Gradient Dependent Reconstruction from Scalar Data

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Arindam Bhattacharya, M.S.

Graduate Program in Computer Science and Engineering

The Ohio State University

2015

Dissertation Committee:

Dr. Rephael Wenger, Advisor

Dr. Han Wei Shen

Dr. Tamal Dey
Abstract

Computed Tomography (CT) is widely accepted as an important tool in medicine. Increasingly, CT is finding a wide variety of application in material science and engineering fields. CT is being used for non-destructive inspection and characterization of aeronautical and automobile components. These components have wide variations in geometry and material characteristics, from single solid piece of metals such as aluminum to exotic composite materials and from micro scale engine parts to large scale airplane tail fins.

Unlike organic parts, machine parts often have ‘sharp’ features. Consequently, feature sensitive reconstruction from volume data has seen sporadic but critical work in the recent years. A number of these papers present algorithms to construct isosurfaces with sharp edges and corners from Hermite data, i.e. data containing the exact surface normals at the exact intersection of the surface and grid edges. Such surface normals are not available with CT data. In this thesis, we discuss some fundamental problems with the previous algorithms and the difficulties in using these algorithms on real CT (scalar data) and further describe a new approach to feature reconstruction from volume data.

Feature sensitive reconstruction is based on the ability to approximate surface normals from scalar field gradients. Change in gradients can also be used to measure local directions of geometric structures in CT data. We describe a method to extract fiber bundle directions from industrial CT of fiber composites using gradients.

Specifically this dissertation proposes, 1) a method to reconstruct isosurfaces from scalar data while preserving sharp features (edges and corners) given a scalar grid and
the gradients at the grid locations; 2) a method to select the correct gradients at the grid locations which will be used as input to the above algorithm; 3) Finally, a method for extracting and visualizing fiber bundles in fiber reinforced composites scanned with X-ray computed tomography (XCT).
Dedicated to my mother and father.
Acknowledgments

First and foremost, I would like to express my deep gratitude to my advisor, Dr. Rephael Wenger, for his support and guidance throughout my time at The Ohio State University. By working with him, I have learned how to solve problems in a scientific way. Without his guidance and encouragement, this dissertation would not have been possible.

I am grateful to Dr. Han Wei Shen and Dr. Tamal Dey for serving on the dissertation committee.

I am also grateful to Dr. Christoph Heinzl and the Computed Tomography Lab in University of Applied Sciences, Upper Austria. Dr. Heinzl provided me an opportunity to work in an exciting industrial research environment. Results from my work in Dr. Heinzl’s laboratory are a crucial part of my thesis.

I would like to thank the following current and former PhD students: Hao Ding, Sofoklis Floratos, Firdaus Janoos, Shantanu Singh, Mustafa Arikan, Xiaoyin Ge, Weissenbck Johannes and Aritra Sengupta for their help and encouragement.

The last part is reserved for the ones that are most important to me: my wife, Paromita Bhattacharjee, and my parents, Jayati Bhattacharya and Ansuman Bhattacharya. They have supported me with all the love they have, and that is what makes me, beyond this dissertation itself.
Vita

West Bengal University of Technology.
Kolkata, India

May 2015 .......................................... M.Sc. Computer Science and Engineering,
The Ohio State University. Columbus, OH, USA

Sept. 2009 – Present ............................. Ph.D. Student, Department of Computer
Science and Engineering, The Ohio State
University. Columbus, OH, USA

Publications

Research Publications

A. Bhattacharya and R. Wenger. Constructing isosurfaces with sharp edges and corners

A. Bhattacharya, C. Heinzl, A. Amirkhanov, J. Kastner and R. Wenger. MetaTracts-A
method for robust extraction and visualization of carbon fiber bundles in fiber reinforced

Fields of Study

Major Field: Computer Science and Engineering
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Dedication</td>
<td>iv</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>v</td>
</tr>
<tr>
<td>Vita</td>
<td>vi</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xii</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Overview</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 Scalar Field</td>
<td>2</td>
</tr>
<tr>
<td>1.1.2 Derivatives of a scalar field</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Reconstruction from Industrial CT of Machine Parts</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Reconstruction from Industrial CT of Composite Materials.</td>
<td>6</td>
</tr>
<tr>
<td>1.3.1 Carbon Fiber Reinforced Polymer (CFRP)</td>
<td>6</td>
</tr>
<tr>
<td>2. Background</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Definitions</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Related Work</td>
<td>12</td>
</tr>
<tr>
<td>2.2.1 Marching Cubes</td>
<td>12</td>
</tr>
<tr>
<td>2.2.2 Dual Contouring</td>
<td>13</td>
</tr>
<tr>
<td>2.2.3 Dual Marching Cubes</td>
<td>13</td>
</tr>
<tr>
<td>2.2.4 Sharp Edges and Corners</td>
<td>14</td>
</tr>
<tr>
<td>2.3 Computing Vertex Locations using Quadric Error Measure</td>
<td>16</td>
</tr>
</tbody>
</table>
2.4 Problems with Vertex Locations ........................................... 19
   2.4.1 Notches and Degeneracies ............................................. 19
   2.4.2 Poor tolerance to approximation errors ............................. 21
   2.4.3 Reliance on Hermite data ............................................. 22
2.5 Gradient Computation ................................................... 22
   2.5.1 Anisotropic diffusion ................................................ 22
2.6 Point Cloud Data: ...................................................... 23
   2.6.1 Delaunay refinement for piecewise smooth complexes ........... 24
2.7 Visual Analysis and Modeling of Fiber Reinforced Composites ....... 26
   2.7.1 Diffusion Tensor Imaging ............................................. 27
   2.7.2 Fiber Tracking in Diffusion Tensor Imaging ....................... 27
   2.7.3 Fiber Clustering in Diffusion Tensor Imaging ..................... 28
   2.7.4 Second Order Local Structure ..................................... 29

3. RELIGRAD: Computing Reliable Gradients from Scalar Data .......... 31
   3.1 Overview ............................................................. 31
   3.2 Determining Correct Gradients ....................................... 33
      3.2.1 Definitions ....................................................... 34
      3.2.2 Central Difference Formula ..................................... 34
      3.2.3 Reliable Gradients ............................................... 36
      3.2.4 CT Data .......................................................... 42
      3.2.5 Extending Reliable Gradients ................................... 47
      3.2.6 Algorithm RELIGRAD ........................................... 48

4. SHREC: SHarp REConstruction from Scalar Data ...................... 49
   4.1 Protecting Ball Overview ............................................. 49
   4.2 SHREC Overview ...................................................... 50
   4.3 SHREC Details ........................................................ 52
   4.4 Vertex Computation ................................................... 53
      4.4.1 SHREC vertex Computation ...................................... 53
      4.4.2 Problems with computing and selecting vertices ............... 56
      4.4.3 Computing Vertex Locations on 1-Dimensional Features ...... 57
      4.4.4 Swapping Locations .............................................. 59
      4.4.5 Locations on Planes ............................................. 60
   4.5 Selecting 0D and 1D Feature Cubes ................................ 62
   4.6 Merging Points with Features Points ................................ 70
      4.6.1 Ordered Merging .................................................. 70
      4.6.2 Merging Tests ..................................................... 71
      4.6.3 Distortion Tests .................................................. 73
      4.6.4 Pair Merging ..................................................... 74
4.6.5 Extended Merging ........................................... 75
4.7 Constructing isosurface triangles and quadrilaterals ............... 76
4.7.1 Grid Spacing .............................................. 78
4.8 Parameters .................................................. 79

5. Experimental Results ........................................ 83

5.1 Synthetic Scalar and Gradient Data ................................... 83
5.1.1 Cube Dataset ............................................ 83
5.1.2 TwoCube Dataset ......................................... 83
5.1.3 Annuli Dataset ........................................... 85
5.1.4 Flange dataset ........................................... 85
5.1.5 Cone and Cannon Dataset ................................ 86
5.1.6 Gradients ................................................ 87
5.1.7 Measurements ............................................ 88

5.2 Experimental results of RELIGRAD ................................... 90
5.2.1 Quantitative Analysis ...................................... 93

5.3 Experimental results of SHREC ..................................... 97
5.3.1 Isosurface Reconstruction on Synthetic Data ....................... 97
5.4 Comparison with Other Algorithms ................................... 105
5.4.1 Comparison with MergeSharp .................................. 106
5.4.2 Comparison with PolyMender .................................. 108
5.4.3 Comparison with EMC (Extended Marching Cubes): ............... 113
5.4.4 Comparison with SingularCocone ................................ 116
5.4.5 Experimental details: ...................................... 117

5.5 Experimental Results on CT Data .................................... 119
5.5.1 Reliable Gradients from industrial CT data ....................... 120
5.5.2 Reconstruction from industrial CT data ......................... 120

5.6 Timings .................................................................. 121
5.7 Software and Datasets ............................................... 122
5.8 Assumptions about extracted “sharp” features ......................... 123


6.1 Overview ........................................................ 129
6.2 Data Characteristics and Assumptions ................................ 132
6.3 Extracting MetaTracts ............................................. 134
6.3.1 Computing Reliable Hessians .................................. 134
6.3.2 MetaTracts Properties ....................................... 137
6.3.3 MetaTracts Generation ....................................... 138
6.4 Fiber Bundle Generation .......................................... 142
6.4.1 Orientation Based Clustering .................................. 142
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Synthetic test dataset information.</td>
<td>96</td>
</tr>
<tr>
<td>5.2 Maximum Angle difference compared to correct gradients</td>
<td>97</td>
</tr>
<tr>
<td>5.3 Maximum of $L_1$ distances to closest grid vertices with exact gradients.</td>
<td>97</td>
</tr>
<tr>
<td>5.4 Results after adding uniform noise to the datasets.</td>
<td>98</td>
</tr>
<tr>
<td>5.5 Surface reconstruction software.</td>
<td>105</td>
</tr>
<tr>
<td>5.6 CT dataset information.</td>
<td>119</td>
</tr>
<tr>
<td>5.7 Timings. SHREC grad3 and SHREC grad9 is SHREC using gradients from $3 \times 3 \times 3$ and $9 \times 9 \times 9$ subgrids, respectively, around selected vertices.</td>
<td>121</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>X-ray CT image slice of the engine cylinder dataset.</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>(a) Non-smooth scalar field. (b) Discontinuity (red) in the gradients. (c) Isocontour (green) produced by the Marching Squares Algorithm. (d) Level set (blue) with sharp corner.</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>Beveling on an edge using Marching Cubes [49] as shown in [46]. The resolution increases (left to right), the beveling becomes less visually pronounced but the problem remains.</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>Reconstruction problems.</td>
<td>20</td>
</tr>
<tr>
<td>3.1</td>
<td>Algorithm ReliGrad.</td>
<td>32</td>
</tr>
<tr>
<td>3.2</td>
<td>Gradient Discontinuities.</td>
<td>34</td>
</tr>
<tr>
<td>3.3</td>
<td>Gradient computation around a sharp surface edge in a CT data.</td>
<td>35</td>
</tr>
<tr>
<td>3.4</td>
<td>(a) Vector $\phi(n, n')$ predicted by $n$ and $n'$. (b) Vector $\phi_2(n, n')$ predicted by $n$ and $n'$.</td>
<td>37</td>
</tr>
<tr>
<td>3.5</td>
<td>Red curve is boundary separating regions $A_1$ and $A_2$. Green vertex $v$ is in $A_1$ but no vertex of $N(v)$ lies in $I_V(A_1)$. Vertices in $N(v)$ are colored blue. Vertices in $I_V(A_1)$ are colored yellow.</td>
<td>40</td>
</tr>
<tr>
<td>4.1</td>
<td>Algorithm SHREC.</td>
<td>50</td>
</tr>
<tr>
<td>4.2</td>
<td>2D illustration of vertex and cube selection.</td>
<td>51</td>
</tr>
<tr>
<td>4.3</td>
<td>SHREC algorithm for generating points on sharp features.</td>
<td>52</td>
</tr>
<tr>
<td>4.4</td>
<td>Cube line intersection.</td>
<td>58</td>
</tr>
</tbody>
</table>
4.5 (a) Two grids cubes sharing an edge $c$. (b) The two other cubes (yellow) which also contain $c$.

4.6 (a) Two grids cubes sharing a vertex $v$. (b) The six other cubes (yellow) which also contain $v$.

4.7 Tightly packed 2D configurations of selected squares.

4.8 Problematic 2D configurations of selected squares.

4.9 (a) Configuration (4,2) has space between the $3 \times 3$ subgrids around each selected square. (b) Additional cube (magenta) which packs tightly with two other squares.

4.10 Tightly packed 3D configurations of selected cubes.

4.11 Problematic 3D configurations of selected cubes.

4.12 Problematic interaction of red and blue cubes in configuration (3,2,2) is blocked by magenta cube.

4.13 SHREC algorithm for selecting 0D and 1D feature cubes.


4.15 (a) Isosurface in yellow square is connected to three selected cubes. Yellow square should merge with the middle green square, not with either of the blue squares. (b) Merging the red vertex with either of the yellow vertices will cause the yellow edge to be contained in four polygons.

5.1 Isosurfaces constructed from synthetic test datasets.

5.2 Polygonal meshes representing level sets.

5.3 SHREC Flange isosurface.

5.4 SHREC TwoCubes isosurface.

5.5 Gradient results on a part of the Annulus dataset (Figure 5.1).

5.6 Cannon dataset gradients (Curved edge, non 90° angles)

5.7 Effect of uniform noise on cube gradients.
5.8 Central difference gradients on grid vertices of a portion of *engine cylinder* dataset.

5.9 The gradients generated on a portion of the *engine cylinder* dataset.

5.10 *Engine Cylinder* dataset. (a) a slice of the original CT image, note the streaking artifacts introduced during the scanning process. The inset shows the original machine part. Figure (b), (c), shows the sharp mesh, and the extracted sparse feature points.

5.11 Results of SHREC using exact gradients and Religrad gradients on Flange datasets.

5.12 Results of SHREC using exact gradients and Religrad gradients on TwoCube datasets.

5.13 SHREC Cannon and Smooth Tip Cone isosurfaces (Religrad gradients.) (a) Cannon isosurface. 1-dimensional feature has dihedral angle 120°. (b) Smooth Tip Cone isosurface. 1-dimensional feature has dihedral angle 60°.

5.14 Results of SHREC on 15 Cannon and 14 Smooth Tip Cone datasets.

5.15 Results of MergeSharp on 39 Flange and 34 TwoCubes datasets (exact gradients).

5.16 (a) Notch in a MergeSharp Flange isosurface. (b) Dimples in a MergeSharp Cone isosurface (left). Compare with smooth SHREC isosurface (right).

5.17 PolyMender and SHREC comparison.

5.18 Results of PolyMender on 39 Flange datasets.

5.19 Results of PolyMender on 34 TwoCubes datasets.

5.20 EMCpoly (Extended Marching Cubes) and SHREC comparison.

5.21 Comparison of SHREC and SingularCocone on 10 Flange and 15 TwoCubes datasets.

5.22 SingularCocone TwoCubes isosurface.

5.23 Motorcycle Engine dataset. (a) a slice of the original CT image, note the streaking artifacts introduced during the scanning process. The inset shows the original machine part. Figure (b) shows the sharp mesh.
5.24 SHREC with Religious gradients computed from part of industrial CT data (Motorcycle Engine). Magnified regions show “sharp” edges reconstructed. In red, picture from the original item. ........................................... 127

5.25 SHREC with Religious gradients computed from part of the VOLT data. Magnified regions show “sharp” reconstructed edges. In red picture of the original item. ......................... 127

5.26 (a) SHREC with Religious gradients computed from part of the CMM dataset. Magnified regions show “sharp” reconstructed edges. (b) Single slice of the CT scan, with scalar values mapped to the “heat” color map. ........................................ 128

6.1 Flow chart of the MetaTracts approach for fiber bundle extraction ......................... 129

6.2 Data characteristics: (a) Rendering of dataset D1. (b, c) 2D slices along Z- and X-axis. (d) Zoom in of the green region marked in (c). Multiple fiber bundles cross and are indistinguishable by visual inspection alone. ........................................... 132

6.3 Reliable Hessians. (a) MetaTracts colored according to the local orientation vector mapped to RGB color scale. (b,c) 2D slices along Z- and X-axis. (d) Magnified region marked in (c). 136

6.4 MetaTracts generation. (b) MetaTracts generation process. (a) Magnified region in (b). (c) Individual MetaTract cylinder. .......................................................... 138

6.5 MetaTracts color-coded according to their mean orientations, along with slices of the YZ, XY-plane. .......................................................... 141

6.6 Clustering Results: Orientation clustering (a,b,c), Distance clustering (d,e). (b) Results of K-means clustering ($K = 2$) with MetaTracts (nodes) projected to the top three eigenvectors as major axes. (a) MetaTracts belonging to orientation cluster 1 in blue. (c) MetaTracts belonging to orientation cluster 2 in red, MetaTracts in gray (a,c) show context. (d,e) Hierarchical (distance based) clustering on each orientation cluster. (f) Combined result showing 11 individual extracted clusters. ........................................... 150

6.7 Applying only (a) Hierarchical Clustering and (b) Dimensionality reduction followed by K-means clustering, methods for various numbers of clusters (distance measure is minimal directed Hausdorffs). For comparison the results using two-step clustering is in Figure 6.6(f). 151

6.8 Voxelization and surface extraction. (a) Voxelization of data set 1, the inset shows the MetaTracts after clustering. (b) a single slice along the XY-Plane (c) Two of the extracted meshes together (d) The meshes rendered separately. ........................................... 152

xv
6.9 Data set with flat thin and compact bundles. (a) shows the volume rendering and a 2D slice with one of the boundaries marked in green, (b) shows the clusters according to individual orientation. (c) shows the complete result. (d) shows the voxelization of (c)...

6.10 (a) Length distribution of individual MetaTracts for a particular bundle (unit for length is grid cube edge length: 2μm).

6.11 Number of MetaTracts and median, minimum and maximum length of an individual MetaTracts in the orientation cluster Figure 6.6(d) when clustered into 10 clusters. The unit for the MetaTracts length is the grid cube length. The labeled clusters 2, 3, 4, 5 and 7 have a cardinality less than 10 and are removed.
Chapter 1: Introduction

1.1 Overview

Computed tomography is an imaging method based on X-ray attenuation. Computed tomography allows generation of two dimensional cross sectional images (called ‘slices’) of the specimen. Figure 1.1 shows an example “slice” of a part of the engine dataset. The gray-level value of each pixel in a CT slice image corresponds to attenuation of the X-ray by the object under investigation. This attenuation primarily depends on the energy of the X-ray along with the material density and composition of the object being imaged. A digital 3-Dimensional volumetric dataset can be generated of the specimen from the 2-Dimensional slices. The key advantage of X-ray CT is the ability to capture both interior and exterior structures of the specimen without destroying it.

Medical X-ray CT is widely accepted as a powerful tool for medical diagnostics. However, the use of 3D X-ray computed tomography in the industrial context is a novel development. Industrial X-ray CT is proliferating as a means of non-destructive testing and dimensional measurement for industrial applications.

In recent years, industrial X-ray CT technology has continuously advanced in order to achieve higher resolutions. Higher resolutions facilitate detailed measurements, and novel algorithms which reconstruct surface geometry (from the X-ray scans) benefit greatly from these detailed measurements. An integral requirement of surface reconstruction algorithms
is to be able to accurately reproduce “features” of the object under investigation. “Features” in industrial CT are often identified as “sharp” edges and corners of the specimen. Consequently, feature sensitive reconstruction algorithms have seen critical and recent developments to accommodate the challenges of reconstructing from X-ray CT scans. A complementary area of work is the development of new methods which can quantitatively inspect the quality of the reconstructed surfaces and compare them to other algorithms and ground truth (such as CAD models).

Recent years have also seen a steady increase in the use of Carbon Fiber Reinforced Polymers (CFRP) in various areas of industrial production. CFRPs hold the promise of meeting the high demands and requirements set by automotive, aeronautics and other industries. Evaluation methods which allow material characterization and non-destructive testing of these new materials are current field of research. Once again X-ray CT with its ability to generate 3-Dimensional images provides the ideal platform for that purpose.

This thesis is broadly divided into two problems related to reconstruction from industrial CT data. Reconstruction of sharp edges and corners is introduced in Section 1.2. Reconstruction of fiber bundles from CT scans of composite materials is introduced in Section 1.3.

1.1.1 Scalar Field

A scalar field is a function $f$ which assigns a scalar value to each point in $\mathbb{R}^d$. The value $d$ is known as the dimension of the scalar field. Popular examples include densities, pressures or temperatures associated with points in $\mathbb{R}^3$. X-Ray CT scanners produce regular grids of scalar values representing material densities of scanned objects. These scalar values can be viewed as a sampling of a scalar field.

Given a scalar field and a constant scalar value $\sigma$, the set $f^{-1}(\sigma) = \{x : f(x) = \sigma\}$, is called a level set of $f$. In three dimension level sets are also called implicit surfaces or
isosurfaces. The isosurface separates grid vertices with scalar values less than $\sigma$ with grid values greater than $\sigma$. If the scanned object has material density $s$ different than the density $s'$ of its surrounding, then the isosurface with value $\sigma$ between $s$ and $s'$ will represent the boundary surface $\Sigma$ of the object.

1.1.2 Derivatives of a scalar field

If $f(x_1,\ldots,x_n)$ is a differentiable, scalar-valued function, its gradient is the vector

$$
\left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right)
$$

(1.1)

whose components are the $n$ partial derivatives of $f$. In the simplest case, gradients can be approximated by finite difference filters such as the central difference operator. (See
Section 3.2.2.) Correct gradients, if available, can be used to attack many difficult problems of reconstruction from industrial CT data.

Gradients are perpendicular to the level sets in the scalar field. Local tangent planes can be constructed from these gradients. These tangent planes can then be utilized to simultaneously, identify and locate sharp edges and corners in scalar fields. (See Section 1.2.)

A problem with using gradients to construct sharp features as mentioned above is that the central difference formula does not compute gradients at or near gradient discontinuities. This is because the central difference and other similar gradient operators assume the underlying field at the location is smooth. We use agreement of neighboring central difference gradients to predict which vertices have correct gradients. Section 1.2 discusses how correct gradients can be selected in the presence of gradient discontinuities.

The change in gradient is also useful for analyzing local geometric structure. Second-order partial derivatives (Hessian) can be used to approximate orientation in a local neighborhood. For example, assume a scalar field consists of a sampling of a single solid tubular structure. For a grid vertex inside the tubular structure, the gradient direction remains the same along the tube direction and changes rapidly in directions orthogonal to the tube orientation. Eigen analysis of the local Hessian can be used to compute this orientation information to extract tube like structures from the data. This is used to extract features from composite materials. (See Section 1.3.)

1.2 Reconstruction from Industrial CT of Machine Parts

We are interested in reconstructing isosurfaces which represent boundary surfaces $\Sigma$ accurately. In particular, we will illustrate algorithms which reconstruct isosurfaces from industrial CT scans of machine parts whose boundaries $\Sigma$ have sharp edges and corners.
Assume that a surface $\Sigma$ is piecewise smooth, i.e. $\Sigma$ is composed of finite set of smooth surface patches whose boundaries are finite smooth curves. Every point $q$ in the interior of a smooth surface patch has a tangent plane and two opposing normal directions. Non-smooth points of $\Sigma$ are points on the boundary of the surface patches where there is a discontinuity in the surface normals. Tangent planes and normal directions are not defined at non-smooth points.

An edge of $\Sigma$ is a curve of non-smooth points on the intersection of the boundaries of two smooth patches. A sharp edge of $\Sigma$ is an edge with dihedral angle (defined in 2.1) bounded away from 180 degrees. A corner of $\Sigma$ is a non-smooth point which is the intersection of three or more sharp edges or does not lie on any sharp edge. A sharp corner of $\Sigma$ is a corner with a solid angle bounded away from $2\pi$. Sharp edges and corners are henceforth referred as “sharp features”. Isosurfaces are represented by polygonal representations as piecewise linear smooth meshes composed of triangles or quadrilaterals. This underlying polygonal representation or mesh should model the sharp features of the surface $\Sigma$. A sharp edge of $\Sigma$ should be represented by a single, connected sequence of mesh edge with similar dihedral angle. A sharp corner of $\Sigma$ should be represented by a single isosurface vertex with similar solid angle. Similarly, mesh edges and vertices representing smooth, low curvature portions of $\Sigma$ should have dihedral angles near 180 degrees and solid angles near $2\pi$.

