and especially to M. Eduard Gröller and Werner Purgathofer for their patient advice throughout all the years which have been related to this work. For funding, thanks go to K plus, an Austrian governmental funding program, which is supporting VRVis since year 2000 and thus also is responsible that most of the work discussed here actually could be done. For more information, see related papers [4–6, 8, 15–18, 25–27, 37, 38] and <http://www.VRVis.at/vis/>.

References

1. Hamdy Agiza, Gian-Italo Bischi, and Michael Kopel. Multistability in a dynamic Cournot game with three oligopolists. *Mathematics and Computers in Simulation*, 51:63–90, 1999.
2. Christopher Ahlberg and Ben Shneiderman. Visual information seeking: Tight coupling of dynamic query filters with Starfield displays. In *Proc. of ACM CHI'94 Conf. on Human Factors in Computing Systems*, pp. 313–317, 1994.
3. M. Sheelagh Carpendale, David Cowperthwaite, and David Fracchia. 3-dimensional pliable surfaces: For the effective presentation of visual information. In *Proc. of the ACM Symp. on User Interface Software and Technology*, Information Navigation, pp. 217–226, 1995.
4. Balázs Csébfalvi, Lukas Mroz, Helwig Hauser, Andreas König, and Eduard Gröller. Fast visualization of object contours by non-photorealistic volume rendering. *Computer Graphics Forum*, 20(3):C 452–C 460, 2001.
5. Helmut Doleisch, Martin Gasser, and Helwig Hauser. Interactive feature specification for focus+context visualization of complex simulation data. In *Proc. of the Joint IEEE TCVG – EG Symp. on Visualization*, pp. 239–248, 2003.
6. Helmut Doleisch and Helwig Hauser. Smooth brushing for focus+context visualization of simulation data in 3D. *Journal of WSCG*, 10(1):147–154, 2002.
7. Helmut Doleisch, Michael Mayer, Martin Gasser, Peter Priesching, and Helwig Hauser. Interactive feature specification for simulation data on time-varying grids. In *Proc. of Conf. Simulation and Visualization*, pp. 291–304, 2005.
8. Helmut Doleisch, Michael Mayer, Martin Gasser, Roland Wanker, and Helwig Hauser. Case study: Visual analysis of complex, time-dependent simulation results of a diesel exhaust system. In *Proc. of the Joint IEEE TCVG – EG Symp. on Visualization*, pp. 91–96, Konstanz, Germany, May 2004.
9. William Augustus Farrand. *Information Display in Interactive Design*. PhD thesis, University of California, Los Angeles, CA, 1973.
10. George Furnas. The Fisheye view: A new look at structured files. Technical Memorandum #81-11221-9, Bell Labs, 1981. Reprinted in Card et al., *Readings in Information Visualization: Using Vision to Think*.
11. George Furnas. Generalized Fisheye views. In Marilyn M. Mantei and Peter Orbeton, editors, *Proc. of the ACM Conf. on Human Factors in Computer Systems*, SIGCHI Bulletin, pp. 16–23, 1986.
12. Donna Gresh, Bernice Rogowitz, Raimond Winslow, David Scollan, and Christina Yung. WEAVE: A system for visually linking 3-D and statistical visualizations, applied to cardiac simulation and measurement data. In *IEEE Visualization 2000*, pp. 489–492, 2000.
13. Markus Hadwiger, Christoph Berger, and Helwig Hauser. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *Proc. of IEEE Visualization 2003*, pp. 301–308, 2003.
14. Helwig Hauser. Towards new grounds in visualization. *ACM SIGGRAPH Computer Graphics*, 39(2), 2005.
15. Helwig Hauser, Florian Ledermann, and Helmut Doleisch. Angular brushing for extended parallel coordinates. In *2002 IEEE Symp. on Information Visualization (InfoVis '02)*, pp. 127–130. IEEE, October 2002.
16. Helwig Hauser and Matej Mlejnek. Interactive volume visualization of complex flow semantics. In *Proc. of the 8th Fall Workshop on Vision, Modeling, and Visualization*, pp. 191–198, München, Germany, November 2003.

17. Helwig Hauser, Lukas Mroz, Gian-Italo Bischi, and Eduard Gröller. Two-level volume rendering - fusing MIP and DVR. In *Proc. of IEEE Visualization 2000*, pp. 211–218, 2000.
18. Helwig Hauser, Lukas Mroz, Gian-Italo Bischi, and Eduard Gröller. Two-level volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):242–252, 2001.
19. Alfred Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1(2):69–92, 1985.
20. Alfred Inselberg. A survey of parallel coordinates. In Hans-Christian Hege and Konrad Polthier, editors, *Mathematical Visualization*, pp. 167–179. Springer Verlag, Heidelberg, 1998.
21. Alfred Inselberg and Bernard Dimsdale. Parallel coordinates: a tool for visualizing multidimensional geometry. In *Proc. of IEEE Visualization '90*, pp. 361–378, 1990.
22. Naftali Kadmon and Eli Shlomi. A polyfocal projection for statistical surfaces. *The Cartography Journal*, 15(1):36–41, 1978.
23. T. Alan Keahey and Edward Robertson. Techniques for non-linear magnification transformations. In *1996 IEEE Symp. on Information Visualization (InfoVis '96)*, pp. 38–45. IEEE, 1996.
24. T. Alan Keahey and Edward Robertson. Nonlinear magnification fields. In *IEEE Symp. on Information Visualization (InfoVis '97)*, pp. 51–58. IEEE, October 1997.
25. Robert Kosara, Silvia Miksch, and Helwig Hauser. Semantic depth of field. In *Proc. of the 2001 IEEE Symp. on Information Visualization (InfoVis 2001)*, pp. 97–104. IEEE Computer Society Press, 2001.
26. Robert Kosara, Silvia Miksch, and Helwig Hauser. Focus + context taken literally. *IEEE Computer Graphics and Applications*, 22(1):22–29, 2002.
27. Robert Kosara, Silvia Miksch, Helwig Hauser, Johann Schrammel, Verena Giller, and Manfred Tscheligi. Useful properties of semantic depth of field for better F+C visualization. In *Proc. of the Joint IEEE TCVG – EG Symp. on Visualization*, pp. 205–210, 2003.
28. Matthias Kreuseler, Norma López, and Heidrun Schumann. A scalable framework for information visualization. In *Proc. Information Vizualization*, pp. 27–36, Salt Lake City, USA, October 2000. IEEE.
29. John Lamping and Ramana Rao. The hyperbolic browser: A focus + context technique for visualizing large hierarchies. *Journal of Visual Languages and Computing*, 7(1):33–35, 1996.
30. John Lamping and Ramana Rao. Visualizing large trees using the hyperbolic browser. In Michael J. Tauber, editor, *Proc. of the 1996 Conf. on Human Factors in Computing Systems, CHI 96: April 13–18, 1996, Vancouver, BC, Canada*, pp. 388–389, New York, NY 10036, USA, April 1996. ACM Press.
31. John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proc. CHI'95*. ACM, 1995.
32. Ying Leung. Human-computer interface techniques for map based diagrams. In *Proc. of the Third International Conf. on Human-Computer Interaction*, volume 2 of *Designing and Using Human-Computer Interfaces and Knowledge Based Systems; Graphics*, pp. 361–368, 1989.
33. Ying Leung and Mark Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction*, 1(2):126–160, June 1994.
34. Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8(5):29–37, 1988.

35. Ishantha Lokuge and Suguru Ishizaki. Geospace: An interactive visualization system for exploring complex information spaces. In *Proc. of the ACM CHI '95 Conf. on Human Factors in Computing Systems*, 1995.
36. Jock Mackinlay, George Robertson, and Stuart Card. The perspective wall: Detail and context smoothly integrated. In *Proc. of ACM CHI Conf. on Human Factors in Computing Systems*, Information Visualization, pp. 173–179, 1991.
37. Krešimir Matković, Helwig Hauser, Reinhard Sainitzer, and Eduard Gröller. Process visualization with levels of detail. In Pak Chung Wong and Keith Andrews, editors, *Proc. IEEE Symp. Information Visualization, InfoVis*, pp. 67–70. IEEE Computer Society, 28–29 October 2002.
38. Lukas Mroz and Helwig Hauser. RTVR - a flexible java library for interactive volume rendering. In *IEEE Visualization 2001*, pp. 279–286, October 2001.
39. Hans-Peter Pfister, Bill Lorensen, Chandrajit Bajaj, Gordon Kindlmann, William Schroeder, Lisa Sobierajski-Avila, Ken Martin, Raghu Machiraju, and Jinho Lee. Visualization viewpoints: The transfer function bake-off. *IEEE Computer Graphics and Applications*, 21(3):16–23, 2001.
40. George Robertson and Jock Mackinlay. The document lens. In *Proc. of the ACM Symp. on User Interface Software and Technology*, Visualizing Information, pp. 101–108, 1993.
41. Manojit Sarkar and Marc Brown. Graphical fisheye views of graphs. In *Proc. of ACM CHI'92 Conf. on Human Factors in Computing Systems*, Visualizing Objects, Graphs, and Video, pp. 83–91, 1992.
42. Manojit Sarkar, Scott Snibbe, Oren Tversky, and Steven Reiss. Stretching the rubber sheet: A metaphor for visualizing large layouts on small screens. In *Proc. of the ACM Symp. on User Interface Software and Technology*, Visualizing Information, pp. 81–91, 1993.
43. Robert Spence and Mark Apperley. Data base navigation: An office environment for the professional. *Behaviour and Information Technology*, 1(1):43–54, 1982.
44. Anne Treisman. Preattentive processing in vision. *Computer Vision, Graphics, and Image Processing*, 31:156–177, 1985.
45. Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, 2000.

Rule-based Morphing Techniques for Interactive Clothing Catalogs

Achim Ebert¹, Ingo Ginkel², and Hans Hagen¹

¹ German Research Center for Artificial Intelligence (DFKI) GmbH, Kaiserslautern,
Germany
`{ebert|hagen}@dfki.de`

² University of Kaiserslautern, Kaiserslautern, Germany
`ginkel@informatik.uni-kl.de`

Summary. In this paper we present the design and the conception of an individual online clothing catalog based on interactive virtual try-on techniques. Our goal is to build a realistic interactive virtual catalog embedded in a web-shop application, which allows customers to view the selected garments fitted onto their own virtual bodies. After the body has been measured in a 3D laser scanning process, the customers can select and combine different garments as well as various colors and patterns. In order to give them the ability to leaf through the offers at an "interactive speed" we avoid time-consuming physically-based simulation methods. Instead, we apply an innovative rule-based morphing technique to generate the different sizes of a garment from one single 3D scan in a given size. Our approach considers standard-sized clothing as well as tailor-made suits, which is an important step towards making made-to-measure clothing affordable for the average customer.

1 Introduction

The clothing industry and clothing stores constitute a very large portion of the volume of sales in the consumer goods market. With the current pricing we are used to in this sector, tracing back to the use of mass production techniques, up to the present customers with individual demands have to put up with disproportional surcharges. These circumstances build the starting point of the project Virtual Try-On, which deals with the effective coverage of the business segment tailor-made suit under the boundary condition of affordability for the average consumer.

A subtask of the project is the development of an interactive individual clothing catalog, particularly consisting of the development of a new innovative rule-based morphing technique which has essential features concerning cloth morphing as its major field of application. The proposed method is making use of agent technology and is linked with a flexible component-based visualization system architecture. It is utilized to efficiently assist a virtual dress fitting process with the target of achieving affordable tailor-made suits for the average customer. The two fundamental inputs for the presented morphing technique are:

- a unique 3D body scan of the customer and an association of the acquired data to suitable clothing measurements.
- 3D reference data by scanning articles of clothing with well-known measurements on a dressmaker's dummy.

The customers are able to choose and combine individual articles of clothing (type of model, cloth, color, pattern and configuration) via a web portal interface at home the same way they could in a store. At this time our system generates an individual 3D model of a dressed figurine, which can be interactively viewed from any perspective.

We are starting this paper with an overview on related work done in this area. Therefore, we examined existing online shops and methods for simulating cloth as well as relevant morphing techniques. In Chap. 3 we are presenting the core elements of our work by describing our morphing techniques for the online clothing catalog. Chapter 4 is addressing the overall system design, the way the virtual dressing is done as well as the implementation details. After presenting our online store prototype and the resulting virtual dressing abilities in Chap. 5 we are closing the paper with the conclusion and future work.

2 Related Work

2.1 Online Shops

Today, most online stores offer virtual fitting rooms based on 2D pictures of garments. In order to present a more interactive and individual way of ordering, some e-tailors have integrated extended functionalities like selecting different sizes, combining several items, changing colors and textures. A good example for an online store based on 2D pictures is the German mail-order firm “Otto Versand” [17]. Here, people have to send in a picture of themselves, which has to fulfill some requirements defined by the “Otto Versand”. As an alternative, a customer can select out of a small number of male or female models that differ in height and stature. The results created by the applied pure 2D algorithms are very unrealistic, especially if the offered zooming is used by the customer. Furthermore, any information about the realistic falling of the clothing is lost by cutting/pasting the design onto the exact body forms of the persons. Figure 1 shows the results of such a virtual dressing.

It is easy to see that the loosely falling blouse in a conventional catalog (2nd picture from the left) is represented as a skin-tight shirt when combined with the model's body (3rd & 4th picture from the left), resulting in a totally false impression presented to the observer. In addition, different sizes aren't handled correctly, making the waist measurement of trousers fitting for every size. In this approach, detailed modifications, such as an individual leg length of trousers, aren't possible. In contrast to this 2D approach, a number of shops are also making use of 3D technologies, e.g. Land's End [12] and Lane Bryant [13] (see Fig. 1 right). Even if the 3D representations are more convincing than a pure 2D visualization, the used 3D avatars only display a poor resemblance to the user. Furthermore, the clothing will



Fig. 1. 2D (left) and 3D (right) approach for a virtual dressing room

always fit, resulting in hardly any benefit for the average customer apart from playing around a little bit.

2.2 Cloth Simulation

Physical techniques applied in the dynamic simulation of cloth represent the cloth as triangular or rectangular meshes of points with finite mass. They can be classified into energy-minimization and force calculation techniques. The energy minimization techniques consider the energy of the cloth as a whole and achieve a local minimum energy state by moving the discrete mass points of the mesh. The force-based techniques calculate the forces acting between the discrete points and integrate over the resulting differential equations using numerical methods to calculate the new position of each point.

The first work on energy minimization techniques was presented by Feynman [5]. His method was extended with a multigrid method by Ng [15], because coarse features are best represented by coarser meshes, whereas smaller ones need finer meshes. Another work on the field of energy minimization techniques was presented by Breen. He derived the energy equations from the Kawabata-measurement System [2]. Besides the simple mass-spring models which have been implemented by various researchers, for example by Howlett [7], the starting point for force-based techniques was the model for elastic deformations by Terzopoulos [19] which is the basis for the work of the Thalmann team and others. A different approach for force-based techniques is to derive the differential equation system from energy formulas like those presented by Breen. Eberhardt [3] took this approach and also added wind and friction between the cloth and rigid objects to the model. There are many more different models which can't be discussed here, a good survey can be found in [16].

The main challenge for a physically-based model of cloth is the construction of an accurate model as well as the calculation of the final shape with fast numerical methods. It is hard to achieve high accuracy together with small computation times. In addition to solving the differential equations, a time-consuming part of the simulation consists of collision detection and collision avoidance. This includes collisions with rigid objects as well as self collisions of the cloth. As there is always a tradeoff

between accuracy and computation time, highly accurate models cannot be computed interactively on consumer hardware. Therefore physically-based simulation is not suitable for our purposes.

2.3 Morphing

Morphing techniques have achieved widespread use in the entertainment industry. 2D image processing techniques have been presented as well as 3D shape transformation algorithms.

Beier and Neely [1] present a morphing algorithm for the metamorphosis of one digital image into another in order to provide a high-level control of the visual effect using natural feature-based specification and interaction. The introduced technique (called field morphing) is based upon fields of influence surrounding two-dimensional control primitives. Compared to other methods, this method is more expressive, but also slower.

The main advantage of 3D morphing techniques over 2D approaches is that objects can be animated independent of the transformation. The 3D methods can be classified into boundary and volumetric approaches, according to the type of information used to calculate the intermediate shapes.

In volumetric approaches, only volumetric information is used to compute the shape transformation of objects. There is no restriction on the topological correspondence between the initial and the final shape. Kaul and Rossignac [10] used a so called Parameterized Interpolating Polyhedron defined by initial and final shapes. The intermediate objects are generated using linear interpolation based on Minkowski sums of the internal points of the two original polyhedra. The technique used by Huges [8] is based on interpolating smoothly between the Fourier transforms of two volumetric models and then transforming the results back. Since linear interpolation between the transformed datasets yields unsatisfactory results in some cases the processing of high and low frequencies are separated during the interpolation process. The most interesting point in this approach is the underlying idea of dissociating the general shape and the details during the transformation. In general these volumetric approaches are robust, but provide no control over the intermediate shapes.

Boundary approaches are applied to polyhedral objects and decompose the shape transformation process into a correspondence problem and an interpolation problem. Since the objects are polyhedral, the solution of the correspondence problem is to compute a correspondence between the topological structures of the two objects. A common technique in all approaches is to build a vertex/face/edge network containing the topological structures of the initial and final shapes. The solution of the correspondence problem provides pairs of vertices, so that intermediate shapes can be produced by simultaneously interpolating between each pair of vertices. The interpolation process can be either performed linearly or for example can use a spline interpolating curve. Both problems are interrelated since the method used to solve the interpolation problem is dependent on the manner in which the correspondences are established.

Kent, Carlson and Parent [11] present an algorithm that, given two 3D polyhedral objects, generates two new models that have the same shape and topology as the original ones, so that the transformation for intermediate shapes can be easily computed. Lazarus and Verroust [14] solve the correspondence problem by sampling the initial and final object. The sampling of the objects is computed using the respectively most suitable parameterization. The correspondence problem is automatically solved by taking the same discretization in the parameter space of the two corresponding parameterizations of the objects. Boundary approaches require significant expense of the solution of the correspondence problem, but allow better control over the intermediate shapes.

Both volumetric and boundary approaches address the transformation of a shape in a global manner. Therefore there is no control over local occurrences in the intermediate shapes. As we want to handle the sleeves of the garments independently of the body, local control is extremely important. In addition, we need to comply with exact measurement specifications, which the intermediate shapes generated by existing morphing techniques cannot provide. Therefore those techniques are not suitable for our purposes.

3 Rule-based Morphing

As we mentioned earlier our goal is to produce virtual humans dressed with realistic garments without making use of physically-based simulation. Our idea is to derive a set of rules that give us absolute and local control over the intermediate shapes in an advanced morphing technique. So we are able to produce every desired size for a given garment model. After the requirements for the derivation of the rules have been formulated in Sect. 3.1 we describe a first basic technique in 3.2 which will be extended and improved in Sect. 3.3.

3.1 Requirements

Instead of applying a cloth simulation the basic clothing model can be generated by draping real clothing over a dressmaker's dummy and to gain the model through a 3D laser scanning process. This produces a model of the cloth with absolute realistic wrinkles. The main disadvantage of this technique is the necessity to produce every garment in every size by laser scanning. To avoid this impracticable time and memory wasting pre-processing process we are proposing an innovative morphing approach to transform the shape of a laser-scanned garment in order to gain models for the different sizes. Here, the desired garments must be scanned in only one basic size and all other sizes will be calculated in the morphing process.

In contrast to existing morphing techniques we must have absolute control over the intermediate shapes as we want to be sure to pass size L when we transform the shape from S to XXL. It must be assured that the intermediate shape that represents size L has the exact associated measures of this cloth-size and not something that just looks similar. Therefore it is necessary to scrutinize the differences between two

Table 1. Example measurements for a shirt (in cm)

	S	M	L	XL	XXL	XXXL
collar size	37/38	39/40	41/42	43/44	45/46	47/48
waist	110	116	124	134	144	154
sleeve length	88	89	90	91	92	93
torso length	82	82	82	85	90	90

sizes (for example between L and XL) in detail. Cloth sizes are usually defined by individual measures like collar size, sleeve length, back width or sleeve circumference (see Table 1). Considering the example of the shirt the sleeve length has to change from 89 cm to 90 cm but the torso length has to stay at 82 cm if the size changes from M to L. Our goal is to derive a set of rules which describe the individual changes that have to be made by the morphing algorithm when we travel from one size to another. Even the simple example of a shirt makes it clear that this is much more complicated than just zooming in and out.

As mentioned before only one piece of real garment will be needed for the scan process. So how can we apply a morphing technique as we would need an initial and a final shape? Our idea is to apply deformation techniques to the shape so that a new size with corresponding measurements is produced. So our approach deforms a shape but morphs between the sizes. A major advantage of this approach over interpolating between two shapes (one of a small size and one of a big size) is that we do not produce additional wrinkles for intermediate shapes if the wrinkles occur at different positions in the initial and final shape. Therefore the computed new size of the garment looks much more realistic than one which would be produced with ordinary morphing techniques. Furthermore, our approach is very flexible and extensible: by adding additional scans of the same garment in other sizes, during the import process the representation which has the most minimal distance regarding the individual measurements can be chosen. Hereby the accuracy is improved going along with higher but tolerable storage requirements.

3.2 Basic Technique

As pointed out in 3.1 individual measures must be changed independently. Thus a morphing technique must provide local control over individual parts of the garment (for example a change at the sleeve must not affect the waist circumference). Additionally it must be possible to obey real sizes. That means if the sleeve has to be made 3 cm longer, the change induced by the morphing process has to be exactly 3 cm.

The morphing technique introduced in this paper is based on a segmentation of a garment, i.e. into upper sleeve, lower sleeve, body front, body back etc. Once this segmentation is calculated we compute a suitable parameterization for each part (see Fig. 2 left). For example, spherical coordinates adapted to the shoulder region and

cylindrical coordinates adapted to the sleeve area are a very reasonable approach. If we allow a polygonal curve instead of a straight line describing the cylinder axis, it can be assured that the axis will be completely inside the sleeve. Changes for the sleeve length can be made by changing the length of the cylinder axis, a new width can be computed by increasing the distance of the sleeve vertices in relation to the axis. Applying this method to every part, the complete morph for a garment is achieved by additive local deformations, what gives us the first formulation of a morphing rule:

$$M_{tot} = M_{sl} + M_{sw} + M_{ww} + \dots + M_{tl}$$

Here, M_{sl} is the morph for sleeve length, M_{sw} is the morph for sleeve width, M_{ww} is the morph for waist width, and M_{tl} is the morph for torso length. In principle it is possible to calculate as many different local morphs as needed and combining them for the whole morph between two sizes.

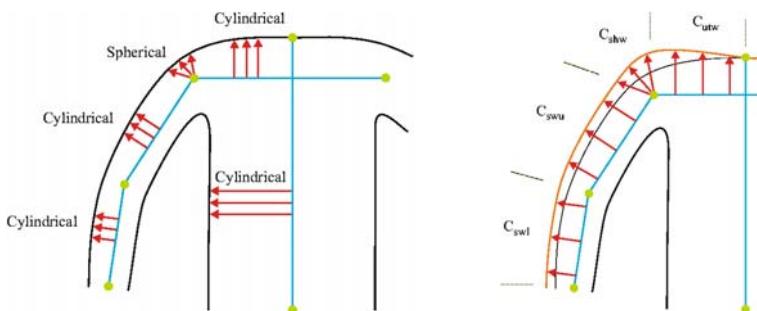


Fig. 2. Basic technique: parameterization, correction terms

A problem that occurs during this process is that smooth transitions between neighboring parts have to be calculated. This is achieved by correcting terms which are added into the morphing process (see Fig. 2 right). For example if the sleeve width is increased we have to apply changes to the shoulder width and to the upper torso part. In addition, we have to translate the whole sleeve to avoid self collisions of the garment at the armpit. The total morph for the change of the sleeve width is:

$$M_{sw_total} = M_{swu} + M_{swl} + C_{shw} + C_{utw} + T_s$$

Here, M_{swu} is the morph for width change at upper sleeve, M_{swl} is the morph for width change at lower sleeve, C_{shw} is the correction of the shoulder width, C_{utw} is the corrections at the upper torso, and T_s is the translation of the whole sleeve.

A similar addition has to be made if for example the length of the upper sleeve is changed. Then a translation of the lower sleeve is necessary. For our purposes linear blending of the correction terms between neighboring parts turned out to be sufficient.

3.3 Extended Technique

In the previous section the core technique for changing the size of garments was presented. To make this approach applicable for practical purposes, sophisticated measures have to be integrated into the morphing process. Furthermore complex coherences that can't be directly derived from the size table have to be considered.

The first extension for integrating sophisticated measures is derived from the following observation: The sleeve circumference is not changed continuously over the whole sleeve by the same amount. Instead, individual measures for the circumference at the shoulder, the elbow and the wrist are given. To interpolate these measures our morphing rule for the M_{swu} and the M_{swl} has to be extended. Linear interpolation between the shoulder measure and the elbow measure for the upper arm is used (elbow and wrist for the lower arm respectively). If we consider the old rule M_{swu} as a function $M_{swu}(x)$ of the amount for the change x it now turns to be a function $M_{swu}(y, z)$ of the two amounts y and z for the circumferences at the upper and the lower bound. The correction for the shoulder area is carried out as before, now depending on the change that was made at the upper bound of the upper arm.

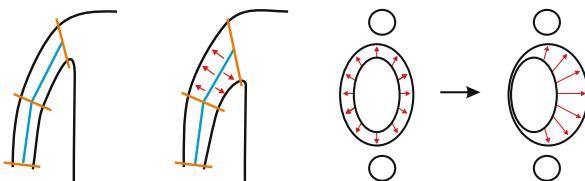


Fig. 3. Extensions: arm width, body width

The second extension is made to the calculation of the body width. The information used here can not be derived from the size tables directly. Instead tailors told us that when the body width is increased this is not done continuously around the garment. The back part stays nearly the same whereas the front part widens up to have a bellied shape. As we have cylindrical parameterization for the points in this area the deformation for the corresponding morphing rule is now dependent on the angle for the individual point in the parameterization. So the deformation is only carried out for those points with an angle that implies a position at the front.

The third extension concerns the change of the body length. Tailors also told us that the changes for the length of the body are only applied to the areas that lie below the armpit. This is due to the fact that we would have unwanted effects in the shoulder area, for example changing the circular appearance of the upper bound of the arm to oval while changing the body in this area. The conversion of this observation into the morphing rule is very easy: As we can find the parameter value x that represents the horizontal plane at the armpits, we apply the change only to points that lie below (see Fig. 4 left).

The forth extension to our set of morphing rules deals with the appearance of the wrinkles. If we change the length of an arm by stretching over the complete

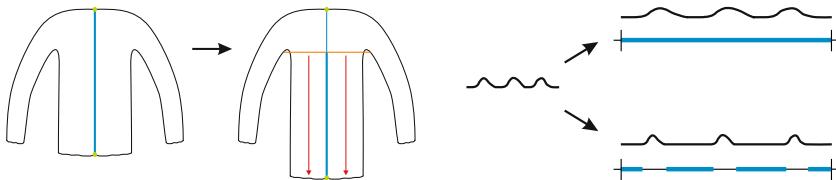


Fig. 4. Extensions: body length, parameterization

parameter space, we would flatten the wrinkles as shown in the upper right part of Fig. 4 (right). If we restrict the stretch to those intervals in the parameter space where no wrinkles occur, or wrinkles are less likely to occur (like at the lower sleeve), we can avoid the flattening effect and the wrinkles are more realistic after the size change (see Fig. 4, lower right part).

The last optimization that we want to present here differs a little from the previous ones. In our morphing approach garments with multiple cloth layers are not considered. This lack of accuracy appears if we have a closer look on attached pockets. In real garment manufacturing they are sewed onto the jacket after the main parts have been assembled. In other words: the size of a pocket does not differ if the jacket size is changed. But this is exactly what happens during the morphing algorithm, because we treat the pocket as a part of the jacket. The pocket would have to be detected as a feature of the shape if we wanted to address this problem geometrically. We prefer to transfer the solution to the texturing calculations, in which the texture coordinates for the pocket are modified, so that the size remains fixed.

To summarize, these extensions to the basic morphing rules denote a significant enhancement of accuracy. Therefore the appearance of the garments is much more realistic. Furthermore the morphing rule can be extended and modified nearly arbitrarily to model different designs of garments. That means if the size changes induced by the size tables are altered, the morphing rules can be adjusted to model the new settings as accurate as possible.

4 System Overview

In Chap. 3, we have presented the core technique that gives us the ability to produce the desired sizes for the garments. We now turn our focus to the overall system design: At first we take a closer look at generating the models of the body by laser scanning. After that, we discuss the virtual dressing process in detail and conclude the chapter with remarks on the implementation.

4.1 System Design

The first step in the described process is the generation of a unique 3D body scan of the customer. We are making use of a VITUS 3D body scanner [20], which is a

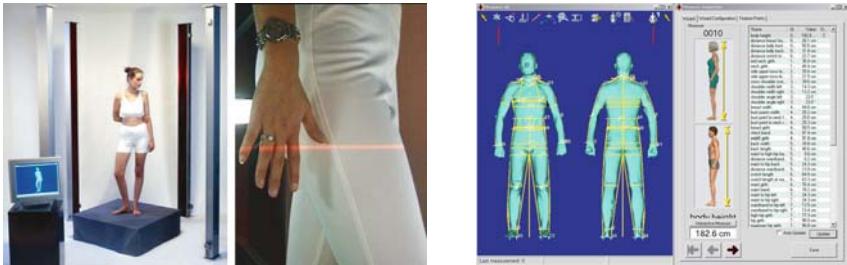


Fig. 5. 3D body scanning

three-dimensional scanner for human beings and other objects. VITUS scans objects within 10 to 20 seconds with a resolution of 1 to 2 mm and provides color textures. Body scanners can be used in many applications, e.g. in the production of customized clothing, in ergonomic research and design or in anthropometric research. In the next step, the individual clothing measurements of the customer are obtained by the ScanWorX tool [9] and stored in a customer database. The scan object together with its measures embodies builds up the virtual customer.

A parallel task is the retrieval of 3D clothing reference data by scanning articles of clothing with well-known measurements on a dressmaker's dummy. The match of the customer's individual choice of clothing to his individual 3D model is achieved by applying our intelligent 3D morphing technique described above.

The customer is now able to mix and match individual articles of clothing (type of model, cloth, color, pattern and configuration) via a web portal interface at home the same way they could in a store. At this time our system generates an individual 3D model of a dressed figurine, which can be interactively viewed from any perspective. Additionally the figurine and the selected clothing can be textured to generate a very realistic representation (see Chap. 5). The graphical output of our system currently supports, but is not limited to Java3D and VRML.