This work proposes a robust algorithm to reconstruct isosurfaces with sharp features. Specifically,

1. **RELIGRAD**: An algorithm which uses central difference formula to construct gradient approximations in the scalar field and then identifies which gradient approximations are reliable. RELIGRAD is described in Chapter 3.
2. **SHREC**: An algorithm for constructing isosurfaces with sharp features. SHREC computes isosurface vertex locations on sharp features based on reliable gradients (RELIGRAD), selects a well-spaced subset of these vertices and merges isosurface vertices in the neighborhood of each selected vertices. SHREC is described in Chapter 4.

### 1.3 Reconstruction from Industrial CT of Composite Materials.

Development of novel materials which integrate function orientation and efficiency are increasingly becoming a key objective of modern industry. Carbon fiber reinforced polymer (CFRP) allow quick integration of these rising demands and have become the material of choice for an ever growing number of applications. The rapidly increasing share of fiber reinforced polymers also generates a strong demand for non-destructive testing (NDT) techniques [59].

The most wide spread method for NDT on fiber reinforced polymers required by various standards is still ultrasonic testing, which provides a quick and cost-efficient but low-resolution and imprecise overview. However, more recently industrial 3D X-ray computed tomography (XCT) has been utilized for NDT applications for fiber reinforced polymers [44]. The following sections provide a brief introduction to Carbon Fiber Reinforced Polymers.

#### 1.3.1 Carbon Fiber Reinforced Polymer (CFRP)

Modern industry is increasingly demanding function orientation, integration, and efficiency of novel materials and components. The material of choice for a growing number of applications is carbon fiber reinforced polymer (CFRP), which allows an integration of these continuously rising demands and increasingly replaces conventional materials such
as aluminum or steel. CFRPs have desirable characteristics such as high specific stiffness, high specific strength and high corrosion resistance. Moreover, CFRP materials show these characteristics at considerably lower weight. At the same time, highly complex and integrated components, which were previously impossible to manufacture, may be produced from CFRPs. Primary structures and highly loaded components in aeronautics are one example.

**CFRP characteristics**

Typical carbon fiber reinforced polymer components, specifically CFRP laminates with endless carbon fibers consist of two main components: a matrix, which acts as a bonding element, and the reinforcements, which helps in achieving the desired characteristics. Various production processes are used to manufacture CFRP laminates. Most of these processes start with the reinforcement element, weaving individual carbon fiber bundles (yarn) into sheets of carbon fiber cloth in a predefined pattern. These sheets of woven carbon fiber cloth are also referred to as fabric and form the geometrical structure associated with the CFRP materials. Depending on the requirements of the final component, fabrics may be stacked in multiple layers in similar or different orientation. Both the alignment of fabrics and the weaving pattern of the individual carbon fiber bundles strongly influence the properties of the CFRP laminate. Resins are then integrated in the material system to fill the gaps in the fabric forming the matrix component. The main function of the matrix is to act as a bonding between the individual carbon fiber bundles. After curing, the production process of the CFRP laminate is finished.

**CFRP and Non-destructive testing (NDT)**

The rapid expansion in utilization of CFRPs, the complexity of both the material system and the final components has generated a strong demand towards non-destructive testing
(NDT) techniques for quality control [59]. The most widely incorporated method for NDT is ultrasonic testing (UT). While UT provides a quick and cost-efficient overview of the material, it generally lacks resolution and may generate arbitrary results, e.g., due to the geometry of the component. Industrial 3D X-ray computed tomography (XCT, also referred to as 3DXCT or cone beam XCT) is increasingly applied for non-destructive testing of fiber reinforced polymers [44]. In contrast to UT, XCT generates a highly detailed 3D volumetric representation of the scanned specimen. In cone beam XCT geometry, the specimen is placed on a rotary table between X-ray source and detector. The X-rays passing through the specimen get attenuated by the materials present. By transferring the X-rays in a scintillator layer into visible light, the detector records the corresponding 2D attenuation image (penetration image). The specimen is rotated and a 2D attenuation image is recorded at each angular step. The full series of attenuation images of a 360° rotation is utilized to generate a complete reconstruction of the data volume [35]. Cone beam XCT can reach voxel sizes below 500 nm generating high resolution volume data for comprehensive and detailed analyzes.

There is unfortunately a trade-off between viewport and image resolution. The magnification reached within an XCT scan is determined by the distances between source and specimen as well as source and detector. The magnification therefore directly influences both resolution and viewport: higher resolutions decrease the viewport but show more details, lower resolutions allow for larger viewports and thus larger portions of the specimen.

Chapter 6 describes a novel method for extraction and visualization of fiber bundles from X-Ray Computed Tomography scans of composite materials.
Chapter 2: Background

A scalar field is a mapping \( f : \mathbb{R}^3 \rightarrow \mathbb{R} \) which assigns a scalar value to each point in \( \mathbb{R}^3 \). An implicit surface is a set \( f^{-1}(\sigma) \) for some real \( \sigma \in \mathbb{R} \). An isosurface with isovalue \( \sigma \) is a polygonal approximation to the implicit surface \( f^{-1}(\sigma) \). Such polygonal representations are widely popular for modeling three dimensional geometry. They are created from varying sources including computer-aided design software or 3D scanners.

Representation of these geometric objects as volumetric data is also quite popular and provides many advantages in geometric processing applications such as boolean operations: intersection, union or to check if a point is inside or outside the surface.

Often the data starts of as volumetric data for example MRI data or CT scans of industrial or biological objects. Sometimes, geometric data is converted into an intermediate volumetric state, for examples Ju’s polygonal model repairing algorithm [40].

Volumetric data refers to the implicit definition of the surface, represented as a zero level set of a scalar valued function \( f : \mathbb{R}^3 \rightarrow \mathbb{R} \). Generally volumetric data is represented as a discrete approximation of the continuous scalar function sampled on a sufficiently dense spatial grid. The grids may or may not be axis aligned. They can be sampled uniformly or be adaptive.
Figure 2.1: (a) Non-smooth scalar field. (b) Discontinuity (red) in the gradients. (c) Iso-
contour (green) produced by the Marching Squares Algorithm. (d) Level set (blue) with
sharp corner.
2.1 Definitions

**Definition 2.1.1** (Scalar Field). A scalar field is a mapping \( f : \mathbb{R}^d \rightarrow \mathbb{R} \) which assigns a scalar value to each point in \( \mathbb{R}^d \).

**Definition 2.1.2** (Active). A grid vertex is positive (negative) if its scalar value is greater than or equal to (less than) the isovalue \( \sigma \).

A grid square or cube or any face of a square or cube is active if it has at least one positive and negative vertex.

**Definition 2.1.3** (Isosurface). Given a scalar field \( f : \mathbb{R}^d \rightarrow \mathbb{R} \) and a constant \( \sigma \in \mathbb{R} \), the set \( \{ x : f(x) = \sigma \} \) is called a level set of \( f \). An isosurface with isovalue \( \sigma \) is a surface which approximates the level set \( f^{-1}(\sigma) \), where the function \( f \) is represented as a finite set of sample points.

**Definition 2.1.4** (Ridge, patch). A ridge is a closed, connected subset of a smooth 1-manifold without boundary in \( \mathbb{R}^3 \). A patch is a 2-manifold that is closed, connected subset of smooth 2-manifold without boundary in \( \mathbb{R}^3 \).

**Definition 2.1.5** (Dihedral Angle). For each smooth point \( q \), let \( \eta_q \) be the normal at \( q \) which points in the direction of increasing \( f \). Let \( B_p^\varepsilon \) be the ball of radius \( \varepsilon \) centered at point \( p \) and let \( \partial \psi \) be the boundary of a surface patch \( \psi \). The *dihedral angle* between two smooth surface patches \( \psi_1 \) and \( \psi_2 \) at point \( p \in \partial \psi_1 \cap \partial \psi_2 \) is \( \lim_{\varepsilon \to 0} \inf \{ \angle(\eta_{q_1}, -\eta_{q_2}) : q_1 \in \psi_1 \cap B_p^\varepsilon \text{ and } q_2 \in \psi_2 \cap B_p^\varepsilon \} \). Note that the normal \( \eta_{q_2} \) is reversed. The *dihedral angle* of an edge \( \gamma \) of \( \Sigma \) between \( \psi_1 \) and \( \psi_2 \) is the maximum dihedral angle over all points \( p \in \partial \psi_1 \cap \partial \psi_2 \).
2.2 Related Work

2.2.1 Marching Cubes

The most popular algorithm to extract polygonal surface from volume data is the Marching Cubes Algorithm [49] by Lorensen and Cline. Marching Cubes takes as input an uniform grid whose vertices are a sampling of $f : \mathbb{R}^3 \to \mathbb{R}$, and extracts a triangular mesh approximation of $f^{-1}(\sigma)$.

The Marching Cubes Algorithm places all the isosurface vertices on grid edges. If the isosurface is approximating a level set $f^{-1}(\sigma)$ which has sharp edge or corner inside a grid cube, then isosurface vertices would have to be placed inside the grid cube to model the sharp edge or corner. Since the Marching Cubes Algorithm does not place isosurface vertices inside grid cubes, it will not reconstruct the sharp feature but instead create a beveled edge or corner. (See Figure 2.1c) Moreover, the beveling will depend upon the intersection of the level set with the grid edges, not upon any intrinsic properties of the level set. Figure 2.2 shows the results of Marching Cubes Algorithm on a cube edge with...
increasing resolution. With increasing resolution the beveling becomes less pronounced but as discussed above, the problem remains.

2.2.2 Dual Contouring

Instead of adding isosurface vertices on grid edges, dual contouring algorithms place isosurface vertices inside grid cubes intersected by the isosurface. Quadrilaterals connect isosurface vertices in the four adjacent grid cubes which share a common grid edge. Gibson [32, 33] gave the first dual contouring algorithm called Surface Nets, placing at most one isosurface vertex inside each grid cube. This is achieved by adding an isosurface vertex \( w_c \) for each grid cube \( c \) with at least one positive and at least one negative vertex. For each bipolar edge \( e = [p, q] \), let \( w_e \) be the point on the edge \((1 - \alpha)p + \alpha q\) where \( \alpha = (\sigma - s_p)/(s_q - s_p) \) (linear interpolation). Take the centroid, \( (w_{e_1}...w_{e_k}/k) \) of all the approximation points as the location of the isosurface vertex \( w_e \). The last step is triangulating the generated quadrilaterals.

2.2.3 Dual Marching Cubes

The Surface Net algorithm mentioned above places a single isosurface vertex in each active grid cube. Because Gibson’s algorithm placed only one isosurface vertex in each active cube, it produces many isosurface edges contained in four quadrilaterals. Thus the resulting surface is not a manifold. Nielson [55] described a dual contouring algorithm, Dual Marching Cubes, which adds between one and four isovertices inside the grid cube. For each interior bipolar grid edge \( e \), retrieve from the look up table the associated isosurface vertices and add a quadrilateral.
Ju and Udeshi in [42] show that the dual contouring algorithm may create meshes which self-intersect, i.e., mesh triangles may intersect on their interior. They present a modification of the dual contouring algorithm which fixes this problem.

If the isovalue does not equal the scalar value of any grid vertex, the Marching Cubes algorithm creates an isosurface which is a manifold. Unfortunately, Gibson’s dual contouring algorithm [32, 33] often generates non-manifold isosurfaces, irrespective of the isovalue. Nielson’s algorithm [55] represents each Marching Cube’s isosurface patch by a single dual isosurface vertex, avoiding many of the non-manifold cases. Wenger in his book [74] provides a Dual Marching Cubes algorithm which generates a guaranteed manifold.

Greß and Klein [34] and Zhang and Qian [78] give dual contouring algorithms which produce multiple isosurface vertices within a cube to represent the different isosurface patches. Because these algorithms split isosurface edges and their incident vertices whenever the edges pass through ambiguous facets, these algorithms produce isosurfaces which are manifolds.


By adding isosurface vertices inside cubes, dual contouring algorithms have the ability to represent sharp features in isosurfaces. However, the algorithms by Gibson and Nielson position the isosurface vertices by using linear interpolation to predict the intersection of the isosurface and grid cube edges and averaging the predicted locations. The resulting isosurface has smoothed edges similar to the isosurface created by Marching Cubes.

2.2.4 Sharp Edges and Corners

The Extended Marching Cubes Algorithm by Kobbelt, Botsch, Schwanecke and Seidel [46] constructs a parametric representation of an implicit surface with sharp features
from a grid of directed distances to that surface. Each grid vertex stores the directed distances in the x, y and z directions to the surface. Using linear interpolation, the algorithm by Kobbelt et al. computes a set of isosurface vertices on grid edges. It also computes surface normals at each of these isosurface vertices based on the directed distances at nearby grid vertices. Grid cubes with widely varying surface normals are identified as containing sharp features. If a grid cube does not have sharp features, an isosurface patch is retrieved from a lookup table as in Marching Cubes. If a grid cube has sharp features, then an additional isosurface vertex is added to the interior of the grid cube and connected to the isosurface vertices on the cube edges. The new isosurface vertex is positioned to minimize its least squares distance to tangent planes of the neighboring isosurface vertices. The final step applies edge flipping to connect vertices on sharp features in adjacent cubes.

Ju, Losasso, Schaefer and Warren [41, 64] gave an alternative approach using dual contouring to construct parametric representations of implicit surfaces with sharp features. Input to their algorithm is Hermite data instead of directed distances but the difference is minimal. Hermite data contains the exact intersection points of a surface with a regular grid and the exact normals. These values are easily computed from implicit surface representations.

The algorithm by Ju et al. retrieves the intersection points of grid edges and the implicit surface from the Hermite data along with the normals at each intersection point. The normals define tangent planes at each intersection point. As in [46], the algorithm positions the isosurface vertex within a grid cube to minimize the least squares distance to tangent planes.

Varadhan, Krishnan, Kim and Manocha [71] extended the dual contouring algorithm of Ju et al. by modeling multiple intersections of an isosurface and adding more than one isosurface vertex per grid cube. Input to their algorithm is directed distances to implicit surfaces.
Zhang, Hong and Kaufman [76] presented a multi-resolution dual contouring algorithm which adds more than one isosurface vertex per grid cube. When input is a scalar grid, the algorithm computes the vertex locations using averaging as in [32, 33, 55] and does not reconstruct the sharp features. When input is a directed distance field, the algorithm follows the approach in [41] and reconstructs the sharp features.

Algorithms by Ho et. al. [36] and Ashida and Badler [3] approximate the intersection of a surface and each grid cube boundary by a polygonal curve. They connect the curve to a single isosurface vertex in the interior of the cube. Sharp features are represented by appropriate positioning of the isosurface vertex and the curve vertices.

Schaefer and Warren [65] gave an innovative, novel approach to constructing isosurfaces with sharp features. From the original scalar grid, they constructed a dual grid whose vertices and edges were on sharp isosurface features. They applied Marching Cubes to the dual grid to extract the isosurface.

Except for the original dual contouring algorithm by Gibson and Nielson’s Dual Marching Cubes, all the dual contouring algorithms listed above support multi-resolution isosurface extraction.

Hermite data provides both the intersection of the isosurface and each grid edge and the isosurface normal at each intersection. This defines a tangent plane at each intersection. The dual contouring algorithms compute the point which minimizes the least squares distance to a set of tangent planes and use that point as an isosurface vertex location.

### 2.3 Computing Vertex Locations using Quadric Error Measure

Hermite data determines tangent planes to the isosurface, one at each intersection of the grid edge and the isosurface. For a grid cube $c$, the $k$ tangent planes on its edges give a set
of \( k \) equations

\[ Mx = b \]

where \( M \) is a \( k \times 3 \) matrix and \( x \) and \( b \) are column vectors of length 3 and \( k \) respectively. In general, this system is over-determined so the least squares solution is computed. The least squares solution is given by solving the normal equations,

\[ M^T Mx = M^T b. \]

The \( 3 \times 3 \) matrix \( A = M^T M \) and the column vector \( b' = M^T b \). The *Fundamental Error Quadric* is commonly computed as \( \sum_{i=1}^{k} (b_i - M_i x)^2 \), which in our case is \( b'b - 2x'b' + x'Ax \) or \( b'b - 2x'M'b + x'M'Mx \).

The singular valued decomposition (SVD) of \( A \) is \( A = U\Sigma V^T \) where

\[
\Sigma = \begin{pmatrix}
\sigma_1 & 0 & 0 \\
0 & \sigma_2 & 0 \\
0 & 0 & \sigma_3
\end{pmatrix}
\]

\( \sigma_1, \sigma_2, \) and \( \sigma_3 \) are the singular values of \( A \). If all three singular values of \( A \) are large, then \( c \) contains a sharp corner. If two singular values are large, then \( c \) contains a sharp edge. Otherwise, \( c \) does not contain a sharp feature.

Let

\[
\sigma'_i = \begin{cases} 
\sigma_i & \text{if } \sigma_i / \sigma^* > \varepsilon \\
0 & \text{otherwise}
\end{cases}
\]

where \( \sigma^* \) is the largest singular value and \( \varepsilon \) is a threshold parameter. Let \( A' = U\Sigma' V^T \) where \( \Sigma' \) is the diagonal matrix with diagonal entries \( \left( \sigma'_1, \sigma'_2, \sigma'_3 \right) \).
When $A$ has three large singular values, $A' = A$ and there is a single point $x$ such that $A'x = b$. When $A$ has two large singular values, $\{x : A'x = b\}$ is a line. When $A$ has one large singular value, $\{x : A'x = b\}$ is a plane.

Let

$$\sigma_i^+ = \begin{cases} 
1/\sigma_i' & \text{if } \sigma_i' \neq 0 \\
0 & \text{otherwise} 
\end{cases}$$

Let $\Sigma^+$ be the diagonal matrix with diagonal entries $(\sigma_1^+, \sigma_2^+, \sigma_3^+)$. As in [47], compute:

$$x = q_c + V \Sigma^+ U^T (b' - A q_c). \quad (2.1)$$

When $A$ has three large singular values, $x$ is the point solving $Ax = b$. When $A$ has two large singular values, $x$ is the point closest to $q_c$ on the line $A'x = b$. When $A$ has only one large singular value, $x$ is the closest to $q_c$ on the plane $A'x = b$.

Garland and Heckbert [30] represented the least squares distance to a set of tangent planes by a 4x4 matrix as described above, which they called the quadric error measure (QEM). The matrix size is independent of the number of tangent planes. Lindstrom [47] also used the quadric error measure to compute the point which minimizes the least squares distance to the represented set of tangent planes. When the tangent planes define an edge or a smooth portion of the isosurface, Lindstrom’s algorithm selects the point closest to the cube center. The feature sensitive dual contouring algorithms all use some variation of the quadric error measure to compute isosurface vertex locations.

Lindstrom uses the center of grid cube $c$ as the point $q_c$. Instead of setting $q_c$ in Equation 2.1 to be the center of cube $c$, Schaefer and Warren [64] propose setting $q_c$ to the centroid of the intersections of the edges of $c$ and the isosurface.

Instead of using $A = M^T M$, we could have used the singular value decomposition of $M$. Using $M$ is preferable for numerical stability, since the condition number of $A$ is the
square of the condition number of $M$. However, because we remove small eigenvalues, the numerical stability is not an issue. We use the singular value decomposition of the $3 \times 3$ matrix $A$ because it is simpler and faster to compute than the singular valued decomposition of the $k \times 3$ matrix $M$.

### 2.4 Problems with Vertex Locations

The Extended Marching Cubes algorithm by Kobbelt et. al. [46] and the dual contouring algorithm by Ju et. al. [41, 64] compute an isosurface vertex for each grid cube using the quadric error measure (QEM). When the surface is relatively smooth, the isosurface vertex lies within the grid cube. However, if the surface has a sharp feature, the vertex location computed using QEM may lie outside the grid cube. Should the isosurface vertex be placed at the location outside the grid cube?

The problem of isosurface vertex locations lying outside the grid cube was noted by Schaefer and Warren in [64]. Next, possible errors due to the vertex location are elaborated.

#### 2.4.1 Notches and Degeneracies

The algorithms in Kobbelt [46] and Ju [41] generate an isosurface vertex for each grid cube containing a positive vertex and a negative vertex. When the isosurface vertices are restricted to the generating cube, notches can appear along the sharp edges and corners can be cut off. When these vertices are allowed to extend into neighboring grid cubes, the isosurfaces triangle or quadrilaterals can become degenerate. The isosurface vertices can also be incorrectly ordered along sharp edges. The incorrect ordering creates folds and overlapping isosurface triangles in the isosurface mesh. The software Polymender by Ju [40] takes Hermite data as input and creates isosurfaces with degenerate and overlapping mesh triangles.
Figure 2.3: Reconstruction problems. (a) The orange cube generates the vertex in the magenta circle. Clamping the vertex to the orange cube creates a notch in the 1-dimensional feature. (b) Same mesh as (a) but vertex in the magenta circle is not clamped to the orange cube. (c) Different view of mesh, vertex and cube in (b). The vertex in the magenta circle is part of a degenerate triangles which lies on the 1D feature. Vertices in the yellow circles are also part of degenerate triangles which lie on the 1D feature. Note that the mesh is fully triangulated so that the apparent mesh quadrilaterals are actually mesh triangles adjacent to degenerate triangles. (d) Mesh folds produced by Extended Marching Cubes near a cube corner. The fold edges are at highlighted in red. The magenta triangle overlaps six other mesh triangles.
Specifically, consider a grid cube $c$ which generates a vertex location $p$ on a sharp edge or corner which does not intersect $c$. Let $c' \neq c$ be the grid cube containing $p$. If cube $c'$ has a bipolar edge, then it also generates an isosurface vertex $v'$. Placing $v$ in $c'$ may create degenerate mesh triangles or it may create overlapping triangles as the mesh folds back on itself. (See Figure 2.3b,c.) On the other hand, if $c'$ does not have any bipolar edge, then it does not generate an isosurface vertex. Clamping $v$ to lie inside $c$ will create ridges in the isosurface or cut off the corner. (See Figure 2.3a.)

One plausible approach might be to clamp $p$ to cube $c$ only if $c'$ contains a bipolar edge, this approach also produces numerous errors.

For cubes whose facets are parallel to the grid facets, grid cubes which intersect the surface corners or edges will almost always have bipolar edges. Thus clamping $v$ will not create ridges along the sharp edges and will not cut off corners. The same holds for cylinders or annuli whose edges whose central axes are parallel to grid edges. On the other hand, rotating these surfaces by any significant angle with respect to the grid creates many sharp corners or edges which intersect cubes with no bipolar edges.

### 2.4.2 Poor tolerance to approximation errors

Algorithms by Varadhan [71] and Zhang [76] don’t have the problems discussed above because they compute multiple intersections of a grid edge and the isosurface to represent thin or sharp features. Other algorithms [3, 34, 36] increase the grid resolution until such problems disappear. Either approach exacerbates a second problem. Specifically, the algorithms listed above rely upon precise calculations of both the intersection point and the normal. They have little tolerance for approximation errors in those values.

Those algorithms which compute multiple intersections or increase the grid resolutions create thin polygons near sharp features. Such polygons are sensitive to approximation errors.
2.4.3 Reliance on Hermite data

Algorithms proposed above construct isosurface representations to a surface $f^{-1}(\sigma)$ with sharp edges and corners from distance fields [34, 36, 46, 76], or from an explicit definition of $f$ [3, 41, 36, 71, 65]. All these algorithms rely on the ability to compute the exact intersection point of the isosurface and each grid edge and an exact surface normal at that intersection point. Ju et. al. [41] coined the term “Hermite” data to describe such inputs. These algorithms have weak tolerance to errors and variations in Hermite data. The problem becomes more acute when dealing with scalar data as surface normals are difficult to compute from a scalar field in the neighborhood of sharp features.

2.5 Gradient Computation

One method of approximation rather than interpolation from Hermite data would be to compute gradients from the input scalar information at the grid vertices. Formulas for improving the numerical accuracy of gradient computations from scalar data are given in [2, 37, 52]. These formulas assume the gradient vector field is smooth and do not work when there are discontinuities in the vector field. They also do not work when there is noise in the input scalar data.

2.5.1 Anisotropic diffusion

Anisotropic diffusion, is a technique by which the filtering of surface normals or field gradients changes based on local curvature. Gradients or normals in low curvature regions are moved to agree with their neighbors. Gradients or normals in high curvature regions are moved only slightly. Anisotropic diffusion for mesh smoothing is described in [5, 21, 68, 69]. Tasziden et. al. [70] used anisotropic diffusion to preserve features in isosurface reconstruction.
Features in papers on anisotropic diffusion are high curvature regions, not regions with normal or gradient discontinuities (infinite curvature.) Anisotropic diffusion applied to surfaces or gradient fields with discontinuities will filter noise from smooth regions, but it will not improve estimations at discontinuities or assist in identifying such discontinuities.

2.6 Point Cloud Data:

There is extensive work on construction of surfaces with sharp features from point cloud data, e.g. [4, 19, 23, 25, 27, 57, 61, 72] and many other articles. In [57] and [72], the final construction of the surface mesh is accomplished by Marching Cubes [49] or some other isosurface reconstruction algorithm. The algorithms described in the following chapters can be used by algorithms in those papers to construct a surface mesh that better represents sharp surface features than the Marching Cubes isosurface.

In [19, 25] and [61], the final construction of the surface mesh is accomplished by Voronoi based algorithms described in [19] and [25]. These algorithms could be applied to isosurface reconstruction from regular grid scalar data by constructing point clouds from the grid data and then applying the algorithms to the point clouds.

Point set data is inherently noisy, so much of the literature focuses on finding the true position of points on surfaces. Daniels et al. [23], Fleishman et al. [27] and Oztireli et al. [57] construct local surface patches fitted to local sets of points and project points onto these surface patches. Wang et al. [72] construct approximations of the tangent planes at the sample points and project points onto these tangent planes. Avron et al. [4] estimate surface normals at sample points using convex optimization, and then reposition the points, again using convex optimization.
The papers cited above focus on correct positioning of surface points in the presence of sharp features. The actual construction of the surface mesh is left to preexisting algorithms. For constructing the surface mesh, Oztireli et al. and Wang et al. use Marching Cubes, Daniels et al. use the advancing front algorithm from [67], and Avron et al. use the Ball Pivoting algorithm from [9]. (Wang uses Poisson Surface Reconstruction described in [45] but that algorithm uses a variation of Marching Cubes.) Neither Marching Cubes nor the Ball Pivoting algorithm is particularly well-adapted to constructing meshes with good representations of sharp features. By starting from the sharp features, Schreiner et al. claim that their advancing front method can do a good job of representing sharp features.

Two algorithms, one by Dey et al. [25] and one by Salman et al. [61], first identify and select points on sharp features and then reconstruct the surface mesh from the selected feature points and a subset of the smooth points. The algorithms differ in how they identify points on sharp features. Dey et al. use the graph Laplacian to identify points on sharp features while Salman et al. use an analysis of the shape of Voronoi cells.

Both algorithms use the “protecting ball” technique from [19] to construct the surface mesh from the weighted Delaunay triangulation of the selected feature points and a subset of the smooth points. The meshing algorithm ensures that surface vertices are well-spaced along feature curves and that surface vertices which are not on feature curves are suitably far from those curves. The method requires constructing and updating the weighted Delaunay triangulation of a set of points. Dey et al.’s algorithm works on non-manifold surfaces and handles three or more smooth pieces joined at a single curve.

2.6.1 Delaunay refinement for piecewise smooth complexes

Cheng, Dey and Ramos [19] described an algorithm to mesh piecewise smooth complexes using weighted Delaunay triangulations. They choose points along the boundaries of each smooth piece and construct protecting balls around each such point by assigning
a weight to each point. They then choose sample points from the smooth portion of the surface outside of the protecting balls and return the weighted Delaunay triangulation of the points.

Delaunay refinement algorithms generate high-quality meshes by inserting vertices into a Delaunay triangulation. The algorithm as described by Cheng, Dey and Ramos [19] is a Delaunay refinement algorithm. It is divided into two stages, a protection stage and a refinement stage. Both stages maintain a weighted Delaunay triangulation of a weighted point set that includes the vertices in the piecewise smooth complex. Each weighted mesh vertex \( v[\omega_v] \), is represented by a ball \( B_v = B(v, \sqrt{\omega_v}) \), which are referred to as protecting balls. The protection stage covers the vertices and ridges (def 2.1) with protecting balls. Refer to the Cheng, Dey, Shewchuk’s book [20] for extensive details.

Salman et al. [61] and Dey et al. [25] use the protecting balls from [19] to reconstruct surfaces with sharp features from point cloud data. Both papers identify sharp features and protect them with balls. They then reconstruct the surface using the protecting balls. There are advantages in constructing sharp features directly from scalar data without (the extra step of) converting it to point cloud data. First, converting scalar data to point cloud data ignores the grid structure of the scalar data. This grid structure can be employed both for constructing sharp features and for meshing those features. Generally, grid based methods are faster than their point-cloud counterparts. Second, point cloud data is extremely noisy so the point cloud reconstruction algorithms average over large neighborhoods. It is extremely difficult to predict the results of the algorithms or to guarantee that the algorithms are not ignoring features in the data. In contrast, our algorithm runs over a small, local neighborhood, so that all features of the data (for better or for worse) are represented in the output.
2.7 Visual Analysis and Modeling of Fiber Reinforced Composites

The approaches presented in visualization and analysis of composites mainly focuses on individual objects such as fiber extraction from high resolution data where the individual fibers are clearly discernible. Fritz et al. [29] proposed interactive workflows for non-destructive testing practitioners to explore and quantify steel fibers in reinforced sprayed concrete.

Salaberger et al. [60] introduced a pipeline to extract and characterize individual fibers of fiber reinforced composites. They encode the extracted fibers as color-coded line segments in 3D and visually identifying fibers with similar orientations. Recently, Weissenböck et al. [73] presented a system for interactive exploration and analysis of fibers in fiber reinforced polymers. They use the visualization paradigms of a scatter plot matrix and parallel coordinates to select fibers according to their characteristics. The defined fiber classes can be managed in a list and are displayed with a 3D renderer.

Lomov et al. [48] discusses the problems and current available solutions in geometric modeling of three dimensional composites. Modeling of the composites, first starts with establishing the topology of the structure, which translates to answering if a particular bundle is in contact with another at a particular position. The second step builds the geometry of the model, answering queries relating to placement of bundles in space, their orientations and dimensions.

Visual analysis of Carbon fiber reinforced composites is a relatively new domain which many ideas borrow heavily from existing fields. One of these existing fields with similar problems is Diffusion Magnetic Resonance Imaging.
2.7.1 Diffusion Tensor Imaging

Diffusion Magnetic Resonance Imaging (dMRI, also referred to as Diffusion Tensor Imaging (DTI)) is a magnetic resonance imaging technique which provides three-dimensional information about the structures in cerebral white matter based on diffusion of water molecules. In particular, diffusion tensor imaging may be used to map and characterize the three-dimensional diffusion of water as a function of spatial location.

DTI has gained popularity in medical diagnosis within recent years; its main clinical application is found in the study and treatment of neurological disorders. DTI may reveal abnormalities in white matter fiber structure and is used for visualizing the organization of fibers in the human brain and brain connectivity. A variety of algorithms have been proposed for generating fiber-tract trajectories. In general, these reconstructions of fiber trajectories are clustered into bundles which are expected to be related anatomically or spatially. The related work on DTI is broadly divided into two parts of immediate relevance to our proposed solution: fiber tracking and fiber clustering.

2.7.2 Fiber Tracking in Diffusion Tensor Imaging

A basic assumption in DTI analysis is that the principal eigenvector of the diffusion tensor is parallel to the underlying dominant fiber direction in each image voxel [6, 53, 54]. The principal diffusion direction at each discrete location is interpolated to form a continuous velocity field. Continuous tracts are generated by propagating virtual particles along the principal diffusion directions until they reach some termination criterion. This is usually done by solving a Runge-Kutta integration (typically second or fourth order). Because decisions are made locally, these methods perform poorly in noisy regions and often generate small fibers. Basser et al. [6, 7] proposed that white matter tracts could be represented as 3D curves in space. They showed that numerical methods could be used to follow fibers
and fiber bundles and to generate tracts in human brain data. Mori et al. [53, 54] divided reconstruction techniques into line-propagation or energy minimization techniques. In line propagation approaches, trajectories are computed based on local neighborhoods and in energy minimization approaches the most favorable trajectory connecting two given endpoints is selected.

### 2.7.3 Fiber Clustering in Diffusion Tensor Imaging

In DTI, similarity measures such as proximity between fibers are used to group fiber tracts into bundles. Extensive research has been done on automatic DTI fiber clustering methods [16, 17, 22, 75, 77]. These approaches build on the assumption that proximity measures that compare DTI fiber trajectories can also represent anatomical relationships. Clustering requires choosing a suitable proximity measure and a method for grouping “close” fibers.

Pairwise proximity measures include endpoint distances [17] and mean of the closest distances between points on two fibers [22]. Zhang et al. [77] introduced a thresholded version of the closest distances mean, so that fibers which are close for certain distance and then diverge, are clustered separately. Brun et al. [16] use normalized cuts along with a pairwise computed distance measure using 9D fiber shape descriptors.

The choice of the clustering algorithm can be broadly divided into those approaches using hierarchical clustering [51, 77] and those using spectral clustering [16, 39, 56]. Brun et al. [17] described how a spectral non-linear dimensionality reduction technique, Laplacian eigenmaps (Belkin and Niyogi [8]), can be applied to the problem of organizing fiber tracts data. The key notion of the Laplacian eigenmaps algorithm is to represent the underlying data as a graph. Each node represents a data point and the edges connect neighboring data points. An eigenvalue problem is solved to represent the data in a lower dimensional
space while preserving the local graph structure. In the case of fiber bundles, the individual points are fiber tracts. In the ideal case fiber tracts belonging to the same bundle must remain “close” to each other in the lower dimensional space. Westin et al. [75] also used spectral clustering on a Hausdorff distance measure defined as the maximum of point wise minimum distances between two fibers. Jonasson et al. [39] ran K-means clustering on the eigenvectors of the affinity matrix defined as the co-occurrence of fibers in the data. Additionally, the agglomerative hierarchical clustering method [26] has gained popularity for proximity based fiber segregation (Zhang et al. [77], Corouge et al. [22]). An agglomerative hierarchical clustering method starts with each data point/fiber in an individual cluster. At each stage of the algorithm the two most similar clusters are merged based on some criterion. The two basic cluster similarity measures are single-link and complete-link. With the single link, the distance between the clusters is the distance between the closest pair of items. Moberts et al. [51] implemented several distance measures in their evaluation of fiber clustering and concluded that clustering methods are generally accurate in capturing fiber bundles.

There are a number of difficulties in hierarchical clustering. First, computing all pairs’ distances for tracts to generate the distance matrix is time consuming [31]. Second, a “correct” distance measure to compare tracts must be chosen. Third, hierarchical clustering is best suited for similar length fibers. Spectral methods are also hindered by long matrix computations.

2.7.4 Second Order Local Structure

A common approach in analyzing the local behavior of an image is to consider its Taylor series expansion in the neighborhood of a point $x_0$.

$$L(x_0 + \delta x_0, s) \approx L(x_0, s) + \delta x_0^T \nabla_{x_0} s + \delta x_0^T H_{x_0} \delta x_0$$  \hspace{1cm} (2.2)
The expansion approximates the structure of the image up to second order. $\nabla_{0,s}$ and $H_{0,s}$ are the gradient vector and Hessian matrix computed at $x_0$ and scale $s$.

The Hessian matrix captures the local second-order structures inherent in the intensity variations around each location $x_0$. The eigen decomposition of the Hessian matrix gives the eigen vectors which represent the local curvature of the image.

The eigenvector corresponding to the smallest eigenvalue gives the direction along which the curvature is smallest. This also coincides with the direction of tubular structures present in the image.

Analyzing curvilinear structures in volumetric images has been utilized for a variety of purposes including center line extraction [15] and vascular image enhancement as proposed by Frangi et al. [28] and Sato et al. [62].

Frangi et al. [28] introduced a method based on studying the eigenvalues of the Hessian matrix specifically for the purposes of developing vessel enhancement filters.
Chapter 3: RELIGRAD: Computing Reliable Gradients from Scalar Data

3.1 Overview

X-ray computed tomography (CT) scanners produce regular grids of scalar values representing material densities of scanned objects. These scalar values can be modeled as samples of some scalar field \( f : \mathbb{R}^3 \to \mathbb{R} \). Object boundaries can be visualized by direct volume rendering or by visualizing isosurfaces (mesh representations of \( f^{-1}(\sigma) \)) representing the object boundaries.

Both approaches have difficulties in representing sharp edges and corners in the object boundaries. Sharp edges and corners are best represented as discontinuities in the gradient field of \( f \). However, standard direct volume rendering and isosurface construction algorithms implicitly assume some continuity in the gradient field of \( f \).

In this chapter, we will describe a fast, local algorithm for reliably reconstructing the gradient field of \( f \) and constructing points on sharp edges and corners of an isosurface.

The points can be rendered in conjunction with isosurface visualizations to highlight sharp features or they can be joined to form a skeleton representation of the sharp feature. They can also be used as input to isosurface or surface meshing algorithms which are designed to handle surfaces with sharp features.
If the level set $f^{-1}(\sigma)$ of a scalar field $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ has sharp features, then there are discontinuities in the gradients of $f$ at the sharp features. Constructing the gradients near those discontinuities is difficult. As noted in Chapter 2, formulas for approximating gradients such as the central difference formula [18] or higher order approximations [2, 37, 52] assume the gradient is continuous at the given location. Anisotropic diffusion [5, 21, 68, 69, 70] removes noise in low curvature regions of the scalar field without affecting high curvature regions, but it does not produce correct gradients near gradient discontinuities.

Instead of attempting to produce correct gradients at all grid vertices, our algorithm identifies correct gradients and uses only those gradients to predict locations of isosurface vertices. We give an algorithm for identifying correct gradients based on their agreement with neighboring gradients. The algorithm produces enough correct gradients in the neighborhood of sharp features to generate points on those sharp features.

Our research contribution in this chapter is; *A local algorithm for constructing reliable gradients from scalar data in the presence of gradient discontinuities.*

The basic steps of our algorithm are given in Figure 3.1. The first two steps produce a set of reliable gradients. The algorithm then selects a set of reliable gradients around each
cube, computes a set of isosurface tangent planes from those gradients, and finds a point at
the “intersection” of those tangent planes. It simultaneously identifies whether that point
lies on a sharp edge or corner of the isosurface or on a smooth region of the isosurface. In
Chapter 4 we propose and algorithm SHREC, which uses the reliable gradients to generate
isosurface vertices on sharp features.

We note that the algorithm for constructing the reliable gradients and the sample points
is completely local, and thus fast and easily parallelizable.

3.2 Determining Correct Gradients

Smooth Scalar field And Noise: We assume that the scalar values \( s_v \) represent the values
at the grid vertices \( v \) of a continuous, piecewise smooth scalar field \( f \). The difference
between \( s_v \) and \( f(v) \) is the “noise” in the data.

A scalar field \( f : \mathbb{R}^3 \rightarrow \mathbb{R} \) is piecewise smooth if \( \mathbb{R}^3 \) can be partitioned into a finite set
of piecewise smooth regions, \( A_1, A_2, \ldots, A_k \) such that \( f \) has derivatives of all orders on each
region \( A_i \). Because \( f \) is only piecewise smooth, the gradient field of \( f \) may have discontinuities. Such discontinuities occur only on the boundaries, \( \partial A_i \), of the \( A_i \). Figure 3.2
contains an example of a continuous piecewise smooth field, consisting of three piecewise
smooth regions separated by two cones (a “double cone”). The isosurface for this field is
the boundary of a cylinder.

The gradient at a point \( p \) is the vector \( (\partial f/\partial x, \partial f/\partial y, \partial f/\partial z) \). Gradients are computed
at some, but not all, of the grid vertices. In particular, gradients are not computed at grid
vertices on or adjacent to points where the scalar field is not smooth.
3.2.1 Definitions

We define the neighborhood of a grid vertex $v$. Let set $N_1(v)$ be $v$ and the six grid vertices which share a grid edge with $v$. Recursively define set $N_k(v)$ as:

$$N_k(v) = \{ v' : v' \in N_1(v'') \text{ for some } v'' \in N_{k-1}(v) \}.$$ 

We sometimes use $N(v)$ as an abbreviation for $N_1(v)$.

A **collinear sequence of adjacent grid vertices** is a sequence $(v_1, v_2, \ldots, v_k)$ of distinct grid vertices such that $v_i \in N(v_{i-1})$ for $i = 2, \ldots, k$ and all the $v_i$ are collinear.

An **interior grid vertex** of a region $A$ is a grid vertex $v$ such that $N(v)$ is a subset of $A$. Set $\mathcal{I}_V(A)$ is the set of all interior grid vertices of $A$.

3.2.2 Central Difference Formula

To compute a gradient in a piecewise smooth scalar field, we need all the scalar values used in the computation to be from a single smooth portion of the scalar field. Thus,
we want to use a small basis for our gradient computation and not extend our gradient computation over many grid vertices. We use the central difference formula

\[
\frac{\partial f}{\partial (x_d)} \approx \frac{(f(x + u_d) - f(x - u_d))}{(2|u_d|)},
\]

(3.1)

where \(x\) is the location of a grid vertex and \(u_d\) is the vector to the adjacent vertex in direction \(d\). Figure 3.3 shows the result of computing gradient using the central difference formula. If spacing between grid vertices is the same in all directions, then the grid can be rescaled so that \(u_d\) is a unit vector in all directions. However, CT scans often have non-uniform spacing, with the \(z\) or slice direction different from the \(x\) and \(y\) directions. In that case, \(u_x\) and \(u_y\) will have different magnitudes from \(u_z\).

Let \(g_v\) be the gradient at a vertex \(v\) and let \(\tilde{g}_v\) be the gradient approximation produced by the central difference formula. There are three types of errors in the approximation of \(g_v\) by \(\tilde{g}_v\).
Noise: First, there are errors caused by noise in the data, i.e. the difference between the scalar value $s_v$ and its "true" value $f(v)$.

Central Difference Approximations: Second, there are errors caused by using the central difference formula as an approximation to the gradient. Such errors occur even if we used the exact values $f(v)$ and if the field was smooth everywhere.

Discontinuity errors: Finally, if $v$ and one of its neighbors $v' \in N(v)$ lie in different smooth regions, then there may be a discontinuity in the gradient along edge $(v, v')$. This discontinuity will also contribute to errors in $\tilde{g}_v$. (Numerical error is a fourth contributor to errors in $\tilde{g}_v$, but it is insignificant compared to the errors caused by noise in the data.)

Replacing the central difference formula by an equation which relies on more vertices will reduce the first two sources of error but increase the effect of gradient discontinuities on the gradient approximation. Anisotropic filtering can be used to decrease noise in the scalar data without affecting the discontinuities, but it will not reduce the error caused by gradient discontinuity. Anisotropic filtering can also be used directly on the gradients. Again, it will improve gradients in the smooth regions, but it won’t correct major errors caused by discontinuities. (It might correct minor ones.)

3.2.3 Reliable Gradients

Comparing with neighbors: For each vertex $v$, let $\tilde{n}_v = \tilde{g}_v / |\tilde{g}_v|$ be the unit vector in the direction of the central difference gradient $\tilde{g}_v$. We can try to determine if the gradient direction $\tilde{n}_v$ at vertex $v$ is reliable by comparing it with the gradient directions $\tilde{n}_{v'}$ at all the neighboring vertices $v' \in N(v)$. If $\angle(\tilde{n}_v, \tilde{n}_{v'})$ is less than some constant $\alpha$ for all vertices $v' \in N(v)$, then we can mark the gradient at $v$ as reliable.
Determining reliable gradients from $\angle(\tilde{n}_v, \tilde{n}_{v'})$ will work for flat regions but will fail if the scalar field has any significant curvature. If $\alpha$ is set to a small value, then the algorithm will fail to detect correct gradients in curved regions. On the other hand, if $\alpha$ is set to a large value, then the algorithm will mark incorrect gradients as correct.

**Solution Approach:** Instead of comparing $\tilde{n}_v$ to neighboring vertices, we use pairs of vertices to predict the gradient direction at $v$ and compare $\tilde{n}_v$ to this predicted direction.

Let $n_v = \frac{g_v}{|g_v|}$ be the unit vector pointing in the direction of the true gradient $g_v$. Assume $(v, v', v'')$ is a collinear sequence of adjacent vertices. Vectors $n_v$ and $n_{v'}$ lie in a plane $h$. If the gradient changes at a constant rate along line segment $(v, v'')$, then $n_{v''}$ also lies in plane $h$ and $\angle(n_{v''}, n_{v'})$ equals $\angle(n_v, n_{v'})$.

Let $n$ and $n'$ be unit vectors in $\mathbb{R}^3$ lying in plane $h$. Let $\phi(n, n')$ be the unit vector in $h$ other than $n$ whose angle with $n'$ is $\angle(n, n')$. We say that the $\phi(n, n')$ is the vector predicted by $n$ and $n'$. (See Figure 3.4(a).) More precisely, define Orth$(n, n')$ as $n - (n \cdot n')n'$, the
component of \( n \) orthogonal to \( n \). Define \( \phi(n, n') \) as:

\[
\phi(n, n') = n - 2 \times \text{Orth}(n, n') = n - 2(n - (n \cdot n')n') = 2(n \cdot n')n' - n.
\]

As defined above, unit vector \( \tilde{n}_v = \tilde{g}_v/|\tilde{g}_v| \) points in the direction of the central difference gradient. We determine reliable gradients by testing \( \angle(\phi(\tilde{n}_v', \tilde{n}_v), \tilde{n}_v) \) against a constant \( \alpha \).

\begin{verbatim}
Input : Vertex v, Angle bound \( \alpha \).
1 foreach grid vertex \( v' \in N(v) \) do
2   Let \( v'' \in N(v') \) be the vertex such that \( (v, v', v'') \) is a collinear sequence of adjacent vertices;
3   if \( (\angle(\phi(\tilde{n}_v', \tilde{n}_v), \tilde{n}_v) > \alpha) \) then return (false);
4 end
5 return (true)
\end{verbatim}

Algorithm 1.

Assume \( N_3(v) \) is a subset of a smooth region \( A_i \) so that \( v, v', v'' \in \mathcal{I}_v(A_i) \) for all the neighbors \( v' \) of \( v \). If all the second order partial derivatives of function \( f \) in \( A_i \) are constant, then the gradients change at a constant rate along any direction and \( \phi(n_v', n_v') \) equals \( n_v \). Moreover, if all the second order partial derivatives are constant and there is no noise (\( s_v = f(v) \) for all \( v \)), then the central difference gradient \( \tilde{g}_v \) equals the exact gradient \( g_v \) (up to numerical error.)
In the more general case, the second order partial derivatives are not constant and there is noise in the data. To analyze this case, define:

\[ \mu = \max \{ \angle(n_v, \tilde{n}_v) : v \in \mathcal{I}_V(A_i) \text{ for some } A_i \}. \]

\[ \Lambda(n_v, n_{v'}, n_{v''}) = \angle(\phi(n_v, n_{v'}), n_{v''}). \]

\[ \lambda = \max \{ \Lambda(n_v, n_{v'}, n_{v''}) : v, v', v'' \in \mathcal{A}_i \text{ for some } A_i \}. \]

Value \( \mu \) is a bound on the angle between the approximate and exact gradient directions in the smooth regions of the field. \( \Lambda(n_v, n_{v'}, n_{v''}) \) is the difference between the prediction \( \phi(n_v, n_{v'}) \) and \( n_{v''} \). This difference is caused by changes in the curvature of \( f \). Value \( \lambda \) is a bound on this difference over vertices in the interiors of the \( A_i \). Note that Algorithm 1 computes \( \Lambda(\tilde{n}_{v''}, \tilde{n}_{v'}, \tilde{n}_v) \), not \( \Lambda(n_v, n_{v'}, n_{v''}) \).

The following proposition bounds \( \Lambda(\tilde{n}_{v''}, \tilde{n}_{v'}, \tilde{n}_v) \) and \( \angle(n_{v''}, \tilde{n}_{v''}) \). The proofs for the same are in [14].

**Proposition 1.** Let \( (v, v', v'') \) be a collinear sequence of adjacent vertices contained in \( A_i \) for some smooth region \( A_i \).