4.2 Virtual Dressing

After computing the different sizes of a garment a couple of problems still remain. First, we have to determine the correct clothing size of the virtual human body and second, we have to dress the figurine virtually. The assignment of correct clothing sizes is done by Human Solutions [9], one of our industrial partners. Their measurement tool is able to automatically analyze the virtual body and to generate various measures (like waist-width, leg length etc.) as well as feature points (like position of the elbows, shoulder, neck and others – all together over 50 feature points). The correct clothing size can now be determined by merging the gathered measurement data. Together with similar feature points for the garments this information is also used for the positioning of the garments around the body - a jacket, for example, is fixed at the shoulder area. However, finding the correct position around the shoulders does not necessarily imply a correct positioning of the clothing around the arms (see



Fig. 6. Virtual dressing

Fig. 6, second from left). This is due to the fact that people will always differ a little in their posture during the scan process. After the jacket has been positioned around the shoulder area, we have to correct the posture of the virtual human's arms. First, we move the visible lower arms into a position where they fit inside the sleeve (rotation and translation process). This is done by calculating an axis for the arm similar to the one we already have for the sleeve and matching them together. Depending on the jacket's size a slightly wrong arm length is obtained by this process, that means the arm may be too long or to short afterwards. In order to resolve this problem, a correction operation is applied: since the correct total length of the upper and lower arm is a known parameter, the lower arm can proportionately be moved in axis direction. This results in a new position of the lower arm, but simultaneously maintains the total length of the arm.

The reason why we do not modify the garment but the body is that otherwise the appearance of the wrinkles in the cloth would change significantly. The used deformation technique that is responsible for increasing/decreasing the size of the garment, already produces a little mistake (for example the wrinkles are slightly enlarged when increasing the size of the garment). Therefore, increasing this error factor by applying further deformations is undesirable in our application scenario. Obviously, this is an incorrect mapping of the real world configuration (the body influences the shape of the garment) to our virtual scene, in which the garment influences the body shape. But, for preview purposes our technique clearly reveals the main fitting aspects, namely if for example the sleeve is too short, if it fits or if it is too long (see Fig. 7). A topic that has not been mentioned so far are the possible intersections of the body and the garment due to the fact that no physical simulation was applied in order to avoid time-consuming intersection detections during the try-on process. From observing real life configurations it is clear that there are body parts which are totally hidden by the garment (for example the upper arm is totally hidden if the virtual person is wearing a jacket). Consequently, these parts can be blended out by just making them invisible during the rendering process (see Fig. 6 on the right). The segments that are only partly visible (like the lower arm) are more complicated to handle. Here, we apply standard intersection detection algorithms

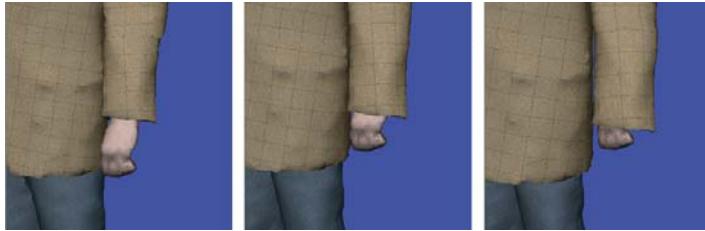


Fig. 7. Visual fitting control: different sleeve lengths

together with standard optimizations like bounding volumes, octrees etc. If a body area is lying outside the garment, the corresponding triangles are made invisible during the rendering process.

But what happens if the virtual human has to try on garments that are by far too small? Our clipping algorithms would just blend out the parts of the body that are outside the garment, presenting the customer a wrong impression: the smaller sizes also seem to fit. Therefore, in our application the sizes selectable by the user are restricted to a predefined range around the calculated optimum size. The customer is still able to see the effect of changing the size, if he/she for example likes a looser fit, but unrealistic degenerated cases which do not give useful information to the user are excluded. It must be stressed out that displaying garments that in fact are too small, is a general problem of virtual try-on applications. Even if a physically-based simulation would be applied, it would be possible to dress a virtual human with too small garments. The real life process of getting dressed also is not simulated – instead, the cloth patches are placed around the body and then sewed together. The way the simulation process is carried out results in too high spring forces in the model that further lead to unrealistic stretching effects. To summarize, it is obvious that a configuration that is impossible in real life should not be feasible in a virtual try-on application.

4.3 Implementation

The complexity of present-day software development cannot be satisfactorily reduced by the concept of object-orientation because of the constraints imposed by particular programming languages and operating systems. Therefore, we are using component-based development technologies, in which a component typically has all the properties of an ordinary object, but in addition can be deployed in different operating environments without change or recompilation. By the use of component technology the strict separation between implementation and provided functionality supports the development of independent and reusable software components, which leads to advantages like dynamic linking, faster application development, smaller development costs, high reliability as well as extremely flexible system architectures.

To benefit from these advantages in our approach we are using our multi agent- and component-based visualization system MacVis [4] as the base development

platform on which the visualization modules are implemented as reusable software components. The MacVis system supports the development of new visualization modules without being tailored to the used visual prototyping environment, including an intelligent control unit capable of automatically supervising and tuning system components during runtime. It uses Sun's Java Beans component technology and requires hardly any extra effort for transferring a Java class into a Java Beans component. In order to offer an easy way for building new applications, a component-based visual prototyping system is integrated, taking advantage of the dynamic linking properties component technology offers. The implemented visualization components are based on Java3D [18], while VRML [22] is used for data exchange.

5 Results

Figure 8 shows the prototypical implementation of our individual clothing catalog environment. The window is divided into four frames. The left frame gives the user the ability to pick the clothing category as well as the favored piece of clothing, while in the middle frame he/she can select out of the possible sizes and patterns. After having selected the garments, the virtual try-on is displayed and can be controlled in the right frame.

The following snapshots show the resulting virtual try-on for a customer wearing different virtual jackets and trousers together with a selection of patterns in different views.



Fig. 8. Individual clothing catalog prototype



Fig. 9. Virtual try-on results

6 Conclusion and Future Work

In this paper we proposed a new rule-based 3D morphing technique which has essential features concerning cloth morphing as its major field of application. In order to be able to independently change individual measures, our approach provides local control over individual parts of the garment. The algorithms have been integrated in a prototypical implementation of an individual clothing catalog. Moreover, the proposed system architecture is flexible enough to be adapted to all kinds of software, hardware and user conditions.

Our future work includes the integration of more garment types as well as optimizations of the used technique. Up to now our shop environment is settled in the area of men's ready-to-wear clothes but should be extended to ladies' wear.

Acknowledgements

This research is supported by the German Federal Ministry of Education and Research (BMBF) and is part of the project Virtual Try-On (#01IRA01A). We like to thank Human Solutions GmbH for providing us with the scan data sets as well as the Hohenstein Institute for intensive discussions with respect to clothing technology and providing us with appropriate size table sets.

References

1. Beier, T., Neely, S. Feature-based image metamorphosis, SIGGRAPH Computer Graphics, 26(2) (1992)
2. Breen, D.E., House, D.H., Wozny, M.J.: A Particle-Based Model for Simulating the Draping Behavior of Woven Cloth. European Computer-Industry Research Centre, ECRC-94-19 (1994)

3. Eberhardt, B., Weber, A., Strasser, W.: A Fast, Flexible, Particle-System Model for Cloth Draping. *IEEE Computer Graphics and Applications* (1996).
4. Ebert, A., Divivier, A., Barthel, H., Bender, M., Hagen, H. Improving Development and Usage of Visualization Applications. In: IASTED International Conference on Visualization, Imaging and Image Processing (VIIP 2001), Marbella, Spain (2001)
5. Feynman, C.: Modelling the Appearance of Cloth. MA Thesis, Dept. of EECS, Institute of Technology, Cambridge, Massachusetts (1986)
6. Hohenstein Institute: <http://www.hohenstein.de/englisch/>
7. Howlett, P.: Cloth Simulation Using Mass-Spring Networks. MA Thesis, Faculty of Science and Engineering, University of Manchester (1997)
8. Huges, J.F.: Scheduled fourier volume morphing. In: Proceedings of the 19th International Conference on Computer Graphics and Interactive Techniques (1992)
9. Human Solutions GmbH: <http://www.human-solutions.de/>
10. Kaul, A., Rossignac, J.: Solid-interpolating deformation: Construction and animation of pips. In: Post, F.H. and Barth, W. (ed) Eurographics '91 (1991)
11. Kent, J.R., Carlson, W.E., Parent, R.E.: Shape transformations for polyhedral objects. In: ACM SIGGRAPH Computer Graphics (1992)
12. Land's End: <http://www.landsend.com/>
13. Lane Bryant: <http://lanebryant.charmingshoppes.com/>
14. Lazarus, F., Verroust, A.: Feature-based shape transformation for polyhedral objects. In: Fifth Eurographics Workshop on Animation and Simulation, Oslo, Norway (1994)
15. Ng, H.: Fast Techniques for the Modelling and Visualization of Cloth. PhD Thesis, Centre for VLSI and Computer Graphics, University of Sussex (1996)
16. Ng, H., Grimsdale, R.L.: Computer Graphics Techniques for Modeling Cloth. In: IEEE Computer Graphics and Applications (1996)
17. Otto Versand: <http://www.otto.de/>
18. Sun Microsystems Inc.: Java3D:
<http://java.sun.com/products/java-media/3D/>
19. Terzopoulos, D.: Elastically Deformable Models. In: Computer Graphics, 21(4) (1987)
20. Vitronic Bildverarbeitungssysteme GmbH: <http://www.vitronic.com>
21. Volino, P., Courchesne, M., Magnenat-Thalmann, N.: Versatile and Efficient Techniques for Simulating Cloth and Other Deformable Objects, MIRAlab, University of Geneva (1997)
22. VRML: <http://astronomy.swin.edu.au/~pbourke/3dformats/vrml1/>

A Practical System for Constrained Interactive Walkthroughs of Arbitrarily Complex Scenes

Lining Yang¹ and Roger Crawfis²

¹ Siemens Medical Solutions

lining.yang@siemens.com

² The Ohio State University

crawfis@cis.ohio-state.edu

Abstract. Complex renderings of synthetic scenes or virtual environments, once deemed impossible for consumer rendering, are becoming available for modern computer architecture. These renderings, due to their high-quality image synthesis, can take minutes to hours to render. Our work focuses on using Image-Based Rendering (IBR) techniques to manage and explore large and complex datasets and virtual scenes. The key idea for this research is to pre-process the scene and render key viewpoints on pre-selected paths inside the scene. We present new techniques to reconstruct approximations to any view along the path, which allows the user to roam around inside the virtual environments with interactive frame rates. We have implemented a pipeline for generating the sampled key viewpoints and reconstructing panoramic-based IBR models. Our implementation includes efficient two-step caching and pre-fetching algorithms, mesh simplification schemes and texture removal and database compression techniques. The system has been successfully tested on several scenes and satisfactory results have been obtained. An analysis and comparison of errors is also presented.

1 Introduction

High-quality renderings of synthetic scenes or virtual environment, due to their complex image synthesis, can take minutes to hours to render. Ray-tracing or global illumination using a tool such as POVRAY [28] and Radiance [25] are very time consuming. An interactive virtual walkthrough of these large and complex scenes is almost impossible on a low to mid-end system using traditional rendering techniques.

Our goal is to allow the user to examine and walkthrough the scene from an internal vantage point on a relatively high-resolution display. To achieve this goal, we decided to apply Image-Based Rendering (IBR) techniques as a post-processing tool for any traditional high-quality renderer.

IBR is a new research area in the computer graphics community, and offers advantages over the traditional rendering techniques. It can utilize real life images and illumination for photo-realistic rendering. It requires a fixed or limited amount of work, regardless of the view or data context. However, this amount of work is proportional to the input image size. Many IBR techniques [2, 13, 14, 21] use the entire

set of input images and therefore can only focus on accurate renderings of relatively low-resolution imagery. Here we explore techniques for large displays having a resolution from $1k \times 1k$ to $8k \times 3k$, as in our new parabolic video wall.

Our objectives are: (1) Accurate results at many pre-selected viewpoints. (2) Smoothly move from one accurate view to another with minimal rendering errors and no disruptive artifacts or popping. (3) Support for extremely high-resolution imagery in the interactive IBR framework. (4) A decoupling of the pre-computed imagery from the resulting viewing platform over high-speed networks. (5) Support for many different rendering tools as a front end. (6) Guaranteed frame-rates regardless of the data size, rendering complexity or display configuration.

Our work can be viewed as an extension to QuickTime VR [3] or other panoramic representations [24]. Panoramic imagery allows one to spin around at their current viewing position, but does not allow the user to move forward or backward. We developed a system to allow movement along a piecewise linear path in three-dimensions. At any position on this curve, the user can interact with the scene as in a panoramic viewer. We termed this type of viewing a rail-track view, in that the user can move forward and backward along the “track”, while viewing the outside scenery in any direction. Darsa, et al [7] investigated techniques to incorporate information about the depth of the image into the panoramic scene. Depth allows for proper re-projection of the pixels from different viewpoints and provides a sense of motion parallax to give a true three-dimensional sense to the imagery. For efficient rendering, we apply mesh simplification methods to simplify the depth image from every pixel to a more manageable geometric quad-mesh. The pre-rendered imagery is then projected to the depth mesh as a texture map. Texture data management and an intelligent caching and pre-fetching scheme are employed to further improve the rendering speed.

Our contributions include: (1) A texture streaming client/server architecture for panoramic walkthrough (2) View-dependent layers to better address occlusion and dis-occlusion problems. (3) Incorporation of mesh simplification and texture mapping hardware for high-quality and high-resolution renderings. (4) Intelligent Data Management, Caching and Pre-fetching schemes. (5) View-dependent IBR texture removal and database compression, considering possible occlusion/dis-occlusion along the track segments.

The paper is organized as follows: First we discuss relevant background and previous work in the IBR area. We then present an overview followed by implementation details of our system. Next we discuss the pre-processing and data organization scheme for efficient rendering. A two-phase caching and pre-fetching technique is also presented in this section. It is then followed by a track dependent occlusion-culling scheme. We will then provide a quantitative measurement of the errors. Finally we conclude with some test results and ideas for future work.

2 Related Work

A lot of effort has been put into designing more accurate IBR systems. This is because IBR has a very important advantage over the traditional geometry-based rendering systems in that it has a bounded computational cost according to the input image resolution.

QuickTime VR [3] is probably the earliest effort of IBR research. It has the ability to allow the user to look around horizontally and vertically (QuickTime VR only allows 360 degrees in horizontal directions and spherical panoramic systems [24] allow for both horizontal and vertical directions). A QuickTime VR system is simple and very efficient because it only uses implicit geometry relationships to reconstruct the scene. However, it also restricts the user to sit at the pre-defined viewpoint. It projects everything to the same radius cylinder. Darsa et al [7] suggests a way to introduce depth information into the cubical panoramic system to allow for a walkthrough. They use three blending methods to reconstruct the views between two pre-defined viewpoints. Cohen-Or et al [5] introduces the idea of pre-defining a path and pre-computing the geometry and textures for the sampled viewpoints on the path. They use the texture mapping hardware to reconstruct the novel views in between. Both of these systems do not address the occlusion and dis-occlusion problems as described in the Layered Depth Image paper [21]. That is, when the user moves away from the pre-selected viewpoints, some features that were previously occluded in the original viewpoint can become visible. Without multiple layers of depth [9, 21], these systems require several viewpoints to fill in the holes. A dense sampling is needed for this purpose, which increases the database size and puts more burden on storage and network transmissions and loading time. By utilizing multiple layers and culling away unnecessary information, our system can achieve more efficiency in this sense.

Most of the previously introduced IBR systems concentrate on accurate renderings of relatively low-resolution imageries. These systems use per-pixel warping, as described in [13, 14, 19, 21]. Hardware texture mapping is not utilized and therefore the performance is not very fast for larger image resolutions. They are not suitable for our purpose, which is interactive management and walkthroughs of large datasets and complex scenes on a high-resolution (over $1k \times 1k$) display. Examples such as the LumiGraph [11] and Light-field Rendering systems [12] usually sample the viewing parameters very densely, requiring large storage spaces. There are some systems that utilize the texture hardware to accelerate the rendering performance, such as the previously mentioned Darsa et al [7] and Cohen-Or [5]'s work. The View Dependent Texture Mapping (VDTM) [8] is a typical example of using 2D texture hardware to accelerate the renderings. However, they do not sample the viewing direction adequately to allow for panoramic viewing. Our system [26] on the other hand, allows the user to move back and forth on a pre-defined track, with full rotational viewing directions.

3 Overview

The goal of this research is to interactively manage complex, time-consuming renderings of large scenes. We concentrated our efforts to allow a user to roam interactively inside a scene, exploring interesting features with the ability to look around at the same time. We achieve this by restricting the user's movement on a pre-selected path and allow him or her to look around at any point in both the vertical and horizontal directions. Figure 1. illustrates the idea of moving on a pre-selected path. It shows one of the POVRAY dataset: the Nature dataset. The black arrowed curves represent the track and the red dots represent the pre-selected reference view-points which will be pre-rendered and saved into a database. The users are allowed to move back and forth along the tracks and change their viewing directions freely. Although some software packages allow discrete jumps from one viewpoint to another, this disturbing teleportation requires a re-orientation of the user and invalidates any sense of a virtual environment.

Figure 2 illustrates the framework of our IBR system. The system consists of a two-step pipeline. The first step extracts partial renderings for the selected reference views of the scene. The resulting geometry and imagery is pre-processed to a more manageable format and stored on the disk of the server. In the second step, whenever the client needs the data, it sends the necessary request across the network to the server and the server retrieves the related data from the database and sends it down the network to the client. Both the server and the client maintain their own cache for fast data access. In the next several sections we will discuss how to reconstruct the novel views efficiently and smoothly with minimal errors.

4 Visibility Polyhedrons

Moving from one viewpoint in a complex scene to another, with the freedom of looking around, is a challenging problem. Let's assume that we want to move from a view, V_1 , to a view, V_2 , in a complex scene. Consider the following definitions extended from those on polygons [17].

Definition 1. *The visibility polyhedron of a viewpoint is the locus of points visible from the viewpoint.*

Definition 2. *A polyhedron P is said to be **star-shaped** if there exists a point z , not external to P , such that for all points $p \in P$, the line segment zp lies entirely within P .*

Definition 3. *The locus of the points, z , having the above property is called the **kernel** of P .*

Theorem 1. *The visibility polyhedron of a viewpoint is a star-shaped polyhedron and has a kernel that contains at least the viewpoint.*

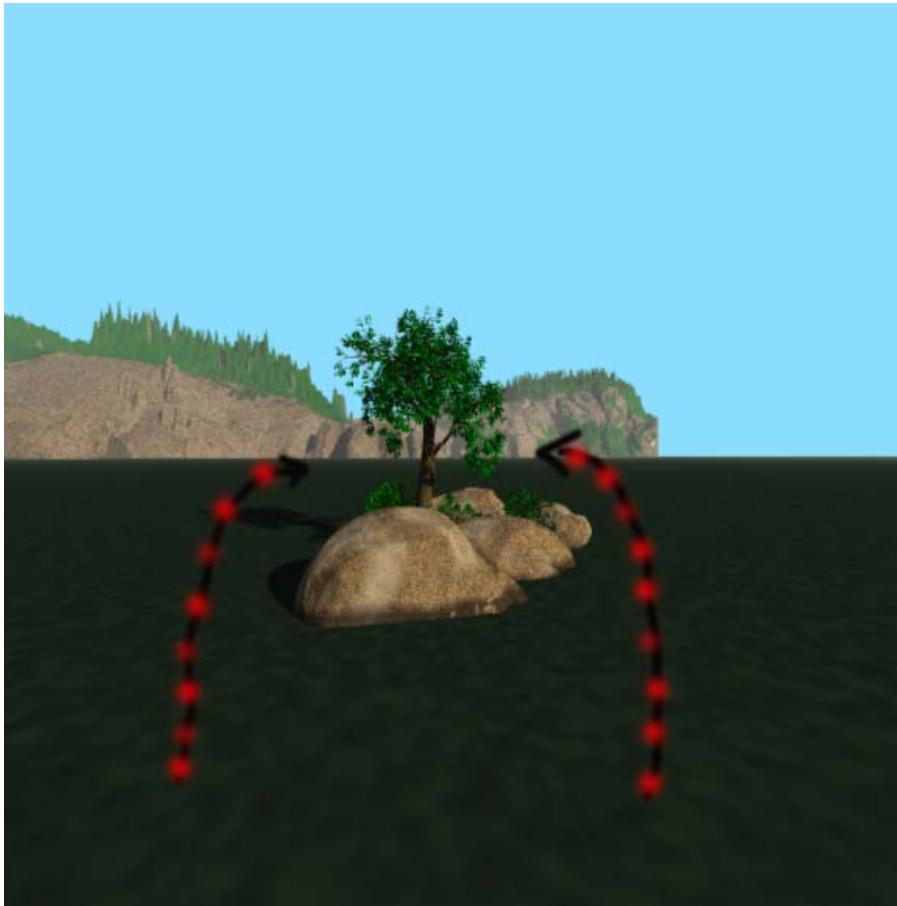


Fig. 1. One pre-selected path for our internal viewer through a model of the Nature scene. The *black curve* represents our pre-selected path in the dataset. *Red dots* represent viewpoints on the path where the plenoptic function is sampled. Users can move along the curve and look around

From these definitions, we can see that any two points inside the kernel of a polygon or polyhedron see exactly the same scene and since all the points on the line connecting these two points are also in the kernel, they likewise see exactly the same scene. Let's assume that the kernel of V_1 's visibility polyhedron contains V_2 and likewise for V_2 . The visibility polyhedrons of the two viewpoints are therefore identical. Any new views, V^* , rendered along the line segment connecting the two points are also inside that kernel, due to above definitions. Therefore, the visibility polyhedron of V^* is the same as V_1 and V_2 . If we define P_{V_i} as the visibility polyhedron of the reference viewpoint V_i and P_{V^*} as the visibility polyhedron of the new view V^* , we can write the equivalence relationship as:

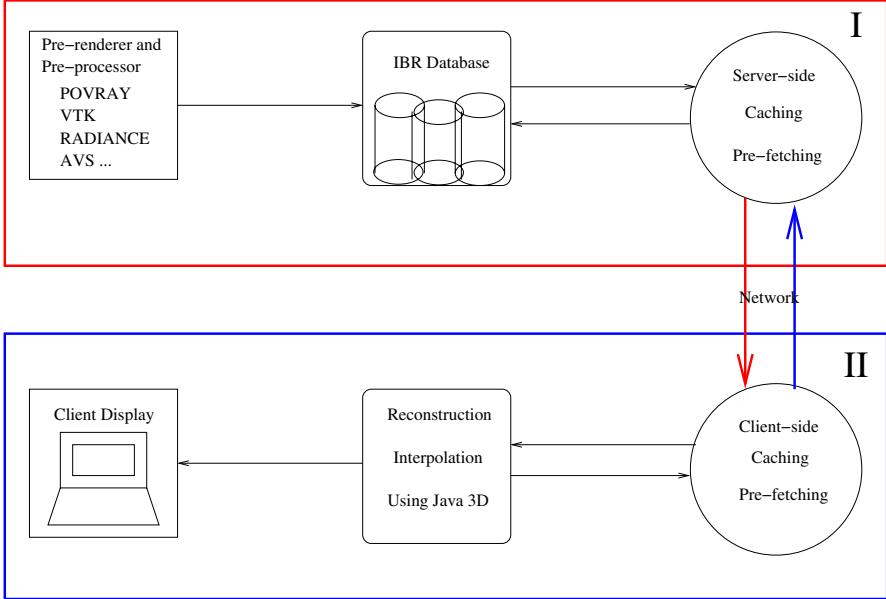


Fig. 2. System diagram – a two-step pipeline is used. The first step uses different rendering engines to pre-render the datasets. The resulting geometries and imageries are pre-processed to a more manageable format and stored on a server. Whenever the client needs the data, it sends a request to the server, the server retrieves the related data from the database and sends it down the network to the client. Both the server and the client maintain their own cache for fast data access

$$P_{V^*} = P_{V_1} = P_{V_2} \quad (1)$$

We can use V_1 's or V_2 's visibility polyhedrons P_{V_1} and P_{V_2} to reconstruct the new view V^* 's visibility polyhedron P_{V^*} .

Note that P_{V_i} assumes a continuous representation of the visibility polyhedrons of the viewpoints and usually are not achievable for current computers system. In practice, we use a pre-renderer to generate the imagery and the associated depth information. The depth values are obtained either from the z -buffer or the first intersection point with each viewing ray, depending on what software package is used. In either case, we connect the depth values to form an approximated polyhedron ZP_{V_i} , which is a discrete representation of the visibility polyhedron P_{V_i} . We texture-map ZP_{V_i} with the pre-rendered imagery and use this as the reconstructed scene for the reference viewpoint. Having the approximations of visibility polyhedrons for all the reference viewpoints, we investigate techniques to reconstruct the information of the in-between novel viewpoints. From (1) we see that we can use the two visibility polyhedrons of the close-by reference views to reconstruct the new view. One way to move from one reference view to another smoothly is to throw the two reference view's polyhedrons ZP_{V_1} and ZP_{V_2} into the z -buffer and perform a simple z -buffer

comparison. The z-buffer comparison chooses the closest z value seen in the novel view. This results in an approximation for the visibility polyhedron of V^* , called ZP_{V^*} , which is actually an intersection operation.

$$ZP_{V^*} = ZP_{V_1} \cap ZP_{V_2} \quad (2)$$

When the two viewpoints lie inside each other's kernel, the two polyhedrons ZP_{V_1} and ZP_{V_2} are identical. z -buffer comparison therefore will not introduce any artifacts. Actually in this case the z -buffer comparison (the intersection operation) is equivalent to using the polyhedron of either reference view to represent that of the new view. This relationship is shown in (3).

$$ZP_{V^*} = ZP_{V_1} \cap ZP_{V_2} = P_{V_1} = P_{V_2} \quad (3)$$

However when the viewpoints are not within each other's kernel, the two visibility polyhedrons are not the same and therefore visibility discrepancy occurs for the two reference viewpoints. If we still use the z -buffer comparison, errors will occur. Consider the example in Fig. 3.

In Fig. 3 (a) we have two viewpoints V_1 and V_2 and two objects O_1 and O_2 . We can see that in Figure 3 (b), V_1 and V_2 are not within each other's kernel and therefore the visibility polyhedrons are not the same. For this example, if we still approximate the in between view, V^* 's visibility polyhedron, by using z -buffer comparison between V_1 and V_2 's visibility polyhedrons, we have problems with some viewing rays. For example, for point p in the figure, p_1 is the depth value in V_1 's polyhedron and p_2 is the depth value in V_2 's polyhedron. If we use z -buffer comparison, p_2 is apparently closer than p_1 . Therefore the depth and color value of p_2 are used for V^* . This is obviously incorrect. The more serious problem of using z -buffer comparison is that: even at the reference view, we may get incorrect results. For the same point p , if z -buffer comparison is used, even at reference viewpoint V_1 , we would use p_2 's value since it is closer and wins the z -buffer fight. Recall that one of our purposes of this research is to guarantee accurate results at reference viewpoints and therefore this is not acceptable.

Another disadvantage of using z -buffer comparison is the popping effect when changing reference views. This is shown in Fig. 3 (c). In this figure, we are moving from V_2 to V_1 then to V_0 . We have the track segments s_0 and s_1 . As discussed before, when the user moves from V_2 to V_1 , p_2 is used to represent p , which is of course incorrect. Immediately after the user moves out of s_1 and into s_0 , p_1 appears in the scene. Recall again that one of our goals is to allow the user move on the rail-track smoothly. This popping effect can seriously distract the user's attention and therefore is not desired.

An alternative way to reconstruct the visibility polyhedron ZP_{V^*} for V^* using the close by reference viewpoints V_1 and V_2 's polyhedrons and is to blend the two polyhedrons. This cannot completely eliminate the errors mentioned before. However it can mitigate the problem and make the transition between the track segments smoother. Again, consider the examples shown in Fig. 3. For the problem shown in

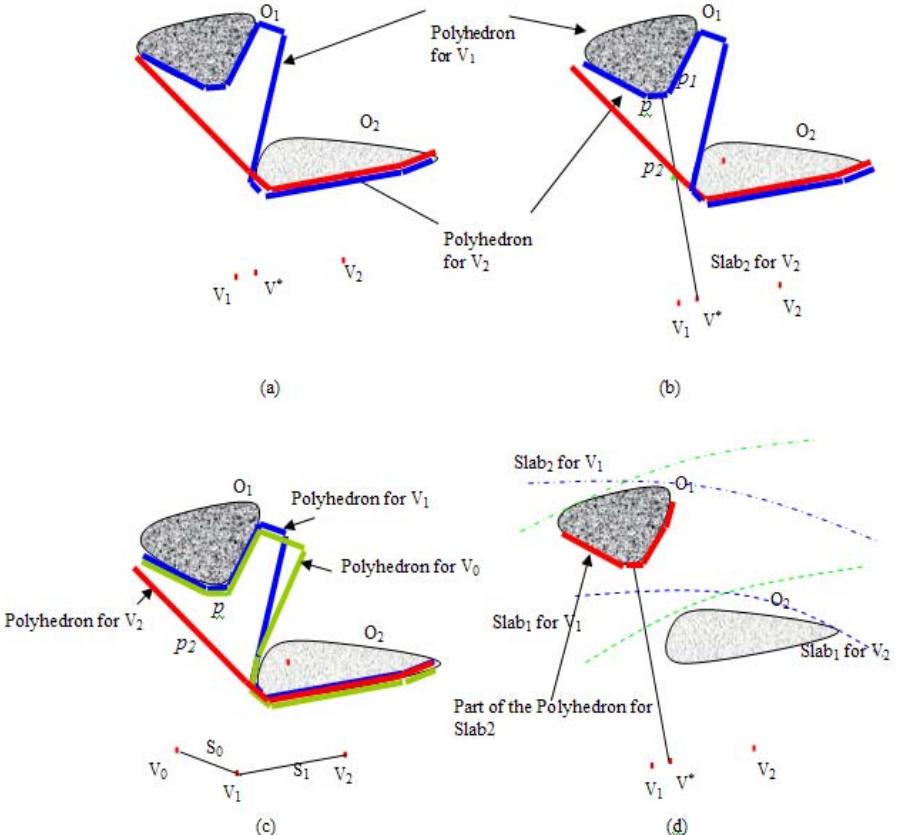


Fig. 3. (a) shows two viewpoints V_1 and V_2 and two objects O_1 and O_2 and the visibility polyhedra. (b) V_1 and V_2 are not within each other's kernel and therefore their visibility polyhedrons are not the same. By z -buffer comparison p_2 is used for V^* , which is incorrect. (c) When user moves from track segment s_1 to s_0 , p_2 suddenly changes to p_1 to represent p which causes the popping effect. By using two slabs to separate the two objects, p is rendered to the second slab of both the viewpoint. Therefore, z -buffer comparison or blending will yield correct result

Fig. 3 (b), instead of using z -buffer comparison to determine whether we use p_1 or p_2 , we blend p_1 and p_2 's information. The blending weight is determined by the distance between V^* and V_1, V_2 . In this case, since V^* is closer to V_1 , the blending weight for p_1 is heavier. Although this does not give us the correct value, it at least mitigates the problem. Most importantly, when we are at V_1 , the blending weight for p_2 is zero and therefore, we only use p_1 . This makes sure that at the reference viewpoints, we don't have the reconstruction (visibility) errors and meets our goal of guaranteed lossless or near-lossless representation at the reference views. Moreover,

it reduces the popping effects and lets the user move on the track smoothly. Consider the example in Fig. 3 (c). At the moment we move out of track s_1 into s_0 , the weights for V_2 and V_0 are zero and therefore, only p_1 is used. This is consistent and allows smooth movement along the track.