1. If \( v, v', v'' \in \mathcal{I}_V(A_i) \), then

\[ \Lambda(\tilde{n}_v, \tilde{n}_{v'}, \tilde{n}_{v''}) \leq \Lambda(n_v, n_{v'}, n_{v''}) + 4\mu \leq \lambda + 4\mu. \]

2. If \( v, v' \in \mathcal{I}_V(A_i) \), then \( \angle(n_{v''}, \tilde{n}_{v''}) \leq \Lambda(\tilde{n}_v, \tilde{n}_{v'}, \tilde{n}_{v''}) + 3\mu + \lambda. \)

Assume that parameter \( \alpha \) in Algorithm 1 is at least \( \lambda + 4\mu \). By the first inequality, if \( N_3(v) \subseteq A_i \) for some \( A_i \), then \( \Lambda(\tilde{n}_{v''}, \tilde{n}_{v'}, \tilde{n}_v) \leq \lambda + 4\mu \leq \alpha \) for all neighbors \( v' \) of \( v \). Thus Algorithm 1 returns true.

On the other hand, assume that Algorithm 1 returns true and that \( N_3(v) \) intersects at most two regions. Let \( A_i \) be the region containing \( v \). Under the assumption that the boundary between these two regions is planar, some \( v' \in N(v) \) is in \( \mathcal{I}_V(A_i) \). If \( (v, v', v'') \) is a
collinear sequence of adjacent vertices, then \( v'' \) is also in \( \mathcal{I}_V(A_i) \). By the second inequality, 
\[
\angle(n_v, \bar{n}_v) \leq \Lambda(\bar{n}_{v'}, \bar{n}_v, \bar{n}_v) + 3\mu + \lambda \leq \alpha + 3\mu + \lambda.
\]
Thus, if Algorithm 1 returns true, then the angle between the approximate gradient direction \( \bar{n}_v \) and the true gradient direction \( n_v \) is bounded by \( \alpha + 3\mu + \lambda \).

So far we’ve made the assumption that the surface between two smooth regions is planar. If we drop that assumption, then it is no longer true that for every vertex \( v \in A_i \) some vertex in \( N(v) \) is in \( \mathcal{I}_V(A_i) \). For instance, in the 2D example in Figure 3.5, no vertex in \( N(v) \) lies in \( \mathcal{I}_V(A_1) \). Without this property, we can no longer guarantee a bound on \( \angle(n_v, \bar{n}_v) \) when Algorithm 1 returns true.

To handle curved boundaries of the \( A_i \), we must modify our algorithm to use vertices at edge distance 3 from \( v \). We define \( \phi_k(n_v, n_{v'}) \) as the normal direction predicted by \( n_v \) and

![Figure 3.5](image-url)
At the vertex which is edge distance $k$ from $n_v'$.

\[
\phi_0(n,n') = n',
\]
\[
\phi_1(n,n') = \phi(n,n') = 2(n \cdot n')n' - n,
\]
\[
\phi_k(n,n') = \phi(\phi_{k-2}(n,n'),\phi_{k-1}(n,n')),
\]
\[
\Lambda_k(n,n',n'') = \angle(\phi_k(n,n'),n'').
\]

We replace $\phi$ in Algorithm 1 with $\phi_2$.

**Input**: Vertex $v$, Angle bound $\alpha$.

1. **foreach** grid vertex $v' \in N(v)$ do
2. Let $v'',v''' \in N_3(v)$ be the vertices such that $(v,v',v'',v''')$ is a collinear sequence of adjacent vertices;  
3. **if** $(\angle(\phi(\tilde{n}_v,v'),\tilde{n}_v)) > \alpha$ **then return** (false);  
4. **if** $(\angle(\phi_2(\tilde{n}_v,v'),\tilde{n}_v)) > \alpha$ **then return** (false);  
5. **end**  
6. return (true)

**Algorithm 2.**

Let $A_i$ be the smooth region containing vertex $v$. Let $\mathcal{X} = \cup_{A_j} \partial A_j$ be the union of all the boundaries of smooth regions $A_j$. If there is a sufficiently large ball containing $v$ and not intersecting $\mathcal{X}$, then $v'',v''' \in \mathcal{I}(A_i)$ for some collinear sequence $(v,v',v'',v''')$.

**Proposition 2.** Let $\Gamma$ be a regular grid whose edges all have the same length $L$. If some ball $B$ of radius $(5/2)\sqrt{3}L$ contains grid vertex $v \in A_i$ and does not intersect $\mathcal{X}$, then there is a collinear sequence $(v,v',v'',v''')$ of adjacent grid vertices such that $v'' \in \mathcal{I}(A_i)$ and $v''' \in \mathcal{I}(A_i)$.

The following proposition bounds $\Lambda_2(\tilde{n}_v,v',\tilde{n}_v)$ and $\angle(n,v').
Proposition 3. Let \((v, v', v'', v''')\) be a collinear sequence of adjacent vertices contained in \(A_i\) for some smooth region \(A_i\).

1. If \(v, v', v'', v''' \in I_V(A_i)\), then
   \[
   \Lambda_2(\tilde{n}_v, \tilde{n}_{v'}, \tilde{n}_{v''}) \leq \Lambda_2(n_v, n_{v'}, n_{v''}) + 6\mu \leq 3\lambda + 6\mu.
   \]

2. If \(v, v' \in I_V(A_i)\), then
   \[
   \angle(n_{v''}, \tilde{n}_{v''}) \leq \Lambda(\tilde{n}_v, \tilde{n}_{v'}, \tilde{n}_{v''}) + 3\lambda + 5\mu.
   \]

As in the discussion of Algorithm 1, Property 3 can be used to show that if \(N_4(v) \subset A_i\) for some \(A_i\), then Algorithm 2 returns true. On the other hand, assume Algorithm 2 returns true. By Proposition 2, there is a collinear sequence of grid vertices \((v, v', v'', v''')\) such that \(v'', v''' \in I_V(A_i)\). By Proposition 3, the angle between the approximate gradient direction \(\tilde{n}_v\) and the true gradient direction \(n_v\) is bounded by \(\alpha + 3\lambda + 5\mu\).

3.2.4 CT Data

Algorithm 2 has two problems when applied to CT data. First, in CT data scalar values near gradient discontinuities are very unreliable. At such vertices, the angle between the true gradient \(n_v\) and the estimated gradient direction \(\tilde{n}_v\) is also very unreliable. Thus, we cannot assume that angle is bounded by a constant \(\mu\) on vertices near gradient discontinuities. Second, gradient magnitudes drop off quickly away from the surface boundaries and gradient directions are quickly meaningless. This is particularly true if the gradient direction is the z-direction and the CT data is reconstructed in planar x-y slices. We address each of these problems.

As stated above, the first problem with CT data is that scalar values near gradient discontinuities are very unreliable. At such vertices, the angle between \(n_v\) and \(\tilde{n}_v\) can be much greater than in the rest of the data set. Unfortunately, gradients at vertices near gradient discontinuities are exactly the gradients which we need to generate sharp features.
Gradients generated from scalar data rely upon the accuracy of the scalar data. CT scanners do not measure scalar values directly at each grid vertex. Instead, they measure the intensity of rays passing through the scanned object. The resulting measurements are called projection data. The projection data is transformed into scalar data by using a Radon or similar transformation of by solving a large set of linear equations. The resolution of the scalar data is usually set to equal the resolution of the projection data.

The process of determining scalar values at grid vertices is provably reliable in regions where field gradients vary slowly and continuously. However, it is highly unreliable near discontinuities in the field gradients. The result is that scalar values at grid vertices adjacent to gradient discontinuities are highly unreliable and the angle between the true gradient and the estimated gradient can be much larger than in the rest of the data.

Fortunately, the scalar errors drop off quickly away from the gradient discontinuities. We found that scalar values at a grid vertex \( v \) was reliable within an acceptable tolerance as long as no edge incident on \( v \) intersected a gradient discontinuity. Equivalently, the scalar value at \( v \in A_i \) was reliable as long as \( N(v) \) was a subset of \( A_i \). Under this assumption, if \( N_2(v) \) is contained in some smooth region \( A_i \), then the scalar values at vertices in \( N(v) \) are close to their true values. This implies that the angle between \( n_v \) and \( \tilde{n}_v \) is small.

Let \( A_i \) be the smooth region containing grid vertex \( v \). Assume that the boundary of \( A_i \) is flat around \( v \). We can no longer assume that the angle between \( n_{v'} \) and \( \tilde{n}_{v'} \) is small for some \( v' \in N(v) \). However, there is a collinear sequence \( (v, v', v'', v''') \) of adjacent vertices such that \( N_2(v'') \) and \( N_2(v''') \) are in \( A_i \). Thus, \( \bar{n}_{v''} \) and \( \bar{n}_{v'''} \) are good estimates of \( n_{v''} \) and \( n_{v'''} \), respectively. By comparing \( \bar{n}_v \) with the direction \( \phi_2(\bar{n}_{v''}, \bar{n}_{v'''} \) predicted by \( \bar{n}_{v''} \) and \( \bar{n}_{v'''} \), we can determine if \( \bar{n}_v \) is a reliable gradient. Note that this exactly what Algorithm 2 does.

The argument above assumes the boundary of \( A_i \) is flat around \( v \). Without this assumption, we can no longer conclude that \( N_2(v'') \) and \( N_2(v''') \) are in \( A_i \) for some collinear sequence \( (v, v', v'', v''') \).
Assume that all grid edges have the same length $L$. Replace the assumption that a scalar value at $v \in A_i$ is reliable if $N(v) \subseteq A_i$ by the assumption that a scalar value at $v$ is reliable if a ball $B_{0.5L}(v)$ of radius $0.5L$ around $v$ is contained in $A_i$. Under this assumption, if $B_{1.5L}(v)$ is contained in some smooth region $A_i$, then the scalar values at the vertices $N(v)$ are all close to their true values. This implies that the angle between $n_v$ and $\tilde{n}_v$ is small.

If $B'$ is a sufficiently large ball containing $v$ and $A_i$ contains $B'$, then there is a collinear sequence $(v, v', v'', v''')$ of adjacent grid vertices such that $A_i$ contains $B_{1.5}(v'')$ and $B_{1.5}(v''')$. Thus, $\tilde{n}_{v''}$ and $\tilde{n}_{v'''}$ are reliable gradients. Algorithm 2 determines if $\tilde{n}_v$ is reliable from $\tilde{n}_{v''}$ and $\tilde{n}_{v'''}$.

The second problem with CT data is that gradient magnitudes drop off quickly away from the surface boundaries. To address this problem, we divide the vertex neighbors of $v$ into two sets. Define the the tangent neighbor set and the orthogonal neighbor set of $v$ as:

$$N^T(v) = \{v' \in N(v) : 20^\circ \leq \angle(\tilde{n}_v, (v' - v)) \leq 160^\circ.\}$$

$$N^O(v) = N(v) - N^T_v$$

$$= \{v' \in N(v) : \angle(\tilde{n}_v, (v' - v)) < 20^\circ \text{ or } \angle(\tilde{n}_v, (v - v')) < 20^\circ.\}$$

$(v' - v)$ is the vector from $v$ to $v'$. The orthogonal neighbor set may be empty.

Assuming that $\tilde{n}_v$ is relatively close to $n_v$, vertices in $N^T(v)$ are near the tangent plane at $v$ and close to the isosurface through $v$. We handle gradient directions at those vertices as in Algorithm 2. Vertices in $N^O(v)$ are (relatively) far from the tangent plane and from the isosurface through $v$. Fortunately, the gradient directions at the vertices in $N^O(v)$ are much more reliable than the gradient directions at vertices in $N^T(v)$. Thus, we can use gradients at vertices in $N^O(v)$ to determine whether gradient $\tilde{n}_v$ is reliable. We reduce the number of nearby vertices whose gradient directions must match $\tilde{n}_v$ in three ways.
DOESORTHMATCHA($v, \alpha_1, \alpha_2$)
1 if ($N^O(v) = \emptyset$) then return (true);
2 foreach grid vertex $v' \in N^O(v)$ do
3 Let $v'', v''' \in N_3(v)$ be the vertices such that $(v, v', v'', v''')$ is a collinear sequence of adjacent vertices flagMatch ← false;
4 if ($\angle(\vec{n}_v, \phi(\vec{n}_{v''}, \vec{n}_{v'''})) \leq \alpha_2$) and
5 ($\angle(\vec{n}_v, \phi(\vec{n}_{v''}, \vec{n}_{v'''})) \leq \alpha_2$) then flagMatch ← true;
6 if ($\angle(\vec{n}_v, \vec{n}_{v''}) \leq \alpha_1$) then flagMatch ← true;
7 if (flagMatch = false) then return (false);
end
10 return (true)

Algorithm 3. Algorithm DOESORTHMATCHA.

DOESORTHMATCHB($v, \alpha_1$)
1 foreach grid vertex $v' \in N^O(v)$ do
2 if ($v \in N^O(v')$) then
3 if ($\angle(\vec{n}_v, \vec{n}_{v''}) \leq \alpha_1$) then return (true);
end
5 return (false)

Algorithm 4. Algorithm DOESORTHMATCHB.

First, we note that there is little curvature along the gradient direction in the scalar field represented by a CT scan. Thus, in addition to the comparison of $\vec{n}_v$ and $\phi(\vec{n}_{v''}, \vec{n}_{v'''})$, we can simply compare $\vec{n}_v$ and $\vec{n}_{v''}$. If angle $\angle(\vec{n}_v, \vec{n}_{v''})$ is below some threshold $\alpha_1$, then $\vec{n}_{v''}$ can be a guarantor of the reliability of $\vec{n}_v$. (See Algorithm 3.)

Second, instead of comparing $\vec{n}_v$ and $\vec{n}_{v''}$, we compare $\vec{n}_v$ to its immediate neighbor $\vec{n}_{v'} \in N(v)$ (Algorithm 4.) We still compare $\vec{n}_v$ with vertices at edge distance 3 in the tangent directions. If some sufficiently large ball contains $v \in A_i$ and does not intersect $X = \bigcup A_i \partial A_j$, then either there is a $v' \in N^T(v)$ and collinear sequence $(v, v', v'', v''')$ where $v'', v''' \in I_{\mathcal{V}}(A_i)$ or there is a $v' \in N^O(v)$ such that $v' \in I_{\mathcal{V}}(A_i)$.
**Algorithm 5.** Algorithm \textsc{FindReliable}

Third, in Algorithm 4 we replace the requirement that gradient directions of both vertices in $N^O(v)$ “match” $\tilde{n}_v$ by a requirement that the gradient direction of one vertex in $N^O(v)$ matches $\tilde{n}_v$. Let $N^O(v) = \{v_1', v_2'\}$. We require that either the $\angle(\tilde{n}_v, \tilde{n}_{v_1'})$ or that $\angle(\tilde{n}_v, \tilde{n}_{v_2'})$ is small. To make sure that we are only applying the single check to gradients which truly point along an axis, we require also that $\tilde{n}_{v_1'}$ or $\tilde{n}_{v_2'}$ be close to the axis direction.

Algorithm 3 contains the check in both orthogonal directions. Algorithm 4 contains the additional check in one orthogonal direction. Because of the check $v \in N^O(v')$ in line 2 of Algorithm 4, there are cases where Algorithm 3 may return true while Algorithm 4 returns false. Algorithm 5 is the full algorithm.

Algorithm 5 loosens the conditions under which a gradient is identified as reliable. By Proposition 3, if $N_4(v) \subset A_i$ for some $A_i$, then Algorithm 5 returns true. What about the converse, i.e. if Algorithm 5 returns true?

We can show that for each $v \in A_i$, there is either a collinear sequence of grid vertices $(v, v', v'', v''')$ such that $v' \in N^T(v)$ and $v'', v''' \in \mathcal{I}_v(A_i)$ or there is a vertex $v' \in N^O(v)$ such that $v' \in \mathcal{I}_v(A_i)$. In the first case, the angle between the approximate gradient direction $\tilde{n}_v$ and the true gradient direction $n_v$ is bounded by $\alpha_2 + 3\lambda + 5\mu$. In the second case, the angle between $\tilde{n}_v$ and $n_v$ is bounded by $\angle(\tilde{n}_v, n_v) + \kappa + \mu$ where $\kappa$ is a bound on curvature.
If \( \angle(\tilde{n}_v, \tilde{n}_{v'}) \leq \alpha_1 \), then \( \angle(\tilde{n}_v, n_v) \leq \alpha + \kappa + \mu \). However, Algorithm DOESORTH-MATCHB (Algorithm 4), only guarantees that \( \angle(\tilde{n}_v, \tilde{n}_w) \leq \alpha_1 \) for one vertex \( w \in N^O(v) \).

\[
\text{EXTENDRELIABLE}(\alpha_2, \text{numIter})
\begin{align*}
\text{for } k &\leftarrow 1 \text{ to numIter do} \\
S &\leftarrow \emptyset; \\
\text{foreach vertex } v \text{ do} \\
\text{foreach vertex } v' \in N^T(v) \text{ do} \\
\text{let } v'', v''' \in N_3(v) \text{ be the vertices such that } (v, v', v'', v''') \text{ is a collinear sequence of adjacent vertices;} \\
\text{if } (v', v'' \text{ and } v''' \text{ are marked reliable) then} \\
\text{if } (\angle(\tilde{n}_v, \phi(\tilde{n}_{v''}, \tilde{n}_{v'})) < \alpha_2) \text{ and } (\angle(\tilde{n}_v, \phi_2(\tilde{n}_{v'''}, \tilde{n}_{v''})) < \alpha_2) \text{ then} \\
S &\leftarrow S \cup \{v\}; \\
\text{end} \\
\text{end} \\
\text{end} \\
\text{end} \\
\text{Mark each vertex } v \in S \text{ as reliable;}
\end{align*}
\]

\textbf{Algorithm 6. Algorithm EXTENDRELIABLE}

3.2.5 Extending Reliable Gradients

Once we’ve identified reliable gradients by Algorithm 5, we can use those reliable gradients to identify vertex neighbors with reliable gradients. By Proposition 3, if \( v'' \) and \( v''' \) are reliable and \( \angle(\phi_2(\tilde{n}_{v''}, \tilde{n}_{v'}), \tilde{n}_v) \) is small, then \( \angle(\tilde{n}_v, n_v) \) is small. This is particularly helpful near gradient discontinuities, where Algorithm 5 will fail to identify correct gradients as reliable because some of their neighbors are incorrect.

Pseudo code for Algorithm EXTENDRELIABLE is given in Algorithm 6. Because the initial set of reliable gradients is computed using \( (v, v', v'', v''') \), the value of \text{numIter} is set to 2.
Algorithm 7. Algorithm RELIGRAD

3.2.6 Algorithm RELIGRAD

Our final algorithm, named RELIGRAD, applies Algorithm FINDRELIABLE to each vertex and then calls EXTENDRELIABLE. Pseudo code is in Algorithm 7. The experimental results for RELIGRAD are given in Section 5.2
Chapter 4: SHREC: SHarp REConstruction from Scalar Data

Preserving sharp edges and corners is an important objective of surface reconstruction. In Chapter 2 some of the previous algorithms to construct isosurfaces with sharp edges and corners, were discussed.

Some of the fundamental problems with these algorithms were also mentioned. This chapter introduces a new method based on merging grid cubes near sharp edges which produces significantly better results than previous works.

4.1 Protecting Ball Overview

In [19], Cheng, Dey and Ramos describe a method for meshing piecewise smooth complexes using protecting balls around the sharp or boundary edges of those complexes. This idea is applied to isosurface reconstruction by merging grid cubes around sharp edges and corners, creating regions which act like protecting balls. A single isosurface vertex is placed in each such region.

Placing a single isosurface vertex near the center of each region, guarantees that the isosurface vertices on sharp edges and corners are well separated from other isosurface vertices, thereby avoiding the creation of degenerate quadrilaterals or incorrect ordering of isosurface vertices along sharp edges. Our algorithm also avoids the creation of notches along sharp edges.
Importantly, the separation of isosurface vertices on sharp edges makes the sharp edge much less sensitive to changes in vertex location. Thus our algorithm is tolerant of small approximation errors in intersection location and/or normals.

Because our algorithm is tolerant of approximation errors in isosurface vertex location, Hermite data is no longer required as input. Instead, gradient grid data: regular grids with scalar values and gradients computed from RELIGRAD (See Chapter 3) is used at each grid vertices.

Isosurface vertex location is directly computed from this data, using gradients from neighboring cubes to increase reliability.

4.2 SHREC Overview

Algorithm SHREC has four parts: computation of isosurface vertex locations, selection of a well-spaced subset of grid cubes on or near sharp features, merging grid cubes around selected cubes and construction of isosurface triangles and quadrilaterals (Figure 4.1).
Algorithm MergeSharp from [1] is a simpler, stripped version of SHREC, it also has a similar four parts, but the first three parts are substantially different and simpler than SHREC.

We next describe the overall steps of SHREC. In the first part, SHREC computes the location of an isosurface vertex, \(\mathbf{c}_{\text{isov}}\), in or near each active grid cube \(\mathbf{c}\). SHREC computes this location using gradients at the vertices of \(\mathbf{c}\) and cubes adjacent to \(\mathbf{c}\). When cube \(\mathbf{c}\) is near a sharp feature, the isosurface vertex will lie on that feature. SHREC also identifies whether the feature is 0-dimensional or 1-dimensional or if the vertex lies on a smooth region of the isosurface.

Let \(C_0\) and \(C_1\) be the set of grid cubes whose isosurface vertices lie on 0-dimensional features or 1-dimensional features, respectively.

In the second part, SHREC selects a well-spaced subset of \(C_0 \cup C_1\). Cubes which generate such vertices are called selected cubes. The vertices in selected cubes will form the sharp features in the isosurface mesh. SHREC chooses selected cubes as follows:

Mark all the cubes as “uncovered”. Sort \(C_0\) and \(C_1\) by increasing distance of \(\mathbf{c}_{\text{isov}}\) from \(\mathbf{c}_{\text{centroid}}\). Select the next uncovered cube \(\mathbf{c}\) in \(C_0\) where \(p_c\) does not form a large angle
Compute \text{c.isovLoc} for each active cube \text{c}.

If \text{c.isovLoc} is in \text{c}', while \text{c}'.isovLoc is in \text{c}, swap \text{c.isovLoc} and \text{c}'.isovLoc.

If \text{c.isovLoc} is in \text{c}', then \text{c}'.isovLoc ← \text{c.isovLoc}.

If \text{c} and \text{c}' are 1D feature cubes intersecting in a grid edge \text{c}, then compute the intersection \text{p} of \text{c.Lsharp} and a plane separating \text{c} and \text{c}'.
\text{c}.isov ← \text{p} where \text{c} is the cube containing \text{p}.

If \text{c} and \text{c}' are 1D feature cubes intersecting in a grid vertex \text{v}, then compute the intersection \text{p} of \text{c.Lsharp} and a plane separating \text{c} and \text{c}'.
\text{c}.isov ← \text{p} where \text{c} is the cube containing \text{p}.

Figure 4.3: SHREC algorithm for generating points on sharp features.

triangle with vertices in previously selected cubes. Mark all cubes which share a vertex with \text{c} as covered. After processing list \text{C}_0, apply the same procedure to list \text{C}_1. The third part is merging covered cubes with selected cubes and finally constructing the isosurface.

4.3 SHREC Details

Section 4.4 describes the process of isosurface vertex generation. Section 4.5 describes the process of isosurface selection. Section 4.6 describes the process of merging the neighboring cubes around selected cubes. Finally, Section 4.7 describes the isosurface construction process.
4.4 Vertex Computation

As noted in the previous section, when a cube $c$ intersects a 1-dimensional feature, we would like to choose a point $c.isov$ inside that feature whenever possible. We would also like that point to be “near” the center of the intersection of the isosurface and cube $c$. Finally, if the 1-dimensional feature is near $c$ but does not intersect $c$, we would like to choose a point that is “closest” to $c$. The SHREC algorithm for isosurface vertex computation is a variation of the algorithm in [41] for computing vertex locations from surface normals. Similar algorithms are in [46, 64, 76].

4.4.1 SHREC vertex Computation

SHREC uses a number of techniques to compute points inside cubes. First, when a cube $c$ is near a 1-dimensional feature, SHREC computes a line $c.Lsharp$ approximating the feature near $c$. SHREC computes the points on $c.Lsharp$ closest to $c.centroid$ and $c.center$. If one of those points is inside $c$, then SHREC sets $c.isov$ to that point. Otherwise, SHREC computes the point on $c.Lsharp$ which is closest to $c.center$ under the $L_\infty$ metric. If $c.Lsharp$ intersects $c$, then that point will lie in $c$.