In summary, computing the correct visibility polyhedron is not feasible without the full geometric scene description. Therefore our choices for merging the views are limited. By taking correctness and smoothness into account, we decide to use blending between the two reference views, instead of a z -buffer comparison to reconstruct the novel views along the rail-track.

5 Depth Meshes and Geometric Simplification

5.1 Slab Representation

As described before, with only one depth value per pixel stored in the IBR database, when the user moves away from the reference viewpoint, previously occluded objects can appear to be visible. However, if no information is stored for these objects, holes and cracks can appear. In order to allow for occluded information at a view to be retained Mark et al's post rendering 3D warping system [15] used more than one reference view to fill the holes. Their heuristics of deciding which pixel from which reference view is used is a per-pixel operation in software. Layered Depth Images [21], on the other hand, stores more than one layer of depth for each pixel to avoid the occlusion and dis-occlusion problems. Again, their system is a per-pixel based system. Although they can achieve relatively accurate results on a low-resolution display, they are not suitable for our purpose of rendering high-resolution imageries interactively. We hope to utilize the graphics hardware to accelerate our renderings. Therefore we introduce our slab representation to reduce the occlusion and dis-occlusion problems.

We divide the viewing frustum into several depth ranges, by setting the near and far clipping planes. We render the part of the scene in each range and obtain the imagery and depth polyhedron. We denote each polyhedron for a reference view V_i and j th slab (the j th polyhedron for viewpoint V_i). We call the corresponding imagery as a slab image. A binary opacity map is assigned to the slab image so that the background (empty) pixels are transparent. We texture map the slab image onto the slab polyhedron and composite the slabs (image and polyhedron) in a front to back order. This way, if at a certain viewpoint a pixel from the front slab has the non-background value, it blocks the pixel from the later slab, which can achieve the correct scene. The relationship of a polyhedron P_{V_i} , its discretized version ZP_{V_i} and the slab polyhedron $ZP_{V_i j}$ is shown in (4).

$$\text{discretize}(P_{V_i}) = ZP_{V_i} = ZP_{V_i 1} + ZP_{V_i 2} + \cdots + ZP_{V_i n} \quad (4)$$

By using slabs, we can reduce the occlusion and dis-occlusion problems. Compared to Mark et al's Post Rendering 3D Warping [15] and Layered Depth Images [21] sys-

tems, our slab-based representation uses 2D texture mapping hardware to accelerate the renderings and therefore is much more efficient during rendering.

Let's examine how we can use a slab representation to further reduce the visibility errors mentioned in the previous section. Recall that errors occur when two close-by viewpoints used for reconstruction are not within each other's kernel. Let's consider the same example as in Fig. 3 (d). Here we use two slabs to separate the two objects. Object O_1 is rendered to $Slab_1$, while object O_2 is rendered to $Slab_2$. In this case, for point p , the rendered values p_1 and p_2 for both viewpoints fall into the second slab. Essentially p_1 and p_2 are of the same value and therefore, using a z -buffer comparison or blending them will not produce any errors. We can see from the figure that by using slabs, we can achieve more accurate results. Since we cannot use infinite number of slabs in our system, because of the efficiency issues, there are still possibilities of occlusion and dis-occlusion problems within one slab and we still suffer errors which will be discussed in a later section.

5.2 Depth Mesh Simplification

A slab polyhedron $ZP_{V_i,j}$ is obtained by connecting the depth value for each pixel. This is essentially a model in which each pixel is a polygon itself, which is way too complex for interactive rendering on large displays. We can reduce the geometric complexity by down-sampling the depth buffer of each slab into a lower resolution grid or quad-mesh. The vertices of the quad-mesh retain the calculated depth values resulting from the pre-rendering phase. For the interior points, a linear interpolation function is assumed by the graphics hardware. Figures 4 (a) and (b) show the depth images rendered before and after down-sampling. We call this simplified slab mesh for the reference viewpoint V_i QZP_{V_i} . By compositing all the simplified slab meshes together we get the following formula as our complete scene.

$$\text{discretize}(P_{V_i}) = ZP_{V_i} \approx QZP_{V_i} = QZP_{V_i1} + ZP_{V_i2} + \dots + ZP_{V_in} \quad (5)$$

As discussed before, a binary opacity mask is assigned to enable per-pixel occlusion for each slab so that if both the front and back slab have information from the current viewpoint, the front slab will correctly occlude the back slab. This per pixel opacity mask can also eliminate some blocky (blurring) effects resulting from the down-sampling. This is known as α -clipping or silhouette-clipping [20]. Figure 4 (c) illustrates the depth image after α -clipping without using any slabs. Figure 4 (d) and (e) represent the depth images of the first and second slabs after α -clipping. All the images here are obtained from the Castle [31] dataset.

6 Data Management

In Sect. 5, we discussed depth mesh simplification and slab representation schemes. In this section, we will concentrate on algorithms of managing IBR imagery data to further improve the storage and rendering efficiency of our system. The IBR imagery

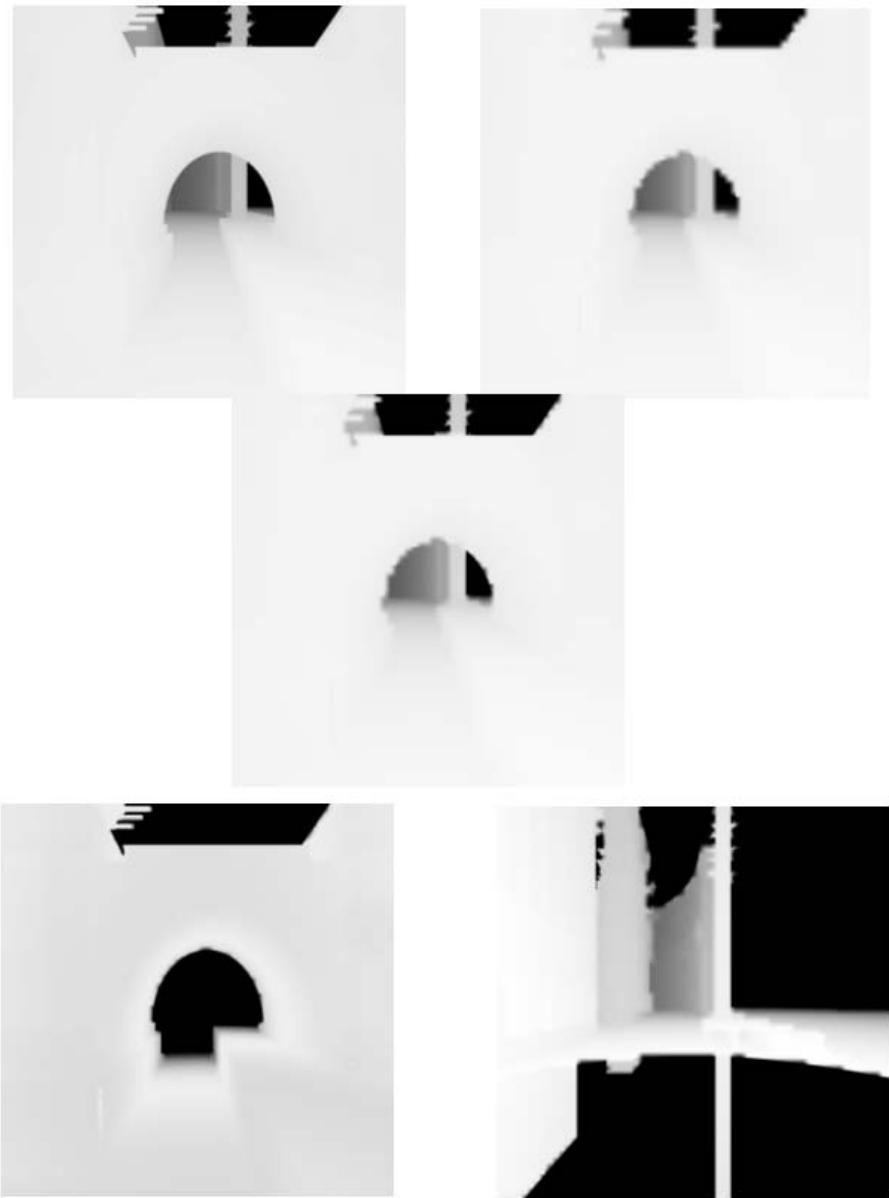


Fig. 4. (a) The original depth image. (b) The depth image after down-sampling. (c) The α -clipped depth image and (d) (e) Two slab depth images after α -clipping. This is for the Castle [31] dataset

data is mapped to the textures. The texture information is much larger and more time consuming to load and render. Consider the pre-rendered results for Nature [29] scene with $1k \times 4k$ for each slab with 3 slabs for one viewpoint, which is 48MB. For smooth navigation, we need to pre-fetch the textures and load them into the graphics hardware. For 1000 reference views on our track, we quickly amass the data in the 10-100 GB range. Without proper management and reduction, our system would be overwhelmed with this texture data.

6.1 Empty Texture Tiles Removal

The images resulting from the pre-rendering can contain parts that don't have any useful information. Keeping them is a waste of resources. To get rid of them, we break the resulting image into fixed size tiles according to the down-sampled quad-mesh. We detect and label them as empty tiles and remove them from the database. Treating each image tile as a distinct texture map easily allows for removal of these empty tiles.

Even with the empty tiles removed, we can still have lots of tiles, especially when the resulting image is very large for high-resolution display. Also remember that for reconstructing a new viewpoint, we need the information from the neighboring two viewpoints and each viewpoint may have several layers of images. This further increases the number of individual texture maps. Different systems can support only limited number of texture binding units, for example, $2^16 = 64k$ units (Please remember that the current Java3D engine supports even fewer). For a pre-rendered image with a resolution of $1k \times 4k$, 4 slabs for each viewpoint and a tile size of 16×16 for two viewpoints, we have 128K individual texture maps which exceeds the number of texture binding units available in the system. In this case, texture thrashing can occur which slows down the performance significantly.

To alleviate this problem we group individual image tiles into larger texture units for rendering. Consider a full image of size W by H , we divide the image into equally sized small tiles, w by h , according to the down-sampled quad-mesh. This results in each row having W/w tiles and each column having H/h tiles. We remove the empty tiles and merge the remaining tiles into a larger texture map. We accomplish this by squeezing each column, removing the empty tiles, and linking the resulting columns of tiles into a one-dimensional tile array. This is used as a single texture unit. Due to the fact that OpenGL/Java3D engine can only handle textures with the size in the power of 2, we split this 1-D array into several arrays and pad the last one. The reason why we split the array is for the consideration of our novel pre-fetching scheme. We want to make the indexing and loading of part of the panorama faster and easier. This will be discussed in more detail in the next section. A header file is constructed which contains pointers to where the beginning of each column in the 1-D array and how many non-empty tiles there are in each column. During run-time, this header file is first loaded into memory and it is fairly easy to keep track of which tile corresponds to which quad. Therefore the corresponding texture coordinates can be generated for appropriate texture mapping. By using this mechanism, we deal with much fewer texture bindings and therefore can avoid potential texture thrashing problems.

6.2 Caching and Pre-fetching

To reconstruct the new viewpoints on the track, the information from the two neighboring viewpoints is needed. The IBR rendering engine determines the closest two reference viewpoints on the path. When the user moves to a new path segment, requiring a new reference view, a severe latency occurs while the needed geometry and textures are read into memory. Therefore, instead of loading just two sampled views, the system needs to load several views into the memory and store them into a cache. The pre-fetching engine, which is implemented as a separate thread, continually requests texture data as the user moves along the track and stores it into the cache. The maximum number of views allowed in the cache is determined by the available memory size and the texture or panoramic image size. Our first experiment treated the whole panorama as a caching unit. It alleviates, but does not eliminate, the latency problem. When the user moves too quickly along the track, noticeable latency still occurs as the server strives to push the data to the client because loading the whole panoramic image is quite time-consuming.

Keeping the whole panorama texture in the memory allows the user to smoothly look around in all directions, but requires substantial memory and network burden. By examining the minimal information needed to reconstruct a novel view, we can reduce these demands and increase our pre-fetching length. We consider two scenarios. The first case is when the user moves along the track while keeping their orientation fixed. In this case, only the information within the user's viewing direction is needed. The second case is when the user stops on the track while examining the virtual environment. This requires more of the panoramic for the current two closest reference views. Therefore, we have the choice of pre-fetching more of each panoramic or pre-fetching partial view of more viewpoints along the track. We handle this using an adaptive two-part pre-fetching scheme - one part along the track and the other part along the viewing direction.

The algorithm works as follows. At the time the system starts up, we first load in the first and second viewpoints on the track. While the system is performing other tasks, like rendering and blending the scene, the pre-fetching thread tries to pre-fetch the information of the next few viewpoints along the track. We adaptively reduce the amount of the panoramic we pre-fetch for views that are farther down the track. This means that as we pre-fetch the information of the view points farther away, we pre-fetch less and less of the information into the cache. If at this time the user stops and looks around, we have enough information (the whole panorama for the first two viewpoints) to reconstruct the information for him/her. Or if the user decides to go along the track, we also have the information he or she needs. When the user moves out of the first track segment to the next one, the pre-fetching engine tries to load the information of the viewpoints even farther away. In the meantime, another thread tries to load in the remaining part of the panorama for the viewpoints that are already in the cache.

This algorithm can balance the user's need for walking down the track or looking around. Which pre-fetching thread should have higher priority should be determined

by the preference of the user: whether he or she wants to move along the track or spin around fast.

6.3 Texture Removal Using a Conservative Track-Dependent Occlusion Culling Algorithm

The slab representation is used to better address the occlusion and dis-occlusion problems. As the user moves away from the reference viewpoint, previously occluded information can be rendered using later slabs. However, the problem with partitioning and pre-rendering scenes into several slabs is that it produces unnecessary information. Consider the example in Fig. 5. In this example, we have three reference viewpoints on the track segment: V_1 , V_2 and V_3 . Objects O_2 , O_3 and O_4 are occluded by Object O_1 for V_1 but are rendered and stored in $slab_2$. O_2 is visible for V_2 and O_4 is visible for V_3 . Hence, when the user moves away from V_1 towards V_2 , the information stored in $slab_2$ of V_1 is used to represent O_2 . Likewise for O_4 . However in this example, O_3 is never visible from any viewpoints along the track segments. Without occlusion culling we would still render O_3 and store the result in $slab_2$ as part of the texture map, which is unnecessary. This unnecessary data affects the storage cost, network transmission time and the rendering performance. A conservative track dependent occlusion-culling scheme is adopted to remove these occluded textures.

We call this algorithm track dependent occlusion-culling because we need to consider current and neighboring viewpoints for the algorithm. How much information is occluded depends on the sampling rate of the reference views on the pre-selected track. Durand et al [10] introduced an algorithm that combines multiple viewpoints into one cell. Occlusions of the objects are calculated for the whole cell. They introduced the extended projection operators. The extended projection for the occluder is the intersection of the views within the cell while the extended projection for the

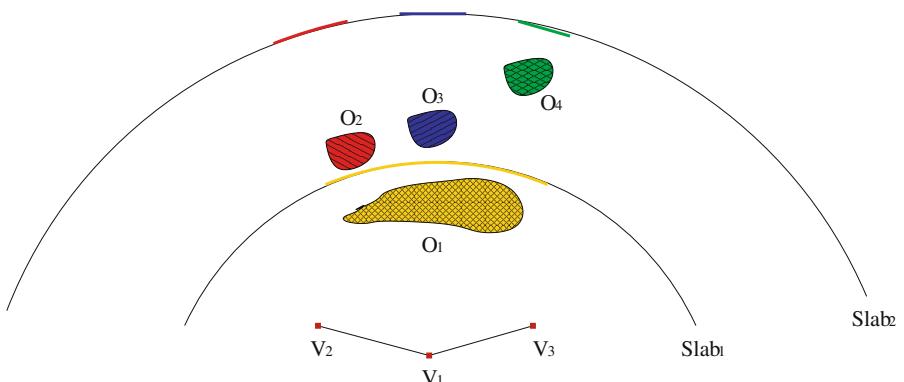


Fig. 5. Objects O_2 , O_3 and O_4 are occluded by Object O_1 for V_1 and therefore are rendered and stored in the $slab_2$. O_2 is visible for V_2 and O_4 is visible for V_3 . However O_3 is not visible from any of the viewpoints along these track segments

occludee is the union of the views within the cell. To calculate the occlusion, they just need to compare the extended projections of the occluders and occludees.

We want to determine whether texture tiles in the later slabs are occluded by those in the previous slabs for both the current and the neighboring two viewpoints. Therefore in our algorithm, we consider current and neighboring viewpoints as a cell and texture tiles of the early slabs to be occluders and texture tiles of the later slabs to be occludees. The algorithm works as follows. For each reference viewpoint, we first build an occlusion map and fill the map with the opacity values of the projected first slab texture tiles. We treat the following slab textures in a front to back order, considering each tile in each slab to see whether it is occluded. The occlusion is performed by comparing the extended projections of the occluders: texture tiles from the previous slab and the extended projections of the occludees: texture tiles from the later slab. Figure 6 (a) and (b) show the extended projections of occluder tiles and occludee tiles, with regard to the current viewpoint V_1 and its neighboring viewpoints V_2 and V_3 . If the extended projection of the occludee falls in that of the occluder, the occludee is occluded. In practice, we first project all the occluder tiles and form an occlusion map and then convolve (average) the window in the occlusion map with the size of extended projection of the occludee. If the result is 1, the occludee tile is occluded. For easier calculation, we make several conservative simplifications. According to [10], for non-flat tiles, the depth of the occluder is the maximum depth of the tile and the depth of the occludee is the minimum depth of the tile. For all the occluder tiles, we chose the slab depth which is larger than any maximum tile depth as another conservative simplification. By taking the minimum depth of the occludee tile and the slab depth, we can consider them as flat tiles and therefore we have a setup as in Fig. 6 (c).

Each tile in our system has a width of w . By considering the viewing angle of V_2 and V_3 , we need to convolve (average) an extended area with a width of $w + s_1 + s_2$ in the opacity map to see if the result equals 1. Considering the 2D case, s_1 can be calculated using the following equation.

$$s_1 = \frac{h_2 - h_1}{h_2} * d_{21} \quad (6)$$

In which h_1 is the distance from the viewpoint to $slab_1$ and h_2 is the minimum depth of the occludee tile. To calculate d_{21} , consider Fig. 6 (d).

$$|d_{21}| = |l| - |l'| \quad (7)$$

$$|l| = (\mathbf{Q} - \mathbf{V}_1) \bullet \mathbf{x} \quad (8)$$

$$|l'| = \frac{|h|}{\tan \beta} \quad (9)$$

$$\tan \beta = \frac{(\mathbf{Q} - \mathbf{V}_2) \bullet \mathbf{x}}{|\mathbf{Q} - \mathbf{V}_2|} \quad (10)$$

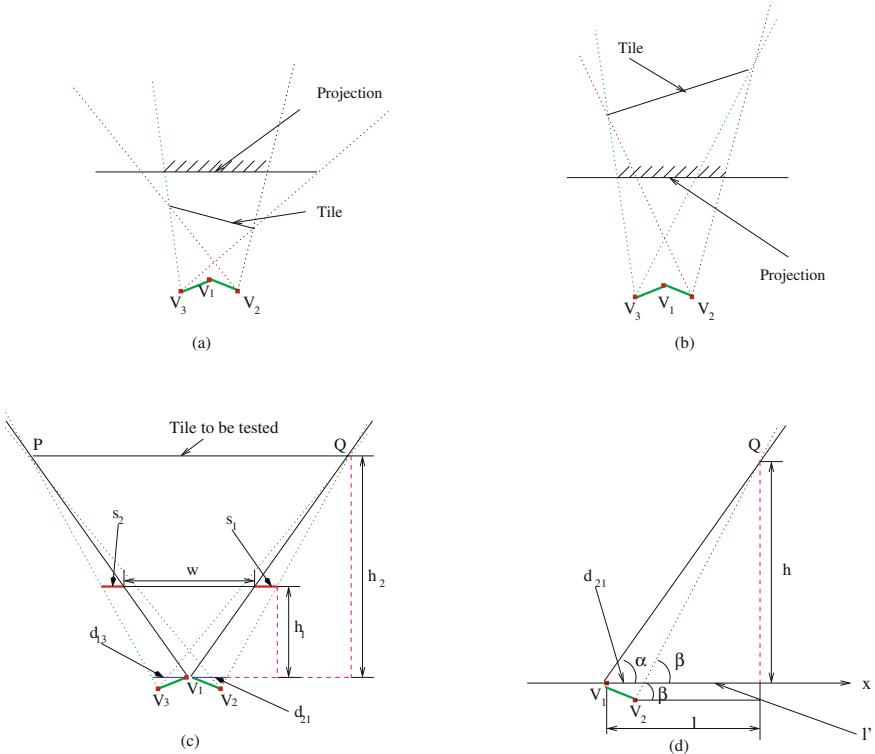


Fig. 6. (a) and (b) the extended projections for occluder and occludee tile respectively, with regard to three viewpoints on the track. (c) The occlusion culling setup after conservative simplification of depth. h_1 is the distance from the viewpoint to $slab_1$ and h_2 is the largest depth of the occludee tile. (d) how to calculate d_{21}

A similar equation can be used to calculate s_2 . We then convolve (average) the extended area with a width of $w + s_1 + s_2$ in the opacity map. If the averaged opacity value of the enlarged window is one, we mark the tile as an empty tile and do not store its textures. If the value is less than one, the tile is not occluded and we add the opacity values of this tile to the opacity map. We treat all the tiles in one slab and continue to the next one until all the slabs are processed. As pointed out in [6] [16, 27], we can use a method similar to α -acceleration, which lowers the opacity threshold to less than one and cull the tiles more efficiently, with minimal degradation to the quality of the rendering result.

Using 3 slabs on the Castle [31] dataset reduces the storage for the second slab by 77% from 1.7 MB to 390 KB. The storage requirement for the third slab is reduced by 80% from 1.3 MB to 261 KB. This is without the α -acceleration. The rendering quality is identical to the one without the occlusion culling.

The results show that the track dependent occlusion culling is quite efficient for this dataset. It can reduce the storage requirement, decrease the network transmission

time and increase the pre-fetching efficiency and improve the rendering performance. Another benefit is that it can reduce the rendering cost/overhead that results from increasing the number of slabs. More slabs can address the occlusion/dis-occlusion problem better. Using the occlusion culling technique, less information will be left for later slabs after culling. Therefore, increasing the number of slabs does not affect the rendering speed too much.

The efficiency of the algorithm is highly dataset-dependent. For the Nature dataset [29] we tested in which the scene is more open and therefore our slab representation does not have too much unnecessary information in the first place, we can only cull about 5-10 percent without α -acceleration.

7 Error Analysis

7.1 Depth-Mesh Errors

We consider two sources of errors after down-sampling and merging two reference views for any novel view. The first source of errors results from the linear interpolation of the down-sampled depth-meshes. As we discussed before, after down-sampling, only the vertices of the quad-mesh have the actual depth values obtained from the pre-rendering (z buffer or intersections). The depth values for the interior points of the quad-mesh are calculated using a bi-linear interpolation function. For regions having high curvature (in depth), the linear interpolation introduces errors. We also break up the pre-rendered image into fixed sized and treat each tile as a texture map. Mapping the textures to a linearly interpolated quad is different from mapping them to a highly curved surface. The errors appear when we reconstruct either the reference views or the novel views. The down-sampling error can be reduced by using a finer quad-mesh (smaller tile size). However, decreasing the tile size increases the rendering time, loading time and storage requirements. Figure 7 (a) shows the experiments that we performed using different sizes of a quad-mesh for one POVRAY [28] rendered view of the Nature [29] dataset with $1k \times 1k$ output resolution on our Sun Blade 1000 system. Here we compare the change of Mean Squared Error (MSE), the loading time, the rendering time and the storage requirement with different tile sizes. From Figure we can see that, the MSE decreases almost linearly when we decrease the tile size. However, the loading time increases quite dramatically with smaller tile sizes. The rendering time also increases with decreased tile size. As tile size increases, the storage first decreases, and then increases. The decrease (from right to left in the Figure) at first is due to the fact that we need to store fewer depth values when we have fewer tiles. The later increase occurs since there are fewer empty tiles to remove, that is, tiles which do not have any information. This is also why the rendering speed levels off when the tile size increases – we have more empty space to rasterize. We should choose a tile size that can achieve proper frame rates and good image quality. Please note that loading time is also a very important factor because it affects the pre-fetching performance. Therefore, we have chosen a tile size of 32×32 in practice.

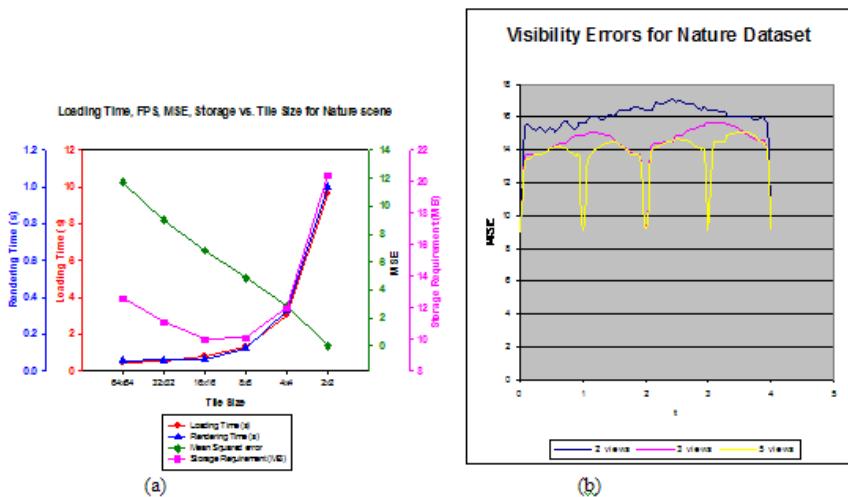


Fig. 7. (a) Shows the Mean Squared Error (MSE), the loading time, rendering time and the storage requirement for different tile sizes for one single view. (b) MSE of interpolated views against actual rendered views. Interpolated views have a tile size of 32×32

7.2 Visibility Errors

Another source of errors result from reconstructing the novel view when we blend two close-by reference views. As described previously, if the reference views are within each other's kernel of the visibility polyhedrons, the actual polyhedrons of the two reference views coincide with each other perfectly and therefore no visibility errors should occur. However in real situations this is seldom true and these visibility errors are inevitable. Figure 7 (b) shows the Mean-Squared Errors (MSE) of the resulting images for interpolated views against Povray rendered views at corresponding positions along our path. As we can easily discover, the closer the sampled viewpoints are, the smaller the visibility errors will be. We performed an experiment using three different sampling rates along the rail-track. To maintain the same down-sampling error for all the cases, we use the image resolution of $1k \times 1k$ as before and keep the tile size constant at 32×32 . We can see that the curve using only 2 reference viewpoints has the highest Mean-Squared Errors (MSE). This is as expected. The peak errors occur somewhere close to the middle of the two sampled viewpoints for all three cases. The peak MSE is about 17 out of 256, or approximately 6.7%. The curve which uses 5 views has the finest sampling rate and peaks out with MSE of about 14–15. From the figure, we can also see that the MSE drops dramatically at the reference viewpoints. This is due to the fact that at the reference viewpoints, there are no visibility errors and the errors solely come from down-sampling and the bi-linear interpolation that we discussed in the previous section.

8 Results and Discussions

Our front end rail-track viewer was implemented in Java/Java3D. We ran our IBR framework on 4 datasets. The first one is a virtual scene called Nature [29] rendered using Povray [28]. The scene is moderately complex, containing trees, bushes, rocks and birds. It takes about 20 minutes to render one frame on our 2GHZ Dell Precision 530 workstation using POVRAY. One path was chosen for this scene with 20 sampled viewpoints along the track. Three panoramic layers with a resolution of 1024×4096 per layer were pre-computed for each view sample. The total size for the database after pre-processing is 220MB. Without empty tile removal and track-dependent occlusion culling it would require over 1.2GB. The geometry and imagery was broken up into 32×32 quads. Our second dataset is a LOX post dataset which Visualization Toolkit (VTK) [32] provides. This dataset simulates the flow of liquid oxygen across a flat plate with a cylindrical post perpendicular to the flow. It contains both scalar and vector fields in the data. A rendering was chosen with the post, a slice plane and several stream-polygons. One path was pre-selected going into the stream-polygon region with 23 view samples. Four panoramic layers with a resolution of 512×2048 per layer were pre-computed for each view sample. The reference image database was pre-rendered using VTK and the total size for the image database was reduced to 138MB. The geometry and imagery was broken up into 16×16 quads. This database required 9.2 hours to pre-render. We also tested our system on two other Povray scenes, namely the Castle [31] and the Night [30] datasets obtained from the web. For each one, we chose 15 viewpoints on a path and each viewpoint has 3 layers with a resolution of 1024×4096 per layer.

We have tested our IBR viewer on two platforms. On the Sun Blade 1000 workstation with dual Ultra-Sparc III 750MHZ, 1 GB of memory and Elite 3D graphics card, we achieve 15 frames per second for the 3 povray scenes at a $1k \times 1k$ rendering resolution and 20 frames per second for the LOX dataset with a 512×512 rendering resolution. On our Dell Precision 530 workstation with a 2 GHZ Xeon processor and 2 GB of memory, 128 MB Nvidia Geforce4 Titanium 4600 video card, we achieve above 30 frames per second for all four datasets. Figure 8 shows a resulting image for each of the four datasets.

9 Conclusions and Future Work

This paper presents our framework for allowing the user to examine and walkthrough any scene using a relatively high-resolution display. We achieved this goal by limiting the user's movement to lie on a track and utilizing pre-computed IBR data and rendering techniques. Intelligent data organization, track-dependent occlusion culling for texture removal, novel caching and pre-fetching scheme makes our system more efficient. Application of our system to several complex scenes illustrates that our system is suitable for exploring complex virtual environments and large datasets with reasonably small errors and visual coherency.