Second, SHREC compares isosurface vertex locations computed by neighboring cubes and swaps or sets isosurface vertex locations based on those comparisons. Third, SHREC computes isosurface vertex locations on planes separating cubes intersecting 1D features. SHREC stores these isosurface vertex locations in the cubes containing them.

The second and third steps of the algorithm are outlined in Figure 4.3. Details of all three steps follow.
At the heart of any algorithm to compute a surface with sharp features is an algorithm to compute the locations of mesh vertices on those features. The following algorithm computes an initial vertex location $c.isoLoc$ for each active cube $c$. The discussion extends from Section 2.3.

SHREC computes one isosurface vertex location $c.isoLoc$ for each active grid cube $c$. The algorithm computes an isosurface vertex location for $c$ by using the gradients at vertices of $c$ and nearby cubes. A grid vertex $v$ with scalar $s_v$ and gradient $g_v$ gives a plane

$$h_v = \{p : (p - v) \cdot g_v + s_v = \sigma\}.$$

This plane is an “approximation” to the tangent planes to the isosurface at isosurface points in the neighborhood of $v$.

A set of $k$ vertices and gradients gives a set of $k$ equations in three variables $Mp = b$ where $M$ is a $k \times 3$ matrix and $b$ is a column vector with $k$ rows and the unknown $p$ is a column vector with three rows. If $M$ has more than three rows, the system $Mp = b$ is over-determined and has no exact solution. The least squares approximation to $Mp = b$ is the solution to $M^T M p = M^T b$.

Let $A$ be the $3 \times 3$ matrix $M^T M$ and let $b'$ equal the column vector $M^T b$. SHREC uses singular valued decomposition (SVD) as described in [41, 46, 47] to compute an isosurface vertex location from the equation $Ap = b'$.

Let $\sigma_1$, $\sigma_2$ and $\sigma_3$ be the singular values of $A$ sorted in decreasing order. A singular value $\sigma_i$ is large, if $\sigma_i / \sigma_1 \geq \varepsilon$ for some predefined parameter $\varepsilon$. There are three possible cases based on the number of large singular values of $A$.

**0D feature cube:** In the first case $A$ has three large singular values. In this case, the tangent planes around $c$ have normals in three or more very distinct directions. The isosurface has some 0-dimensional feature near cube $c$. We call a cube $c$ a **0D feature cube**
if the matrix $A$ associated with $c$ has three large singular values. The solution to $Ap = b'$ approximates the 0-dimensional feature near cube $c$.

**1D feature cube:** In the second case, matrix $A$ has only two large singular values. In this case, the tangent plane normals are close to two different directions. The isosurface has some 1-dimensional feature near cube $c$. We call a cube $c$ a 1D feature cube if the matrix $A$ associated with $c$ has two large singular values.

To compute the 1-dimensional feature near $c$, we use singular valued decomposition to remove the small singular value from $A$. The singular valued decomposition of $A$ is $A = U \Sigma V^T$ where

$$
\Sigma = \begin{pmatrix}
\sigma_1 & 0 & 0 \\
0 & \sigma_2 & 0 \\
0 & 0 & \sigma_3
\end{pmatrix}.
$$

When $A$ has only two large singular values, $\sigma_1$ and $\sigma_2$, MergeSharp replaces $\Sigma$ by a new diagonal matrix $\Sigma'$ with diagonal entries $\sigma_1$, $\sigma_2$, 0. Let $A'$ equal $U \Sigma V^T$. Matrix $A'$ has rank two. The solution to $A'p = b'$ is a set of points on a line $c.Lsharp$. Line $c.Lsharp$ represents a line tangent to the 1-dimensional feature.

**Smooth Cube:** In the last case, matrix $A$ has only one large singular value. In this case, the tangent plane normals are close to a single direction. The isosurface is smooth around cube $c$. Replace $\Sigma$ by a new diagonal matrix $\Sigma'$ with diagonal entries $\sigma_1$, 0, 0. Let $A'$ equal $U \Sigma' V^T$. Matrix $A'$ has rank one. The solution to $A'p = b'$ is a set of points on a plane. That plane represents a tangent plane to the isosurface in cube $c$.

In the case that $A$ has only one or two large singular values, the solution to $A'p = b'$ is a line or plane. As suggested in [64], SHREC selects the point on the line or plane that is closest to $c.centroid$. 

55
More precisely, SHREC defines the diagonal matrix $\Sigma^+$ with diagonal entries:

- $1/\sigma_1, 1/\sigma_2, 1/\sigma_3$, if $A$ has three singular values,
- $1/\sigma_1, 1/\sigma_2, 0$, if $A$ has two singular values,
- $1/\sigma_1, 0, 0$, if $A$ has one singular value.

The point

$$\phi_c(q) = q + V\Sigma^+U^T(b' - Aq) \tag{4.2}$$

is the point on $\{p : A'p = b'\}$ closest to $q$. SHREC selects point $\phi_c(\text{centroid})$ on the line or plane $\{p : A'p = b'\}$.

The number of large singular values of $A$ determines whether the computed isosurface vertex location lies on a 0-dimensional feature, a 1-dimensional feature, or a smooth portion of the isosurface. This information is used in subsequent steps in the SHREC algorithm.

When $A'p = b'$ is a line, the direction of that line is given by the equation $(I - V\Sigma^+U^T)w$ for any vector $w$ which is not in the kernel of $(I - V\Sigma^+U^T)w$. Substituting $(1, 0, 0), (0, 1, 0)$ and $(0, 0, 1)$ for $w$ and using $(I - V\Sigma^+U^T)w$ with largest magnitude gives the direction. The direction and point $\phi_c(\text{centroid})$ defines the line $c.Lsharp$.

### 4.4.2 Problems with computing and selecting vertices

There are problems with computing isosurface vertex location and selecting isosurface vertices.

**Correct Gradients**

The vertex computation algorithm is based on availability of correct gradients in the neighborhood. As discussed in Chapter 3 computing correct gradients near sharp features is difficult, if not impossible. Thus, the gradients in $c$ and its neighbors may not be known.
Algorithm SHREC selects gradients from a larger area than just the immediate neighbors of $c$.

**Isosurface vertices outside the cube**

Second, the isosurface vertex $c.isov$ generated by cube $c$ may lie outside $c$. Moreover, if cube $c$ is near a 1-dimensional feature, point $c.isov$ may lie outside of $c$ even if this 1-dimensional feature intersects $c$. In addition, because of curvature, noise and numerical instability, point $c.isov$ could lie in some cube $c'$ adjacent to $c$ while point $c'.isov$ lies inside $c.isov$. Algorithm SHREC locates $c.isov$ inside $c$ whenever possible.

### 4.4.3 Computing Vertex Locations on 1-Dimensional Features

When a cube $c$ intersects a 1-dimensional feature, we would like to choose a point $c.isov$ inside that feature whenever possible. We would also like that point to be “near” the center of the intersection of the isosurface and cube $c$. Finally, if the 1-dimensional feature is near $c$ but does not intersect $c$, we would like to choose a point that is “closest” to $c$.

SHREC uses a number of techniques to compute points inside cubes. First, when a cube $c$ is near a 1-dimensional feature, SHREC computes a line $c.Lsharp$ approximating the feature near $c$. SHREC computes the points on $c.Lsharp$ closest to $c<center>$ and $c.center$. If one of those points is inside $c$, then SHREC sets $c.isov$ to that point. Otherwise, SHREC computes the point on $c.Lsharp$ which is closest to $c.center$ under the $L_\infty$ metric. If $c.Lsharp$ intersects $c$, then that point will lie in $c$.

**Details:** Consider a 1D feature cube $c$. By definition, cube $c$ is near some 1-dimensional feature. That 1D feature is approximated by the line $c.Lsharp$. The location $c.isovLoc$ of the isosurface vertex associated with $c$ should be on $c.Lsharp$. However, that condition still gives one degree of freedom in selecting the location of $c.isovLoc$. One additional
Algorithm SHREC computes three different possible isosurface vertex locations. First, Algorithm SHREC computes the point $p^0_c = \phi_c(\text{c.centroid})$ (Equation 4.2), the point on c.Lsharp which is closest to c.centroid. The idea is that c.centroid is a good approximation for the intersection of c and the isosurface, and so one should choose a point near c.centroid. If $p^0_c$ lies in c, then SHREC sets c.isovLoc to $p^0_c$. In most cases, if line c.Lsharp intersects cube c, then the point $p^0_c$ will lie in c. However, if line c.Lsharp is “near” some facet or edge of c, then it is possible for c.Lsharp to intersect c but $p^0_c$ to lie outside c. (See Figure 4.4(a).)
If $p^0_c$ lies outside $c$, then SHREC computes $p^1_c = \phi_c(c.\text{center})$, the point on $c.\text{Lsharp}$ which lies closest to the center $c.\text{center}$ of cube $c$. If $p^1_c$ is in $c$ while $p^0_c$ is not, SHREC sets $c.\text{isovLoc}$ to $p^1_c$.

Unfortunately, it is possible that both $p^0_c$ and $p^1_c$ are not in $c$ even though $c.\text{Lsharp}$ intersects $c$. (See Figure 4.4(b).) As a final step, SHREC computes a point $p^2_c$ on $c.\text{Lsharp}$ which is closest to the center $c.\text{center}$ of $c$ under the $L_\infty$ metric. If $c.\text{Lsharp}$ intersects $c$, then this point is guaranteed to lie in $c$. If $p^2_c$ is in $c$ while $p^1_c$ and $p^2_c$ are not, SHREC sets $c.\text{isovLoc}$ to $p^2_c$.

Instead of computing $p^0_c$, $p^1_c$ and $p^2_c$, we could compute and use only $p^2_c$. However, the computations of $p^0_c$ and $p^1_c$ and $p^2_c$ are much faster, and their locations are preferable to $p^2_c$ when they are contained in cube $c$. Algorithm MergeSharp computes only $\phi^0_c$.

### 4.4.4 Swapping Locations

Many of the problems in Algorithm SHREC (and MergeSharp) occur when isosurface vertex location $c.\text{isovLoc}$ lies outside of cube $c$. Thus, it is always preferable that $c.\text{isovLoc}$
be in \( c \). This is not always possible since the sharp features near \( c \) may not intersect \( c \). However, in cases where a sharp feature intersects \( c \), we would like \( c.isovLoc \) to be in \( c \).

The computation of a sharp feature near \( c \) depends upon gradients in the neighborhood of \( c \). The set of such gradients changes for each cube \( c \). Because of the inaccuracy in computing sharp features, it is possible that \( c.isovLoc \) is in a cube \( c' \) adjacent to \( c \) while \( c'.isovLoc \) is not contained in \( c \).

In some cases, location \( c.isovLoc \) is in \( c' \) while \( c'.isovLoc \) is in \( c \). In those cases, we simply swap \( c.isovLoc \) and \( c'.isovLoc \). In other cases, location \( c'.isovLoc \) lies in some third cube \( c'' \). In that case, we simply set \( c'.isovLoc \) to \( c.isovLoc \).

The setting of isosurface vertex locations from adjacent cubes increases the number of cubes \( c \) containing their associated vertex locations \( c.isovLoc \). Note that if the initial location \( c.isovLoc \) lies in \( c \), then we never change \( c.isovLoc \).

### 4.4.5 Locations on Planes

**Two Cubes, One edges:** Consider two grid cubes, \( c \) and \( c' \), which intersect in an edge \( c \) but not in any facet. (See Figure 4.5.) Two other grid cubes, \( \tilde{c} \) and \( \tilde{c}' \) share edge \( c \). A 1-dimensional feature which passes through \( c \) and \( c' \) must intersect either \( \tilde{c} \) or \( \tilde{c}' \). (In the exceptional case, the 1-dimensional feature passes through \( c \), in which case it intersects both \( \tilde{c} \) and \( \tilde{c}' \).) Since the 1D feature intersects either \( \tilde{c} \) and \( \tilde{c}' \), either \( \tilde{c}.isovLoc \) should be in \( \tilde{c} \) or \( \tilde{c}'.isovLoc \) should be in \( \tilde{c}' \). However, because of inaccuracies in computing points on 1D features, neither condition may hold.

To ensure that either \( \tilde{c}.isovLoc \) lies in \( \tilde{c} \) or \( \tilde{c}'.isovLoc \) lies in \( \tilde{c}' \), Algorithm SHREC computes the plane \( h \) containing \( c \) and perpendicular to the line from \( c.center \) to \( c'.center \). SHREC then computes the intersection of the line \( c.Lsharp \) and plane \( h \). This intersections point, \( p_I \), lies in either \( \tilde{c} \) or \( \tilde{c}' \). If \( p_I \) lies in \( \tilde{c} \) and \( \tilde{c}.isovLoc \) is not in \( \tilde{c} \), SHREC sets \( \tilde{c}.isovLoc \) to \( p_I \). If \( p_I \) lies in \( \tilde{c}' \) and \( \tilde{c}'.isovLoc \) is not in \( \tilde{c}' \), SHREC sets \( \tilde{c}'.isovLoc \) to \( p_I \).
Two Cubes, one Vertex: Next, consider the case of two grid cubes, \(\mathbf{c}\) and \(\mathbf{c}'\), which intersect in a vertex \(v\) but not in any edge. (See Figure 4.6.) Six other grid cubes share vertex \(v\) with \(\mathbf{c}\) and \(\mathbf{c}'\). A 1-dimensional feature which passes through \(\mathbf{c}\) and \(\mathbf{c}'\) must intersect one of these six other grid cubes. (In the exceptional case, the 1D feature passes through \(v\), in which case all six.) Since the 1D feature intersects one of the six grid cubes, point \(\mathbf{c}.\mathbf{i沙发Loc}\) should lie in \(\mathbf{c}\) for one of the six grid cubes \(\mathbf{c}\). Again because of inaccuracies in computing points on 1D features, point \(\mathbf{c}.\mathbf{i沙发Loc}\) may not be in \(\mathbf{c}\) for any of the six grid cubes \(\mathbf{c}\).

To ensure that \(p_c\) lies in \(\mathbf{c}\) for one of the six grid cubes, Algorithm SHREC computes the plane \(h\) containing \(v\) and perpendicular to the line from \(\mathbf{c}.\mathbf{center}\) to \(\mathbf{c}'.\mathbf{center}\). SHREC then computes the intersection of the line \(\mathbf{c}.\mathbf{Lsharp}\) and plane \(h\). This intersections point,
\(p_I\), lies in at least one of the six grid cubes. If \(p_I\) lies in cube \(\tilde{c}\) and \(\tilde{c}.\text{isovLoc}\) is not in \(\tilde{c}\), SHREC sets \(\tilde{c}.\text{isovLoc}\) to \(p_I\).

The computation in this section ensures “continuity” in the cubes which intersect a 1-dimensional feature. If a 1-dimensional feature intersects a cube \(c\), then the 1-dimensional feature should intersect two cubes \(c'\) and \(c''\) which share a facet with \(c\). SHREC’s computation of line-plane intersections described above, ensures that if \(c'\) and \(c''\) are active cubes, then \(c'\) will contain \(c'.\text{isovLoc}\) and \(c''\) will contain \(c''.\text{isovLoc}\).

### 4.5 Selecting 0D and 1D Feature Cubes

Algorithm SHREC selects a well-spaced subset of the 0D and 1D feature cubes and merges isosurface vertices in adjacent cubes with the vertices generated by the selected cubes. To ensure that the isosurface vertices on sharp features are well-spaced, SHREC never selects two cubes which have a common vertex. Whenever SHREC selects a cube \(c\), any adjacent cube sharing a vertex with \(c\) is marked as “covered” by \(c\). SHREC never selects a covered cube.

The selection step chooses cubes based on the proximity of \(c.\text{isov}\) to \(c.\text{centroid}\). If a point \(c.\text{isov}\) is near \(c.\text{centroid}\), it probably is located in cube \(c\) and is a good approximation of the vertex location. While this is reasonable, it ignores the interaction between selected cubes. SHREC does better when 1-dimensional features are well-covered by the selected and covered cubes. For instance, the selected squares in Figure 4.2(b), are packed together more closely than the ones in Figure 4.2(a), and their 3 × 3 regions do a better job of covering the given curve.

SHREC does better when the selected cubes are packed “tightly” together. In particular, SHREC tries to avoid certain configurations of nearby cubes. Consider cubes \(c\) and \(c'\) with grid indices \((x_0, x_1, x_2)\) and \((x'_0, x'_1, x'_2)\), respectively. (A cube with grid indices \((x_0, x_1, x_2)\) is...
in column $x_0$, row $x_1$ and $z$-plane $x_2$ of the grid.) The *distance vector* between the cubes is $(|x_0 - x'_0|, |x_1 - x'_1|, |x_2 - x'_2|)$. Two grid cubes are in configuration $(a_0, a_1, a_2)$ if the distance vector between the cubes is some permutation of $(a_0, a_1, a_2)$. Figures 4.7, 4.8, 4.9 and 4.10, contain examples of 2D and 3D configurations of grid cubes.

We can get some insight into 3D configurations of grid cubes by considering the 2D configurations of grid squares. Figure 4.7 contains configurations $(2,0), (3,0), (2,1)$ and $(3,1)$ of grid squares. The $3 \times 3$ subgrids around the selected squares are “tightly packed” so that any line segment with endpoints in the selected squares is contained within the union of the two $3 \times 3$ subgrids.

Figure 4.8 contains configurations $(3,2)$ and $(2,2)$ of grid squares. With configuration $(3,2)$, some line segments with endpoints in the selected squares are not contained within the union of the $3 \times 3$ subgrids. Configuration $(2,2)$ is somewhat better, since all line segments with endpoints in the selected squares are contained within the union of the $3 \times 3$ subgrids. Unfortunately, they are only barely contained. The green line segment in Figure 4.8(b) intersects the boundary of the $3 \times 3$ subgrid. Slightly curving the segment could mean that it no longer was contained in the union.

Note that if selected squares are suitably far apart, then additional squares can be selected between them. For instance, a magenta square can be selected between the two colored squares in Figure 4.9(a) and the $3 \times 3$ subgrids will fit together tightly.

Some examples of tightly packed 3D configurations of cubes are given in Figure 4.10. Any line segment with endpoints in the two selected cubes $c$ and $c'$ will be well contained within the union of the two $3 \times 3 \times 3$ subgrids, $R^{3 \times 3 \times 3}_c$ and $R^{3 \times 3 \times 3}_{c'}$, around $c$ and $c'$.

Figure 4.11 contains three problematic configurations, $(3,2,0), (3,2,1)$ and $(3,2,2)$, of selected cubes. Line segments with endpoints in the two selected cubes $c$ and $c'$ may contain points outside $R^{3 \times 3 \times 3}_c \cup R^{3 \times 3 \times 3}_{c'}$. SHREC attempts to avoid selecting cubes with these configurations.
Figure 4.7: Tightly packed 2D configurations of selected squares. Selected squares and $3 \times 3$ subgrid around each square. Line segments with endpoints on the two selected squares (e.g., the green line segments) are contained within the union of the two $3 \times 3$ subgrids.

If two selected cubes, $c$ and $c'$, are in configuration $(2, 2, 0)$, $(2, 2, 1)$ or $(2, 2, 2)$, then a line segment with endpoints in $c$ and $c'$ could intersect the boundary of $R_c^{3 \times 3} \cup R_{c'}^{3 \times 3}$. Such configurations are undesirable. Unfortunately, we found that avoiding such configurations is too restrictive.

Consider two cubes, $c$ and $c''$ in a $(3, 2, 0)$, $(3, 2, 1)$ or $(3, 2, 2)$ configuration. If a third selected cube $c'$ lies between $c$ and $c''$, then the 1-dimensional feature will pass from $c$ to $c'$ to $c''$. The selection of $c'$ “blocks” the problematic interaction of $c$ and $c''$. If cubes $c$ and $c'$ are selected, then SHREC will permit the selection of $c''$, even though $c$ and $c''$ have a configuration $(3, 2, *)$. Figure 4.12 contains an example of two cubes in a $(3, 2, 2)$ configuration and a third cube between them.

More formally, let $c$, $c'$ and $c''$ be grid cubes with grid indices $(x_0, x_1, x_2)$, $(x_0', x_1', x_2')$ and $(x_0'', x_1'', x_2'')$, respectively. Cube $c'$ separates $c$ from $c''$ if $x_i' \in [x_i, x_i'']$ for $i = 0, 1, 2$ and $x_i < x_i' < x_i''$ or $x_i > x_i' > x_i''$ for some $i$. SHREC avoids selecting cube $c''$ if some already
Figure 4.8: Problematic 2D configurations of selected squares. Selected squares and $3 \times 3$ subgrid around each square. (a) The green line segment is not contained within the union of the two $3 \times 3$ subgrids. (b) The green line segment intersects the boundary of the union of the two $3 \times 3$ subgrids.

Figure 4.9: (a) Configuration (4,2) has space between the $3 \times 3$ subgrids around each selected square. (b) Additional cube (magenta) which packs tightly with two other squares.

selected cube $c$ forms a $(3, 2, 0)$ or $(3, 2, 1)$ or $(3, 2, 2)$ configuration with $c''$ and no already selected cube separates $c$ from $c''$.

Figure 4.13 contains an outline of the algorithm for selecting 0D and 1D feature cubes. SHREC first selects 0D feature cubes. When the 0-dimensional feature is inside an active cube, SHREC selects the active cube containing the feature point. When the feature point is not in an active cube, SHREC selects the active cube “closest” to the feature point by choosing the cube $c$ whose center has minimum $L_\infty$ distance to the feature point.

SHREC next selects 1D feature cubes which are “near” the selected 0D feature cubes, i.e., they are contained in an $7 \times 7 \times 7$ subgrid around each selected 0D feature cube. Selecting 1D feature cubes near 0D feature cubes poses special challenges since there are
multiple 1-dimensional features ending at a 0-dimensional feature. Selecting a cube which is near two such 1-dimensional features can obscure one of the edges.

Let $c$ be a selected 0D feature cube and let $c'$ be a 1D feature cube which is in a $(2, 0, 0)$, $(2, 1, 0)$ or $(2, 1, 1)$ configuration with $c$. Let $c''$ be any cube sharing an edge or facet with $c'$ which is contained in the $5 \times 5 \times 5$ subgrid but is not covered by $c$. If $c''$ is active and a 1D feature cube, then SHREC does not select cube $c'$.
Once 1D feature cubes near 0D feature cubes are selected, SHREC could iterate by selecting uncovered 1D feature cubes near already selected cubes. If no uncovered 1D feature cubes was near a selected cube, SHREC could select an arbitrary uncovered 1D feature cube and extend the set of selected cubes from that cube. By not selecting a cube $c''$ if it forms a $(3,2,0)$, $(3,2,1)$ or $(3,2,2)$ with an already selected cube $c$ (and is not separated by a selected cube from $c$,) SHREC would produce a tight packing of selected cubes.
Select 0D feature cubes

Select 1D features cubes in $7 \times 7 \times 7$ subgrids around selected 0D feature cubes

Select 1D feature cubes whose indices are congruent to $(0,0,0)$ mod 6.

Select 1D feature cubes whose indices are congruent to permutations of $(k,0,0)$ mod 6.

Select 1D feature cubes whose indices are congruent to permutations of $(k_a,k_b,0)$ mod 6.

Select 1D feature cubes whose indices are congruent to permutations of $(k_a,k_b,k_c)$ mod 6.

Figure 4.13: SHREC algorithm for selecting 0D and 1D feature cubes.
The algorithm outlined in the previous paragraph would produce a good set of selected cubes but it is highly sequential. One of the best aspects of the Marching Cubes and dual contouring algorithms is their local, distributed, parallelizable nature. Sequentially extending the set of selected cubes would totally destroy that aspect of the algorithms. Instead of selecting cubes using the sequential algorithm given above, SHREC divides the regular grid into overlapping $6 \times 6 \times 6$ subgrids, selects cubes from the boundaries of those subgrids and then from their interior. The algorithm is completely local and easily distributed and parallelizable.