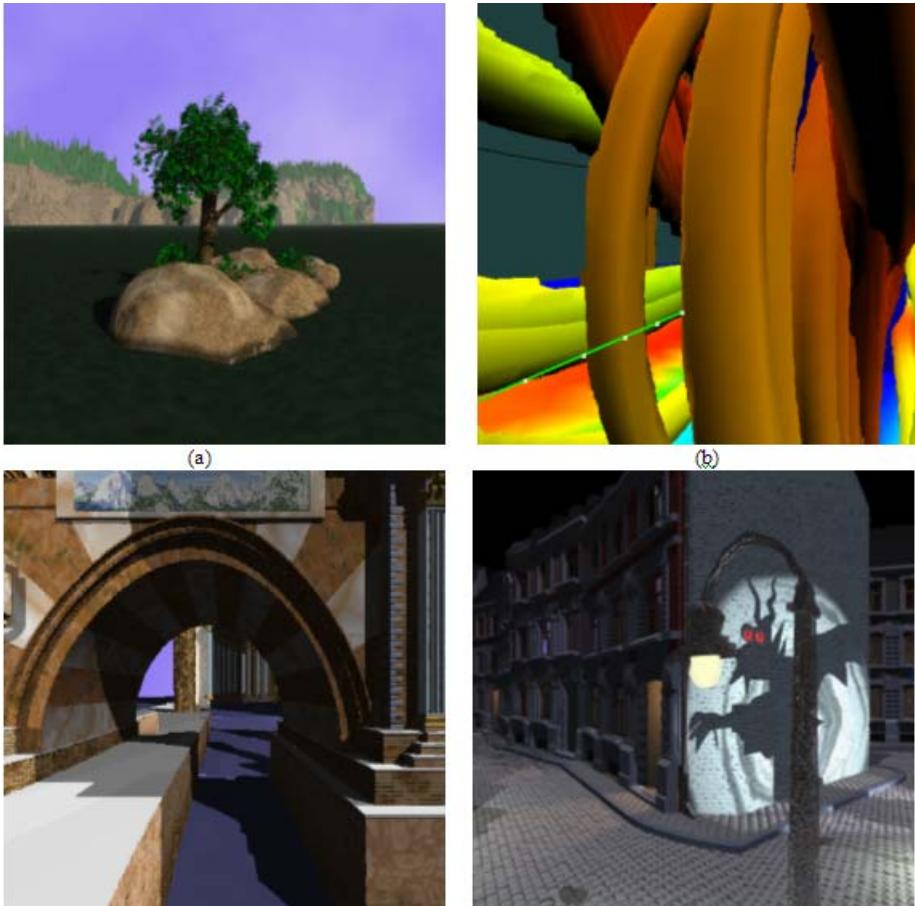


Fig. 8. Rendering results for (a) Nature dataset, (b) LOX dataset, (c) Castle dataset and (d) Night dataset

With our new data management techniques, our database still tends to be very large, especially with larger resolution and finer sampling. However, this was expected. We trade storage for rendering efficiency and low-cost texture streaming. This allows us to render at an extremely large resolution and offers an advantage over other high-quality IBR techniques.

Our contributions to this research have been: We have built a texture streaming client/server architecture for panoramic walkthroughs inside the scene and use view-dependent layers to better address occlusion and dis-occlusion problems. We also implemented an intelligent caching and pre-fetching scheme for the efficient data management. A track-dependent texture removal algorithm, considering possible occlusion/dis-occlusion along the track segments, is designed to remove unnecessary data and compress the database. Finally we incorporate mesh simplification

and texture mapping hardware for interactive high-quality and high-resolution renderings.

We are considering ways to replace textures with average color for the highly-occluded regions, as described in Zhang's paper [27]. For example, when $\alpha = 0.9$ for a specific tile, instead of throwing away the tile or rendering the whole tile as a texture map with full resolution, we can use the average color of the tile to represent it. This way, we can reduce the visual artifacts introduced by alpha-acceleration while keeping a minimum amount of information. Another future work involves how to sample the track. One approach would be to sample the pre-selected track densely and let the user specify an error metric that he/she can tolerate. We can then use this error metric to determine the actual samples that are needed.

References

1. Andelson, E. H., Bergen, J. R.: The plenoptic Function and the Elements of Early Vision. In: Landy, M., Movshon, A. (ed) Computational Models of Visual Processing. The MIT Press, Cambridge Massachusetts (1991)
2. Chang, C., Bishop, G., Lastra, A.: LDI Tree: A Hierarchical Representation for Image-Based Rendering. Proc. SIGGRAPH, 291-298 (1999)
3. Chen, S.: QuickTime VR – An Image-Based Approach to Virtual Environment Navigation. Proc. SIGGRAPH, 29-38 (1995)
4. Choi, J., Shin, Y.: Efficient Image-Based Rendering of Volume Data. Proc. Pacific Graphics, 70-78 (1998)
5. Cohen-Or, D., Mann, Y., Fleishman, S.: Deep Compression for Streaming Texture Intensive Animations. Proc. SIGGRAPH, 261-268 (1999)
6. Danskin, J., Hanrahan, P.: Fast Algorithms for Volume Raytracing. Proc. Workshop Volume Visualization, 91-98 (1992)
7. Darsa, L., Costa, B., Varshney, A.: Navigating Static Environments Using Image-Space Simplification and Morphing. Symposium on Interactive 3D Graphics, 25-34 (1997)
8. Debevec, P., Yu, Y., Borshukov, G.: Efficient View Dependent Image-Based Rendering with Projective Texture Mapping. In 9th Eurographics Rendering Workshop, (1998)
9. Decoret, X., Schaufler, G., Sillion, F., Dorsey, J.: Multilayered Imposters for Accelerated Rendering. Proc. Eurographics, 145-156 (1999)
10. Durand, F., Drettakis, G., Thollot, J., Puech, C.: Conservative Visibility Pre-processing Using Extended Projections. Proc. SIGGRAPH, 239 - 248 (2000)
11. Gortler, S., Grzeszczuk, R., Szeliski, R., Cohen, M.: The Lumigraph. Proc SIGGRAPH, 43-54 (1996)
12. Levoy, M., Hanrahan, P.: Light Field Rendering. Proc. SIGGRAPH, 54-61 (1996)
13. Mark, W., McMillan, L., Bishop, G.: Post-Rendering 3D Warping. Symposium on Interactive 3D Graphics, 7-16 (1997)
14. McMillan, L., Bishop, G.: Plenoptic Modeling: An Image-Based Rendering System. Proc. SIGGRAPH, 39-46 (1995)
15. Mueller, K., Shareef, N., Huang, J., Crawfis, R.: IBR-Assisted Volume Rendering. Late Breaking Hot Topic IEEE Visualization, 5-8 (1999)
16. Mueller, K., Shareef, N., Huang, J., Crawfis, R.: High-quality Splatting on Rectilinear Grids With Efficient Culling of Occluded Voxels. IEEE Transactions on Visualization and Computer Graphics, 5, 2, 116-134 (1999)

17. Preparata, F., Shamos, M.: Computational Geometry, An Introduction. Springer-Verlag New York Inc (1985)
18. Qu, H., Wan, M., Qin, J., Kaufman, A.: Image Based Rendering With Stable Frame Rates. Proc. IEEE Visualization, 251-258 (2000)
19. Rademacher, P., Bishop, G., Multiple-Center-of-Projection Images. Proc. SIGGRAPH, 199-206 (1998)
20. Sander, P., Gu, X., Gortler, S., Hoppe, H., Snyder, J., Silhouette Clipping. Proc. SIGGRAPH, 327-334 (2000)
21. Shade, J., Gortler, S., He, L., Szeliski, R., Layered Depth Images. Proc. SIGGRAPH, 231-242 (1998)
22. Schroeder, W., Zarge, J., Lorensen, W.: Decimation of Triangle Meshes In: Computer Graphics, **26**, 2, 65-70 (1992)
23. Shum, H., He, L., Rendering With Concentric Mosaics. Proc. SIGGRAPH, 299-306 (1999)
24. Szeliski, R., Shum, H., Creating Full View Panoramic image Mosaics and Texture-Mapped Models. Proc. SIGGRAPH, 251-258 (1997)
25. Ward, G., The RADIANCE Lighting Simulation and Rendering System. Proc. SIGGRAPH, 459-72 (1994)
26. Yang, L., Crawfis, R., Rail-Track Viewer, an Image Based Virtual Walkthrough System. Eurographics Workshop on Virtual Environment, 37-46 (2002)
27. Zhang, H., Manocha, D., Hudson, T., Hoff, K., Visibility culling using hierarchical occlusion maps. Proc. SIGGRAPH, 77-88 (1997)
28. <http://www.povray.org>
29. <http://www.irtc.org/stills/1998-06-30.html>
30. <http://www.irtc.org/stills/1998-04-30.html>
31. <http://www.irtc.org/stills/1999-02-28.html>
32. <http://www.kitware.com/vtk>

Component Based Visualisation of DIET Applications

Rolf Hendrik van Lengen¹, Paul Marrow², Thies Bähr¹, Hans Hagen¹,
Erwin Bonsma², and Cefn Hoile²

¹ German Research Centre for Artificial Intelligence
Intelligent Visualisation and Simulation
Erwin-Schrödinger-Strasse, 67663 Kaiserslautern, Germany
[lengen|baehr|hagen]@dfki.uni-k1.de

² BT Exact
Intelligent Systems Laboratory
Adastral Park, Ipswich IP5 3RE, United Kingdom
[paul.marrow|erwin.bonsma|cefn.hoile]@bt.com

Summary. Controlling distributed information in a complex information infrastructure requires novel and innovative information processing and management techniques. The Decentralised Information Ecosystem Technologies (hereafter, DIET) approach provides a software platform based on a lightweight, robust, adaptable, and scalable multi-agent system. DIET uses implicit forms of communication found in natural ecosystems as an analogy for computer-based distributed information management systems. The platform can be used to tackle a variety of information management applications in distributed and open real-world scenarios.

However, the development, debugging, and monitoring of distributed applications is a complex task. Adequate visualisation techniques and tools are required to assist the software developer. This paper describes the visualisation platform developed on top of the DIET framework. The main purpose of the visualisation platform is the inspection and manipulation of ecosystem inspired applications built upon DIET. The platform provides fundamental visualisation components that can be easily linked together to build up a visual network in order to formulate complex visualisation tasks. Possible tasks can range from monitoring an individual agent to visualising the overall system behaviour. Due to this component based approach the effort for monitoring and debugging agent applications is considerably reduced.

Key words: Intelligent Agents, Information Visualisation, Multi-Agent Systems

1 Introduction

The Decentralised Information Ecosystem Technologies project is a European collaborative research project, part of the Universal Information Ecosystems initiative. The initiative is centred around the information ecosystem concept, that relates transactions in the information infrastructure to the dynamics of natural ecosystems.

The term information ecosystem is used by analogy with natural ecosystems. Similarly to a natural ecosystem it is dynamic, noisy, to some extent unpredictable, and decentralised [9]. The information ecosystem includes all those individuals and units involved in creating, managing, using, and adaptively re-using information sources where information is interpreted in its widest sense. In an effective information ecosystem data, information, and knowledge are all viewed as organisational resources.

Accordingly the DIET project is concerned with the construction of a multi-agent development toolkit to support complex information manipulation applications using nature-inspired computing techniques. The development, debugging, and monitoring of ecosystem inspired applications is a complex task. Adequate visualisation techniques and tools are required to assist the application developer.

The paper is structured as follows: In the first session the main components of the DIET software platform are described. We then illustrate the outline of our visualisation architecture that extends the platform by supplying visual and interactive components. Finally, we present a simple DIET example application and demonstrate how the visualisation platform can be used to monitor it.

2 DIET Software Platform

The DIET platform has been designed drawing upon the information ecosystem philosophy introduced in the Universal Information Ecosystems initiative [2]. Accordingly it follows a bottom-up design approach, and is robust, adaptive, and scalable.

The *bottom-up* design approach means that intelligent behaviour can emerge from the interactions between large numbers of agents, each of which is very small and simple, and does not need to have a high level of individual intelligence.

The DIET platform kernel itself is *robust* to network failure and/or system overload. It is designed such that the effects of such failures are localised, and the kernel provides feedback when failure occurs allowing applications to adapt accordingly. The decentralised nature of DIET also makes the platform less susceptible to overall failure.

It supports *adaptive* applications in that many lightweight DIET agents can interact to produce adaptive responses to changing application circumstances. The simplicity of agents means that they can easily be deleted or created as applications require.

It is *scalable* at a local and at a global level. Local scalability is achieved because DIET agents can be very lightweight. This makes it possible to run large numbers of agents, up to several hundred thousands, in a single machine. DIET is also globally scalable, because the architecture is such that it does not impose any constraints on the size of distributed DIET applications. This is mainly achieved because the architecture is fully decentralised, thus not imposing any centralised bottlenecks due to centralisation of the information flow management.

For more details about the "DIET approach" in agent system design, see Marrow et al. [8] and Hoile et al. [3]. The DIET platform is available as Open Source [1].

2.1 Layered Architecture

The DIET system is designed as a three-layer architecture (see Fig. 1). The core layer is the lowest layer. It provides the minimal software needed to implement multi-agent functionality on top of the DIET framework. This is through the DIET platform kernel, which provides the underlying “physics” of the DIET ecosystem. It also includes basic support for debugging and visualisation.

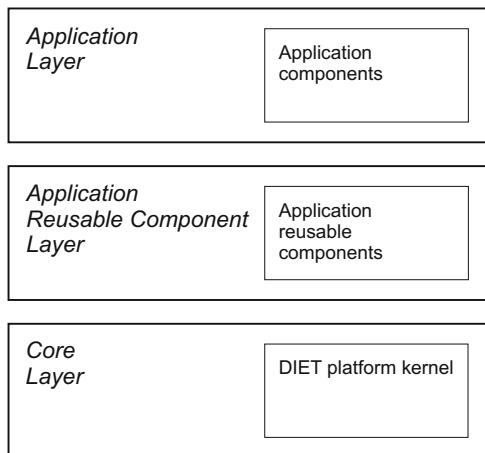


Fig. 1. The three layer architecture of the DIET platform

The *Application Reusable Component (or ARC) layer* contains components that are not essential for the core layer, but can be shared among different applications. These components include

- various schemes for remote communication,
- a framework for pluggable agent behaviours and
- support for scheduling events for execution at a later time.

The *application layer* contains code specific to particular applications. This is where code that extends the DIET platform in diverse application directions resides. The platform by default includes a number of sample applications, which demonstrate specific features of the platform.

2.2 DIET Elements

The basic classes and interfaces in the DIET platform kernel define five fundamental elements. These elements are arranged in the following hierarchy:

- *worlds*,
- *environments*,

- *infohabitants*,
- *connections*, and
- *messages*.

The *world* is a placeholder for *environments* in a JVM (Java Virtual Machine). Environments are where the *infohabitants* reside. Infohabitants are the agents in the DIET platform. Infohabitants can communicate by forming *connections* and sending *messages* via these connections.

The world manages functionality that can be shared by environments, such as infohabitant migration to other worlds. A world is also the access point for attaching debugging and visualisation components to the DIET platform.

Environments implement the DIET “physics”, enabling infohabitant creation, destruction, communication, and migration. An environment can host large numbers of infohabitants. An infohabitant can migrate from the environment it currently resides in to any other environment, providing that it knows the address of the destination environment. Environments can have one or more neighbourhood links, which are addresses of other neighbouring environments. Infohabitants can use these neighbourhood addresses to explore the DIET universe without any a-priori knowledge of existing environments.

The infohabitants are at the heart of this conceptual hierarchy. They are the agents within the DIET multi-agent platform, and each one executes autonomously using an assigned execution thread. Infohabitants in the core are designed to be very lightweight. An infohabitant only has minimal capabilities to execute and to communicate. Furthermore, an infohabitant can give up its execution thread when it temporarily does not need it. The kernel attempts to give the infohabitant a thread again when it requires one, e.g. to handle an incoming message.

Infohabitants can communicate with other infohabitants in the same environment by creating *connections*. A connection is a bi-directional communication channel between a pair of infohabitants.

After a connection has been set up, infohabitants can use it to pass messages. The DIET platform does not enforce a particular communication protocol. It provides infohabitants with the ability to exchange text messages and optionally objects. This allows each infohabitant to use a protocol most suited to its functionality and capabilities.

2.3 Infohabitant Actions

Infohabitants can carry out four different actions:

- creation (an infohabitant can create other infohabitants),
- destruction (an infohabitant can destroy itself),
- communication (an infohabitant can send messages to other infohabitants), and
- migration (an infohabitant can move to a different environment).

The actions are implemented in the DIET platform kernel such that their execution time does not go up when the number of infohabitants increases but stays

constant. Furthermore, they are implemented in a *resource constrained* and *fail-fast* manner.

These actions are resource constrained because there are explicit limits on the resources that they can use. For example, execution threads are a constrained resource. The number of threads that are used by infohabitants in a DIET world is limited (to a number specified by the user when the world is created). The kernel actions are fail-fast because when an action cannot be executed instantaneously, it fails immediately. This failure does not prevent the execution of other actions. The kernel does not retry the actions later or block execution until it has successfully executed the action.

The fail-fast, resource constrained implementation of the kernel actions protects the system against overload. Each infohabitant receives incoming messages through a buffer of limited size. When an infohabitant attempts to send a message to an infohabitant whose message buffer is already full, the message is rejected. If this did not happen, there would for instance be problems when an infohabitant processes messages more slowly than the rate at which they arrive. Its buffer of incoming messages would constantly grow, and eventually the JVM would run out of memory (although this might take a while). A more immediate effect is that as the number of pending messages grows, the time it takes before a reply is received to each message goes up as well. This means that the reply may be too late and obsolete by the time it is received. Limiting the size of the infohabitant event buffers is a simple yet effective way to cope with overload. As long as the system load is low, the size of the buffer does not matter, as the buffer limits will never be reached. When the system becomes overloaded, this mechanism offers basic protection and allows infohabitants to adapt their behaviour rapidly.

2.4 Event Generation

The DIET platform uses the Observer design pattern [4], also known as Publish-Subscribe, to support visualisation. Visualisation components can observe applications by registering as an observer, also called listener, with the objects that they are interested in. They will then receive notifications when specific events occur. More specifically, the following events can be monitored:

- Environments: added and removed.
- Neighbourhood links: created and destroyed.
- Infohabitants: created, creation failed, arrived, arrival failed, departed, departure failed, and destroyed.
- Infohabitant run state: started, suspended, resumed, resume failed, terminated, and crashed.
- Infohabitant property: changed.
- Connections: created, rejected, and destroyed.
- Messages: sent, received, accepted, and rejected.

All these events, except infohabitant property changes, are generated by the DIET kernel. This means that no extra code is required in applications to generate these

events. They are generated automatically as and when needed. It also means that infohabitants cannot suppress event notifications or fire spurious events. This is important for reasons of security as well as visualisation. For instance, it allows the event listening infrastructure to be used to implement resource accounting as well.

The infohabitant property event notifications can be used to visualise the internal state of infohabitants. For example, some infohabitants may have an internal activity level associated with them, which affects their behaviour. The infohabitant can fire a property change event whenever the value of its activity level changes. If it does so, this property can be monitored and displayed by visualisation components. Different infohabitants may support different properties, and this can be application-specific. Therefore, property events are the only ones that are not generated by the DIET kernel, but by user code instead.

All of the events listed above are generated at the level where they occur. For instance, an environment created event is generated by a DIET world. However, a message sent event is generated by the connection along which the message was sent. Furthermore, events are generated lazily. If there are no observers registered with a specific object, it will not generate event notifications. This makes it possible to efficiently visualise simulations. For instance, infohabitants that are not being visualised do not generate any connection events, property events and run state events.

3 Visualisation Platform

In recent years, the development of agent-building toolkits has created the need for efficient software visualisation tools. These tools assist the programmer in individual application development stages.

Toolkits like e.g. DCM [10], ZEUS [5], and AMS [11] have provided agent development environments together with basic visualisation support. These agent-building frameworks have substantially improved the study and development of collaborative multi-agent systems. However, the provided visualisation components have several limitations that make them not suitable for the DIET approach. In order to work properly, the visualisation components are realised as agent systems themselves using a certain message protocol to communicate with a heavyweight multi-agent system to be monitored and manipulated.

The DIET approach favours the design of lightweight agents that lack the implementation of a special message protocol for the sake of openness as well as a minimal core implementation allowing the creation of a large number of agents with very little overhead. The design philosophy of the DIET approach entails a variety of general requirements to the visualisation architecture.

Non-intrusive: The visualisation architecture should have minimal effect on the observed application to prevent different behaviour with or without the visualisation running. Many real-time visualisation tools are intrusive; they inject wait states, add special code, or interrupt real-time operations. This can cause unwanted side-effects between concurrently running processes that are hard to discover because the

co-occurrence may appear only sometimes and is hardly predictable so that error situations are not reproducible for debugging purposes. Furthermore, the design of the architecture should enable the introspection of DIET applications without the need for changing any application code.

Scalable: The visualisation architecture has to be scalable to support varying operational scenarios ranging from environments that contain only a few infohabitants up to environments that contain several thousands. After a simulation it might be helpful to repeat it offline with a different time scale or even in single step execution to investigate conditions and behaviour in detail without the need of cooling down the real-time application. Therefore the visualisation must also support the display of logged events and properties.

Flexible: There are many possible applications so the architecture has to consist of a set of components which can be combined to build task- and application-specific visualisation and debugging tools. The components themselves should be highly flexible and adjust themselves automatically to specialised core elements.

Interactive: Nevertheless, debugging complex systems demands a high degree of steering and control functionality which should be supported by the architecture. Possible features for example could be the manipulation of properties or resources, the creation of new infohabitants, their cloning or even killing as well as the possibility to cool down the whole world or parts of it for further investigations.

3.1 Particular Visualisation Challenges

In addition to these more general requirements the visualisation of DIET applications is especially challenging because of features offered by the platform that are usually not provided by common multi-agent systems.

- There can be large numbers of agents. The light-weight nature of agents means that an ordinary desktop machine¹ may host over 100,000 agents. To maintain scalability, the memory that the visualiser associates with each agent should be low. Also, the execution overhead should be independent of the number of agents.
- The frequency of events can be high. For example, in some applications more than 1000 messages are sent per second. This means that events must be processed rapidly, with minimal delay and overhead.
- The data generated by the kernel is low-level. The kernel generates event notifications but even basic information, such as the number of infohabitants in a given environment, is not directly available. So, for users to understand what is happening, the visualiser may need to combine multiple low-level events to generate higher level information. For instance, the number of infohabitants in an environment can be determined by continuously observing the events generated by the environment. Visualisation of some applications may also require synthesising application-specific, high-level properties.

¹e. g. 32bit Intel CPU, 1.4 GHz, 1 GB RAM

- The execution of DIET applications is multi-threaded. Each agent executes using a dedicated execution thread. This affects visualisation because it limits the ability to pause applications or to slow them down in a controlled manner.
- Agents execute in real-time. The behaviour of many agents is controlled by the system clock. For instance, if an agent does not receive an acknowledgement message within x milliseconds, it may retry sending the message or try elsewhere. The visualiser inevitably incurs some execution overhead. If CPU usage is close to maximal, this may slow down agent execution and thus affect their behaviour.
- The kernel actions are fail-fast and resource constrained. This is therefore another way that heavy CPU utilisation can affect an application. When the CPU is heavily used, some infohabitant actions may fail. The infohabitants may adapt their behaviour in response, but this in itself affects the application. The overhead introduced by the visualiser must therefore be minimal, to minimise the effect on the observed applications.

3.2 Visualisation Architecture

DIET agents have no complex communication protocols that can be analysed and used for visualisation purposes. Instead, the DIET core software provides only a basic event protocol. Events relate to the creation and destruction of elements, modification of element state, and interactions between elements in the DIET application.

As an attempt to address some of the limitations in recent agent-building frameworks, the visualisation architecture is designed as a separate platform.

Instead of providing a series of visualisation tools for a fixed set of different applications the visualisation and debugging of a range of tasks within the DIET environment is supported. The platform offers specific visualisation categories (see section 3.3). Within each category a diversity of visualisation components are provided, each of them pervading a particular task.

Visualisation components can be combined by the user to create so-called visual networks (see Sect. 3.4). A visual network formulates a particular visualisation task to observe and to interact with the application. There is no need for changing any code in the DIET application. Networks can be connected to or disconnected from the application at any time.

The visualisation process comprises three stages: (1) data acquisition, (2) data analysis, and (3) data presentation. In order to improve the visualisation performance and to minimize side effects the DIET visualisation platform is constructed as a layered architecture according to the visualisation flow (see Fig. 2).

A graphical user and programming interface supports the creation of user defined visualisation networks. In the following sections the visualisation categories provided by the platform are discussed.

3.3 Visualisation Categories

Currently the platform offers six different visualisation categories: (1) Application, (2) Filter, (3) Compute, (4) Control, (5) Interactor, and (6) View category.

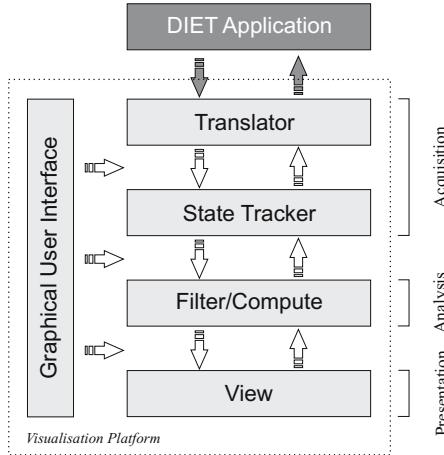


Fig. 2. The DIET visualisation platform. Events (arrows) are created by the DIET application and processed by the visualisation platform in order to analyse the application

The visualisation process itself is accomplished in a Publish-Subscribe manner, which allows the subscribed visualisation components (observer) to keep track of the internal state of DIET objects (environments, infohabitants, connections, messages). In order to improve the event processing speed and to achieve a flexible event distribution throughout the visualisation architecture we have incorporated the use of proxies into the visualisation platform as denoted in Fig. 3.

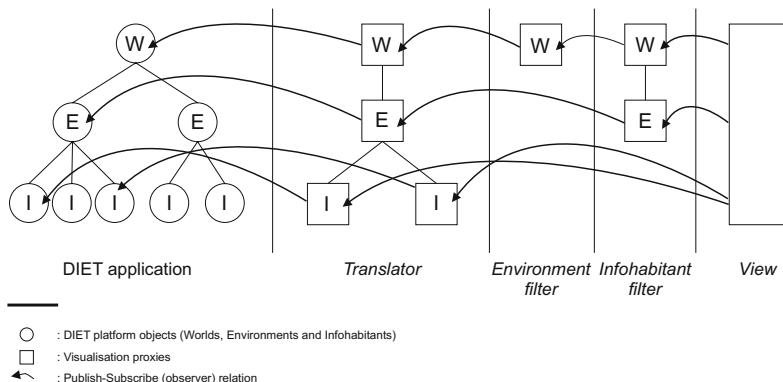


Fig. 3. An example visualisation configuration. A DIET application is visualised in a view, after applying an Environment filter and an Infohabitant filter

The visualisation architecture uses particular proxies: *WorldProxy*, *EnvironmentProxy*, *InfohabitantProxy*, and *ConnectionProxy*. Each of those corresponds to one of the fundamental DIET elements in the DIET application. Each proxy instance is

associated with a specific DIET core object. It generates events that can be used by subsequent visualisation components. In addition, further proxies exist that have no corresponding objects in the DIET application. These special proxies generate events that are not provided by the application (e.g. *PropertyProxy*).

With respect to the visualisation requirements stated in Sect. 3.1 the use of proxies has several reasons. Proxies are necessary to enable filtering of events. If the user wants to use a filter for family tags, there is an *EnvironmentProxy* that only passes events related to infohabitants with the proper family tag. All other events are suppressed so that subsequent views and other visualisation components in the chain are entirely unaware of the existence of these infohabitants.

Proxies make it possible to attach views to an application during run-time. The objects in the DIET platform do not maintain state that is required to do so. However, some of the additional proxies (e.g. Intermediate) can. This enables efficient visualisation of an application, as events are only generated and extra state is only maintained where this is needed.

Proxies make GUI components such as views independent of the DIET platform. Views are never observing DIET core objects directly. They are only attached to one or more visualisation proxies.

Proxies also enable the logging of events to a file (see next section). DIET objects such as infohabitants cannot be serialised, but their corresponding proxies can.

Figure 3 shows an example that illustrates how the visualisation architecture works. On the left, it shows the current state of a DIET application. It consists of two environments, containing five infohabitants in total. The first visualisation component is the Translator. It translates DIET platform events into visualisation proxy events. It also ensures that there is a visualisation proxy component for each element in the DIET application that is being visualised. Attached to the Translator is an Environment filter. It only passes through the events generated by the Translator's *WorldProxy* that are about a specific environment, e.g. Environment 1. Connected to the Environment filter is an Infohabitant filter. The specific filter here only passes through events about infohabitants with a specific family tag. Connected to this filter is a view, which displays all DIET elements that it is notified about. In this case, it is showing a single environment containing two infohabitants.

The figure demonstrates some of the advantages of the visualisation architecture that is used. First of all, visualisation proxy objects and events are only generated where they are needed. In the figure, one of the environments and its two infohabitants do not generate any events at all, because there are no observers registered with them. Furthermore, the use of generic proxy interfaces and generic proxy events enables the connection of visualisation components in a very flexible manner. For example, filtering of events and offline visualisation are both possible without any support in the views at all.

In the following sections the different visualisation categories and a selection of their components are discussed.

Application Category

The application category provides visualisation components which determine the application to be observed. In addition, this category provides services for offline visualisation.

Translator

The Translator component is the first visualisation component to be placed on the workbench as part of a visual network (see Sect. 3.4). The user can select a particular DIET application to be visualised by choosing the application name in a context menu. The Translator component translates received DIET events to visualisation proxy events.

This ensures a clear separation between the DIET core platform and the visualisation platform. In this way any event based multi-agent platform can be visualised only by adapting the Translator component.

Intermediate

The main purpose of the Intermediate is to observe the DIET application. The Intermediate component keeps track of received and propagated visualisation proxy events. It serves as an anchor and can be placed at any position within a visualisation network.

Other visualisation components can be attached to the Intermediate component during runtime. This property, for example, is useful, if the user is interested only from time to time in a particular view that visualises certain aspects of the application. Several Intermediate components can be placed on the workbench.

Recorder

The Recorder component dumps visualisation events to a file. As soon as the Recorder is placed on the workbench a file name is requested by the user. For offline visualisation purpose this file can be reloaded by the Player component to replay all recorded events. The component is attached to an Intermediate component and stores the following information:

- a timestamp,
- an event identification number
- an event object.

The recording can be started and stopped by the user by using the context menu of the component. An arbitrary number of Recorders can be placed within the visual network.

Player

The Player component enables offline visualisation by replaying a file previously dumped by a Recorder component. Basically, the component reads events one by one from a file. The event is interpreted and the corresponding DIET objects and their events are simulated by the component.

The playing can be started, paused, resumed, stopped, and executed stepwise by using the context menu of the component. Only one Player component can be placed on the workbench.

Filter Category

In order to improve the visualisation performance the user can determine which events are passed from one component to other components of the visualisation architecture. This task is the main purpose of the filter component. Through combination of different filter components more complex filter tasks can be achieved.

In general filters are parameterised; as an example the so-called Environment filter is parameterised by a name string. The name string defines the environment whose events are passed to the following component.

Compute Category

The data generated by the DIET kernel is very low level. Even basic information, like the population size of a particular environment, is not directly available for visualisation purposes.

In general, compute components calculate new visualisation properties by observing the events generated by DIET core components. The Population component for example processes environment events in order to count the infohabitant population within an environment.

Interactor Category

An interactor component enables the user to interact with the running DIET application. For security reasons the use of interactor components is restricted. Interactor components can only be applied if the developer of the application has explicitly granted access to DIET objects. Otherwise interactor components could be misused, e.g. a user could cheat the application by adding money resources to infohabitants of his own.