Each grid cube has an index $(x_0, x_1, x_2)$. SHREC processes the grid cubes by reducing the indices modulo six to $(x_0 \mod 6, x_1 \mod 6, x_2 \mod 6)$ (Figure 4.14(a)). SHREC first selects 1D feature cubes with indices congruent to $(0, 0, 0)$ modulo six. SHREC next selects 1D feature cubes with indices congruent modulo six to $(\pm 1, 0, 0)$ or $(0, \pm 1, 0)$ or $(0, 0, \pm 1)$. (Figure 4.14(b)). SHREC then selects 1D feature cubes with indices congruent modulo six to $(\pm 2, 0, 0)$ or $(0, \pm 2, 0)$ or $(0, 0, \pm 2)$. Finally, SHREC selects 1D feature cubes with indices congruent modulo six to $(3, 0, 0)$ or $(0, 3, 0)$ or $(0, 0, 3)$. SHREC does not select a 1D feature cube $c$ if some already selected cube $c'$ forms configuration $(3, 2, 0)$, $(3, 2, 1)$ or $(3, 2, 2)$ with $c$ and no already selected cube separates $c$ from $c''$.

SHREC next selects 1D feature cubes which are some permutation of $(k_a, k_b, 0)$ modulo six, starting first with permutations of $(\pm 1, \pm 1, 0)$ and ending with permutations of $(3, 3, 0)$. (Figure 4.14(c)). Finally, SHREC selects 1D feature cubes which are permutations $(k_a, k_b, k_c)$ modulo six, starting first with permutations of $(\pm 1, \pm 1, \pm 1)$ and ending with $(3, 3, 3)$.

It is possible that the selection of two 1D feature cubes $c$ and $c''$ at distance five apart can force the selection of an edge cube $c'$ which has a $(3, 2, 1)$ or $(3, 2, 2)$ configuration with either $c$ or $c''$. This happens if the distance vector between $c$ and $c''$ is $(5, 3, 2)$. Thus, in addition to avoiding $(3, 2, \ast)$ configurations, SHREC also avoids selecting two cubes, $c$
and \( c'' \), which form a \((5, 3, 2)\) configuration. Of course, if \( c \) and \( c'' \) are separated by some other selected cube, then they can both be selected.

While the above rules select a set of cubes which almost completely cover the 1-dimensional features of a surface, it is possible that the configuration restrictions leave a few areas uncovered. Therefore, SHREC repeats the selection process to select any remaining uncovered 1D feature cubes but drops the configuration restrictions.

4.6 Merging Points with Features Points

The simple way of merging would be to merge neighbors of a selected cube \( c \) with \( c \) when \( c \) is selected. Doing so sometimes distorts triangles, creating extremely thin triangles and sometimes creating folds in the “surface” mesh.

SHREC is much more careful about its cube merging. SHREC also extends the merging to some cubes in a \( 5 \times 5 \times 5 \) subgrid around each selected cube. Note that SHREC actually merges the isosurface vertices generated by the cubes, not the cubes themselves.

SHREC relies upon some angle tests to determine permissible merging. Before performing such tests, SHREC maps the grid and the computed isosurface locations \( c.isovLoc \) to the regular grid composed of unit cubes.

4.6.1 Ordered Merging

SHREC first merges neighbors of selected 0D feature cubes. SHREC merges these neighbors in three steps. First, for each selected 0D feature cube \( c \), SHREC merges with \( c \) the active cubes which share a facet with \( c \). Next, for each selected 0D feature cube \( c \), SHREC merges with \( c \) the active cubes which share an edge with \( c \). Finally, for each selected 0D feature cube \( c \), SHREC merges with \( c \) the active cubes which share a vertex.
with \( c \). Of course, once an active cube is merged with some selected cube, it is never merged with any other cube.

SHREC next merges neighbors of selected 1D feature cubes. The procedure is similar to the one for 0D feature cubes. SHREC first merges active cubes which share a facet with a selected 1D feature cube, then merges active cubes which share an edge with a selected 1D feature cube, and finally merges active cubes which share a vertex with a selected 1D feature cube.

### 4.6.2 Merging Tests

For both the 0D feature cube and 1D feature cube merging, SHREC applies six different tests to avoid creating very thin triangles or flipping triangles or violating manifold conditions. Consider a cube \( \tilde{c} \) which SHREC would like to merge with a selected cube \( c \). In order for a cube \( \tilde{c} \) to merge with a selected cube \( c \), cubes \( \tilde{c} \) and \( c \) should satisfy the following conditions:

1. Some isosurface edge connects \( \tilde{c} \) to \( c \).

2. Merging \( \tilde{c} \) with \( c \) does not create an edge which is contained in four isosurface triangles or quadrilaterals (manifold condition).

3. If \( \tilde{c} \) is connected to three different selected 1D feature cubes, \( c \), \( c' \) and \( c'' \), then \( c \) should lie “between” \( c' \) and \( c'' \). (See Figure 4.15(a) for a 2D illustration.)

4. Merging \( \tilde{c} \) with \( c \) does not create a triangle with very small angles.

5. Merging \( \tilde{c} \) with \( c \) does not “flip” a triangle, creating a “fold” in the isosurface.

6. If cubes \( \tilde{c} \) and \( \tilde{c}' \) share an ambiguous facet, then \( \tilde{c} \) should merge with \( c \) if and only if \( \tilde{c}' \) merges with \( c \).
We describe these conditions in more detail below.

First, SHREC checks that the isosurface vertex in \( \tilde{c} \) is connected to the isosurface vertex in \( c \). A cube \( \tilde{c} \) is connected to a selected cube \( c \) if \( \tilde{c} \) shares an active facet or edge with \( c \) or some cube \( \tilde{c}' \) shares an active facet or edge with \( \tilde{c} \) and is merged with \( c \). If \( \tilde{c} \) is not connected with \( c \), then \( \tilde{c} \) is not merged with \( c \).

Second, SHREC checks for some manifold violations that could be caused by merging \( \tilde{c} \) with \( c \). For each such selected cube \( c' \neq c \) which is connected to \( \tilde{c} \), SHREC checks if \( c' \) is connected to \( c \). Let \( c' \) be a selected cube which is connected to \( c \) and \( \tilde{c} \). Each active edge \( e \) of \( \tilde{c} \) is dual to an isosurface quadrilateral with a vertex in \( \tilde{c} \). Let \( \tilde{c}_1, \tilde{c}_2, \) and \( \tilde{c}_3 \) be the three other cubes containing \( c \). If some \( \tilde{c}_i \) merges with \( c \) and another \( \tilde{c}_j \) merges with \( c' \), then merging \( \tilde{c} \) with \( c \) collapses this quadrilateral to a triangle or an edge. However, if no isosurface quadrilateral dual to an active edge of \( \tilde{c} \) has vertices in \( \tilde{c}, c \) and \( c' \), then merging \( \tilde{c} \) with \( c \) creates a non-manifold edge from \( c \) to \( c' \) with four incident polygons. (See Figure 4.15(b).) If some selected \( c' \) is connected to both \( c \) and \( \tilde{c} \), but no isosurface quadrilateral has vertices in \( \tilde{c}, c \) and \( c' \), then SHREC does not merge \( \tilde{c} \) with \( c \).

Third, SHREC checks that if \( \tilde{c} \) is connected to three selected 1D feature cubes, \( c, c' \) and \( c'' \), then \( c \) is between \( c' \) and \( c'' \). If not, then mapping \( \tilde{c} \) to \( c \) could create a triangle with a small angle. (See Figure 4.15 for a 2D illustration.) Let \( c, c' \) and \( c'' \) be grid cubes with grid indices \( (x_0, x_1, x_2), (x'_0, x'_1, x'_2) \) and \( (x''_0, x''_1, x''_2) \), respectively. If \( (x''_0, x''_1, x''_2) \) is contained in the box with corners \( (x_0, x_1, x_2) \) and \( (x'_0, x'_1, x'_2) \) and does not lie on any of the eight corners of that box, then we say that \( c'' \) lies between \( c \) and \( c' \). If \( \tilde{c} \) is connected to selected 1D feature cubes \( c, c' \) and \( c'' \), and \( c'' \) lies between \( c \) and \( c' \) or \( c' \) lies between \( c \) and \( c'' \) then SHREC does not merge \( \tilde{c} \) with \( c \).

In the fourth and fifth tests, SHREC checks whether mapping \( \tilde{c} \) to \( c \) creates an isosurface triangle with a small angle or “flips” an isosurface triangle, creating a “fold” in the
isosurface. These tests are a little more complicated than the others and are explained in the next section.

Finally, SHREC checks whether cube \( \tilde{c} \) has more than one isosurface vertex. If cube \( \tilde{c} \) has more than one isosurface vertex, then it has at least one ambiguous facet. (A facet is ambiguous if two diagonally opposite vertices have scalar value above the isovalue while the other two vertices have scalar value below the isovalue.) Let \( \tilde{c}' \) be a cube sharing an ambiguous facet with \( \tilde{c} \). If cube \( \tilde{c} \) has two isosurface vertices and \( \tilde{c}' \) is not merged with \( c \), then \( \tilde{c} \) is not merged with \( c \).

### 4.6.3 Distortion Tests

Let \( q \) be a quadrilateral dual to some active edge of \( \tilde{c} \). Let \( \tilde{w} \) be the vertex of \( q \) which is generated by \( \tilde{c} \). Quadrilateral \( q \) can either be triangulated by adding a diagonal incident on \( \tilde{w} \) or by adding a diagonal connecting the neighbors of \( \tilde{w} \) in \( q \). Triangulating \( \tilde{w} \) by adding a diagonal incident on \( \tilde{w} \) places more restrictions on possible locations of \( \tilde{w} \). Since SHREC does not know how \( q \) will be triangulated, it assumes this more restrictive triangulation.

The diagonal of \( q \) incident on \( \tilde{w} \) splits \( q \) into two triangles. SHREC checks whether mapping \( \tilde{c} \) to \( c \) will severely distort either of those triangles. Let \( \tilde{c}, \tilde{c}' \) and \( \tilde{c}'' \) be the cubes containing the triangle vertices. SHREC only checks triangles where \( \tilde{c}' \) and \( \tilde{c}'' \) are not covered or selected.

Let \( \tilde{w}, \tilde{w}' \) and \( \tilde{w}'' \) be the vertices generated by \( \tilde{c}, \tilde{c}' \) and \( \tilde{c}'' \). Let \( w \) be the vertex generated by selected cube \( c \). In the fourth test, SHREC computes \( \angle(w, \tilde{w}', \tilde{w}'') \) and \( \angle(w, \tilde{w}'', \tilde{w}') \). If \( \angle(w, \tilde{w}', \tilde{w}'') \) or \( \angle(w, \tilde{w}'', \tilde{w}') \) is less than 5°, then SHREC does not map \( \tilde{c} \) to \( c \).

The fifth test is composed of two different parts. In the first part, SHREC checks whether mapping \( \tilde{w} \) to \( w \) significantly changes the normal of triangle \( (\tilde{w}, \tilde{w}', \tilde{w}'') \). If the angle between the normal of \( (w, \tilde{w}', \tilde{w}'') \) is less than 30°, then the triangle passes this test.
In the second part, SHREC checks the orientation of \((w, \tilde{w}', \tilde{w}'')\). Let \(c\) be the grid edge shared by cubes \(\tilde{c}, \tilde{c}'\) and \(\tilde{c}''\). Let \(\pi(w), \pi(\tilde{w}')\) and \(\pi(\tilde{w}'')\) be the orthogonal projection of \(w, \tilde{w}'\) and \(\tilde{w}'',\) respectively, onto a plane \(h\) perpendicular to \(c\). If the orientation of \(\pi(w), \pi(\tilde{w}')\) and \(\pi(\tilde{w}'')\) matches the orientation of \(\tilde{c}, \tilde{c}'\) and \(\tilde{c}''\) around \(c\), then triangle \((w, \tilde{w}', \tilde{w}'')\) passes the orientation test.

To pass the fifth test, SHREC requires that a triangle pass either the normal or the orientation test, not necessarily both tests. The original triangle \((\tilde{w}, \tilde{w}', \tilde{w}'')\) could have a normal which is very far from the true surface normal. In that case, the normal of triangle \((w, \tilde{w}', \tilde{w}'')\) should be far from the normal of triangle \((\tilde{w}, \tilde{w}', \tilde{w}'')\). On the other hand, the projected vertices \(\pi(w), \pi(\tilde{w}')\) and \(\pi(\tilde{w}'')\) could be nearly collinear. In that case, \(\pi(w), \pi(\tilde{w}')\) and \(\pi(\tilde{w}'')\) could have opposite orientation from \(\tilde{c}, \tilde{c}'\) and \(\tilde{c}''\), even though the normal of \((w, \tilde{w}', \tilde{w}'')\) is quite close to the original.

### 4.6.4 Pair Merging

A cube \(\tilde{c}\) may fail to merge with a selected cube \(c\) because of some neighboring cube \(\tilde{c}'\) while \(\tilde{c}'\) fails to merge with \(c\) because of \(\tilde{c}\). Often this “deadlocking” arises when \(\tilde{c}\) and \(\tilde{c}'\) share a common ambiguous facet.

After attempting to merge individual cubes, SHREC tries to merge pairs \((\tilde{c}, \tilde{c}')\) of covered cubes with selected cubes. The elements of the pairs should share a common facet or edge and should both be covered by the selected cube \(c\). SHREC temporarily merges \(\tilde{c}\) with \(c\), and then applies all the above tests to \(\tilde{c}'\). SHREC then temporarily merges \(\tilde{c}'\) with \(c\), and applies all the above tests to \(\tilde{c}\). If both \(\tilde{c}\) and \(\tilde{c}'\) pass the tests, then SHREC merges both of them with \(c\).

As the merging proceeds, it is possible that a cube which was previously unable to merge with any selected cube is now able to merge with some such cube. Thus, SHREC
reapplies the algorithm in Section 4.6.1 and in this section to any remaining covered, unmerged cubes.

### 4.6.5 Extended Merging

The merging of cubes in $3 \times 3 \times 3$ subgrid around each selected cube will clear most but not all of the vertices around sharp features. There are multiple reasons that some vertices near sharp features may remain. First, the location of some vertex outside of $R_c^{3\times3\times3}$ may stop some cube covered by $c$ from merging with $c$. Second, if two selected $c$ and $c'$ are in a $(2,2,0)$ or $(2,2,1)$ or $(2,2,2)$ configuration, then a 1-dimensional feature with endpoints in $c$ and $c'$ can intersect the boundary of the $R_c^{3\times3\times3} \cup R_{c'}^{3\times3\times3}$. Vertices in uncovered cubes can be arbitrarily close to such a 1-dimensional feature. Third, while the selection step avoids most $(3,2,*)$ configurations, it does not avoid all of them. If $c$ and $c'$ are in a $(3,2,*)$ configurations, then a 1-dimensional feature passing through $c$ and $c'$ may contain points outside of $R_c^{3\times3\times3} \cup R_{c'}^{3\times3\times3}$.

To handle such problems, SHREC extends the merging to cubes in a $5 \times 5 \times 5$ subgrid around each selected cube. Merging cubes in a $5 \times 5 \times 5$ subgrid $R_c^{5\times5\times5}$ around a selected cube $c$ can create its own problems of thin or flipped triangles. Thus, SHREC only attempts to merge cubes in $R_c^{5\times5\times5}$ which are potentially near a 1-dimensional feature through $c$.

First, SHREC checks whether any covered cubes are unmerged after the steps in Sections 4.6.1 and 4.6.4. If some cube $\tilde{c}$ is covered by selected cube $c$ but not merged with any cube, SHREC pairs $\tilde{c}$ with any adjacent active cubes $\tilde{c}'$ sharing a vertex with $\tilde{c}$ and attempts to merge the pair $(\tilde{c}, \tilde{c}')$ with $c$. SHREC applies the pair merging procedure in Section 4.6.4 to determine whether to allow the pair $(\tilde{c}, \tilde{c}')$ to merge with $c$. Note that in Section 4.6.4 both cubes in the pair must be contained in $R_c^{3\times3\times3}$ while here only one cube must be contained in $R_c^{3\times3\times3}$.
It is possible that the interaction of vertices in three cubes prevents a covered cube from merging with a selected cube. SHREC considers triples of unmerged cubes which share a common edge $c$. At least one cube in the triple must be in $R_{c}^{3 \times 3 \times 3}$. The triple merging procedure is the same as the pair merging procedure described in Section 4.6.4. Its description is omitted.

Next, SHREC tries to merge cubes which lie at the intersection $\partial R_{c}^{3 \times 3 \times 3} \cap \partial R_{c'}^{3 \times 3 \times 3}$ of two $3 \times 3 \times 3$ subgrids around two selected cubes $c$ and $c'$. More specifically, SHREC attempts to merge a cube $\tilde{c}$ with a selected cube $c$ if some facet of $\tilde{c}$ lies on the boundary of $R_{c}^{3 \times 3 \times 3}$ while some other facet of $\tilde{c}$ lies on the boundary of $R_{c'}^{3 \times 3 \times 3}$. Note that such a cube is contained in $R_{c}^{5 \times 5 \times 5}$. SHREC applies all the tests in Section 4.6.2 to determine whether to merge $\tilde{c}$ with $c'$.

Finally, SHREC tries to merge pairs of cubes $(\tilde{c}, \tilde{c}')$ which lie at the intersection $\partial R_{c}^{3 \times 3 \times 3} \cap \partial R_{c'}^{3 \times 3 \times 3}$. Both $\tilde{c}$ and $\tilde{c}'$ must have facets which lie on $R_{c}^{3 \times 3 \times 3}$ and $R_{c'}^{3 \times 3 \times 3}$. SHREC applies the pair merging procedure described in Section 4.6.4 to determine whether to allow the pair $(\tilde{c}, \tilde{c}')$ to merge with $c$.

### 4.7 Constructing isosurface triangles and quadrilaterals

The last step in SHREC is the construction of isosurface triangles and quadrilaterals. SHREC first applies the Manifold Dual Marching Cubes algorithm described in [74] to the full resolution grid (no merged grid cubes) to construct a set of isosurface quadrilaterals. The algorithm is essentially the same as the Nielson’s Dual Marching Cubes algorithm [55] but with some small changes to the isosurface in certain ambiguous cases. We briefly describe Manifold Dual Marching Cubes.
Each grid vertex with scalar value below the isovalue receives a negative (“−”) label. Each grid vertex with scalar value equal to or above the isovalue receives a positive (“+”) label. A grid edge is active if it has one positive endpoint and one negative endpoint.

The eight labels on the eight vertices of a cube $c$ determine the configuration of the cube. There are $2^8 = 256$ possible configurations. A lookup table stores the number of isosurface vertices for each configuration $\kappa$. Cubes with configuration $\kappa$ contain the corresponding number of isosurface vertices. For each configuration $\kappa$, the lookup table also stores an assignment of active edges to isosurface vertices. If a cube has configuration $\kappa$, then the isosurface quadrilateral dual to active edge $c$ is incident on the isosurface vertex assigned to edge $c$.

As with all dual contouring algorithms, Manifold Dual Marching Cubes constructs one isosurface quadrilateral dual to each active grid edge. The four vertices of the isosurface quadrilateral $q$ lie in the four grid cubes containing the active edge $c$. Using the lookup table, we determine the four isosurface vertices in the four grid cubes which form the vertices of $q$ and store $q$ by its four vertices.

When two cubes, $c$ and $c'$, share an ambiguous facet and each has only one isosurface vertex, the procedure in the previous paragraph will generate a non-manifold edge contained in four quadrilaterals. In this case, we replace the configurations of $c$ and $c'$ with complementary configurations where all the positive and negative vertex labels are flipped. The complementary configurations generate two isosurface vertices in each cube, avoiding the non-manifold edge. Because the initial configuration for each cube generates only one isosurface vertex, each cube has only one ambiguous facet, the facet shared by $c$ and $c'$. Thus, using the complementary configurations for $c$ and $c'$ does not create any “cracks” in the connection of $c$ and $c'$ to other cubes.

After constructing a full resolution isosurface mesh, SHREC collapses mesh quadrilateral using the previously constructed grid cube merging. For each selected cube $c$, let $w_c$
be the isosurface vertex generated by \( c \). If \( c \) generates more than one isosurface vertex, arbitrarily choose one to be \( w_c \).

For each isosurface vertex of a non-selected cube \( c' \), if \( c' \) merges a selected cube \( c \) let \( M(w) \) equal \( w_c \). For each isosurface vertex \( w \) of a selected cube \( c \), let \( M(w) \) equal \( w_c \). For every other isosurface vertex \( w \), let \( M(w) \) equal \( w \).

Replace each quadrilateral \((w, w', w'', w''')\) by \((M(w), M(w'), M(w''), M(w'''))\). Replacing the vertices of these quadrilaterals will map some of the quadrilaterals to triangles and collapse others to edges, vertices or pairs of edges. Remove the quadrilaterals which are mapped to vertices or edges.

For each selected cube \( c \), assign \( w_c \) the isosurface vertex location computed for \( c \) as described in Section 4.4. For each unselected, unmerged cube containing a single isosurface vertex \( w \), also assign \( w \) the location computed for \( c \) as described in Section 4.4. The remaining isosurface vertices are in unselected, unmerged cubes which contain multiple isosurface vertices. For each such isosurface vertex \( w \), let \( E_w \) be the cube edges dual to the quadrilaterals incident on \( w \). Using linear interpolation, compute the intersection point of the isosurface and each edge \( c \in E_w \). Locate \( w \) at the centroid of the intersection points.

Manifold Dual Marching Cubes generates an isosurface which is always a manifold. However, the merging of isosurface vertices can create non-manifold edges. In the experiments described in Section 5.3, SHREC almost always produced a manifold isosurface. If a manifold isosurface is required, then tests such as in [63] can be added to the merging step to ensure that the resulting surface is a manifold.

### 4.7.1 Grid Spacing

The distortion tests for cube merging (Section 4.6.3) depend upon two angle parameters, one for testing triangle angles and one for testing changes in normals. If the grid is not spaced by the same length in the \( x \), \( y \) and \( z \) directions, these tests will be biased in certain
directions. To avoid this bias, we map the isosurface vertex locations to locations in a grid with uniform $1 \times 1 \times 1$ spacing before applying these tests.

In gradient selection, we use a $2 \times 2 \times 2$ region around cube $c$ to determine the selected gradients. Of course, this region size is only for a grid with uniform $1 \times 1 \times 1$ spacing. If the grid spacing is $\delta_x \times \delta_y \times \delta_z$, we use a region of size $(2/\delta_x, 2/\delta_y, 2/\delta_z)$ around cube $c$.

In sharp cube selection (Section 4.5), we use an angle test to avoid selecting a cube whose isosurface vertex may create a thin triangle with vertices in other selected cubes. For that test, we use the vertex locations in the original grid.

### 4.8 Parameters

Algorithm SHREC has two major parameters, one determining the number of large singular values in matrix $A$ and the second determining the size of the $k \times k \times k$ subgrid from which gradients are selected around each cube (Chapter 3). The first parameter is a value $\varepsilon$ between 0 and 1. As described in Section 4.4, a singular value $\sigma_i$ of matrix $A$ is called “large” if $\sigma_i/\sigma_1$ is greater than or equal to $\varepsilon$. The number of large singular values determines whether a vertex is on a sharp feature or a smooth subgrid of the isosurface. To the best of our knowledge, all algorithms which reconstruct surfaces with sharp features require some parameter to distinguish the sharp features from the smooth regions of the surface.

The second parameter is an odd integer $k \geq 3$. Each cube $c$ uses gradients from a $k \times k \times k$ subgrid around the cube in selecting gradients for computing $c$.$\text{ISOVLOC}$. The size of $k$ depends upon the input data. If correct gradients are provided at each grid vertex, then $k$ should equal 3 for a $3 \times 3 \times 3$ subgrid around each cube. If gradients are computed from correct scalar values using the algorithm in [14], then $k$ should equal 7 for a $7 \times 7 \times 7$
subgrid around each cube. If gradients are computed from CT data using the algorithm in [14], then $k$ should equal 9 for a $9 \times 9 \times 9$ subgrid around each cube.

SHREC uses three other constants: one for the triangle test in selecting vertices, one for the angle test in merging cubes and a second for the normal test in merging cubes. In the triangle test, if selecting cube $c$ would possibly create a triangle between three selected cubes with angle greater than $140^\circ$, cube $c$ is not selected. In the angle test, if merging cube $\tilde{c}$ with $c$ would create a triangle with angle less than $5^\circ$, then cube $\tilde{c}$ is not selected. In the normal test, if merging $\tilde{c}$ with $c$ would change the normal of some triangle $(\tilde{w}, \tilde{w}', \tilde{w}'')$ by less than $30^\circ$, then merging $\tilde{c}$ with $c$ does not distort triangle $(\tilde{w}, \tilde{w}', \tilde{w}'')$.