The Connection interactor, for instance, can be attached to a view (see Sect. 3.3). Two infohabitants can be selected in the view. This selection is passed to the interactor. On this basis, the component builds up a DIET connection between these infohabitants. Afterwards the user can send a text message from one infohabitant to the other one.

View Category

A view basically is a window that contains a collection of various graphical objects. Different views are provided by the platform, e.g. tables and bar charts. Each view contains graphical objects representing DIET entities or properties, respectively (see Fig. 4).

Simple graphical objects comprise icons, charts, or textual information. The appearance of a graphical object can be customized to increase its visual perception or to meet special visualisation requirements. A variety of attributes (e.g. colour, size, shape, etc.) can be changed by the Mapper component (see below). A selection mechanism is integrated in the view component as well; see the following section for details.

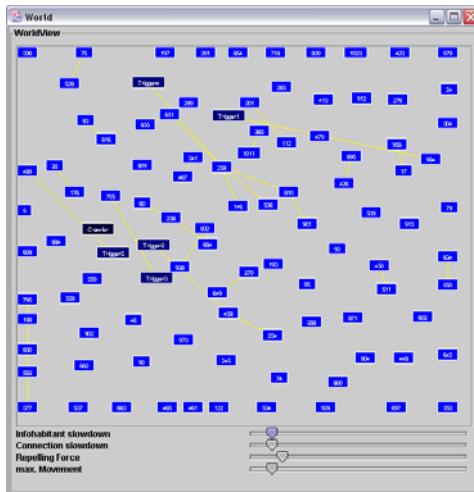


Fig. 4. The so-called WorldView visualises all infohabitants in a single view. This view helps to get an overall impression of the application

Control Category

The control category provides services to manipulate the graphical objects representing the different DIET core objects.

Group

The Group component makes basically two services available: selection and grouping. Within a view component each DIET entity can be selected by clicking the left mouse button. Selected entities can be grouped together by assigning a common group number. This group can be visualised in another view component.

For instance, the WorldView component shows all infohabitants in a world, but without much detail about each of them. The Group component allows the user to select a few infohabitants, and get detailed information about these in a separate view. A suitable view component for example could be the *TableView* that visualises the values of all their properties.

Mapper

DIET entities or properties are visualised by graphical objects in different view components. The Mapper component allows the mapping between DIET entities and properties on the one hand and graphical objects on the other.

The Mapper component allows the customisation of the appearance of each graphical object during runtime. For this purpose all incoming visualisation proxy events are processed and presented to the user in a dialog window.

3.4 Visual Programming Interface

The visual programming interface supports the creation of user defined visualisation networks. Visual networks are created on the workbench that appears as soon as the visualisation platform is started.

As part of the workbench a toolbox is displayed that allows the selection of individual visualisation categories. Within each category different visualisation components are accessible. Visualisation components developed by the user can be accessed by clicking with the right mouse button in the component list of the particular category (see Fig. 5). Visualisation components are placed on the workbench by drag and drop using the mouse.

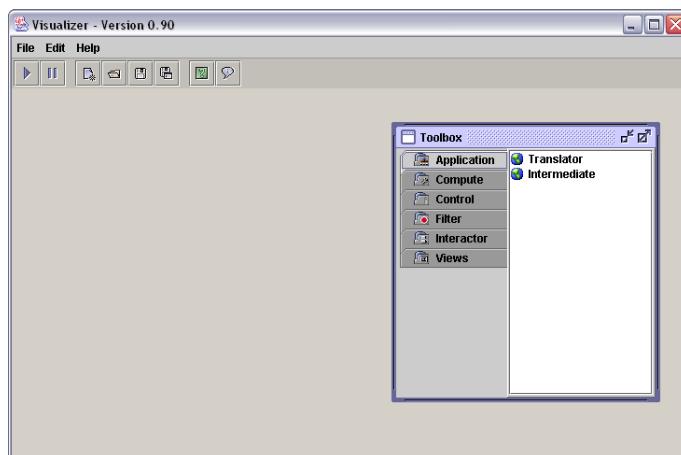


Fig. 5. The graphical user interface. As part of the workbench a toolbox is displayed that allows the selection of individual visualisation categories. The toolbox contains six categories. Each category offers different visualisation components

As soon as a visualisation component is placed on the workbench the component is represented by a graphical icon. The input and output ports displayed on the left and the right hand side of the icon indicate the components to which this component can be registered and to which components it can pass the processed events. Different components within the same category may have different ports associated with them.

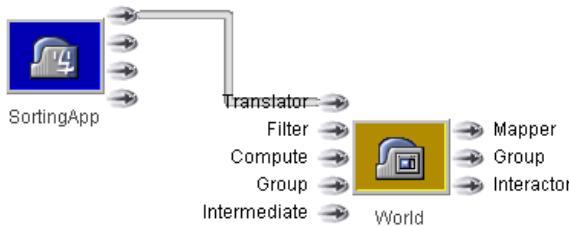


Fig. 6. A view icon representing the WorldView. The view is connected to a Translator component

Each visualisation component can be connected to one or more components by clicking the component with the left mouse button and dragging a connecting line to the other component while the left mouse and the SHIFT key is pressed. After releasing the mouse button a connection will be drawn, if a suitable port is available. In this way a visual network can be defined in a very efficient way. Visual networks can be saved in XML format for later use.

The following section introduces a simple DIET application and demonstrates the use of visual networks to gather particular information.

4 Example

The so-called *Migrate* application is provided as an example application by the DIET platform. The application demonstrates how infohabitants can move from one environment to another. The application creates Migrator infohabitants, each with its own different sleep interval. The sleep interval determines the time step each infohabitant waits before it migrates to another region. After instantiation of the application the infohabitants start moving randomly between the various neighbouring environments. The *Migrate* application creates three environments (Env0, Env1, Env2) that are populated by ten infohabitants in total.

The user may be interested in the individual number of infohabitants that populate a particular environment. How can we visualise this information with the components provided by the visualisation platform?

Figure 7 shows the visualisation network that visualises the infohabitant population in a particular environment. An Environment filter selects all events (e.g. from Env0) and passes them to a Population component. Based on the creation, the

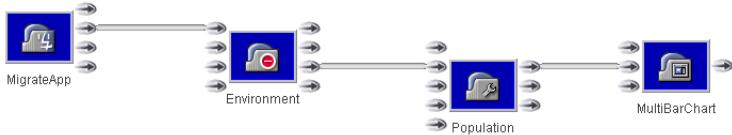


Fig. 7. A visualisation network that visualizes the infohabitant population of a particular environment in a MultiBarChart view

destruction, and the migration events this compute component calculates the actual population of the environment. The outcome of this calculation is visualised by a MultiBarChart view that is connected to the compute component.

By defining similar networks for the remaining environments (Env1, Env2) and passing the results of the compute components to the view we get the network illustrated in Figure 8. The corresponding graphical output is shown in Fig. 9.

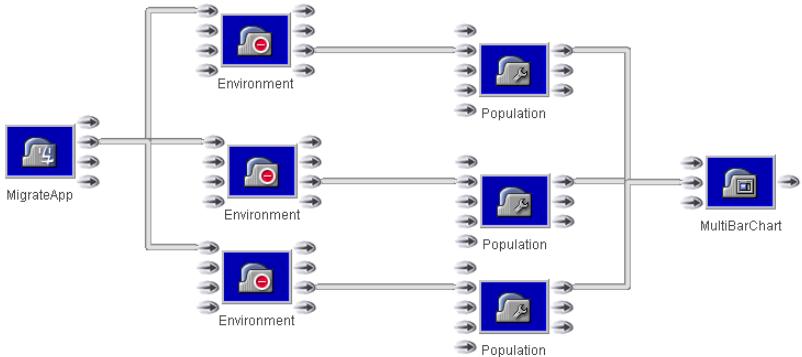


Fig. 8. The complete visualisation network that visualises the infohabitant population of each environment within the migration application

This simple example shows how efficient the visualisation components can be used to gather information that is not provided by the application itself. More complex information can be derived by combining different components. Due to the use of basic components, it is not necessary to develop overloaded single-use visualisation modules anymore. Furthermore, the design of the architecture enables the introspection of DIET applications without the need for changing any application code or even some knowledge about the code.

5 Conclusion

The multi-agent software platform DIET provides components to implement decentralised, open, robust, adaptive, and scalable ecosystem inspired applications. The

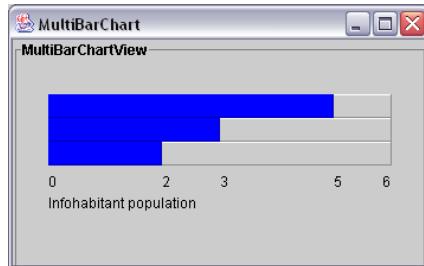


Fig. 9. The output of the MultiBarChart view. Currently Env0 is populated by five infohabitants, Env1 by three infohabitants, and Env2 by two infohabitants. The all-time maximum population in one of the environments is equal to six infohabitants

development, debugging and monitoring of decentralised systems is a complex task. Visualisation can be an effective way to support the developer in the design and analysis of multi-agent systems. However, the dynamic visualisation of multi-agent systems for this purpose is largely ignored.

Some related work can be found in [7]. Schroeder and Noy visualised a post mortem set of call data derived from agent systems. Their work focuses on the aspect of showing the relations between agents in contrast to the visualisation of a single agent. Schroeder worked on the visualisation of arguing agents as well. In [6] the internal reasoning and the external argumentation of agents is examined. Both aspects are visualised as avatars, Chernoff faces or distance graphs as proposed in [6].

In this paper we have described our approach to visualise agent systems build upon the DIET software platform in a more general way. The visualiser provides fundamental components and functions extending the DIET core layer by visual and interactive components. These allow for the introspection and manipulation of DIET applications without the need of changing any application code. The integrated graphical user interface is intuitive to learn and easy to use. Visualisation tasks can be formulated by building visual networks.

Acknowledgements

This work was carried out as part of the DIET project (IST-1999-10088), within the Universal Information Ecosystems initiative of the Information Society Technology Programme of the European Union. We thank the other participants in the DIET project, from the Departamento de Teoría de Señal y Comunicaciones, Universidad Carlos III de Madrid and the Department of Electronic and Computer Engineering, Technical University of Crete, for their comments and contributions.

References

1. DIET open source project. <http://diet-agents.sourceforge.net>.

2. Universal information ecosystems initiative. <http://www.cordis.lu/fetuie.htm>.
3. Hoile C, Bonsma B, Wang F, and Marrow P. Core specification and experiments in DIET: a decentralised ecosystem-inspired mobile agent system. In *Autonomous Agents and Multi-Agent Systems (AAMAS2002)*, Bologna, Italy, 2002.
4. Gamma E, Helm R, Johnson R, and Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Massachusetts, 1995.
5. Nwana HS, Ndumu DT, Lee L, and Collis JC. ZEUS: A toolkit and approach for building distributed multi-agent systems. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third Annual Conference on Autonomous Agents (AGENTS-99)*, pp. 360–361, New York, May 1–5 1999. ACM Press.
6. Schroeder M. Towards a visualization of arguing agents. *Future Generation Computer Systems*, 17(1):15–26, September 2000.
7. Schroeder M and Noy P. Multi-agent visualisation based on multivariate data. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pp. 85–91, Montreal, Canada, May 2001. ACM Press.
8. Marrow P, Koubarakis M, van Lengen RH, Valverde-Albacete F, Bonsma E, and et al. Agents in decentralised information ecosystems: the DIET approach. In *Artificial Intelligence and Simulation of Behaviour Convention 2001 (AISB01). Symposium on Intelligent Agents and E-commerce*, pp. 623–630, University of York, United Kingdom, 2001.
9. Waring RH. Ecosystems: fluxes of matter and energy. In Cherrett JM, editor, *Ecological Concepts*. Blackwell Scientific, 1989.
10. van Liedekerke MH and Avouris NM. Information and software technology 1995 37 (2) 103-112, July 11 2002.
11. Cui Z, Odgers B, and Schroeder M. An in-service agent monitoring and analysis system. In *Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence*, pp. 237–244, Chicago, USA, 1999. IEEE Press.

Facilitating the Visual Analysis of Large-Scale Unsteady Computational Fluid Dynamics Simulations

Kelly Gaither¹ and David S. Ebert²

¹ The University of Texas at Austin
kelly@tacc.utexas.edu

² Purdue University
ebertd@purdue.edu

1 Introduction

Computational power has increased dramatically over the past decade and has allowed computational fluid dynamics (CFD) researchers to more accurately simulate many types of flow with more detailed computational meshes. This increase has supported the calculation of highly detailed unsteady simulations that model real world conditions. However, this new power has yielded terabytes of data, and CFD researchers now face the very difficult task of trying to find, extract, and analyze important flow features buried within these monstrous data sets. Although being able to accurately model these problems is a major accomplishment in the world of CFD, it comes with the caveat that the amount of information that must be sifted through has now grown to enormous proportions that overwhelm investigators.

Unlike the explosive growth in computational power, visualization tools for these very large data sets have experienced a more modest evolution, and are not yet sufficiently developed to significantly aid the feature detection, extraction and analysis process. Most visualization tools require the user to work at a very cumbersome, low level with the data. Users are required to dice their data sets into appropriate data parameter ranges to search for the features of interest, since detailed visualization of such large data sets is impractical. Additionally, current visualization tools do not have the necessary capabilities to allow high-level exploration and analysis of these enormous datasets on desktop PCs, thus limiting their widespread use by scientists and designers. Previously developed systems for the interactive visualization of large CFD simulations [1, 2, 4], either work at a low-level with the data [1] or require high-speed network connections between the CFD simulation and the desktop visualization system and only work with a fixed set of basic features [2, 4].

The typical approach for visualizing large-scale CFD simulations is to pre-compute offline the visual representations that might be of relevance during the analysis process. This stage brings with it the hazard that often occurs when subsampling data. If the sample rate is not chosen with care, relevant features may be missed and the analysis to the solution may provide incorrect or misleading results.

This process can also be very time consuming and tedious requiring multiple passes into large stores of data without automatic methods for guiding placement and location of visualization samples.

CFD researchers desperately need new techniques that simplify and automate the iterative search and extraction process of finding the vital information in their data set. Improved visualization tools for CFD researchers could have a drastic impact on the design and safety of vehicles such as aircraft, ships, submarines, spacecraft, and automobiles. This community needs a new system that allows the users to articulate appropriate features of interest, provides a compact representation of these features which preserves their intrinsic qualities, and allows users to interactively visualize the feature information on a desktop computer. Such a system would also have to overcome the additional challenges of loading a sufficient portion of the data set into the available memory on a modest desktop machine, transferring these data sets over a network connection between the archive and local machine, mapping the entire data set to a visual representation in a reasonable amount of time, and rendering the results at interactive rates.

Therefore, we believe that a new visualization pipeline is needed for large-scale CFD visualization. As described in the following sections, our new system allows researchers to work at an intuitive, perceptually meaningful level with their data at their desktop. This new pipeline incorporates a flexible domain knowledge interface and a compact procedural encapsulation of large unstructured datasets to allow interactive exploration and analysis of CFD data.

2 Relevant Issues

Improving the current process for visualizing large-scale unsteady CFD data sets requires an understanding of the problematic issues that have dominated the current techniques.

2.1 Large Scale Data

The issue that gets the most attention is the size of the data sets. Often, a single time step can be larger than the memory capabilities of modern hardware. This is further complicated when the time dimension is being explored as well. It would be difficult to determine an average data size since this is entirely dependent on the resolution of the discretized volume in which the geometry is modeled and the solution computed. It would be fair to say that the size of the data set and the resolution with which it needs to be visualized increases with the complexity of the underlying physical problem being modeled.

2.2 Computational Grid Structures

The discretization, commonly called a computational grid can be generated in a variety of manners. The simplest of these cases is a voxelized grid that contains three-dimensional elements, or voxels, that are all the same shape and size. There is no

need for additional adjacency information, as this is implied in the way the grid is stored. Another method for generating a computational grid is curvilinear, a grid generated in the dimensions I, J, and K holding six sided cells that have a non-linear shape function applied to the faces. This grid structure also has implied adjacency information, but the interpolation functions are more complex inside the cells and on the faces. A third type of grid structure that is commonly found in the computational simulation domain is the unstructured grid, typically a collection of tetrahedra, prisms, pyramids, and hexahedra that match at the faces. Because this structure is not stored with any adjacency information, this structure requires the explicit calculation of neighbor information when necessary.

2.3 Multivariate Data

The CFD domain has a base set of variables that are typically stored, from which all other parameters can be derived. Although the solutions are typically computed cell-centered, the solution parameters are often stored at the nodes of the computational grid. The solution values that are stored are the Q values: density, u component of velocity, v component of velocity, w component of velocity and energy. Additional variables such as pressure, temperature, and helicity can be derived from this base set of stored quantities.

2.4 Regions of Interest

Efficiently navigating and querying through these vast amounts of data requires some way of determining what is important and what is not. These regions of interest can be further refined into specific structures that the analyst would like to isolate and query. Current methods for detecting and tracking features are discussed in the next section.

2.5 Visualizing at Interactive Frame Rates

Interacting with the data is a key component to successful analysis. Prior to the introduction of a sophisticated Graphics Processing Unit (GPU V a single chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines capable of processing a minimum of 10 million polygons per second) [5], methods for displaying large-scale data at interactive frame rates required a sub-sampling of the data that was amenable for interactive display. With the maturation of highly parallel graphics hardware, we can now handle more complex computations that operate locally on the graphics chip. Additionally, combining compactly represented procedural encoding techniques to compress the data and the graphics processing unit to uncompress and visualize the results gives us much more flexibility in the sophistication of the displays.

3 Feature Detection and Feature Tracking

Current feature detection techniques consist primarily of analytical methods for identifying cells within the computational grid that meet some functional criteria and methods that track the formation and dissipation of cells that have been pre-defined to contain feature characteristics.

Analytical feature detection began as early as 1875 when Poincare made the connections between differential equations and the topology of vector fields. A historical survey can be found in [6]. One of the first applications of Poincare's theory of differential equations to three dimensional fluid flows was done by Lighthill [7]. In [8], this approach was extended to include a more general class of flows. Automatic techniques for identifying the topology in simulated flow fields was first introduced by Helman and Hesselink [10–12] and Shirayama and Kuwahara [9]. Visualizing this topology to understand the underlying physical flow structure was introduced by Globus, Levit and Lasinski [13]. They presented methods for calculating and classifying three dimensional flow structures such as sources, sinks, and saddles by computing the eigenvalues of the Jacobian matrix. Since then, analytical methods have been used to detect structures such as vortices [14, 15], shocks [3, 16], wakes [17], and recirculation regions [18]. The current techniques allow little if any flexibility for defining features through a higher level language, and these methods give little insight into the behavior of features over a period of time. Examples of these kinds of analytically described features can be seen in Figs. 1–4.

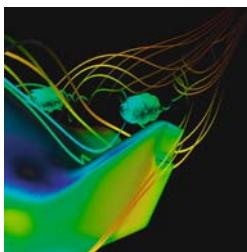


Figure 1: Vortices Over X38

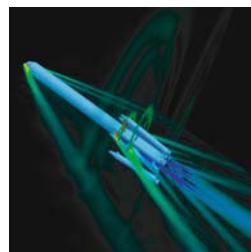


Figure 2: Shock Rings Around DeltaII

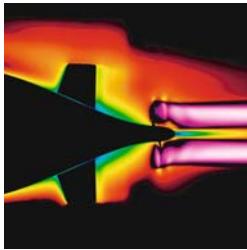


Figure 3: Wake from Propellers

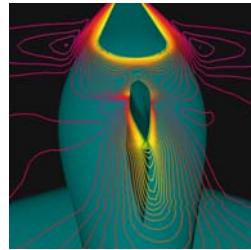


Figure 4: Recirculation Region

Figure 1 illustrates a vortex tube that forms from the air flow over an X38 crew recovery configuration computed at a 30 degree angle of attack. This figure displays the most common method for displaying vortices—particle traces. The traces in Fig. 1 are colored with pressure values.

Shown in Fig. 2 is an example of shock rings that form around a Delta II configuration during the transition from subsonic transport to hypersonic transport. The shock rings are shown by isolating density values contoured on cutting planes computed orthogonal to the body. These values are highlighted by blending out all other values with the use of transparency.

Figure 3 displays the wake region that forms during a hydrodynamic simulation of a submarine during a stopping and turning maneuver. This wake region is generated from the turbulence shed off of the propeller and has been found to be the primary cause of noise and drag during live maneuvers.

A recirculation region generated from flow around the rudder and fins of a submarine is shown in Fig. 4. This region can result in an area of suction and cavitations causing significant drag on the body during maneuvers.

Understanding the added time dimension as it relates to the physical problem is a crucial part of analyzing the solution, both for verification and for integration into complex physical design. Work has been done to develop a full 3D volume feature extraction and tracking algorithm that tracks thresholded, connected regions [19, 20]. This work is responsible for an initial classification of events: Continuation (an object continues, it may shrink or grow), Creation (a new object appears), Dissipation (an object dies), Bifurcation (an object splits into two or more objects), and Amalgamation (two or more objects merge). These features are classified by tracking properties of connected regions such as mass, moment and volume changes. This method looks at the next time step as an indicator of what type of event is occurring. This method, however, suffers in practice because it has no method for handling the periodic Dissipation and Creation of events such as tip vortices that are created and die in periodic intervals. Additionally, there is no scheme for giving the complete information for features that behave non-deterministically.

An example of the types of composite features that are of interest is shown in Fig. 5. This figure (courtesy of David Marcum, Mississippi State University) shows a set of potential features that exist on a finned missile configuration. In viscous simulations, the boundary layer contains a significant amount of important information and is the primary indicator for transition from laminar to turbulent flow. The shock wave is found as a connected or nearly connected set of cells containing a discontinuity. The expansion fan gives an indication of an expansion shock that occurs because of the movement of the fin. The wake sheet is of critical importance when studying both noise and vibration as both can result in an unstable transition. Finally, vortices are of critical importance when studying noise and instability.

Although each of these features can be individually identified by analytical techniques, we are often interested in the relationships between features and the given geometries and between features of similar and differing types. To properly identify this behavior, we must turn to a combination of analytical techniques, statistical properties of the volumetric domain, and heuristics applied during the analysis. This

allows us to work at a higher level, producing a hierarchy of decisions that together form the conclusions about features, such as existence, persistence, coherence, and correlation to other features.

Working at this higher level is critical to the ability to identify features that are neither well understood nor well defined through analytical descriptions. For example, lambda shocks (Fig. 6) are well understood in theoretical aerodynamics, but are not well described analytically. The general shape is known, but the orientation, strength and persistence of this type of feature requires a significant amount of heuristics to properly identify. The lambda shock in Fig. 6 was captured through a series of shaded cutting planes sliced orthogonal to the wing. This lambda shock is shown using density to contour the cutting planes. Thresholding was used to make uninteresting regions transparent.

Understanding the correlation and intermingling of features is necessary to properly study events such as mixing. Figure 7 illustrates one time step in a sequence of time steps that demonstrate one solute being mixed over time with another.

Turbulent flow presents unique challenges. The scale of features in a turbulent boundary layer is at a very fine scale, typically orders of magnitude more precise than laminar boundary layers. This precision is necessary to capture all of the small scale features that contribute to turbulent behavior. Additionally, we face the problem that the features that occur in a turbulent boundary layer are not all well defined. Given the proper scale, we can easily find vortical structures or eddies. It is significantly harder to analytically describe seemingly random behavior such as that shown in Fig. 8, showing turbulent channel flow. We can easily see that features of interest exist in the flow by looking at cutting planes contoured with velocity magnitude. However, these features are very difficult to describe analytically. Therefore, it is critical to add statistically based information and heuristics to properly identify the features of interest.

The issue becomes even more complex when trying to understand and analyze the features present in complex flow of complex geometries, such as those attempting to simulate real-world conditions. Figure 9 shows four views of a Delta II configuration with strap on boosters. These boosters are used to propel the Delta II into space, but drop off shortly after. It is the peeling off of these boosters that can cause very complex structures and instabilities to form that can affect the viability of the Delta II in flight. Understanding these structures is the key to being able to compensate and adjust for them. However, this is a process that is not well understood at all from the feature detection community. It is here that we must implement and deploy our hierarchy of features, giving us the ability to detect features that we suspect are present and those that we do not know are present. Additionally, understanding the interplay of these features allows us to understand the causal relationship of possible instabilities, thus reducing the possibility of vehicular failure.

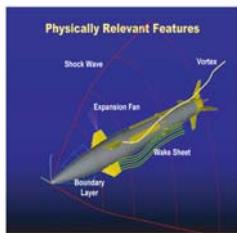


Figure 5: Finned Missile Features

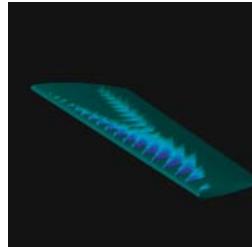


Figure 6: Lambda Shock

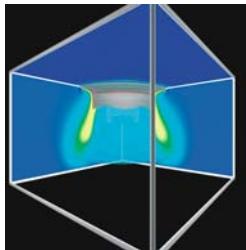


Figure 7: Mixing Solutes

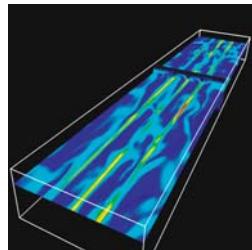


Figure 8: Turbulent Channel Flow

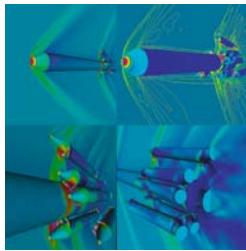


Figure 9: Delta II Features

4 A New Visualization Pipeline for Large-Scale CFD Simulations

It is clear that the existing methods for visualizing these large-scale data sets need improvement. To this end, we are solving these CFD Visualization problems by developing techniques for creating a procedural abstraction for very large data sets, developing effective and efficient methods for mapping from the procedural to visual representation, and applying these techniques to the problem of visualizing large CFD simulations. Our new methods provide interactive visualization of very large data sets on a desktop computer, and will scale gracefully across a range of computer power and bandwidth situations. These new methods can be summarized in a new pipeline for visualizing large-scale CFD data sets. This new pipeline can be adapted easily to any sequences of large data sets whether measured or simulated. This pipeline operates in the following manner:

4.1 Detect Candidate Features

General characteristics of a given data set can be found by applying a set of statistically based methods. This gives the opportunity to find properties such as connected cells, including 1 nearest neighbor, 2 nearest neighbor, etc, persistence, energy, entropy, and area. These are useful in determining areas of potential features or to detect candidate cells that may contain features that can be found by a combination of analytical techniques and domain dependent heuristics.

4.2 Further Refine Features

These candidate cells can be further refined by applying a combination of analytical techniques with procedural descriptions of domain-dependent heuristics. For example, we consider every feature to be a reaction to an action. If the problem that is being solved is the physics surrounding a geometric body, then a feature cannot exist without having some causal relationship to either the geometry or to other features.

4.3 Procedurally Encode Features

To facilitate the visualization of these features on a desktop PC, we represent these features using a procedural representation of implicit models based on radial basis functions.

4.4 Apply Multiresolution Techniques

We can then adapt the procedural representation to the appropriate level of detail using multi-resolution techniques.

4.5 Incorporate Knowledge Gained into Metadata

The feature information is encapsulated as domain specific knowledge in the metadata. This allows us to explore these extremely large data sets both at the feature level and, more importantly, at the higher level of relationships among features, including feature to feature relationships and feature to geometry relationships.

4.6 Visualize Directly From the Procedural Representation Using the GPU When Possible

We then visualize the data directly from the procedural representation, using and extending numerous familiar CFD visualization techniques (e.g. cutting planes, iso-surfaces, volume splatting, direct volume rendering, particle clouds, streams, rakes, line-integral convolutions, and glyphs).

4.7 Verify Accuracy

Accuracy is a key component to the viability of this pipeline. Although it is not currently standard practice to measure the accuracy of the CFD simulations, it is necessary to ensure that the procedural is accurate to some pre-define threshold. For our work, we have defined an acceptable tolerance to be 5communicating with researchers in the CFD domain. We are facilitating this by careful tracking of approximation error throughout the entire process, including scanning, modeling, reconstruction and visualization.

5 Conclusion

Our system containing the new visualization pipeline will allow CFD researchers to work more effectively by interactively exploring their data to pinpoint the features of interest. This will facilitate and allow them to adaptively refine their solution grids within those areas, and iteratively compute solutions on these adapted grids to improve their simulation and solve the underlying flow problem more quickly. Moreover, the results of this project will provide solutions not only for CFD researchers, but also for a wide variety of visualization challenges and applications. Our main goal is to develop techniques that allow visualization exploration, feature detection, extraction and analysis at a higher, more effective level through the use of procedural data abstraction and representation.

Acknowledgements

This work is supported by the National Science Foundation under grant NSF ACI-0121288. The authors would like to thank and acknowledge those responsible for computing the simulations that produced the data in the Figs. 1–9. The data for Figs. 1–6 and Fig. 9 was generated at the Simulation and Design Center, Mississippi State University. The data for Fig. 7 was generated at the Computational Fluid Dynamics Laboratory at The University of Texas at Austin. The data for Fig. 8 was generated by Dr. David Goldstein’s laboratory at The University of Texas at Austin.

References

1. Freitag, L., Loy, R.: Using Desktop Graphics Workstations for Interactive Remote Exploration of Large Data Sets. Proceedings Using Desktop Graphics Workstations for Interactive Remote Exploration of Large Data Sets, (2000)
2. Haimes, R.: Pv3: A Distributed System for Large-Scale Unsteady CFD Visualization, AIAA 94-0321, (1994)
3. Lovely, D., Haimes, R.: Shock Detection from Computational Fluid Dynamics Results, AIAA 99-3285, (1999)

4. Haimes, R., Kenwright, D.: On the Velocity Gradient Tensor and Fluid Feature Extraction, AIAA 99-3289, (1999)
5. NVIDIA: Graphics Processing Unit (GPU) at <http://www.nvidia.com/object/gpu.html> (1999)
6. Abraham, R. Shaw, C.: Dynamics: The Geometry of Behavior, parts 1-4, Ariel Press, Santa Cruz, CA, (1984)
7. Lighthill, M.: Attachment and Separation in Three Dimensional Flow, Laminar Boundary Layers II, ed. L. Rosenhead, pp. 72-82, Oxford University Press (1963)
8. Perry, A. Fairly, B.: Critical Points in Flow Patterns, Advances in Geophysics, 18(B), pp. 299- 315, (1974)
9. Shirayama, S. Kuwahara, K.: Flow Past a Sphere: Topological Transitions of the Vorticity Field, AIAA-90-3105-CP (1990)
10. Hesselink, L., Helman, J.: Evaluation of Flow Topology from Numerical Data, AIAA-87-1181 (1987)
11. Helman, J., Hesselink, L.: Representation and Display of Vector Field Topology in Fluid Flow Data Sets, IEEE Computer, pp. 27-36 (1989)
12. Helman, J., Hesselink, L.: Surface Representation of Two- and Three-Dimensional Fluid Flow Topology, Proceedings of IEEE Visualization 1990 (1990)
13. Globus, A., Levit, C., Lasinski, T.: A Tool for Visualizing the Topology of Three-Dimensional Vector Fields, Proceedings of IEEE Visualization 1991 (1991)
14. Banks, D., Singer, B.: Extracting Vortices From an Unsteady Flow, Proceedings of IEEE Visualization 1994 (1994)
15. Sujudi, D., Haimes, R.: Identification of Swirling Flow in 3-D Vector Fields, AIAA 95-1715, (1995)
16. Sobieczky, H., Hannemann, M.: Computational Shock and Mach Waves Visualization Aiding the Development of Aerodynamic Design Techniques, Proceedings of the 21st International Symposium on Shock Waves, (1997)
17. Haimes, R.: Automated Feature Extraction from Transient CFD Simulations, Proceedings of the 7th Annual Conference of the CFD Society of Canada, Keynote Address. Halifax, NS, May (1999)
18. Haimes, R.: Using Residence Time for the Extraction of Recirculation Regions, AIAA 99-3291 (1999)
19. Silver, D. Wang, X.: Volume Tracking, IEEE Transactions on Visualization and Computer Graphics, 3(2), June (1997)
20. Silver, D., Wang, X.: Tracking Scalar Features in Unstructured Datasets, Proceedings of IEEE Visualization 1998 (1998)

Evolving Dataflow Visualization Environments to Grid Computing

Ken Brodlie, Sally Mason, Martin Thompson, Mark Walkley and Jason Wood

School of Computing, University of Leeds, Leeds LS2 9JT, UK

1 Introduction

A major new development in computing in the past few years has been the emergence of Grid computing, where applications can exploit in a secure manner substantial remote computing resources. This offers new opportunities to scientists and engineers. Very large simulations can now be performed, almost routinely, by reserving a distributed set of resources in advance. A key difference from traditional distributed computing is that the resources may span multiple institutions, and so issues of trust become much more significant. In a crisis situation, resources not normally required can be commandeered and dedicated to a crucial simulation or analysis task. In these situations the importance of visualization to gain rapid understanding of the results is well established.

As these new computing technologies emerge, so the challenge for visualization system designers is to evolve existing systems to take advantage of these new technology developments – rather than re-design from scratch on every occasion. The dataflow visualization environment, pioneered over a decade ago, has proved particularly flexible and resilient, and has successfully adapted to the many recent changes in computing.

In this paper, we explain how the dataflow visualization environment can exploit the arrival of Grid computing to provide a desktop interface for remote computational steering. This is by no means the only approach to visualization in Grid computing, and so we begin in Sect. 2 with a brief review of other work. Section 3 traces the development of dataflow systems, showing how they quickly emerged as environments for computational steering as well as data visualization; and how they evolved to encompass collaborative visualization, supporting the activities of geographically dispersed research teams. This sets the context for our work, as we proceed to show how earlier reference models for dataflow visualization can extend to Grid-based visualization. Our research is driven by what can be seen as a typical Grid application: a crisis scenario in which a pollutant escapes from a chemical factory and its dispersion under different wind directions must be predicted with maximum speed and accuracy in order to plan evacuation. Section 4 describes this application, with

Sect. 5 detailing the simulation process we used. Section 6 explains how we evolve the model for dataflow systems to enable computational steering and visualization on the Grid; and in Sect. 7 we describe a demonstrator built from the model, and addressing the pollution application. In Sect. 8, we show how the work can be extended to allow several collaborators to take part in the analysis process.

2 Visualization and Grid Computing

Grid computing is defined as the provision of flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions and resources [8]. Regardless of their heterogeneous nature and geographic location, access to these resources should appear simple and seamless. Only recently, with the advent of high-speed wide area networks, has this ambition become a practical reality. Examples of this can be seen in national-scale computational grid efforts such as the NSF-funded TeraGrid in the US [22], linking nine major computing centres, and the UK National Grid Service [16].

Visualization is key to understanding the “tidal wave” of data that is being generated by scientific applications running on the Grid. In an early paper, Foster et al. [7] put forward their vision of distance visualization where the computation and visualization are decomposed into steps that may be located on different resources. An application to tomographic reconstruction is described. However they choose to develop their own visualization architecture. This is the approach put forward also by Shalf and Bethel [21]. They argue that existing systems such as VTK, AVS and OpenDX are attractive in their flexibility and extensibility, but were designed primarily for a single machine (or at most a limited number of machines). They go on to claim

But deploying such systems onto the grid requires consideration of new conditions not likely anticipated during their initial design and implementation. A distributed visualization architecture – a framework suitable for component-based, grid-enabled visualization – can best meet these additional design considerations.

Their vision is of a world where distributed, heterogeneous components are available as building blocks to allow application builders to create optimum tools for the task. Our position in this paper is rather different. While the Foster and Shalf-Bethel vision may be a holy grail, there is much we can do right now by a small evolution of existing systems. The reward of our approach is a solution for Grid-based visualization that is available now, and exploits the years of development effort which has been invested in these systems. A particular feature of our approach is that we are able to quite naturally extend from single-user to multi-user working.

Before we introduce our approach, we mention some other instances of using visualization in Grid computing. Computational steering on the Grid has been discussed by Engquist [6]. He uses the VTK software to construct a steering application, but without the ability to collaborate in controlling the simulation. The UK e-Science

RealityGrid project has also used VTK, and has developed an API for computational steering [4]. Recently this project has carried out a highly impressive demonstration of Grid-based visualization and computational steering at Supercomputing 2003, in collaboration with the USA TeraGrid – one of the largest ever lattice-Boltzmann simulations was carried out. Data visualization (as opposed to simulation visualization) is also benefitting from Grid computing – Norton and Rockwood [17] describe a progressive approach to view dependent volume visualization. This addresses the important topic of compression – a vital aspect in distributed visualization. Jankun-Kelly et al. [12] show how existing web-based visualization tools can be deployed through Grid portals – with particular emphasis on spreadsheet-based visualization. Finally, Bethel and Shalf [2] discuss the importance of network efficiency in distributed visualization on the Grid, with experience from Visapult (a parallel volume renderer) and Cactus (a framework for high performance simulation and visualization that has been used for a number of Grid applications).

Our contribution in this paper is to return to the fundamental reference model for dataflow visualization, and show how this model can be extended to include a link to a remote, Grid-based simulation. In this way we provide a blueprint for how to extend the several modular visualization environments which are based on this fundamental model. Therefore in the next section we trace the evolution of dataflow visualization systems, so that we can set the scene for our extension.

3 Dataflow Visualization Environments

Many see the NSF Report of McCormack, de Fanti and Brown [15] as marking the beginning of the modern visualization era. This report sparked the development of a number of dataflow visualization systems, such as AVS [1], IRIS Explorer [26] and IBM Open Visualization Data Explorer [18] – all three of which are still in active use today. These systems were based on a pipeline model of visualization, elegantly presented by Haber and McNabb [11] and shown in simplified form in Fig. 1. Data is read in; it is converted to some abstract geometric representation; and this is then rendered as an image on the display.

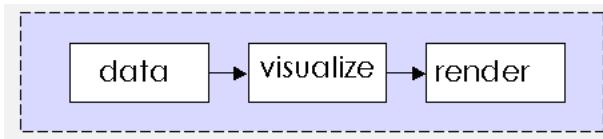


Fig. 1. Basic Visualization Pipeline

From the outset, the systems were used both for visualization of data acquired by observation or measurement, and also for visualization of the results of simulations. In the latter case, three models quickly emerged and were nicely characterised by Marshall et al. [14]. The simplest model is to use visualization as a **post-processing**

step: the simulation is executed, the results written to storage, and then visualized in a separate step using the pipeline of Fig. 1. This has the advantage of allowing the scientist to analyse the results at their own pace, and share the results with others, but it has some major limitations. Errors in simulations cannot be detected until the simulation completes, making mistakes expensive, and there is no opportunity to interact with a simulation on the basis of the intermediate results.

Fortunately a key feature of all the dataflow visualization systems is their extensibility. In addition to the set of modules provided with the system, users can embed their own simulation code as a module. This enables a different model for linking simulation and visualization, termed **tracking** by Marshall et al.. Essentially the process marked “data” in Fig. 1 is replaced by a “simulate” process. Now the results can be viewed as they are computed and any errant simulations immediately halted. However even greater flexibility is provided in the third model, termed **steering**, in which control parameters of the simulation can be modified as it proceeds – schematically, this gives a basic pipeline as shown Fig. 2. In the pollution application we shall use as demonstrator in this paper, the ability to change wind direction as the simulation executes will be vital.

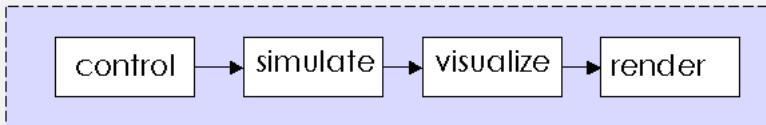


Fig. 2. Steering

The early visualization systems mentioned above (IRIS Explorer, IBM Open Visualization Data Explorer and AVS) can all be used for computational steering in these ways. Moreover a number of newer systems have emerged – such as SCIRun [13, 19] for example – which aim to provide special support for close coupling of simulation and visualization. Toolkits such as VTK [20, 25] can also be used, through the application developer incorporating toolkit components within their simulation code. In all cases however the underlying conceptual model is as in Fig. 2.

As the use of the Internet expanded, and more and more research was carried out by geographically separate research teams, so the push came for visualization systems to be collaborative. Again the dataflow systems were able to evolve, allowing users at different locations to link their pipelines across the Internet – thus enabling exchange of raw data, geometry or images, as well as shared control of parameter settings on modules. This was realised first as COVISA [28], then an extension to IRIS Explorer but now an integral part; and also later as extensions to AVS in the MANICORAL project [5] and in the cAVS project [3]. The model is shown in Fig. 3: a collaborative server process links the sessions allowing data from one pipeline to be transmitted to a collaborator’s pipeline. Here the data is passed to the collaborator to visualize as they want, but equally the sharing could be programmed to take place further down the pipeline, after the “visualize” step. This programmability allows

the collaborative application to exploit different bandwidths between the locations of the research team. For example, in a low-bandwidth situation, a group might choose to share data at the start of a collaboration, with only parameters being exchanged during the session. Alternatively, with high-bandwidth connections, geometry and image data might be exchanged as the session proceeds.

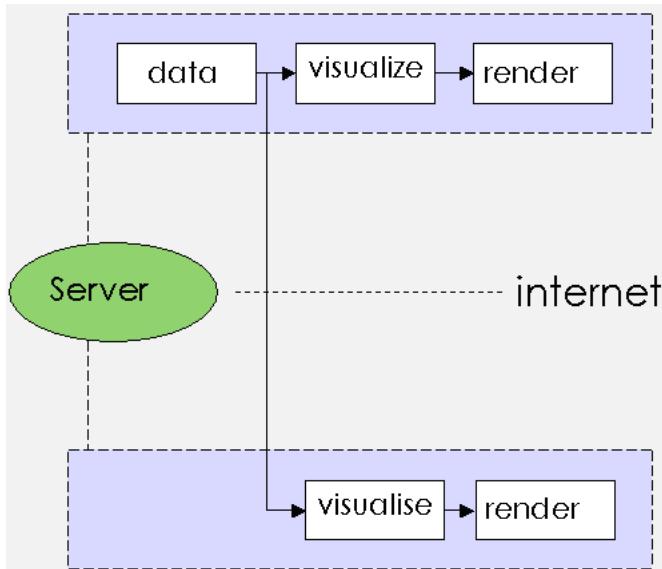


Fig. 3. Collaborative Visualization Pipeline

Our concern in this paper is to evolve the dataflow model one step further, to encompass Grid computing. Rather than immediately describe the extended model in the abstract form of Figures 1 to 3, we shall first motivate the work by describing a typical application of Grid computing where visualization, computational steering and collaboration are all important ingredients. We do this in the next two sections.

4 An Environmental Crisis – Pollution Alert

Imagine this: a dangerous pollutant escapes from a chemical factory – where is it headed? We need to know the answer faster than real-time, so that people can be safely evacuated. This calls for immediate access to powerful Grid compute facilities in order to run a numerical simulation with a desktop interface for the scientist to adjust simulation parameters and to visualize the results. Additional expertise is needed, such as meteorological information concerning likely wind velocities – this is needed instantly, yet the meteorologist will be located elsewhere. Electronic collaboration is absolutely essential to allow the meteorologist to share their knowledge

of the problem, a computational steering approach allows them direct control of the relevant simulation parameters – to address questions such as “what happens if the wind changes to this direction?” – and to share visualization of the resulting effect. Once the scientists have reached an understanding, this needs to be communicated to the political decision maker in a clear and effective manner (the Challenger space shuttle disaster in 1986 is recognised now as a failure of scientists to present an effective visualization to launch decision makers [24]). The decision maker also needs to collaborate over the Internet, with the ability to propose further “what if” scenarios, but perhaps using a simplified interface to the underlying software. Indeed the actual way the data is visualized will vary according to the different people involved, and the information they need to understand.

The project team will typically have different skills and motivations: the numerical analyst will be more interested in the performance of the algorithms and, for example, the error in the computed solution; the chemist will be interested in the chemical concentrations and details of the physical processes occurring; the decision maker will be interested in regions where the concentrations exceed safe levels and the potential risk to populated areas. The collaborative model allows separate users to analyse a set of data in a completely independent fashion, as shown schematically in Fig. 3. Additionally the researchers may work at geographically remote locations. The collaborative model will facilitate the dissemination and discussion of the work by allowing problem parameters, computed data and rendered images to be processed and shared between the researchers. With that motivation in mind, we now describe the simulation involved in our application.

5 Simulating the Dispersal of the Pollutant

Computational models describing the evolution and reaction of chemical species in the atmosphere are an important tool in understanding the formation of hazardous pollutants such as greenhouse gases and acid rain. Such problems are typically modelled with a system of partial differential equations describing the transport of the chemical species through space and the chemical reactions between the species present. The modelled processes can occur on widely disparate time scales, hence the most effective numerical methods make use of adaptivity both in space and time [23]. The high spatial and temporal accuracy required, coupled with long simulation times, make this an intensive computation. The full model is a major computational challenge, for which Grid computing can provide the resources necessary for its solution. Our simple prototype allows us to explore the sort of problem-solving environment needed to allow interaction with code during the long simulation, to visualize the solution as it evolves and to interactively steer the computation when required. The simulation considered here is a prototype of the full scenario, in which a single component of pollution is emitted from a source and transported through the spatial domain. Figure 4 shows: on the left the scenario of the real-life problem being studied; in the centre, the model used as basis for the simulation, with the computational mesh; and on the right, the visualization of the solution at a particular time-step. The

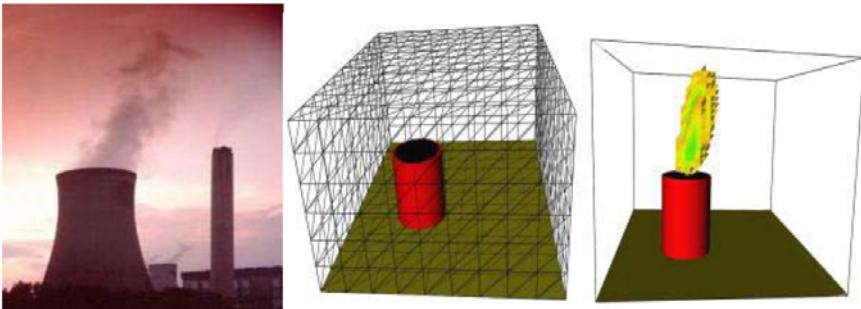


Fig. 4. The Reality and the Model

simple visualization shown here, such as may be required by the decision maker, is to select a threshold value for the concentration and render the cells that exceed that threshold.

The mathematical model is an advection equation for a scalar quantity $c(\mathbf{x}, t)$ (the concentration of the pollutant) in a spatial domain Ω with boundary Γ and time domain $[0, T]$

$$\frac{\partial c}{\partial t} + \mathbf{a} \cdot \nabla c = f(\mathbf{x})$$

where the spatial distribution of the scalar c is driven by an advection velocity $\mathbf{a}(t)$. Note that \mathbf{a} is spatially uniform but can vary in time allowing the convection direction to change. The source function $f(\mathbf{x})$ is used to simulate the production of the scalar c inside the domain, such as from the chimney of the chemical factory in our application. A cell-centred finite volume method is used to approximate the governing partial differential equation with an unstructured tetrahedral grid representing the spatial domain. The results are returned as the concentration of pollutant within each cell, at each time step of the simulation.

6 Computational Steering in a Grid Environment – Reference Model

Our aim here is to illustrate through the example of the environmental crisis, that the traditional dataflow visualization system is perfectly well suited to Grid Computing. We aim to provide an easy access to important aspects of Grid computing: authentication, via single log-on; resource discovery, via enquiry of Grid information services; and resource access, via remote launching of simulations from the desktop.

We assume the authentication step is performed before commencing the visualization processing, but we need additional modules that will support resource discovery and resource access. The “discover” module queries the Grid information service for available resources, allowing the user to select a specific resource to be used for the simulation. The “link” module receives the detail of the selected resource, and

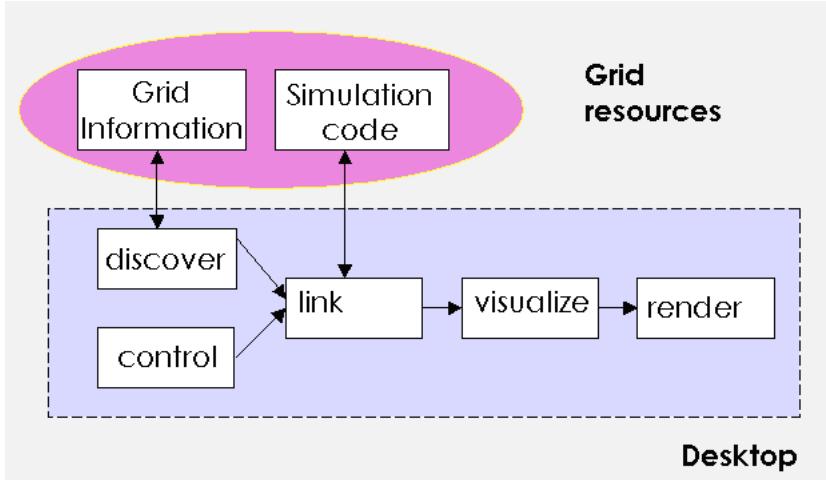


Fig. 5. Grid-based Computational Steering

uses that information to initiate a simulation on the Grid and receive results from it. Figure 5 shows a schematic view.

7 Computational Steering in a Grid Environment – Demonstrator

As a validation of the model, we have implemented a demonstrator that tackles the pollution problem of Sect. 4, using the simulation described in Sect. 5.

Our Grid computing environment was the White Rose Grid [27]. The White Rose Grid is a regional-scale Grid environment that incorporates four major high performance computing resources distributed between the universities of Leeds, Sheffield and York in the UK. It has been constructed to explore the potential for optimising resource usage between institutions and for supporting scientific collaboration between academics and also their industrial partners.

The Grid middleware used within the White Rose Grid is Globus Toolkit v2.4 [9]. Key components of the Globus Toolkit include: the Grid Security Infrastructure (GSI), the Globus Resource Allocation Manager (GRAM) and the Monitoring and Discovery Service (MDS). These have all been used in the demonstrator. It is important to realise that Grid computing is a young and evolving subject, and the concepts and technology have still to mature. We believe the abstract model of Sect. 6 will remain applicable as this evolution occurs, but the Globus technologies used in the demonstrator will be replaced by new developments, as increased focus is placed by the Grid community on Web services.

We have used IRIS Explorer, a long established visualization system, as the basic visualization environment for our demonstrator. Figure 7 shows the system in use.

The “GlobusSearch” module queries the MDS, the Globus information service. Each Grid resource maintains a catalogue of information that describes various static and dynamic features of that resource. For example, information recorded for a computational resource would include number of CPUs, size of memory and current workload. The Grid Resource Information Service (GRIS) is the Globus component responsible for maintaining this metadata for an individual resource. The Grid Information Index Server (GIIS) pools together the information from each GRIS and thus provides a convenient single point of reference for information about all resources within a single Grid. The “GlobusSearch” module therefore connects to the GIIS, making an LDAP query to discover the details of the Grid resources. The module displays, for each host, the machine name, its operating system, memory size, number of processors and current load. From this information the scientist can select the most appropriate host on which to run the simulation. The host name is output from the module, and wired to the GLSSpawn module which links the desktop to the simulation running on the Grid.

The GLSSpawn module initiates the specified executable on the selected host, using the `globus-job-run` command. In this example, the executable code is transferred from the desktop to the selected host prior to execution, using Globus. The simulation is compiled for IRIX, Solaris and Linux OS’s so any of these platforms may be targeted. Our experimental Grid offers a “fork” job manager allowing our simulation to start immediately upon submission. Once started, the simulation connects back to the module by means of sockets and then will lie dormant, polling the desktop for a “run” flag. On receipt of this, the simulation is triggered and executes on the Grid, returning results to the GLSSpawn module. The frequency of reporting data from remote host to desktop is controlled by settings on the module interface, as is the frequency with which steering parameters are requested. The simulation can be paused at any time to allow time to make steering decisions, or equally parameters may be changed while it executes which the simulation will pick up at the next update point. The interface panel contains generic parameters such as time stepping, pause/run controls and grid size. An application-specific module is used to steer the simulation, by direct manipulation of a 3D arrow widget that controls the wind velocity.

8 Collaborative Computational Steering – Reference Model and Demonstrator

The final step is to allow a research team to handle the simulation. Here we combine the Grid approach of the previous section with the collaborative visualization approach of Wood et al. [28] – giving the schematic model of Fig. 6.

This model has been realised in terms of IRIS Explorer and the COVISA collaborative toolkit. In Fig. 8 we see a collaborative computational steering session in progress. The facility of IRIS Explorer to deliver end-user applications with the underlying pipeline hidden, has been used to create interfaces for the researcher and collaborator. Each user shares control of the wind velocity, the output and update

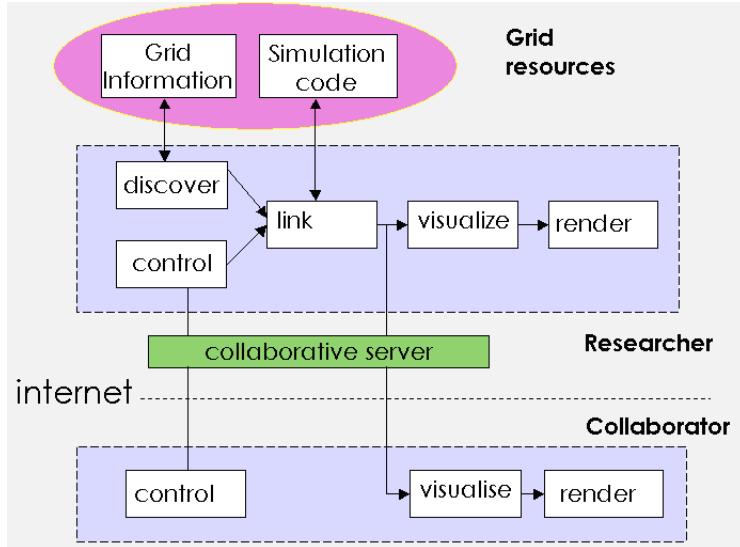


Fig. 6. Grid-based Collaborative Computational Steering

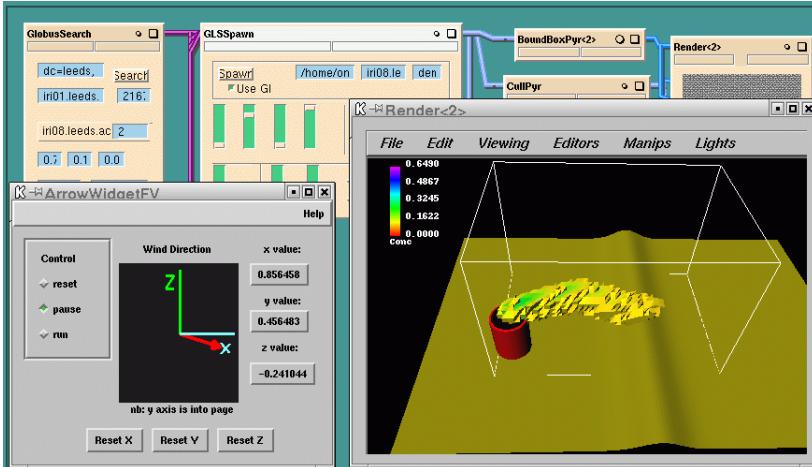


Fig. 7. Grid-based Computational Steering with IRIS Explorer

time steps, the grid size for the simulation and pause/run/reset control of the simulation. In this example, simulation data is being shared between both parties with joint control over the value of the concentration level parameter for visualization. Also the collaborator is using a different visualization technique (a slice plane) to visualize an estimate of the error associated with the solution.

Note that the COVISA toolkit is cross-platform, and so the researcher and collaborator can be using different hardware and operating system.

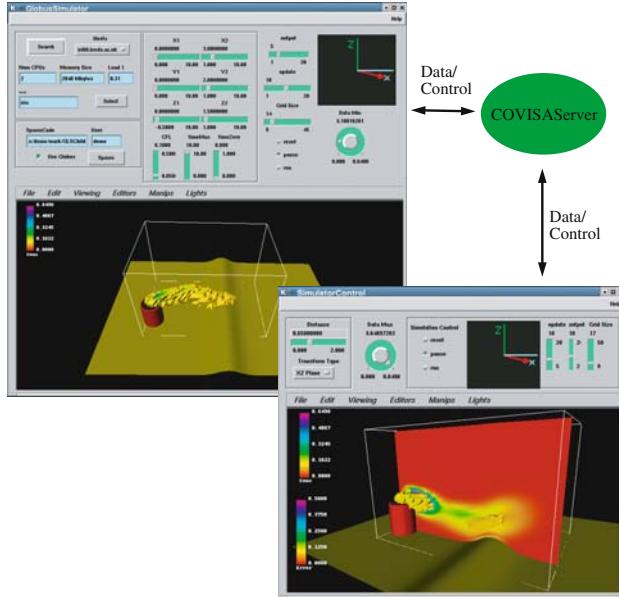


Fig. 8. Grid-based Collaborative Computational Steering Application with IRIS Explorer

We have run the application in a number of different scenarios

- in the simple environment of a LAN at the University of Leeds, with the simulation running on the White Rose Grid
- in a transatlantic collaboration with colleagues at CACR, Caltech in Pasadena
- in a trans-hemisphere collaboration, in which the simulation and one researcher were based at Leeds, UK in the northern hemisphere, and a collaborator was based in Christchurch, New Zealand in the southern hemisphere.

9 Conclusions and Future Work

This work has shown that the dataflow visualization environment can provide an important tool for Grid computing. Its original open and flexible design have allowed it to grow with computing technology over the last fifteen years, and it sits quite happily with the new Grid software infrastructure.

The work was implemented in IRIS Explorer, but it should be straightforward to take the same ideas and implement for AVS, or IBM Open Visualization Data Explorer.

In our experimental Grid, we take advantage of the fact that our simulations start immediately on submission. In reality a variety of job managers and queuing systems would be found in use on the Grid and work needs to be done to manage access to simulations that are scheduled for execution at a later time.

We continue to explore further ways of integrating dataflow visualization systems with Grid computing environments. In particular we are studying how pipelines can be distributed across several host machines, each running an instance of the visualization system. This was an integral part of the early design of these systems, but the original authentication mechanism has been deprecated as being insecure and so this feature is generally no longer available. We are investigating whether this can be revived in a secure manner using Globus middleware.

There are two difficulties with our current approach:

- the simulation has to regularly poll the visualization system on the desktop to check whether any parameters have changed – this is a major overhead and slows down the simulation
- the simulation only runs as long as the visualization system is kept running on the desktop – for a long running simulation one wants to be able to have the simulation running with a life of its own, and allow the visualization system to “check in” periodically to monitor progress

We are aiming to address these limitations in a further development cycle.

An important issue yet to be addressed in this work is data management for asynchronous collaboration. It is envisaged that users may wish to review a time-history of the computed solution together with the related steering options that were chosen at that time. The current user can then restart the simulation at a particular time and study the effects of varying some parameters of the model in detail from that point. Some of the data management issues are addressed by the Hyperscribe system already incorporated into IRIS Explorer.

The case study considered here was based around a prototype application but the developed environment is intended for generic scientific computing models. A companion demonstrator has been built, extending its use to the far more challenging simulation of elastohydrodynamic lubrication [10].

Grid-enabled visualization is an extremely important research area. Our aim in this paper has been to question the need to start afresh and develop new visualization systems for Grid computing *ab initio* – existing dataflow systems have sufficient flexibility and extensibility for them to evolve into this exciting new world.

References

1. AVS/Express website, 2004. <http://www.avs.com>.
2. E. W. Bethel and J. Shalf. Grid-distributed visualizations using connectionless protocols. *IEEE Computer Graphics and Applications*, 23(2):51–59, March/April 2003.
3. cAVS website, 2004. <http://cavs.sdsc.edu>.
4. J. Chin, J. Harting, S. Jha, P.V. Coveney, A.R. Porter, and S. Pickles. Steering in computational science: mesoscale modelling and simulation. *Contemporary Physics*, 44:417–434, 2003.
5. D.A. Duce, J.R. Gallop, I.J. Johnson, K. Robinson, C.D. Seelig, and C.S. Cooper. Distributed Cooperative Visualization - Experiences and Issues from MANICORAL Project. In *EG Workshop on Visualization in Scientific Computing*. Eurographics Association, 1998.