Because SHREC is based on the regular grid, the constants used in all three of these tests do not depend upon the input data and should work well for any scalar fields. Note that SHREC maps the input grid and the computed isosurface locations $c.i SovLoc$ to the regular grid composed of unit cubes before applying the angle or normal tests.
Figure 4.14: 2D illustration of order of cube selection based on cube indices mod 6. (a) Green squares are (0, 0) mod 6. (b) Blue squares are (k, 0) mod 6 or (0, k) mod 6 for k = 1, ..., 5. Squares marked 1 are selected first, then squares marked 2 and then squares marked 3. (c) Yellow squares are (k_a, k_b) mod 6 for k_a = 1, ..., 5 and k_b = 1, ..., 5. Squares marked 1 are selected first, then squares marked 2, then 3, 4 and 5.
Figure 4.15: (a) Isosurface in yellow square is connected to three selected cubes. Yellow square should merge with the middle green square, not with either of the blue squares. (b) Merging the red vertex with either of the yellow vertices will cause the yellow edge to be contained in four polygons.
Chapter 5: Experimental Results

In this chapter we discuss the experimental results of algorithms SHREC and RELIGRAD.

5.1 Synthetic Scalar and Gradient Data

To measure the quality of our reconstruction, we used a number of synthetic scalar and gradient datasets. (See Figure 5.1.) First, we provide some information regarding the creation of the synthetic datasets.

5.1.1 Cube Dataset

Given a point \( p \), let \( f^{L_1}_p(q) \) be the \( L_1 \) distance from \( p \) to \( q \). A Cube dataset is generated by sampling \( f^{L_1}_p \) and its gradients on vertices of the regular grid. Level sets of \( f^{L_1}_p \) are cubes whose edges are parallel to the coordinate axes and whose facets are orthogonal to those axes. By rotating \( f^{L_1}_p \) around \( p \), we can generate a scalar field whose level sets are cubes that are not axis-aligned.

5.1.2 TwoCube Dataset

By taking the minimum of two (rotated) scalar fields, \( f^{L_1}_p \) and \( f^{L_1}_{p'} \), centered around two different points, \( p \) and \( p' \), respectively, we get a scalar field whose level sets are the
Figure 5.1: Isosurfaces constructed from synthetic test datasets.

Figure 5.2: Polygonal meshes representing level sets.
boundaries of the unions of the two cubes. (See Figure 5.1(a).) We call a regular grid sampling of such a scalar field and its gradients a TwoCubes dataset. The isosurface of a TwoCubes dataset has twenty 0-dimensional features, fourteen of which are cube corners and six of which are “saddle points” where the two boundaries of two cubes meet. We use the TwoCubes datasets with various rotations as test sets for the reconstruction of 0-dimensional features.

5.1.3 Annuli Dataset

Let \( \ell \) be a line. Let \( f_{Cyl}^\ell(q) \) be the Euclidean distance from point \( q \) to \( \ell \). The level sets of \( f_{Cyl}^\ell(q) \) are infinite cylinders around \( \ell \). Let \( f_{Cyl\times2}^{\ell,r}(q) \) equal \( |f_{Cyl}^\ell(q) - r| \). The level sets of \( f_{Cyl\times2}^{\ell,r}(q) \) are pairs of infinite cylinders at equal distances from the cylinder of radius \( r \) around \( \ell \). Let \( f_{Pl\times2}^{p,\ell,r}(q) \) equal the (unsigned) distance from \( q \) to the plane that contains point \( p \) and is orthogonal to line \( \ell \). Let \( f_{Ann}^{p,\ell,r}(q) \) be the maximum of \( f_{Cyl\times2}^{\ell,r}(q) \) and of \( f_{Pl\times2}^{p,\ell}(q) \). The level sets are the boundaries of thickened annuli. (See Figure 5.1(b).)

The width of the thickened annuli defined by \( f_{Ann}^{p,\ell,r}(q) \) equals their height. We can adjust change the difference between the width and height by adding constants to \( f_{Cyl\times2}^{\ell,r}(q) \) or \( f_{Pl\times2}^{p,\ell}(q) \). Let \( f_{Ann}^{p,\ell,r,c}(q) \) be the maximum of \( f_{Cyl\times2}^{\ell,r}(q) + c_1 \) and of \( f_{Pl\times2}^{p,\ell}(q) + c_2 \). If \( c_1 \) is greater than \( c_2 \), then the height is \( c_1 - c_2 \) units greater than the width. If \( c_2 \) is greater than \( c_1 \), then the width is \( c_2 - c_1 \) units greater than the height.

5.1.4 Flange dataset

Define \( f_{Fr,c}^{p,\ell,r,c}(q) \) as the minimum of \( f_{Ann}^{p,\ell,r,c,0}(q) \) and \( f_{Ann}^{p,\ell,r,0,c}(q) \). The level sets of \( f_{Fr,c}^{p,\ell,r,c}(q) \) are the boundaries of the unions of two thickened annuli. (See Figure 5.1(c).) One annuli has height \( c \) units greater than its width while the other has width \( c \) units greater than its height. A Flange dataset is a regular grid sampling of \( f_{Fr,c}^{p,\ell,r,c}(q) \) and its gradients.
The Flange dataset has no 0-dimensional features. We use the Flange datasets as test sets for the reconstruction of 1-dimensional features.

5.1.5 Cone and Cannon Dataset

We use two other types of datasets to test our algorithm on dihedral angles other than 90°. A cone is defined by an apex \( p \), an axis direction \( \zeta \), and an angle \( \alpha < 90° \). The cone is the union of all the rays from \( p \) forming angle \( \alpha \) with \( \zeta \). Let \( \ell \) be the directed line through \( p \) with direction \( \zeta \), i.e. the cone axis. Let \( \pi_\ell(q) \) be the projection of point \( q \) on \( \ell \). Let \( D_1(q) \) be the distance from \( p \) to \( \pi_\ell(q) \) and let \( D_2(q) \) be the signed distance from \( \pi_\ell(q) \) to \( p \). (The distance is negative if \( \pi_\ell(q) - p \) points in the opposite direction from \( \zeta \).) Define \( f_{\text{Cone}}^{p,\zeta,\alpha}(q) \) as \( \cos(\alpha)D_1(q) - \sin(\alpha)D_2(q) \). Function \( f_{\text{Cone}}^{p,\zeta,\alpha}(q) \) is the signed distance from point \( q \) to its orthogonal projection on the cone, when that orthogonal projection exists. The level sets of \( f_{\text{Cone}}^{p,\zeta,\alpha}(q) \) are cones with axis \( \ell \).

We truncate the cone by defining a plane orthogonal to \( \ell \). Let \( p' \) be a point on ray \( p + t\zeta \) where \( \zeta \) is a unit vector. Let \( f_{\text{Pl}}^{p',\zeta}(q) \) be equal \((q - p') : \zeta\), the (unsigned) distance from \( q \) to the plane containing \( p' \) orthogonal to \( \zeta \). Define

\[
 f_{\text{Fr}}^{p,\zeta,\alpha,\alpha'}(q) = \max \left(f_{\text{Cone}}^{p,\zeta,\alpha}(q), f_{\text{Pl}}^{p',\zeta}(q), f_{\text{Pl}}^{p',-\zeta}(q)\right).
\]

The level sets of \( f_{\text{Fr}}^{p,\zeta,\alpha,\alpha'}(q) \) are frustra. (See Figure 5.1(d).)

The level sets of \( f_{\text{Fr}}^{p,\zeta,\alpha,\alpha'}(q) \) have two closed curves forming 1-dimensional features. The dihedral angle on one of these curves is \( 90° + \alpha \) while the dihedral angle on the other is \( 90° - \alpha \). To generate a field with dihedral angles of \( 90° + \alpha \) or \( 90° - \alpha \), but not both, we modify \( f_{\text{Fr}}^{p,\zeta,\alpha,\alpha'}(q) \) so that the level sets are capped by spheres at one end.
Define

\[ f_{SC}^{p, \xi, \alpha, p'}(q) = \begin{cases} |q - p'| & \text{if } \angle(q, p', p) \leq 90^\circ - \alpha, \\ f_{C}^{p, \xi, \alpha}(q) & \text{if } \angle(q, p', p) > 90^\circ - \alpha, \end{cases} \]

\[ f_{TC}^{p, \xi, \alpha, p'}(q) = \max \left(f_{SC}^{p, \xi, \alpha, p'}(q), f_{P}^{p'}(q), f_{P'}^{p, \xi, \alpha, p'}(q)\right). \]

A level set of \( f_{SC}^{p, \xi, \alpha, p'}(q) \) is a cone with its tip smoothed. A level set of \( f_{TC}^{p, \xi, \alpha, p'}(q) \) is a truncated cone with its tip smoothed. The level set has a single closed curve forming a 1-dimensional feature with dihedral angle \( 90^\circ - \alpha \). The \textit{Smooth Tip Cone} dataset is a regular sampling of \( f_{TC}^{p, \xi, \alpha, p'}(q) \) and its gradients. We use Smooth Tip Cone datasets to evaluate reconstruction of dihedral angles less than \( 90^\circ \).

Let \( p' \) and \( p'' \) be points on the ray \( p + t\xi \) where \( |p - p'| \) is less than \( (1 - \sin(\alpha))|p - p''| \). Define

\[ f_{X}^{p, \xi, \alpha, p''}(q) = \begin{cases} |q - p''| & \text{if } \angle(q, p', p) > 90^\circ - \alpha, \\ f_{C}^{p, \xi, \alpha}(q) & \text{if } \angle(q, p', p) \leq 90^\circ - \alpha. \end{cases} \]

\[ f_{Can}^{p, \xi, \alpha, p', p''}(q) = \max \left(f_{X}^{p, \xi, \alpha, p''}(q), f_{P}^{p'}(q), f_{P'}^{p, \xi, \alpha, p''}(q)\right). \]

A level set of \( f_{Can}^{p, \xi, \alpha, p'}(q) \) has the shape of a cannon. The level set has a single closed curve forming a 1-dimensional feature with dihedral angle \( 90^\circ + \alpha \). The \textit{Cannon} dataset is a regular sampling of \( f_{Can}^{p, \xi, \alpha, p'}(q) \) and its gradients. We use Cannon datasets to evaluate reconstruction of dihedral angles greater than \( 90^\circ \).

### 5.1.6 Gradients

Algorithm SHREC requires gradients at the grid vertices. Exact formulas can be given for gradients for each of the scalar fields described in the previous section. However,
scalar data is acquired from scanning devices such as CT scanners, only scalar information is available. Chapter 3, describes an algorithm, Religrad, for constructing a set of reliable gradients from scalar data. SHREC is evaluated both on gradients computed by exact formulas and on the gradients produced by Religrad from the scalar data.

5.1.7 Measurements

For each test scalar field \( f \) and test isovalue \( \sigma \), we constructed a polygonal mesh which represented the level set, \( f^{-1}(\sigma) \). (See Figure 5.2.) The polygonal mesh was designed specifically for each surface to accurately represent the 0-dimensional and 1-dimensional features on the surface.

To compare isosurfaces constructed by different software on different scales, we rescaled every isosurface and polygonal mesh to lie in the unit cube. Our isosurfaces were computed from grids of \( 150 \times 150 \times 150 \) or \( 200 \times 200 \times 200 \) so this rescaled the cube size to about 0.005 units.

**Angular Distance**

Let \( \Sigma_P \) and \( \Sigma_Q \) be two surfaces. The Angular distance is the minimum angle between the normal at \( p \in \Sigma_P \) and any normal of \( \Sigma_Q \) in some suitable neighborhood around \( p \). Angular distance is computed in the extended \( \varepsilon \)-neighborhood between each isosurface and the corresponding polygonal mesh. (See [11], for details.)

The number of triangles whose normals differed by more than 30, 40 or 50 degrees from polyhedral triangles in the extended \( \varepsilon \)-neighborhood around each triangle is also reported. \( \varepsilon \) is set to 0.01 or about twice the cube size for the extended neighborhood.
Figure 5.3: SHREC Flange isosurface. “Sharp” edges (with dihedral angle less than 140°) are marked in red. “Smooth” edges are in dark gray. The magnified region shows a blend of the “sharp” edges with a subset of the “smooth” edges.

Degree test

The degree test as described in [1] is also used to measure errors in reconstructing sharp features. The 1-skeleton of isosurface edges with dihedral angle less than 140° is extracted and we count the number of vertices in the 1-skeleton with degree other than two. For an isosurface $\Sigma$, let $N_{\neq 2}(\Sigma)$ be the number of vertices in the 1-skeleton with degree other than two. Isosurfaces from the Flange, Smooth Tip Cone and Cannon datasets should have zero vertices with degrees other than two. The number of degree errors for these isosurfaces is $N_{\neq 2}(\Sigma)$. Isosurfaces from the TwoCubes datasets should have exactly twenty vertices with degrees other than two. The number of degree errors for these isosurfaces is $|N_{\neq 2}(\Sigma) - 20|$. 
5.2 Experimental results of RELIGRAD

We tested our algorithm on a large number of synthetic datasets with varying spacings and axis rotations, here we show 6 representative elements; Table 5.1 and Figure 5.1.

We tested 4 algorithms apart from the central difference gradients. Algorithm 1 uses pairs of vertices to predict the gradient direction at $v$ and compare this predicted direction to $\hat{n}_v$, where $\hat{n}_v = \hat{g}_v/|\hat{g}_v|$ is the unit vector in the direction of the central difference gradient $\hat{g}_v$. Algorithm 1 uses vertices at edge distance 2 to predict the gradient at $v$. Algorithm 2 is similar to Algorithm 1, but uses vertices at edge distance 3 to predict $\hat{n}_v$. As mentioned in Section 3.2.4, in CT data gradient magnitudes drop of quickly away from the surface boundaries. Algorithm FINDRELIABLE adjusts for CT data by handling the normal direction to the surface boundaries differently. FINDRELIABLE uses one adjacent vertex in the
normal direction. Algorithm RELIGrad extends the FINDRELIABLE gradients and is our final algorithm. (See Section 3.2.3 for details on all the algorithms.)

**Reliable Gradients:** Figure 5.5 shows gradients at grid vertices for grid cubes which intersect the isosurface on a zoomed in section of the *Annulus* dataset. To get the colormap for the gradient vectors, we projected all gradient vectors on to the XY plane, the angle of the resulting vector to the X-Axis is mapped between red-green. Gradients close to X-Axis are red, those perpendicular are green. Figure 5.5(a) shows the known correct gradients at all the grid vertices.
Figure 5.6: Cannon dataset gradients (Curved edge, non 90° angles). (a) correct gradients, (b) central difference gradients, (c) gradients marked correct by FindReliable. (d) gradients at grid vertices marked correct by RELIGRAD. Inaccurate CDiff gradients Figure(b) across the discontinuity are not marked correct, hence not shown.

Figure 5.5(d), shows the central difference (CDiff) gradients at all the grid vertices. The blue inset shows a magnified view: CDiff gradients are inaccurate along discontinuities, compared to the known correct gradients (figure 5.5(a)).

Figure 5.5(b) shows gradients only at vertices marked correct by Algorithm 1, which uses \( n_{\nu''} \) and \( n_{\nu'} \) to predict \( n_{\nu} \), refer to Section 3.2.3.

Figure 5.5(c) shows results from Algorithm 2, which uses \( n_{\nu''''} \), \( n_{\nu'''} \), and \( n_{\nu'} \) to predict \( n_{\nu} \), refer to Section 3.2.3.

Algorithm 1 does not handle curved boundaries and vertices close to the discontinuity are marked correct. Figure 5.5(e) shows gradients marked reliable by RELIGRAD, the cyan inset shows a close-up, the inaccurate gradients are marked unreliable by RELIGRAD and not shown.

Figure 5.6 shows the gradients around the edge of the *Cannon* dataset (Figure 5.6(a) red rectangle shows the location). The dataset is not axis-parallel, the dihedral angle around the curved edge is 120°.

Figure 5.6(a) shows known correct gradients at grid vertices which are intersected by the isosurface. Figure 5.6(b) shows the central difference gradients, at grid vertices of the
Figure 5.7: Effect of uniform noise on cube gradients. Uniform 0.1 noise was added to vertex scalars. (a) central difference gradients on the original cube, (b) central difference gradients computed on the noisy cube. (c) gradients marked correct by RELIGRAD on the original cube. (d) gradients at the vertices marked correct by RELIGRAD on the noisy cube.

Figure 5.6(c) shows gradients at grid vertices marked reliable by FINDRELIABLE. Figure 5.6(d) shows gradients marked correct by RELIGRAD. Figure 5.7 shows the gradients at corner of the cube dataset. Figure 5.7(a) shows the central difference gradients. As expected, the gradients across the discontinuity are erroneous. Figure 5.7(c) shows gradients at grid vertices marked correct by RELIGRAD. The problems of central difference gradients becomes compounded if the scalar values are noisy. We added 0.1 uniform noise to grid vertex scalars of cube. Figure 5.7(b) shows the central difference gradients computed from the noisy scalars, Figure 5.7(d) shows the result from RELIGRAD.

5.2.1 Quantitative Analysis

For the synthetic tests cases, the correct gradients at each grid vertices are known. Consequently, the reliable gradient computation results can be quantitatively compared with them.
**Measuring False Positives**

Table 5.2 shows the maximum angle difference between the known correct gradients and those computed as reliable gradients at each grid vertex $v$. Intuitively, this captures false positives. Large maximum angle would mean poor gradients are being marked as correct.

Algorithm 1 and Algorithm 2 are described in Section 3.2.3. Algorithm 2 which uses vertices at edge distance 3 for $n_v$ has lower angles than Algorithm 1 for all the test cases thus performing better. Figure 5.5(b) (Algorithm 1) showed one particular example where gradients near the discontinuity were marked correct compared to (Algorithm 2) Figure 5.5(c). As expected, Central Difference generates erroneous gradients near the edges.

For the synthetic datasets, FINDRELIABLE described in Section 3.2.3, performs similarly to Algorithm 2. RELIGRAD which extends the FINDRELIABLE gradients by using Algorithm 6 generates a larger maximum angle than FINDRELIABLE.

**Measuring False Negatives**

It is desirable for algorithms which use the reliable gradients results, that the distance of a vertex $v$ with unreliable gradients to another vertex $v_1$ with reliable gradients be “small”.

This can be tracked by finding the maximum of the $L_1$ distance from each vertex $v$ to the closest grid vertex with a reliable gradient near the generated isosurface.

Table 5.3 shows the results. This test intuitively captures false negatives. Large distances mean more vertices with reliable gradients are being marked unreliable.

For Annulus there is a grid vertex at maximum $L_1$ edge distance of 4 from each $v$ when using Algorithm 2. For Cube there is a vertex with correct gradient within $L_1$ edge distance 4 from each $v$ when using Algorithm 2.
When we extend the reliable gradients, the maximum $L_1$ distance decreases to 2 and 3 respectively. Experimentally we found that the distances for Cube is maximum near the corners. (See Figure 5.7(c).)

**Uniform Noise**

To test the effect of noise on RELIGRAD, uniform noise was added to the datasets. Table 5.4 shows the results. With 0.1 uniform noise RELIGRAD performs well. The maximum $L_1$ distance to reliable gradients in the Cube dataset is 5. This distance was achieved at a vertex near the isosurface corner (Figure 5.7(d).) With a high uniform noise of 0.2 and $\alpha_2$ set to 30 degrees, RELIGRAD performs reasonably well.

Figure 5.8: All the central difference gradients on grid vertices of a small portion of the engine cylinder dataset (Figure 3.3 shows the same regions, but only the gradients at vertices intersected by isosurface). The gradient magnitudes are proportional to length of the vectors. On the right we see a magnified section of the cyan rectangle. The grid magnitudes drop off quickly, and the gradient directions become meaningless especially along the Z-axis.

RELIGRAD results on industrial CT data are described in Section 5.5.
Table 5.1: Synthetic test dataset sizes, isovalues, and average statistics on isosurfaces produced by SHREC. Average number of active cubes (Avg Num Act. C), average number of isosurface vertices, average number of isosurface triangles, and average SHREC running time. All isosurface quadrilaterals are triangulated before counting the number of isosurface triangles. Because SHREC merges isosurface vertices, the average number of isosurface vertices is less than the average number of active cubes.

<table>
<thead>
<tr>
<th>Dataset Type</th>
<th>Num Datasets</th>
<th>Grid Size</th>
<th>Isovalue</th>
<th>Avg Num Act. C</th>
<th>Avg Num Iso Vert</th>
<th>Avg Num Iso Tri</th>
<th>Avg Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwoCubes</td>
<td>34</td>
<td>150 × 150 × 150</td>
<td>20.1</td>
<td>25K</td>
<td>22K</td>
<td>45K</td>
<td>0.8 sec</td>
</tr>
<tr>
<td>Flange</td>
<td>39</td>
<td>200 × 200 × 200</td>
<td>10.1</td>
<td>90K</td>
<td>75K</td>
<td>150K</td>
<td>1.8 sec</td>
</tr>
<tr>
<td>Smooth Tip Cone</td>
<td>15</td>
<td>100 × 100 × 100</td>
<td>10.2</td>
<td>4K</td>
<td>3.5K</td>
<td>7K</td>
<td>0.2 sec</td>
</tr>
<tr>
<td>Cannon</td>
<td>15</td>
<td>100 × 100 × 100</td>
<td>0.0</td>
<td>7K</td>
<td>7K</td>
<td>14K</td>
<td>0.2 sec</td>
</tr>
</tbody>
</table>

Figure 5.9: The gradients generated on a portion of the *engine cylinder* dataset. The red rectangle shows the size of a grid cube with spacing (0.27, 0.27, 0.68). (a) Gradients marked reliable by Algorithm 1. (b) Gradients marked reliable by Algorithm 2. (c) Gradients marked reliable by FINDRELIABLE (Algorithm 5). (d) Gradients marked reliable by RELIGRAD.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>CDiff</th>
<th>Algo 1</th>
<th>Algo 2</th>
<th>Find Reliable</th>
<th>Reli Grad</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube</td>
<td>48.3</td>
<td>0.7</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Annulus</td>
<td>44.03</td>
<td>2.79</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>Flange</td>
<td>45.4</td>
<td>2.1</td>
<td>0.04</td>
<td>0.04</td>
<td>9.8</td>
</tr>
<tr>
<td>TwoCube</td>
<td>61.86</td>
<td>6.1</td>
<td>0.0</td>
<td>0.0</td>
<td>19.8</td>
</tr>
<tr>
<td>Cannon</td>
<td>29.7</td>
<td>12.3</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Cone</td>
<td>57.5</td>
<td>1.6</td>
<td>1.11</td>
<td>1.11</td>
<td>13.1</td>
</tr>
</tbody>
</table>

Table 5.2: Maximum Angle difference compared to correct gradients in degrees. $\alpha, \alpha_2$ is set to $20^\circ$. A large number of vertices with high angle difference to the correct gradients, means erroneous gradients are being marked correct. Low angles mean the gradients marked reliable are very close to the correct gradient which is desired outcome.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Algorithm 1</th>
<th>Algorithm 2</th>
<th>RELIGRAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Annulus</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Cube</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Flange</td>
<td>4</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Cannon</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Cone</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>TwoCube</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 5.3: Maximum of $L_1$ distances to closest grid vertices with exact gradients.

### 5.3 Experimental results of SHREC

#### 5.3.1 Isosurface Reconstruction on Synthetic Data

We tested SHREC on 39 Flange datasets with 39 different orientations and 34 TwoCubes datasets with 34 different orientations. Table 5.1 presents dataset sizes, isovalues, and average isosurface sizes. We tested SHREC both on gradients produced by exact formulas and on gradients produced by Religrad. We set the singular value threshold $\varepsilon$ to 0.1, and used $9 \times 9 \times 9$ subgrids for gradient selection. We measured the oriented angle distance
Table 5.4: Results after adding 0.1 and 0.2 uniform noise to the datasets using RELIGRAD. To handle 0.2 noise we set $\alpha_2$ to be 30 degrees. MaxAngleDiff, measures the maximum angle difference between known correct gradients and those marked correct by RELIGRAD. DistToGrad measures the maximum of the $L_1$ distance of the unreliable vertices to vertices marked correct by RELIGRAD.