6. E. Engquist. Steering and visualization of electromagnetic simulations using Globus. In *Simulation and Visualization on the Grid*, pages 82–97. Springer Series LNCSE, 2000.
7. I. Foster, J. Insley, G. von Lasewski, C. Kesselman, and M. Thiebaux. Distance visualization: Data exploration on the Grid. *IEEE Computer*, December:36 – 43, 1999.
8. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organisations. *International Journal of High Performance Computing Applications*, 15(3):200 – 222, 2001.
9. Globus website, 2004. <http://www.globus.org>.
10. C. Goodyer, J. Wood, and M. Berzins. A parallel grid based PSE for EHL problems. In J. Fagerholm, J. Haataja, J. Jarvinen, M. Llyl, M. Raback, and V. Savolainen, editors, *Applied Parallel Computing Advanced Scientific Computing 6th International Conference, PARA 2002*, pp. 521–530. Springer-Verlag, 2002.
11. R. B. Haber and D. A. McNabb. Visualization Idioms: A Conceptual Model for Scientific Visualization Systems. In B. Shriver, G. M. Neilson, and L. J. Rosenblum, editors, *Visualization In Scientific Computing*, pp. 74–93. IEEE Computer Society Press, 1990.
12. T.J. Jankun-Kelly, O. Kreylos, K. Ma, B. Hamann, K.I. Joy, J. Shalf, and E.W. Bethel. Deploying web-based visual exploration tools on the grid. *IEEE Computer Graphics and Applications*, 23(2):40–50, March/April 2003.
13. C.R. Johnson, S. Parker, D. Weinstein, and S. Heffernan. Component-based problem solving environments for large-scale scientific computing. *Journal on Concurrency and Computation: Practice and Experience*, (14):1337–1349, 2002.
14. R. Marshall, J. Kempf, S. Dyer, and C. Yen. Visualization methods and computational steering for a 3d turbulence model for Lake Erie. *ACM SIGGRAPH Computer Graphics*, 24(2), 1990.
15. B. H. McCormick, T. A. DeFanti, and M. D. Brown. Visualization in scientific computing. *Computer Graphics*, 21(6), 1987.
16. National Grid Service website, 2004. <http://www.ngs.ac.uk>.
17. A. Norton and A. Rockwood. Enabling view-dependent progressive volume visualization on the grid. *IEEE Computer Graphics and Applications*, 23(2):22–31, March/April 2003.
18. 2004. <http://www.opendx.org>.
19. S. G. Parker and C. R. Johnson. SCIRun: A scientific programming environment for computational steering. In H. W. Meuer, editor, *Proceedings of Supercomputer '95*, New York, 1995. Springer-Verlag.
20. W. J. Schroeder, K. M. Martin, and W. E. Lorensen. *The Visualization Toolkit: An Object Oriented Approach to 3D Graphics*. Kitware, Inc., 3rd edition, 2003.
21. J. Shalf and E. W. Bethel. The Grid and future visualization system architectures. *IEEE Computer Graphics and Applications*, 23(2):6–9, March/April 2003.
22. TeraGrid website, 2004. <http://www.teragrid.org>.
23. A Tomlin, M. Berzins, J. Ware, and M.J. Pilling. On the use of adaptive gridding methods for modelling chemical transport from multi-scale sources. *Atmospheric Environment*, 31(18):2945–2959, 1997.
24. Edward R. Tufte. *Visual Explanations: Images and Quantities, Evidence and Narrative*. Graphics Press, 1997.
25. VTK website, 2004. <http://public.kitware.com/VTK/>.
26. J. P. R. B. Walton. NAG's IRIS Explorer. In C. R. Johnson and C. D. Hansen, editors, *Visualization Handbook*. Academic Press, 2003, in press. Available from <http://www.nag.co.uk/doc/TechRep/Pdf/tr2.03.pdf>.
27. White Rose Grid website, 2004. <http://www.wrg.org.uk>.
28. J. D. Wood, H. Wright, and K. W. Brodlie. Collaborative visualization. In R. Yagel and H. Hagen, editors, *Proceedings of IEEE Visualization '97*, pp. 253–259, 1997.

Earthquake Visualization Using Large-scale Ground Motion and Structural Response Simulations

Joerg Meyer and Thomas Wischgoll

University of California, Irvine, Department of Electrical Engineering and Computer Science, 644E Engineering Tower, Irvine, CA 92697-2625
`{jmeyer|twischgo}@uci.edu`

Summary. Earthquakes are not predictable with current technology. However, it is possible to simulate different scenarios by making certain assumptions, such as the location of the epicenter, the type and magnitude of the eruption, and the location of a fault line with respect to buildings of a particular type. The effects of various earthquakes can be studied, and the aftermath of the simulation can be used by first responders and emergency management agencies to better prepare and plan for future disasters.

This article describes methods for visualizing large-scale, finite element simulations of ground motion based on time-varying tetrahedral meshes, and explains how such a simulated earthquake scenario can be visually combined with a simulation of the structural response. The building response is based on a simulation of single-degree-of-freedom (SDOF) structural models.

The amount of data generated in these simulations is quite substantial (greater than 100 gigabytes per scenario). Real-time, interactive visualization and navigation in a 3-D virtual environment is still challenging. For the building simulation, a number of structural prototypes that represent a typical urban building infrastructure is selected. For the ground motion simulation, an efficient, topology-sensitive tetrahedral mesh decimation algorithm suitable for time-varying grids and based on a feature-preserving quadric error metric is used. The algorithms presented in this chapter have the potential for being applied to other scientific domains where time-varying, tetrahedral meshes are used.

Key words: finite element simulation, tetrahedral mesh simplification, level-of-detail, mesh reduction, hybrid rendering, earthquake simulation, ground motion, structural response, scientific visualization

1 Introduction

According to the National Earthquake Information Center (NEIC), millions of earthquakes occur every year. The majority of events are insignificant microquakes (less than 2.2 on the logarithmic Richter scale), but on average approximately 7,000 earthquakes occur annually with a potential for a significant impact on the infrastructure in an urban setting (Richter magnitude greater than 4.5).

In the United States and in many other countries, the Richter scale is used to measure energy released during an earthquake. On the Richter scale, intensity increases in geometric ratio (Table 1). An increase of one number means the energy released is ten times greater. Thus an earthquake of 4.0 on the Richter scale is ten times stronger than an earthquake of 3.0 on the Richter scale [20]. Significant earthquakes (Richter magnitude greater than 4.5) are characterized by the potential to destroy or to cause considerable damage to buildings, bridges, dams, power and gas lines, etc.

Table 1. Richter scale and potential damage/loss

<i>Richter Scale</i>	<i>Type of damage in a populated area</i>
< 2.2	Microquake
2.2	Most people aware that an earthquake has occurred
3.5	Slight damage
4.0	Moderate damage
5.0	Considerable damage
6.0	Severe damage
7.0	Major earthquake, capable of widespread heavy damage
8.0	Great earthquake, capable of total damage/loss

According to the Southern California Earthquake Center (SCEC), a large number of significant earthquakes occurred over the past few decades (Fig. 1), many of them close to known fault lines (Fig. 2).

The data structure for the ground motion simulation is a large-scale, deformable tetrahedral grid. This time-varying grid represents a layered soil model, as it is typical for sedimentary basins, such as the one that we have chosen for our simulation (Fig. 3). The basin is modelled as a three-dimensional isotropic, heterogeneous anelastic medium. The domain is limited by absorbing boundaries that restrict the

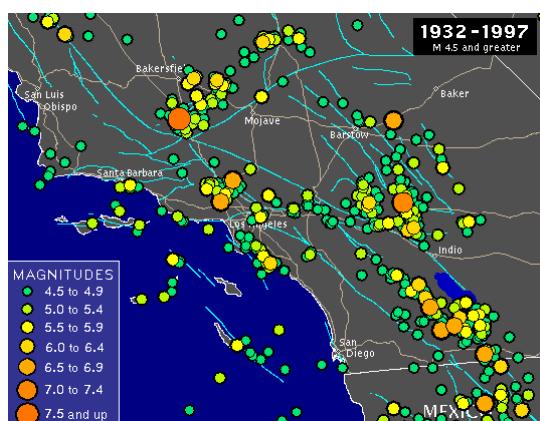


Fig. 1. Earthquake magnitudes greater than 4.5 (1932–1997)

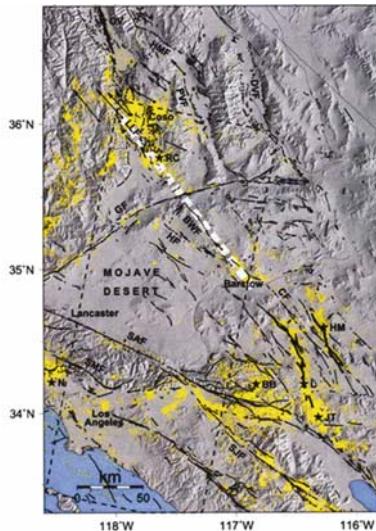


Fig. 2. Southern California earthquake fault lines and locations

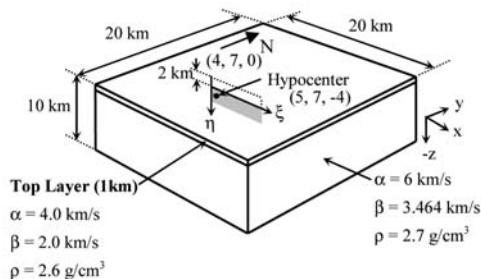


Fig. 3. Layered half-space with extended source fault

amount and magnitude of spurious reflections. Different configurations can be tested by varying the size and the layer composition of the volume represented by the grid.

A finite element simulation is performed over the idealized model shown in Fig. 3. Idealized models are essential to understand physical phenomena such as seismic wave propagation, and to verify calculation methodologies. The model incorporates an idealized extended strike-slip fault aligned with the coordinate system. The shaded area in Fig. 3 represents such a fault.

The tetrahedral mesh generated in this simulation consists of approximately 12 million nodes. For every node, a maximum velocity vector for 800 time steps is computed, representing a time history of the node for a time period of eight seconds in 10 milliseconds increments. The total amount of data generated by this simulation is approximately 130 GB. This is a typical size for a ground motion simulation of the given size and complexity.

This chapter describes a complete, self-contained method for efficient, topology-sensitive time-varying tetrahedral mesh decimation based on a feature- and boundary-preserving quadric error metric. The goal is to visualize such a large-scale dataset interactively, and to enable the user to navigate the space defined by the grid both from an external point of view as well as from a view point within the volume (immersive visualization). This type of visualization enables an interactive analysis of the data, which is essential for an in-depth understanding of the complex processes of ground motion and structural response.

2 Related Work

In earthquake-related ground motion simulations, volumes are typically represented as large-scale, time-varying tetrahedral meshes. The resulting datasets are usually large, and datasets that exceed the size of the processor's main memory are difficult to store, and in the case of remote visualization require significant amounts of time for data transfer over the Internet. The amount of geometry data generated from such simulations is usually too complex for rendering in an interactive display environment. The complexity of those meshes mainly results from the large number of nodes considered in the simulation, from the significant number of time steps necessary to capture various earthquake frequencies, and from the actual vector data (usually displacement or velocity data) associated with the nodes [1, 4, 19].

For interactive visualization, it is critical to reduce the geometric complexity of such large, time-varying meshes with minimal loss of detail in the spatial and in the temporal domain. The algorithm should be scalable, so that an arbitrary number of time steps can be processed and visualized.

In the past, a significant number of algorithms have been developed to address aspects of this complex problem. Tetrahedral meshes have been established as a standard to represent large finite element meshes. Most other data structures, such as voxel grids or scattered data fields can be either broken down or tessellated into tetrahedral meshes. Tetrahedral meshes are preferred for discrete volume representations because of the simplicity of the primitives (tetrahedra).

We distinguish between surface and volume mesh simplification techniques. Most volume decimation algorithms evolved from surface or polygon mesh simplification algorithms [10, 13, 14, 16, 24–27, 32], and most algorithms provide solutions for topological inconsistencies that may occur during or after the mesh simplification. Hierarchical representations are important for interactive rendering of large tetrahedral meshes. Zhou et al. [33], Gerstner et al. [11], and Ohlberger et al. [21] present frameworks for hierarchical representations of tetrahedral meshes. However, none of the described methods are applicable to time-varying data without modifications.

After an initial overview of surface decimation algorithms (section 2.1) and volume decimation algorithms (Sect. 2.2), we discuss a new method called *TetFusion* [2] and some of its properties and limitations (Sect. 2.3). Specific results for this method are presented in Sect. 2.4. Section 3 explains the error metric, and Sect. 4 addresses

time-varying data sets. Section 5 finally gives some results for the new method and explains how it can be used for the given application domain (ground motion and structural response simulation).

2.1 Surface Mesh Simplification

Some ideas that were developed for surface mesh simplification can also be applied to volume meshes. Therefore, we provide a short overview of existing surface mesh simplification methods.

Surface or polygon mesh simplification algorithms are either based on geometry reduction or iterative refinement. Since we are mostly interested in a direct reduction rather than the creation of a new mesh using refinement which usually requires global or local access to the reference mesh making the algorithm less scalable, we focus primarily on the first technique, i.e., geometry reduction. Most methods represent the mesh with a smaller number of nodes that are taken from the original mesh or by calculating new nodes that represent several nodes in the original mesh.

Schroeder et al. [26, 27] propose a method for decimation of triangle meshes. Turk [32] uses a set of new points to re-tile a polygonal mesh. Hoppe [14] takes a different approach and introduces progressive meshes that can be used for streaming data (level-of-detail representation). This method was considered a major milestone in polygon mesh simplification. For further study, Garland [9] provides a comprehensive overview of other mesh simplification techniques.

Garland and Heckbert [10] introduce a quadric metric as an error metric for polygonal mesh simplification. A detailed discussion of the application of this metric to volume meshes is given in Sect. 3.

2.2 Volume Mesh Simplification

Trotts et al. [30] were amongst the first to implement a tetrahedral volume mesh decimation algorithm. In their work, they extend a polygonal geometry reduction technique to tetrahedral meshes. They define a tetrahedral collapse operation as a sequence of three *edge collapses*, while keeping the overall geometric error within a certain tolerance range. The error metric is based on a spline segment representation of the tetrahedra. Since an *edge collapse* is used as the atomic decimation operation, several topological problems, such as volume flipping, are addressed and discussed. The algorithm suffers from overwhelming overhead for storing and updating the connectivity information (edge list).

Staadt and Gross [28] also discuss an extension of the *edge collapse* algorithm for polygonal meshes to tetrahedral meshes. They interpret tetrahedra as a specialized class of simplicial complexes [24] and extend Hoppe's work on progressive meshes [14] to tetrahedral meshes. The article offers solutions to the previously mentioned problem of volume flipping (negative volume), and other topological changes that might occur, such as self-intersection, and tetrahedra intersecting the boundary regions. Kraus et al. [17] use a similar approach and also address the specific case of non-convex tetrahedral meshes.

Trotts et al. [31], in an extension to their earlier work [30], incorporate an error metric that not only incorporates modifications to the geometry, but also to the scalar attributes associated with the vertices. These attributes are usually interpolated between the two nodes that constitute the collapsing edge.

Cignoni et al. [5] use the same idea and apply it to tetrahedral meshes by presenting a framework for integrated error evaluation for both domain and field approximation during simplification. The article elaborately explores local accumulation, gradient difference, and brute force strategies to evaluate and predict domain errors while incrementally simplifying a mesh. The algorithm also uses a quadric error metric, which is compared to other metrics in this article [5].

Topology preservation is the main topic of the work published by Edelsbrunner [8]. He provides an extensive algorithmic background for ensuring topological correctness during *edge-collapse*-based mesh simplification. Dey et al. [7] provide detailed criteria for topological correctness, which can be generalized to polyhedral meshes.

In the next section, we present a computationally efficient tetrahedral volume mesh simplification method that combines metrics for accurate field (attribute) data representation with techniques for restraining volumetric topology.

2.3 A Combined Mesh Decimation Technique: *TetFusion*

Recently, an efficient volume mesh decimation algorithm, *TetFusion*, was published [2]. It addresses both geometry and attribute data preservation. We summarize the properties and limitations of this algorithm, which lead to the development of an improved method, *QTetFusion*, which uses a different error metric and addresses problems such as volume flipping, boundary intersection, and other critical changes in the topology (Sect. 3).

The *TetFusion* algorithm employs a tetrahedral collapse as an atomic operation (*TetFuse*) for mesh decimation. The idea is simple and intuitive: take all four vertices of a tetrahedron, and fuse them onto the barycenter (the geometric center) of the tetrahedron (Fig. 4).

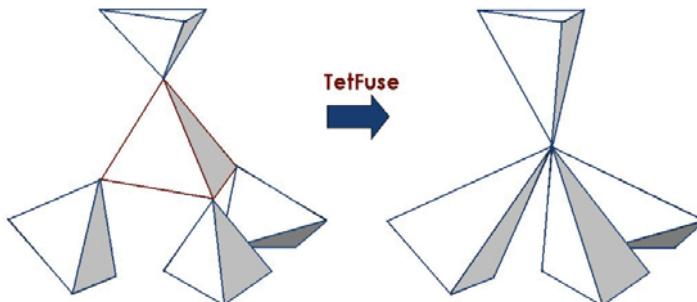


Fig. 4. An illustration of an instance of the *TetFuse* operation

The center tetrahedron is the target object that is supposed to be collapsed onto its barycenter. The four other tetrahedra are affected by this change and stretch in the direction of the target tetrahedron's barycenter. Note that for any affected tetrahedron, the vertex it shares with the target tetrahedron moves *away* from the base plane formed by its other three vertices. In a fully connected mesh, at least eleven tetrahedra collapse as a result of *TetFuse* applied to an interior tetrahedron. This includes the *target tetrahedron*, the four tetrahedra sharing one of the four faces with the *target tetrahedron*, and at least six more tetrahedra that share one of the six edges with the *target tetrahedron*. This means that each instance of an application of *TetFuse* causes an efficient decimation of the mesh.

The *TetFusion* algorithm is based on multiple, error-controlled executions of a primitive operation called *TetFuse*. The following paragraphs summarize the inherent properties and limitations of *TetFusion*.

Symmetry: The volume of the collapsed tetrahedron is distributed symmetrically with respect to the barycenter between the affected tetrahedra in the local neighborhood.

Efficient decimation: Each instance of *TetFuse* causes at least eleven tetrahedra to collapse for a *non-boundary target tetrahedron*, as explained in one of the previous paragraphs. This results in a much higher mesh decimation rate per atomic operation than in the case of an *edge-collapse-based* algorithm.

Avoiding Flipping: Because of the symmetry of the decimation operation, the vertex that an affected tetrahedron shares with the *target tetrahedron (shared vertex)* tends to move away from its base plane (the plane formed by the other three vertices of the affected tetrahedron, see Fig. 4). Hence, most of the time the ordering of vertices in an affected tetrahedron does not get changed from the original configuration, and the volume is represented correctly. However, if the barycenter of the *target tetrahedron* is located on the other side of the base plane of an affected tetrahedron, flipping is possible (Fig. 5). Such special cases can be avoided simply by checking

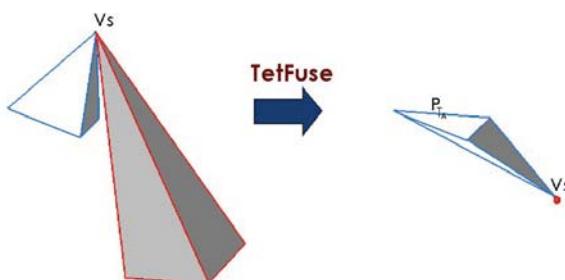


Fig. 5. Flipping may occur if the barycenter is on the other side of the base plane

if the point has moved to the other side of the base plane of the affected tetrahedron, which would result in a rejection of the current instance of *TetFuse*.

Prohibiting Self-Intersections of the Boundary: *TetFusion* does not allow any boundary tetrahedra to be affected. It has been verified that self-intersections of boundaries occur only at sharp edges and corners [28], when an affected tetrahedron pierces through one or more of the boundary faces of a boundary tetrahedron. However, this is a serious limitation of the algorithm that drastically affects the decimation ratio. An improved solution is discussed in Sect. 3.

Prohibiting Boundary Intersections at Concave Boundary Regions: Cases of boundary intersection occur when an interior tetrahedron stretches through and over a concave boundary region. Such cases cannot be avoided completely by employing the given error metric. *TetFusion* addresses this problem by limiting the expansion of an affected tetrahedron, and by not allowing tetrahedra in the vicinity of the boundary surface to stretch as a result of the collapse of a *target tetrahedron*. This is also a limitation, which is addressed in Sect. 3.

Locking the Aspect Ratio: Tetrahedra that exceed a pre-specified threshold of the edge aspect ratio (long, skinny tetrahedra) are usually difficult to render and therefore trigger an early rejection of the execution of *TetFuse*.

2.4 Example and Results

Figure 6 shows an example of a decimated mesh. The 1,499,160 element blunt-fin dataset was decimated using *TetFusion* in 187.2 seconds, and rendered on an SGI R10000 194MHz with 2048 MB RAM, running Irix 6.52. The boundary is perfectly preserved, while some of the data attribute values appear to be slightly blurred due to repeated interpolation in the reduction step.

In summary, the original *TetFusion* algorithm [2] was limited to interior tetrahedra, thus leaving the surface intact, but at the same time unfortunately also limiting the decimation ratio. The next section discusses an extension of this algorithm,

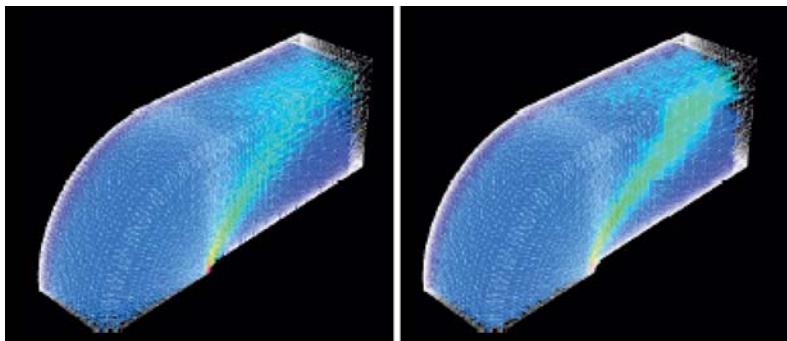


Fig. 6. Original (100%) and decimated mesh (36.21%) of the blunt-fin dataset

called *QTetFusion* (*Quadratics-guided Tetrahedron Fusion*) [3]. It is a volume mesh simplification algorithm that employs a *planar quadric error metric* to guarantee minimum geometric and attribute error upon each instance of *TetFuse*. We show how this atomic tetrahedral collapse operation, along with an efficient geometric error metric, can take care of complex mesh inconsistency problems with minimal additional computational overhead. The algorithm ensures that a mesh stays within (and infinitesimally close to) its *boundary envelope* at all levels of resolution during simplification.

Almost all of the existing work addressing volume mesh simplification evolved from *edge-collapse*-based decimation strategies that were originally proposed for polygonal meshes. However, surface mesh simplification algorithms cannot simply be scaled up to handle higher order simplicial complexes because of additional geometric and topological constraints. Cases like degenerate simplices, violation of Delaunay tessellation, loss of topological genus (undesired closing of holes or creation of new ones), violations of the convex hull and boundary preservation properties, etc., must be specially taken care of during volume mesh simplification. Such cases have already been identified and can be avoided with special computational methods [7, 8]. However, algorithms that present computationally less expensive simplification schemes either do not handle all of these cases, or are spatially selective during decimation and hence are limited in the achieved decimation ratio [2].

3 Tetrahedral Fusion with Quadric Error Metrics

To overcome the shortcomings listed in the previous section, we use *QTetFuse* as a reversible atomic decimation operation for tetrahedral meshes [3]. The algorithm operates on a list of nodes with associated attribute data, and a separate array that contains the connectivity information for every tetrahedron. Both the point list and the list of tetrahedra are updated in the decimation process. This is necessary because both tetrahedra are collapsed and new points are generated by fusing some of the tetrahedra to a single point. For a better understanding of the algorithm, we summarize the most important definitions (Sect. 3.1) that are needed for describing the algorithm (Sect. 3.2), analyze the properties (Sect. 3.3) of the algorithm, and provide some details on the derivation of the employed *planar quadric error metric* (*PQEM*) from previous polygonal methods (Sect. 3.4) [10]. Finally, we explain how the *fusion point* is calculated (Sect. 3.5), and provide an example of the results of the algorithm (Sect. 3.6).

3.1 Definitions

In this section, we summarize notations and definitions that are necessary for understanding the algorithm.

- a) *Target Tetrahedron*: A tetrahedron that is selected for decimation.
- b) *Boundary Tetrahedron*: A tetrahedron with one or more of its vertices lying on

the boundary surface. All other tetrahedra that are not on the boundary are called *interior tetrahedra*.

- c) *Boundary Face*: Triangle face of a *boundary tetrahedron* where all three vertices lie on the boundary surface.
- d) *Fusion Point*: Point of collapse of the four vertices of a *target tetrahedron*. One *target tetrahedron* may have more than one valid fusion point depending on the specified planar quadric error tolerance value.
- e) *Affected Tetrahedron*: A tetrahedron that shares exactly one vertex with a *target tetrahedron*. This shared vertex (*target vertex*) stretches the *affected tetrahedron* towards and onto the *fusion point* of the *target tetrahedron* as a result of *QTetFuse*.
- f) *Target Vertex*: The vertex of an *affected tetrahedron* that it shares with a *target tetrahedron*.
- g) *Base Triangle*: A triangle formed by the vertices of an *affected tetrahedron*, excluding the target vertex.
- h) *Deleted Tetrahedron*: A tetrahedron that shares two or more vertices with a *target tetrahedron*, which collapses as a result of the collapse of the *target tetrahedron*.

3.2 Algorithm

The basic idea is to fuse the four vertices of a tetrahedron into a point (the *fusion point*, see definitions in Sect. 3.1). The *fusion point* (Fig. 7) is computed so that minimum geometric error is introduced during decimation. We employ a *planar quadric error metric (PQEM)* to measure and restrict this error (see Sect. 3.4).

The algorithm is driven by an efficient space redistribution strategy, handles cases of mesh-inconsistency, while preserving the *boundary envelope* of the mesh, and employs a *PQEM* to guarantee minimum error in the geometric domain when collapsing the tetrahedral elements. It maintains the input mesh's topological genus in the geometry domain at low computational costs.

In Fig. 7, the upper-left *target tetrahedron* collapses onto its *fusion point*. The upper-right tetrahedron degenerates to an edge, which is consequently removed (*deleted tetrahedron*). The lower *affected tetrahedron* stretches in the direction of the corresponding *fusion point*. Note that for the *affected tetrahedron*, the vertex it shares with the *target tetrahedron* tends to move away from the base plane formed by its other three vertices. If the shown *target tetrahedron* is an *interior* one, at least eleven tetrahedra collapse as a result of this operation (see Sect. 2.3).

The following section describes the main algorithm. *QTetFusion* is a locally greedy algorithm. It features a pre-processing phase that evaluates *PQEMs* for all the tetrahedra in the input mesh, and stores them in a heap data structure. We employ a *Fibonacci* heap (because of its better amortized time complexity compared to

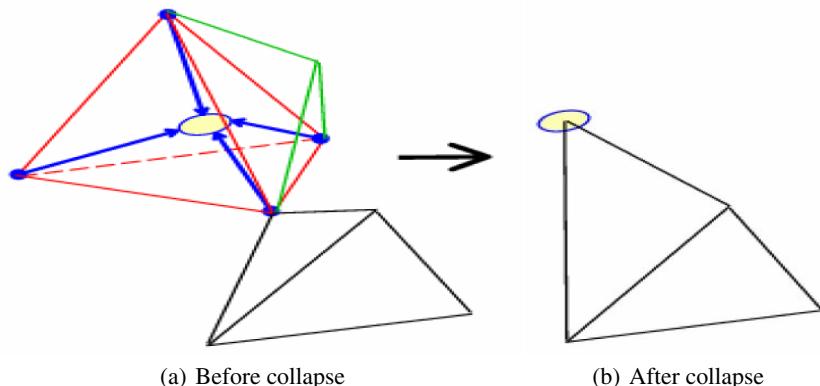


Fig. 7. An illustration of one instance of the *QTetFuse* operation

a simple binomial heap) to maintain the priority queue of tetrahedral elements addressed by their *PQEM* keys [3]. In fact, both a binomial and a Fibonacci heap have a worst-case time complexity of $O(m + n \log n)$, where m is the number of edges, and n is the number of nodes. However, in a Fibonacci heap the insert operation is more efficient: $O(1)$ vs. $O(\log n)$, while the delete operation remains the same: $O(\log n)$. The main algorithm is outlined below:

```

while heap is not empty
  extract  $T$  with minimum  $\Delta(T)$  from the heap
  if none of adjacentTetrahedra( $T$ ) would flip as a result of  $T$ 's collapse,
     $QTetFuse(T)$ 
  update heap

```

The procedure **adjacentTetrahedra**(T) returns a list of all the tetrahedra adjacent to T . The geometric error $\Delta(T)$ is explained in Sect. 3.4.

3.3 Properties

This section discusses the inherent properties of *QTetFusion* as a volume mesh decimation algorithm for tetrahedral meshes.

Efficient decimation: similar to *TetFusion* (Sect. 2.3).

Avoiding flipping: similar to *TetFusion* (Sect. 2.3).

Simplified mesh restricted to the inside of (and infinitesimally close to) the boundary envelope of the source mesh: Self-intersections of boundary elements might occur when an *affected tetrahedron* pierces through one or more of the boundary faces of a *boundary tetrahedron*. We prevent such cases by restricting the simplified mesh to

remain inside its *boundary envelope*.

Avoiding changes of the topological genus of a mesh: The *boundary envelope* of a polyhedral mesh defines its topological genus. Consequently, if the topological genus of the envelope is preserved, topology preservation for the enclosed volume is guaranteed. As a result, the algorithm guarantees that the simplified mesh remains confined to its *boundary envelope*. Therefore, the algorithm cannot change the topology of the mesh, i.e., it is prevented from closing any existing holes or from creating new ones. The latter is an inherent problem of all *edge-collapse*-based decimation algorithms and usually requires complex consistency checking. The proposed method requires only local testing of the *affected* and *deleted tetrahedra* and is therefore relatively efficient.

3.4 Planar Quadric Error Metric (PQEM)

This section describes the error metric we employ to control the domain errors during simplification. Garland and Heckbert [10] developed a computationally efficient and intuitive algorithm employing a *Quadric Error Metric (QEM)* for efficient progressive simplification of polygonal meshes. The algorithm produces high quality approximations and can even handle 2-manifold surface meshes.

To obtain an error minimizing sequence of *QTetFuse* operations, we first need to associate a *cost* of collapse with each tetrahedron in the mesh. As described in [10], we first associate a quadric error measure (a 4×4 symmetric matrix Q) with every vertex v of a tetrahedron that indicates the error that would be introduced if the tetrahedron were to collapse. For each vertex v of a tetrahedron, the measure of its squared distance with respect to all incident triangle faces (planes) is given by:

$$\Delta(v) = \Delta([v_x \ v_y \ v_z \ 1]^T) = \sum_{p=faces(v)} (a_p p^T v)^2 \quad (1)$$

where $p = [p_x \ p_y \ p_z \ d]^T$ represents the equation of a plane incident on v such that the weight a_p represents the area of the triangle defining p . Further, if n represents the normal vector of p , then d is given by

$$d = -nv^T \quad (2)$$

Equation (1) can be rewritten as a quadric:

$$\begin{aligned} \Delta(v) &= \sum_{p=faces(v)} v^T (a_p^2 p p^T) v \\ &= v^T \left(\sum_{p=faces(v)} (a_p^2 p p^T) \right) v \\ &= v^T \left(\sum_{p=faces(v)} (Q_p) \right) v \end{aligned} \quad (3)$$

where Q_p is the area-weighted error quadric for v corresponding to the incident plane p .

Once we have error quadrics $Q_p(i)$ for all the four vertices of the tetrahedron T in consideration, we simply add them to obtain a single $PQEM$ as follows:

$$PQEM(T) = \sum_{i=1}^4 Q_p(i) \quad (4)$$

If T were to collapse to a point v_c , the total geometric error (for T) as approximated by this metric would be:

$$\Delta(T) = v_c^T PQEM(T) v_c \quad (5)$$

3.5 Computing the Fusion Point

Consider a tetrahedron $T = \{v_1, v_2, v_3, v_4\}$. We compute a point of collapse (*fusion point* v) for T that minimizes the total associated $PQEM$ as defined in (5). According to [10], this can be done by computing the partial derivatives of $\Delta(T)$, and solving them for 0. The result is of the form

$$\bar{v} = Q_1^{-1} [0 \ 0 \ 0 \ 1]^T \quad (6)$$

where

$$Q_1 = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

Note that the terms q_{ij} are coefficients of the respective $PQEM$. There might be cases when the quadric matrix used in the equation is not invertible. In that case, we settle on the barycenter of T as the *fusion point*.

Note that the central ellipsoid in Fig. 8b) represents a level-set surface for the *Planar Quadric Error* for the *target tetrahedron* shown. This quadric error is the sum of quadric errors of the four constituent vertices of the tetrahedron.

3.6 Example

Figure 9 shows an example of a decimated mesh. The 12,936 element *super phoenix* dataset was decimated using *QTetFusion* in 31.174 seconds, and rendered on an SGI R12000 400MHz with 2048 MB RAM, running Irix 6.52. The increase of computing time over the standard *TetFusion* algorithm is significant (for instance, a factor of 61 for the *super phoenix* dataset, and 46 for the *combustion chamber*; see Table 2 and compare with [2]). However, the decimation rate in most cases was either slightly increased or at least preserved (+12.4% for the *super phoenix* dataset, and -2% for the *combustion chamber*). This is a good result, as the new *QTetFusion* algorithm has significant topological advantages over the standard *TetFusion* method. Diagram 10 and Tables 2 and 3 provide some additional statistics for other datasets.

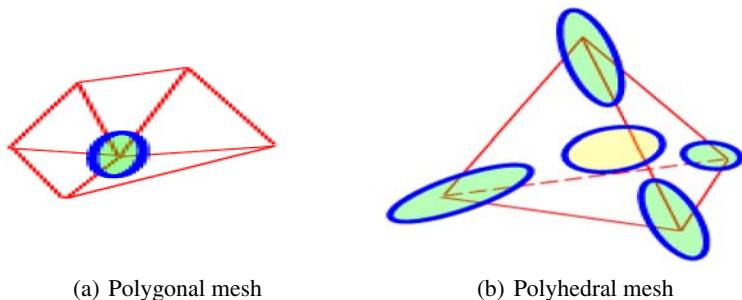


Fig. 8. Error ellipsoids for affected vertices when the primitive to be decimated is (a) an edge and (b) a tetrahedron

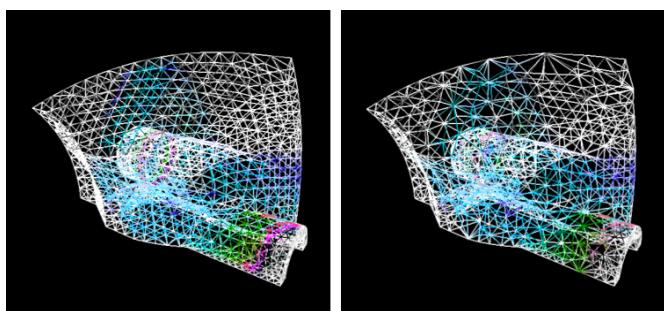


Fig. 9. (a) Original (100%) and (b) decimated mesh (46.58%) of the *super phoenix* dataset

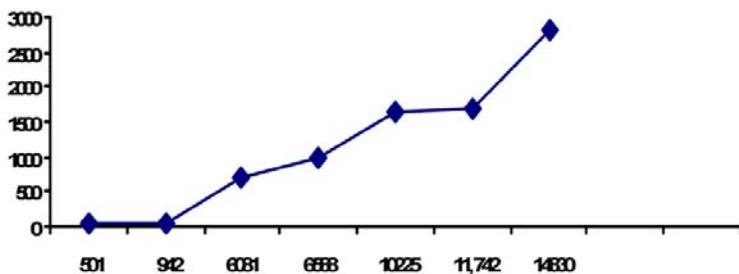


Fig. 10. CPU time in seconds (vertical axis) vs. number of *QTetFuse* operations (tetrahedral collapses) performed (horizontal axis)

Table 2. Number of tetrahedra, decimation ratio and CPU time for various datasets (*QTetFusion*)

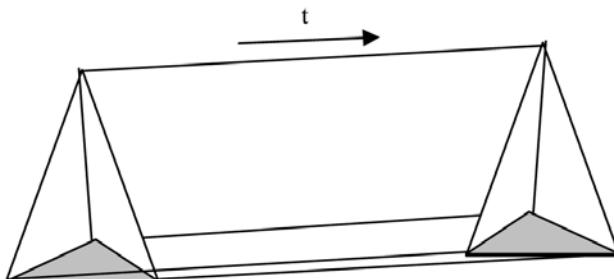
#	mesh	n	dec. rat.	<i>QTetFusion</i> (s)
1.	<i>super phoenix</i>	12,936	53.6%	31.174
2.	<i>blunt fin</i>	187,395	49.3%	715.074
3.	<i>comb chamber</i>	215,040	47.0%	976.603
4.	<i>oxygen post</i>	513,375	46.0%	2,803.575

Table 3. Number of tetrahedra, number of decimated tetrahedra, number of *QTetFuse* operations required, average number of decimated tetrahedra for a single *QTetFuse* operation

#	mesh	n	n_{decim}	# <i>QTetFuse</i>	Avg. n_{decim}
1.	<i>super phoenix</i>	12,936	6,940	501	13.852
2.	<i>blunt fin</i>	187,395	92,375	6,081	15.684
3.	<i>comb chamber</i>	215,040	101,502	6,588	15.407
4.	<i>oxygen post</i>	513,375	238,527	14,830	16.084

4 Time-varying Tetrahedral Mesh Decimation

Extremely high decimation rates can be obtained by taking the temporal domain into account. Instead of tetrahedra, we consider a mesh of hypertetrahedra that consists of tetrahedra that are connected across the temporal domain (Fig. 11). We define a hypertetrahedron as a set of at least two (possibly more) tetrahedra where all four vertices are connected in the time domain. Intuitively, a hypertetrahedron represents a tetrahedron that changes shape over time. Every time step represents a snapshot of the current shape. Without loss of generality, we can assume that the time domain is a linear extension of a three-dimensional cartesian coordinate system. As a consequence, we connect corresponding vertices with linear, i.e. straight, line segments that indicate the motion of a vertex between two or more discrete time steps. Since many tetrahedra do not change significantly over time, hypertetrahedra can be collapsed both in the temporal domain as well as in the spatial domain. This results in hypertetrahedra that are either stretched in space or in time. Mesh decimation in time

**Fig. 11.** Hypertetrahedron

means that a hypertetrahedron (4-D) that does not change over time can be represented by a simple tetrahedron (3-D), just like a tetrahedron can be represented by a single point. The opposite direction (expansion of a tetrahedron to a hypertetrahedron over time) is not necessary, because a hypertetrahedron is collapsed only if it does not change significantly in a later time step.

The latter of the previously mentioned cases turns out to restrict the decimation ratio significantly. In the given example of earthquake simulations, many tetrahedra in the peripheral regions do not change at the beginning and towards the end of the simulation, but they are affected during the peak time of the earthquake. Since we do not allow hypertetrahedron expansion from a tetrahedron (split in the temporal domain), a large potential for decimation is wasted. Also, for practical purposes the given approach is not very suitable, because we need to be able to access the position of each vertex in the mesh at every time step if we want to navigate in both directions in the temporal domain. The reconstruction of this information and navigation in time with VCR-like controls requires a global view of the data. This means that the data cannot be processed sequentially for an infinite number of time steps. Consequently, the algorithm is not scalable.

Figure 12 shows an example where one node is affected by a velocity vector. The velocity is proportional to the displacement, because all time steps have the same temporal distance. Therefore, the arrow indicates the position of the node in the next time step. Two tetrahedra are affected by this displacement and change over time. In this example, it would be sufficient to store the time history of the affected node (solid line) or the affected tetrahedra (dotted lines). In order to reconstruct the mesh, it would be necessary to search for the most recent change in the time history of each node, which would require keeping all time histories of all nodes in memory during playback. This becomes particularly obvious if forward and backward navigation in time is considered. Even though this method offers a very compact representation of a time-varying tetrahedral mesh, we propose a different approach that enables easier playback (forward and backward) of all the time steps in a simulation.

The standard method would be to decimate the mesh for each time step separately by applying *QTetFusion* or some other mesh decimation technique, resulting in very high decimation rates in the initial time steps (regular, undistorted grid, no disruption due to earthquake), and moderate decimation of the later time steps where more information needs to be preserved. However, this approach would result in different meshes for every time step, leading to “jumps” and flicker in the visualization. This would be very disrupting in an animation or on a virtual reality display (Sect. 5).

Therefore, we use a different approach. The idea is to preserve every time step, which is necessary for playback as an animation and for navigation in time. The mesh that has the greatest distortion due to the earthquake (the velocity vector values associated with each grid node) is identified, and then decimated. All the decimation steps that were necessary to reduce the complexity of this mesh are recorded. For the record, it is sufficient to store the IDs of the affected tetrahedra in a list, because for the given application the IDs of the tetrahedra are identical in all time steps. Since tetrahedra are only removed but never added, the IDs are always unique. These recorded steps are then used to guide the decimation of the remaining meshes, i.e.,

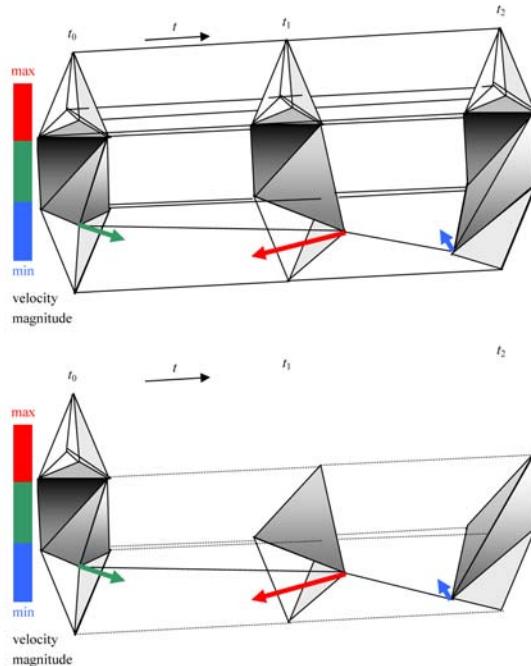


Fig. 12. One affected node, two affected tetrahedra

the meshes of the other time steps are decimated in the exact same manner as the one whose features are supposed to be preserved.

Figure 13 shows that the decimation of t_0 and t_1 is guided by t_2 , because t_2 is more distorted than any of the others. The decimated mesh should represent all displaced nodes in the most accurate way, because these are the ones that represent the significant features in the given application. Isotropic regions, i.e., areas that are not affected by the earthquake, such as the three tetrahedra at the top that are simplified into a single tetrahedron, expose only little variance in the data attributes, and consequently do not need to be represented as accurately, i.e., with the same error margin, as the significantly changing feature nodes in the other time steps. Comparing the top scenario (before decimation) and the bottom scenario (after decimation), the image shows that the tetrahedra on the top that were simplified in the selected t_2 time step are also simplified in the other two time steps (t_0 and t_1).

The question that remains is how to identify this “most distorted” mesh. Instead of using complex criteria, such as curvature (topology preservation) and vector gradients (node value preservation), we simply divide the length of the displacement vector for each node by the average displacement of that node, calculate the sum of all these ratios, and find the mesh that has the maximum sum, i.e., the maximum activity. If there is more than one mesh with this property, we use the one with the smallest time index. The average activity of a node is the sum of all displacement

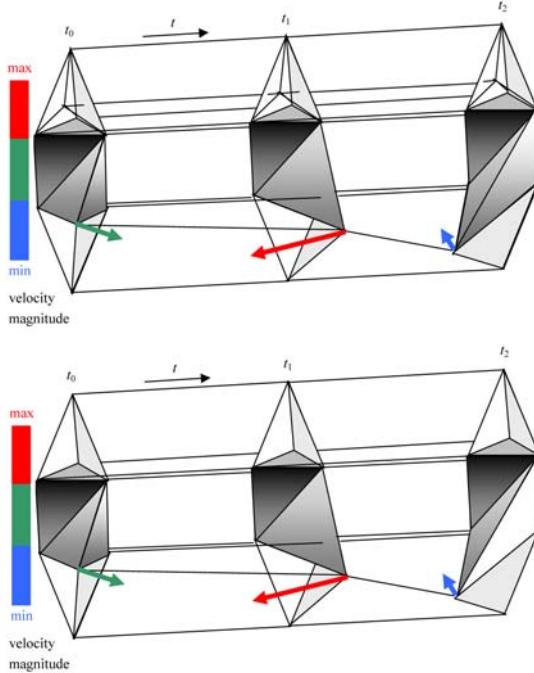


Fig. 13. Preservation of temporal resolution, decimation guided by t_2

vector lengths for all time steps divided by the number of time steps. This means that we consider those nodes in the mesh that expose a lot of activity, and try to represent the mesh that contains these active nodes in the best possible way. The mesh decimation algorithm is applied only to this one mesh. All other meshes are decimated according to this guiding mesh, using the same node indices for the collapse sequence.

The index $mesh_{guide}$ of the mesh that guides the decimation can be calculated in linear time using the following equation:

$$mesh_{guide} = \min\{i \in \mathbb{N}_0 | f(i) = \max\{f(i) | i \in \mathbb{N}_0\}\}$$

$$f(i) = \sum_{k=0}^{n-1} \frac{|\vec{d}_{k,i}|}{\sum_{l=0}^{n-1} |\vec{d}_{l,i}|}$$

n number of nodes

$\vec{d}_{k,i}$ displacement vector k in mesh i .

One drawback of this method is the search for the most active mesh nodes. However, the search is performed in linear time, and the extra time for the search is more than compensated for by the fact that all the other time steps do not need to be decimated by a complex decimation algorithm. Instead, the same list of collapse operations is applied to each time step, resulting in a fast and efficient decimation of the entire data set. This method is scalable, as it does not require loading of more than one time step into memory at one time. It works for an arbitrarily large number of time steps. Also, the algorithm is not restricted to the *QTetFusion* method. It should also work with other decimation algorithms (see Sect. 2).

5 Results from Ground Motion and Structural Response Simulation

The simulation of the effect of an earthquake on a set of buildings was performed using the OpenSees simulation software [22].

A map of the surface (Fig. 15) was used to place a group of buildings along the projected fault line. Two different heights of buildings were used (3 story and 16 story structures, Fig. 14) to simulate various building types in an urban setting. A structural response simulation was calculated for this scenario and then combined with the visualization of the ground motion. The buildings were represented as boxes that resemble the frame structure that was simulated using a SDOF model.

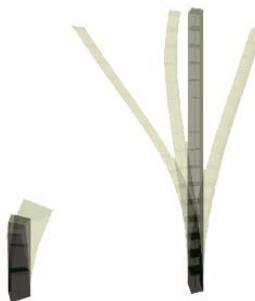


Fig. 14. 3 story vs. 16 story building (story drift horizontally exaggerated)

The scenario can be easily changed by selecting a different set of buildings and different locations on the surface map. The placement of the buildings and the selection of building types could be refined, based on real structural inventory data.

The finite element ground motion simulation was performed on a Cray T3E parallel computer at the Pittsburgh Supercomputer Center. A total of 128 processors took almost 24 hours to calculate and store an 8 second velocity history of an approximately 12 million-node, three-dimensional grid. The required amount of disk space for this problem was approximately 130 GB. Figure 15 shows a 2-D surface plot of the simulation in two coordinate directions.

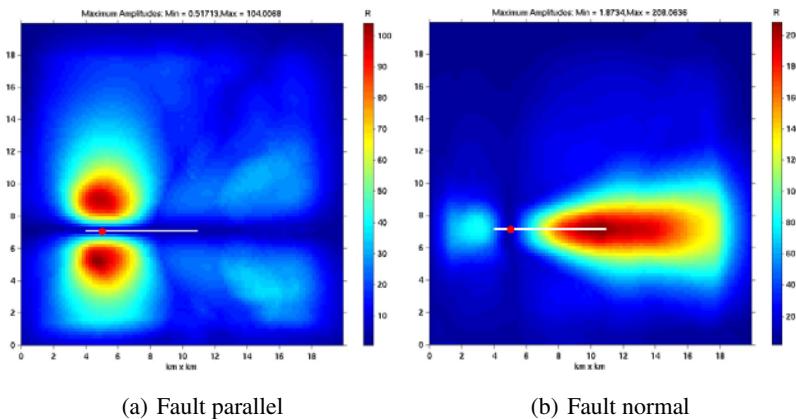
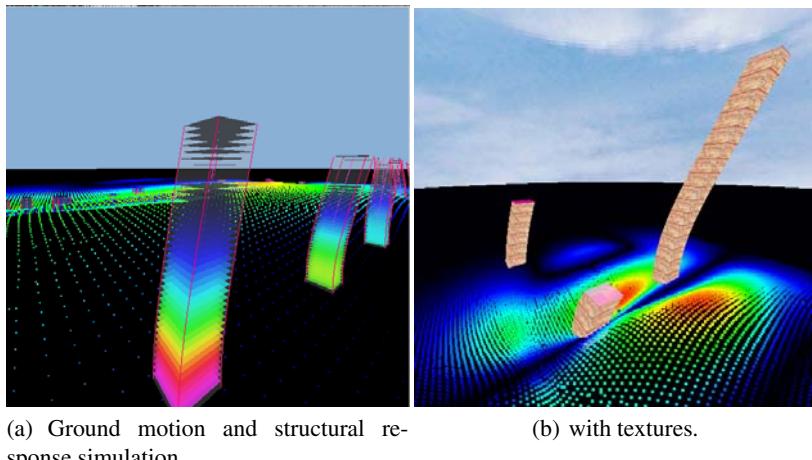
**Fig. 15.** Velocity plot

Figure 16a shows a 3-D rendering of the surface mesh combined with the structural response simulation. The various intensities (colors) on the buildings indicate the maximum story drift for each floor. Figure 16b shows a hybrid rendering of ground motion and structural response. Textures have been added for a more photorealistic representation of the buildings and the environment.

It turns out that some buildings experience more stress than others, even if they are in close proximity or in the same distance from the fault line as their neighbors. The determining factors are building height, orientation of the frame structure, and building mass.



(a) Ground motion and structural response simulation,

(b) with textures.

Fig. 16. (a) Ground motion and structural response simulation, (b) with textures

Figure 17 shows a scenario in a CAVETM-like virtual environment (four stereoscopic rear-projection screens with LC shutter glasses and electro-magnetic head and hand tracking system) [18]. The user is fully immersed in the visualization and gets both visual and audio feedback while the shockwave approaches and the buildings start to collapse [4].

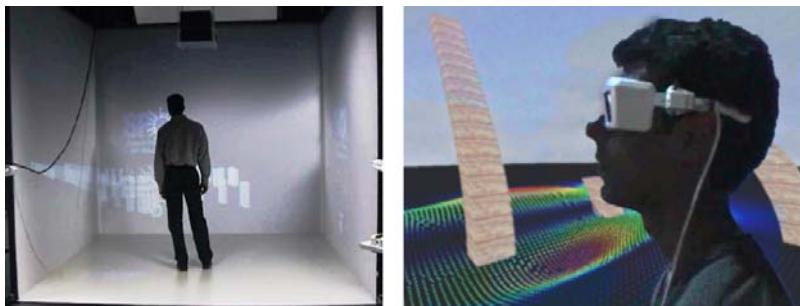


Fig. 17. Virtual environment visualization

6 Conclusions

We presented an integrated framework for domain- and field-error controlled mesh decimation. The original tetrahedral fusion algorithm (*TetFusion*) was extended by employing a *planar quadric error metric* (*PQEM*). The additional computational overhead introduced by this error metric is justified by added features, such as topology preservation, and a better decimation ratio. The trade-off between time complexity of *QTetFusion* and the error in either the vertex domain or the attribute field introduced as a result of tetrahedral fusion needs to be investigated in more detail.

The atomic decimation operation employed (*TetFuse*) is symmetric, and better suited for 3-D volumetric meshes than *edge-collapse*-based methods, because it generates less topological inconsistencies (tetrahedra are usually stretched *away* from their base plane). Remaining cases of negative volumes are solved by an early rejection test for tetrahedral flipping. In *QTetFuse*, the barycenter as the center of the tetrahedral collapse has been replaced by a general *fusion point* that minimizes the *PQEM*. This improves mesh consistency and reduces the overall error of the decimated mesh.

A control parameter can be used to provide a smooth and controlled transition from one step to the next. Therefore, the method can be employed to implement a hierarchical level-of-detail set that can be used in multi-resolution rendering algorithms allowing for a smooth transition between multiple levels of detail (hierarchical refinement). One could also think of a view-dependent, localized refinement for applications such as flight simulation.

Our future work includes offline compression of the datasets, as suggested by Alliez et al. [1] and Isenburg et al. [15], and similar to the schemes suggested by Gumhold et al. [12], Pajarola et al. [23], or Szymczak et al. [29]. This would enable dynamic (on the fly) level-of-detail management for volume meshes similar to those methods that currently exist for polygonal meshes [6].

In this chapter, we presented a general method for decimation of time-varying tetrahedral meshes. The algorithm preserves the discrete time steps in the temporal domain, which is critical for interactive navigation in both directions in the time domain. It also represents an intuitive method for consistent mesh generation across the temporal domain that produces topologically and structurally similar meshes for each pair of adjacent time steps. The algorithm presented in this chapter is not restricted to a particular mesh decimation technique. It is an efficient method that exploits and preserves mesh consistency over time, and most importantly, is scalable.

Acknowledgements

This work was supported by the National Science Foundation under contract 6066047–0121989 and through the National Partnership for Advanced Computational Infrastructure (NPACI) under contract 10195430 00120410. We would like to acknowledge Gregory L. Fenves and Bozidar Stojadinovic, Department of Civil and Environmental Engineering, University of California at Berkeley, for providing us with the Structural Response Simulation and the OpenSeesTM simulation software, Jacobo Bielak and Antonio Fernández, Department of Civil and Environmental Engineering, Carnegie Mellon University, Pittsburgh, MA, for providing us with the ground motion simulation data, and Prashant Chopra, Z-KAT, Hollywood, Florida, for the software implementation. We would like to thank Peter Williams, Lawrence Livermore National Laboratory, for the *super phoenix* dataset. We would also like to thank Roger L. King and his colleagues at the Engineering Research Center at Mississippi State University for their support of this research contract. Finally, we would like to thank Elke Moritz and the members of the Center of Graphics, Visualization and Imaging Technology (GRAVITY) at the University of California, Irvine, for their help and cooperation.

References

1. Pierre Alliez and Mathieu Desbrun. Progressive Compression for Lossless Transmission of Triangle Meshes. In *Proceedings of SIGGRAPH 2001, Los Angeles, CA*, Computer Graphics Proceedings, Annual Conference Series, pp. 198–205. ACM SIGGRAPH, ACM Press, August 2001.
2. Prashant Chopra and Joerg Meyer. Tetfusion: An Algorithm for Rapid Tetrahedral Mesh Simplification. In *Proceedings of IEEE Visualization 2002, Boston, MA*, pp. 133–140. IEEE Computer Society, October 2002.

3. Prashant Chopra and Joerg Meyer. Topology Sensitive Volume Mesh Simplification with Planar Quadric Error Metrics. In *IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP 2003)*, Benalmadena, Spain, pp. 908–913. IASTED, September 2003.
4. Prashant Chopra, Joerg Meyer, and Michael L. Stokes. Immersive Visualization of a Very Large Scale Seismic Model. In *Sketches and Applications of SIGGRAPH'01 (Los Angeles, California, August 2001)*, page 107. ACM SIGGRAPH, ACM Press, August 2001.
5. P. Cignoni, D. Costanza, C. Montani, C. Rocchini, and R. Scopigno. Simplification of Tetrahedral Meshes with Accurate Error Evaluation. In Thomas Ertl, Bernd Hamann, and Amitabh Varshney, editors, *Proceedings of IEEE Visualization 2000, Salt Lake City, Utah*, pp. 85–92. IEEE Computer Society, October 2000.
6. C. DeCoro and R. Pajarola. XFastMesh: Fast View-dependent Meshing from External Memory. In *Proceedings of IEEE Visualization 2002, Boston, MA*, pp. 363–370. IEEE Computer Society, October 2002.
7. T. K. Dey, H. Edelsbrunner, S. Guha, and D. V. Nekhayev. Topology Preserving Edge Contraction. *Publications de l'Institut Mathematique (Beograd)*, 60(80):23–45, 1999.
8. H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, 2001.
9. M. Garland. Multi-resolution Modeling: Survey & Future Opportunities. In *EUROGRAPHICS 1999, State of the Art Report (STAR) (Aire-la-Ville, CH, 1999)*, pp. 111–131. Eurographics Association, 1999.
10. M. Garland and P. Heckbert. Surface Simplification Using Quadric Error Metrics. In *Proceedings of SIGGRAPH 1997, Los Angeles, CA*, pp. 115–122. ACM SIGGRAPH, ACM Press, 1997.
11. T. Gerstner and M. Rumpf. Multiresolutional Parallel Isosurface Extraction Based on Tetrahedral Bisection. In *Proceedings 1999 Symposium on Volume Visualization*. IEEE Computer Society, 1999.
12. S. Gumhold, S. Guthe, and W. Strasser. Tetrahedral Mesh Compression with the Cut-Border Machine. In *Proceedings of IEEE Visualization 1999, San Francisco, CA*, pp. 51–59. IEEE Computer Society, October 1999.
13. I. Guskov, K. Vidimce, W. Sweldens, and P. Schroeder. Normal Meshes. In *Proceedings of SIGGRAPH 2000, New Orleans, LA*, pp. 95–102. ACM SIGGRAPH, ACM Press, July 2000.
14. H. Hoppe. Progressive Meshes. In *Proceedings of SIGGRAPH 1996, New Orleans, LA*, pp. 99–108. ACM SIGGRAPH, ACM Press, August 1996.
15. M. Isenburg and J. Snoeyink. Face Fixer: Compressing Polygon Meshes with Properties. In *Proceedings of SIGGRAPH 2000, New Orleans, LA*, pp. 263–270. ACM SIGGRAPH, ACM Press, July 2000.
16. A. D. Kalvin and R. H. Taylor. Superfaces: Polygonal Mesh Simplification with Bounded Error. *IEEE Computer Graphics and Applications*, 16(3):64–77, 1996.
17. M. Kraus and T. Ertl. Simplification of Nonconvex Tetrahedral Meshes. *Electronic Proceedings of the NSF/DoE Lake Tahoe Workshop for Scientific Visualization*, pp. 1–4, 2000.
18. Joerg Meyer and Prashant Chopra. Building Shaker: Earthquake Simulation in a CAVE™. In *Proceedings of IEEE Visualization 2001, San Diego, CA*, page 3, October 2001.
19. Joerg Meyer and Prashant Chopra. Strategies for Rendering Large-Scale Tetrahedral Meshes for Earthquake Simulation. In *SIAM/GD 2001, Sacramento, CA*, page 30, November 2001.

20. B. Munz. *The Earthquake Guide*. University of California at San Diego. <http://help.sandiego.edu/help/info/Quake/> (accessed November 26, 2003).
21. M. Ohlberger and M. Rumpf. Adaptive projection operators in multiresolution scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):74–93, 1999.
22. OpenSees. *Open System for Earthquake Engineering Simulation*. Pacific Earthquake Engineering Research Center, University of California at Berkeley. <http://opensees.berkeley.edu> (accessed November 28, 2003).
23. R. Pajarola, J. Rossignac, and A. Szymczak. Implant Sprays: Compression of Progressive Tetrahedral Mesh Connectivity. In *Proceedings of IEEE Visualization 1999, San Francisco, CA*, pp. 299–305. IEEE Computer Society, 1999.
24. J. Popovic and H. Hoppe. Progressive Simplicial Complexes. In *Proceedings of SIGGRAPH 1997, Los Angeles, CA*, pp. 217–224. ACM SIGGRAPH, ACM Press, 1997.
25. K. J. Renze and J. H. Oliver. Generalized Unstructured Decimation. *IEEE Computer Graphics and Applications*, 16(6):24–32, 1996.
26. W. J. Schroeder. A Topology Modifying Progressive Decimation Algorithm. In *Proceedings of IEEE Visualization 1997, Phoenix, AZ*, pp. 205–212. IEEE Computer Society, 1997.
27. W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of Triangle Meshes. *Computer Graphics*, 26(2):65–70, 1992.
28. O. G. Staadt and M. H. Gross. Progressive Tetrahedralizations. In *Proceedings of IEEE Visualization 1998, Research Triangle Park, NC*, pp. 397–402. IEEE Computer Society, October 1998.
29. A. Szymczak and J. Rossignac. Grow Fold: Compression of Tetrahedral Meshes. In *Proceedings of the Fifth Symposium on Solid Modeling and Applications, Ann Arbor, Michigan*, pp. 54–64. ACM, ACM Press, June 1999.
30. I. J. Trotts, B. Hamann, and K. I. Joy. Simplification of Tetrahedral Meshes. In *Proceedings of IEEE Visualization 1998, Research Triangle Park, NC*, pp. 287–296. IEEE Computer Society, October 1998.
31. I. J. Trotts, B. Hamann, and K. I. Joy. Simplification of Tetrahedral Meshes with Error Bounds. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):224–237, 1999.
32. G. Turk. Re-tiling Polygonal Surfaces. *Computer Graphics*, 26(2):55–64, 1992.
33. Y. Zhou, B. Chen, and A. Kaufman. Multiresolution Tetrahedral Framework for Visualizing Regular Volume Data. In R. Yagel and H. Hagen, editors, *Proceedings of IEEE Visualization 1997*, pp. 135–142, Phoenix, AZ, 1997.

Author Index

Bähr 367	Hamann 3, 35	Pagendarm 264
Bonsma 367	Hauser 305	Peikert 65
Brodlie 395	Hoile 367	Rütten 264
	Huang 97	Rodgman 162
Chen 162	Joy 3, 35	Scheuermann 231, 249
Childs 3		Sigg 65
Comba 16	Kanitsar 207	Silva 16
Crawfis 344	Kaufman 131, 149	Straßer 115
Danovaro 78	Kraus 115	Thompson 395
De Floriani 78	Lakare 131, 149	Tricoche 249
Ebert (Achim) 328	Lee 49, 97	van Lengen 367
Ebert (David S.) 385	Linsen 35	Varshney 49
Ebling 231	Marrow 367	Vivodtzev 35
Ertl 115	Mason 395	Walkley 395
Fleischmann 207	Meister 207	Wegenkittl 207
Gaither 385	Meyer 286, 408	Weiler 115
Garth 249	Mitchell 16	Wiley 3
Gröller 207	Mueller 131	Wischgoll 286, 408
Gregorski 3	Nielson 97	Wood 395
Guthe 115		Yang 344
Hagen 97, 328, 367	Olshausen 35	