<table>
<thead>
<tr>
<th>dataset</th>
<th>noise</th>
<th>$\alpha_2$</th>
<th>Max AngleDiff</th>
<th>Dist ToGrad</th>
</tr>
</thead>
<tbody>
<tr>
<td>cube</td>
<td>0.1</td>
<td>20°</td>
<td>7.5°</td>
<td>5</td>
</tr>
<tr>
<td>annulus</td>
<td>0.1</td>
<td>20°</td>
<td>8.1°</td>
<td>3</td>
</tr>
<tr>
<td>cone</td>
<td>0.1</td>
<td>20°</td>
<td>12.7°</td>
<td>3</td>
</tr>
<tr>
<td>cube</td>
<td>0.2</td>
<td>30°</td>
<td>12.8°</td>
<td>7</td>
</tr>
<tr>
<td>annulus</td>
<td>0.2</td>
<td>30°</td>
<td>14.1°</td>
<td>5</td>
</tr>
<tr>
<td>cone</td>
<td>0.2</td>
<td>30°</td>
<td>19°</td>
<td>6</td>
</tr>
</tbody>
</table>

between the SHREC isosurfaces and the polygonal meshes representing the level sets. We also counted the number of isosurface triangles whose normal differences from the polyhedral mesh were above 30°, 40° and 50°. Finally, we measured the number of degree errors in the 1-skeleton of sharp isosurface edges.

Figure 5.3 shows a Flange isosurface produced by SHREC using exact gradients. The magnified region shows edges with dihedral angle less than $140°$ in red blended with a subset of the “non-sharp” edges.

Figure 5.4 shows a TwoCubes isosurface and isosurface edges constructed by SHREC. SHREC reproduces the 0-dimensional features with single mesh vertices and 1-dimensional features with a sequence of mesh edges with dihedral angle $90°$.

With exact gradients, all the Flange isosurfaces had oriented angle distance under 15 degrees from the polygonal meshes. They also had no degree errors in the 1-skeletons of sharp isosurface edges, i.e. all the vertices in the 1-skeletons had degree two.

Figure 5.11 displays oriented angle distance and degree error information for the 39 Flange isosurfaces when gradients were produced using Religrad.
Figure 5.10: *Engine Cylinder* dataset. (a) a slice of the original CT image, note the streaking artifacts introduced during the scanning process. The inset shows the original machine part. Figure (b), (c), shows the sharp mesh, and the extracted sparse feature points.

Figures 5.11(a) and 5.11(c) shows that, only one isosurface had oriented angle distance greater than 50° and more than half had oriented angle distance less than 40°. No isosurfaces had more than 5 triangles with angle distance greater than 40° and no isosurfaces had more than 15 triangles with angle distance greater than 30°.

Figure 5.11(b) shows the 1-skeleton degree errors for the 39 Flange isosurfaces when gradients were produced using Religrad. Thirteen isosurfaces had no degree errors. No isosurface had more than fourteen degree errors.

Figure 5.12 displays angle distance and degree error information for the 34 TwoCubes isosurfaces produced by SHREC. Figure 5.12(a) shows the oriented angle distance using exact gradients and Religrad gradients. As with the exact gradients in the Flange datasets, the angle distance is very low (less than 5°) for all the TwoCubes isosurfaces produced using exact gradients. For most isosurfaces produced using Religrad gradients, the angle distance is still low (under 15°), although six isosurfaces have angle distances above 30°. The high angle distances indicate errors in the SHREC reconstruction of sharp features.
Figure 5.11: Results of SHREC using exact gradients and Religrad gradients on 39 Flange datasets. (a) Oriented angle distances between Flange isosurfaces (exact and Religrad gradients) and polygonal mesh. (b) Number of degree errors in the Flange 1-skeletons of sharp edges (Religrad gradients). (SHREC using exact gradients produces no degree errors.) (c) Number of Flange isosurface triangles with normal difference to polygonal mesh above 30, 40 and 50 degrees.
Figure 5.12: Results of SHREC using exact gradients and Religrad gradients on 34 TwoCubes datasets. (a) Oriented angle distances between TwoCubes isosurfaces (exact and Religrad gradients) and polygonal mesh. (b) Number of degree errors in the TwoCubes 1-skeletons of sharp edges (Religrad gradients). (SHREC using exact gradients produces no degree errors.) (c) Number of TwoCubes isosurface triangles with normal difference to polygonal mesh above 30, 40 and 50 degrees.
Figure 5.13: SHREC Cannon and Smooth Tip Cone isosurfaces (Religrad gradients.)
(a) Cannon isosurface. 1-dimensional feature has dihedral angle 120°. (b) Smooth Tip Cone isosurface. 1-dimensional feature has dihedral angle 60°.

Figure 5.12(c) gives the number of TwoCubes Religrad isosurface triangles whose normal differences to the polyhedral mesh are above 30°, 40° and 50°. No isosurface has more than 10 triangles with normal difference above 30°.

With exact gradients, the 34 TwoCubes isosurfaces had no 1-skeleton degree errors, i.e. they all had exactly 20 vertices with degree three in the 1-skeleton of their sharp edges. Figure 5.12(b) shows the degree errors in the 1-skeleton of sharp edges when gradients were produced using Religrad. 29 out of the 34 TwoCubes isosurfaces had no degree errors. Of the 5 isosurfaces with degree errors, the maximum number of degree errors was four.

To measure the dihedral angles other than 90°, we ran SHREC on 15 Cannon datasets with 15 different orientations and 14 Smooth Tip Cone datasets with 14 different orientations. The 1-dimensional features in the Cannon level sets had dihedral angles of 120°. The 1-dimensional features in the Smooth Tip Cone level sets had dihedral angles of 60°. As expected, SHREC produced more errors than on the Flange or TwoCubes datasets, although it still did quite well.
Figure 5.14: Results of SHREC on 15 Cannon and 14 Smooth Tip Cone datasets. (a) Number of degree errors on 1-skeleton of Cannon isosurfaces constructed using Religrad gradients. (Cannon isosurfaces constructed using exact gradients had no degree errors.) (b) Number of degree errors on 1-skeleton of Smooth Tip Cone isosurfaces constructed using Religrad gradients (green) and exact gradients (red).

Figure 5.13 shows Cannon and Smooth Tip Cone isosurfaces produced by SHREC using Religrad gradients. In each, the 1-dimensional is represented by a sequence of isosurface mesh edges with the appropriate dihedral angles.

Figure 5.14 gives the number of degree errors in the 1-skeleton of sharp edges for Cannon and Smooth Tip Cone isosurfaces. SHREC using exact gradients produced no degree errors on the 15 Cannon datasets. SHREC using Religrad produced degree errors on only four Cannon datasets. No SHREC Cannon Religrad isosurface had more than two degree errors. SHREC using exact gradients produced degree errors in only two of the Smooth Tip Cone datasets, one with two errors and one with four. SHREC using Religrad gradients produced degree errors on 7 out of the 14 Smooth Tip Cone datasets. The maximum number of degree errors in any Smooth Tip Cone isosurface was 4.
Figure 5.15: Results of MergeSharp on 39 Flange and 34 TwoCubes datasets (exact gradients). (a) Oriented angle distance to polygonal Flange meshes. (b) Degree errors in 1-skeleton of sharp edges of Flange datasets. (c) Oriented angle distance to polygonal TwoCubes meshes. (d) Degree errors in 1-skeleton of sharp edges of TwoCubes datasets. (f) Number of triangles with normal difference to polygonal Flange mesh above 30, 40 and 50 degrees. (No Flange isosurface triangles have normal difference above 50°.) (e) Number of triangles with normal difference to polygonal TwoCubes mesh above 30, 40 and 50 degrees.
<table>
<thead>
<tr>
<th>Software</th>
<th>Algorithm</th>
<th>URL</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMC (IsoEx)</td>
<td>Extended Marching Cubes</td>
<td><a href="http://www.graphics.rwth-aachen.de/software">www.graphics.rwth-aachen.de/software</a></td>
<td>[46]</td>
</tr>
<tr>
<td>EMCpoly</td>
<td>Extended Marching Cubes</td>
<td>web.cse.ohio-state.edu/research/graphics/isotable</td>
<td></td>
</tr>
<tr>
<td>PolyMender</td>
<td>Dual Contouring</td>
<td><a href="http://www.cse.wustl.edu/taoju/code/polymender.htm">www.cse.wustl.edu/taoju/code/polymender.htm</a></td>
<td>[40, 41, 64]</td>
</tr>
<tr>
<td>SingularCocone</td>
<td>Singular Cocone</td>
<td>web.cse.ohio-state.edu/tamaldey/cocone.html</td>
<td>[19, 25, 24]</td>
</tr>
<tr>
<td>MergeSharp</td>
<td>MergeSharp</td>
<td>web.cse.ohio-state.edu/research/graphics/isotable</td>
<td>[1, 13]</td>
</tr>
<tr>
<td>SHREC</td>
<td>SHREC</td>
<td>web.cse.ohio-state.edu/research/graphics/isotable</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.5: Surface reconstruction software. EMC is a sample implementation of Extended Marching Cubes. EMCpoly is a modification of EMC which creates Two Cubes and Flange isosurfaces. PolyMender is an implementation of dual contouring for fixing polygonal meshes. SingularCocone is an implementation of a Voronoi based algorithm for surface reconstruction from point clouds. The algorithm handles 0 and 1 dimensional features including non-manifold features. MergeSharp is an implementation of a dual contouring reconstruction algorithm which uses cube merging to obtain better reconstruction around sharp features. SHREC is an implementation of the algorithm in this paper.

5.4 Comparison with Other Algorithms

We compared SHREC with software implementations of four other algorithms: MergeSharp [1, 13], PolyMender [40], EMC (Extended Marching Cubes) [46] and SingularCocone [25]. Table 5.5 contains descriptions of the software.

MergeSharp [1] is a simplified version of SHREC. We briefly describe the differences between SHREC and MergeSharp. First, MergeSharp uses gradients from immediate neighbors to compute the location of isosurface vertex. Algorithm SHREC selects gradients from a larger area than just the immediate neighbors. Second, algorithm SHREC tries to locate the isosurface vertex inside the generating cube whenever possible. Third, MergeSharp has a very simple selection step which often leads to bad “cover”. (Figure 4.2.) Finally, MergeSharp merges cube c with the first selected cube adjacent to c. Poor “coverage” of sharp edges often leads to distorted triangles.
Figure 5.16: (a) Notch in a MergeSharp Flange isosurface. (b) Dimples in a MergeSharp Cone isosurface (left). Compare with smooth SHREC isosurface (right).

Input to both SHREC and MergeSharp is a regular grid sampling of a scalar field and the gradients at the grid vertices. Input to the software implementations of the other algorithms is very different from input to SHREC or inputs to each other, so one should be extremely careful in make comparisons between these algorithms based on the results presented here. For instance, on average, EMC has fewer degree errors than PolyMender or SingularCocone, but EMC computes scalar, distance and normal values from formulas hard coded into the software. PolyMender and SingularCocone (and SHREC and MergeSharp) would certainly do much better if their input came from formulas hard coded into their software.

SingularCocone has the most degree errors, but SingularCocone is designed for reconstruction from point samples of a (possibly non-manifold) surface, not for reconstruction from sampling of a scalar field. Input to SingularCocone is point samples of a surface, not scalar grid or gradient information.

5.4.1 Comparison with MergeSharp.

Input to MergeSharp is a regular grid sampling of a scalar field and the gradients at the grid vertices. We ran MergeSharp on the same 39 Flange and 34 TwoCubes synthetic
Figure 5.17: PolyMender and SHREC comparison. Mesh edges with dihedral angle below 140° are colored red. (a) PolyMender Flange isosurface. (b) Corresponding SHREC Flange isosurface. (c) Distribution of differences of triangle normals between PolyMender isosurface and polygonal mesh (red) and between SHREC isosurface and polygonal mesh (green). (d) PolyMender Cone isosurface. (e) Corresponding SHREC Cone isosurface. (f) Magnified view of PolyMender isosurface (left) and corresponding SHREC isosurface (right).

datasets that we applied to SHREC in Section 5.3.1. We used the exact gradients on MergeSharp, not the Religrad gradients. Figures 5.15(a) and 5.15(c) show the oriented angle distances between the MergeSharp isosurfaces and the polygonal mesh isosurfaces. Five of the MergeSharp TwoCubes isosurfaces have angle distance near 180° indicating “flipped” triangles produced by folds in the mesh. Figures 5.15e and 5.15f present the number of triangles with angle difference greater than 30, 40 and 50 degrees in the MergeSharp Flange and TwoCubes isosurfaces. 24 out of 39 of the Flange isosurfaces and 18 out of 34 of the TwoCubes isosurface have angle distance above 30° indicating significant problems in the reconstruction of sharp features.
Figures 5.15(b) and 5.15(d) gives the number of degree errors in the 1-skeleton’s of the sharp edges. 8 out of 39 of the Flange isosurfaces and 15 out of 34 of the TwoCubes isosurfaces have degree errors. The maximum number of degree errors is eight.

Note that all the results in Figure 5.15 are for MergeSharp isosurfaces produced from EXACT gradients. Thus, they should not be compared with the SHREC results in Figure 5.11 for Flange isosurfaces produced using Religrad gradients. The 39 SHREC Flange isosurfaces produced using exact gradients all have angle distance under $15^\circ$ compared with angle distances above $30^\circ$ for 24 of the corresponding MergeSharp isosurfaces. None of the SHREC Flange isosurfaces produced using exact gradients have degree errors compared with 8 MergeSharp isosurfaces with 4 to 8 degree errors.

As shown in Figure 5.12(a), the 39 SHREC TwoCubes isosurfaces produced using exact gradients all have angle distance under $2^\circ$. 18 of the corresponding MergeSharp TwoCubes isosurfaces have angle distance above $30^\circ$. None of the SHREC TwoCubes isosurfaces produced using exact gradients have degree errors compared with 15 of the TwoCubes isosurfaces with 2 to 5 degree errors.

Figure 5.16(a) shows “notches” in a MergeSharp Flange isosurface. Figure 5.16(b) shows gentle dimples in a MergeSharp Cone isosurface (left image). The corresponding SHREC isosurface is smooth and does not have these dimples.

### 5.4.2 Comparison with PolyMender.

PolyMender is an implementation of mesh repairing algorithm from Ju [40]. Input to PolyMender is a set of triangles representing a surface, but not necessarily properly connected in a polygonal mesh. PolyMender uses the input triangles to build a regular scalar grid representing the signed distance to the surface. It also uses the input triangles to determine surface normals on the surface. It extracts an isosurface mesh using the dual
Figure 5.18: Results of PolyMender on 39 Flange. (a) Oriented angle distance to polygonal Flange meshes. (b) Degree errors in 1-skeleton of sharp edges of Flange datasets. (c) Number of triangles with normal difference to Flange mesh above 30, 40 and 50 degrees.
Figure 5.19: Results of PolyMender on 34 TwoCubes datasets. (a) Oriented angle distance to polygonal TwoCubes meshes. (b) Degree errors in 1-skeleton of sharp edges of TwoCubes datasets. (c) Number of triangles with normal difference to TwoCubes mesh above 30, 40 and 50 degrees.
contouring algorithm described in [41, 64]. The extracted isosurface mesh is the “mended” polygonal mesh.

Because PolyMender contains an implementation of the dual contouring algorithm from [41, 64], we used it to compare SHREC and the algorithm from [41, 64]. Our inputs to PolyMender were the polygonal meshes (Figure 5.2) designed specifically for each surface to accurately represent the 0-dimensional and 1-dimensional features on the surface.
PolyMender builds a multiresolution isosurface using an octree instead of a fixed regular grid. The highest resolution is determined by the depth of the octree. For all tests, we ran PolyMender with octree depth set of 7. At the highest resolution, this octree sampled data from a regular grid with dimensions $2^7 \times 2^7 \times 2^7$ or $128 \times 128 \times 128$. We set the scale to 0.9 and used defaults for all other parameters.

Because PolyMender computes its scalar field and surface normals from an input polygonal mesh, we did not think it fair to compare PolyMender to SHREC with exact gradients. Instead, we compared PolyMender to SHREC using Religrad gradients. Since PolyMender surface normals come from triangles on the polygonal meshes used for the angle distance measurements while Religrad gradients are computed from the scalar data, we think this comparison is actually biased in favor of PolyMender.

Figures 5.17(a) and 5.17(b) show part of a Flange isosurface reconstructed by PolyMender and a corresponding isosurface produced by SHREC (Religrad gradients). Note the flipped triangles and distorted “sharp” curve (red) in the PolyMender isosurface. Figure 5.17(c) shows the distribution of differences of triangle normals between the PolyMender isosurface and the polygonal mesh and between the SHREC isosurface and the polygonal mesh. All SHREC triangle normals are within $25^\circ$ of the polygonal mesh normals, while PolyMender has numerous triangles with normals greater than $30^\circ$ of the polygonal mesh normals.

Figures 5.17(d), 5.17(e) and 5.17(f) show a PolyMender Smooth Tip Cone isosurface and the corresponding SHREC isosurface (Religrad gradients). The 1-dimensional feature around the base of the cone has a $60^\circ$ dihedral angle. In the PolyMender isosurface, there are numerous “notches” along the 1-dimensional feature.

We ran PolyMender on the same 39 Flange and 34 TwoCubes meshes that we applied to SHREC in Section 5.3.1. Figure 5.18, 5.19 shows the oriented angle distances from the polygonal meshes and the degree errors in the 1-skeletons of sharp edges. Most of
the PolyMender Flange and TwoCubes isosurfaces had an oriented angle distance greater than $90^\circ$ from the corresponding polygonal mesh normals. 24 out of 39 of the PolyMender Flange isosurfaces had over 150 triangles whose normals were more than $50^\circ$ from the normals of the corresponding polygonal meshes.

In comparison, only one SHREC Flange isosurface (Religrad gradients) had triangles whose normals were more than $50^\circ$ from the normals of the corresponding polygonal meshes (Figure 5.11). No SHREC Flange isosurface (Religrad gradients) had more than 15 triangles whose normals were more than $30^\circ$ from the normals of the corresponding polygonal meshes. Almost all of the PolyMender Flange isosurfaces had degree errors and 25 out of 39 had over 200 degree errors. Only 8 out of 39 SHREC Flange isosurfaces (Religrad gradients) had degree errors and the maximum number of degree errors was eight.

PolyMender did a bit better on the TwoCubes isosurfaces, probably because the 1-dimensional features in the TwoCubes level sets are line segments. Only one of the 34 PolyMender TwoCubes isosurfaces had over 150 triangles whose normals were more than $50^\circ$ from the normals of the corresponding polygonal meshes. 18 out of 34 PolyMender TwoCubes isosurfaces had over 50 triangles whose normals were more than $50^\circ$ from the normals of the corresponding polygonal meshes. Most of the PolyMender TwoCubes isosurfaces had degree errors, but none had more than 180 degree errors. 8 out of 34 had more than 100 degree errors and 22 out of 34 had more than 40 degree errors. 5 out of 34 SHREC TwoCubes isosurfaces (Religrad gradients) had angle distance greater than $40^\circ$ to the corresponding polygonal meshes. The same five SHREC isosurfaces had degree errors, but no SHREC isosurface had more than four such errors.

### 5.4.3 Comparison with EMC (Extended Marching Cubes):

EMC is a sample implementation by Mario Botsch of the Extended Marching Cubes algorithm [46]. The program has a function which provides the scalar value of a point
cloud dataset, the directed distances along the \( x \), \( y \) and \( z \) axes to some level set of the point cloud (a sphere), and the gradients at query points near the level set. Isosurfaces with sharp features are constructed by combining the functions to represent the union, intersection or difference of balls defined by the point cloud datasets.

The union, intersection and difference of balls produces visually interesting surfaces but such surfaces do not seem a realistic approximation to the surfaces found in industrial products. To compare EMC with SHREC, we implemented functions which produce scalar values, directed distances, and gradients, for Cube and Annulus scalar fields. By combining these functions, we the corresponding values for the TwoCubes and Flange scalar fields. We added only functions to provide the relevant scalar field measurements, but did not change the EMC code which generates the isosurface. Our modification of EMC is called EMCpoly.

We ran EMCpoly on four Flange scalar fields with four randomly generated axes directions and four TwoCubes scalar fields with four randomly generated orientations. We ran SHREC on eight corresponding datasets representing the same scalar fields. For both EMCpoly and SHREC, the Flange scalar fields were sampled on a \( 200 \times 200 \times 200 \) regular grid and the TwoCubes scalar fields were sampled on a \( 150 \times 150 \times 150 \) regular grid.

Figure (a) shows the angle distance between the EMCpoly isosurfaces and the polygonal meshes. The oriented angle distance of the SHREC isosurfaces is less than 20°, but is over 90° for all of the EMCpoly isosurfaces. Figure 5.20(b) shows the number of triangles in EMCpoly isosurfaces whose normals differ more than 30, 40 or 50 degrees from the polygonal mesh normals. All four of the EMCpoly Flange isosurfaces had over 20 triangles with normals differing more than 50° from the polygonal mesh. The four EMC TwoCubes isosurfaces were better, with only a few triangles with normals differing more than 50°.
Figure 5.20(c) shows the degree errors in the 1-skeletons of the EMCpoly isosurfaces. All eight EMCpoly isosurfaces had some degree errors, but each of the TwoCubes isosurfaces had under 40 degree errors while each of the Flange isosurfaces had over 200 errors. None of the eight SHREC isosurfaces had any degree errors. Examples of errors in EMCpoly Flange and TwoCubes isosurfaces are shown in Figures 5.20(d) and 5.20(e).

Visually, the sharp features in the EMCpoly isosurface look quite good. The large number of triangles with normals very different from the polygonal mesh normals and the high number of degree errors are probably caused by the large number of near degenerate triangles. Each of the EMCpoly TwoCubes isosurfaces has over 250 triangles (out of 46K-48K) with angle less than 1°. Each of the EMCpoly Flange isosurfaces has over 1500 triangles (out of 200K) with angle less than 1°. In contrast, none of the eight corresponding SHREC isosurfaces had any triangles with angles less than 4°. The SHREC TwoCubes and Flange isosurfaces had about 40K triangles, and 150K triangles, respectively.

Small perturbations in the locations of vertices of thin triangles create almost arbitrary normal orientations contributing to the normal differences and degree errors in the EMCpoly isosurfaces. These thin triangles are aligned with the 1-dimensional features so they do not create large visual anomalies in the EMCpoly isosurfaces.

As previously noted, EMCpoly uses hard coded functions to directly compute scalar values, directed distances and gradient information. Thus, the vertex locations both on and near the sharp features are extremely precise. If EMCpoly computed such information from an input mesh as does PolyMender or from scalar and gradient data, as does SHREC, those vertex positions would be much less precise. We conjecture that under those circumstances, the numerous thin triangles would be much more visible, creating numerous visual anomalies.
5.4.4 Comparison with SingularCocone.

Extensive research has been done on reconstruction of surfaces with sharp features from point cloud data. Regular grid scalar data can easily be converted to point cloud data by approximating the intersection points between a given level set and the grid edges, and applying a point cloud reconstruction algorithm to the intersection points. Perhaps, this is an effective way to reconstruct isosurfaces with sharp features?

SingularCocone extracts a surface mesh from a weighted Delaunay triangulation of a set of sample points of the features and the smooth portions of the mesh.

To test the efficacy of point cloud reconstruction for isosurface reconstruction, we tested one algorithm, SingularCocone, on reconstructing the TwoCubes isosurfaces. Cube merging in MergeSharp and SHREC plays a similar role to the “protective balls” in SingularCocone, so we thought that SingularCocone would perform better than other point cloud reconstruction algorithms.

We note that we are only evaluating whether SingularCocone is more effective than SHREC in reconstructing isosurfaces. SingularCocone is really built for point cloud data which is much noisier and more difficult to handle than scalar data. SingularCocone also can reconstruct non-manifold surfaces and their non-manifold features. SHREC has no application to reconstruction from point cloud data or to reconstruction of non-manifold surfaces.

SingularCocone requires two inputs: a point cloud and a weighted sampling of the 0 and 1 dimensional features of the surface. Weight of the sampling determines the size of “protecting balls” around surface features. SingularCocone outputs a feature sensitive mesh.

In order to run SingularCocone, we must construct a weighted sampling of surface features from a set of points sampling the surface. We use the algorithm FeatureRecon by Dey et al. [24] to compute sample points on the surface features.
5.4.5 Experimental details:

The input to FeatureRecon is a point cloud. Two different input point clouds were tested;

1. Marching Cubes [49] was applied to fifteen TwoCubes datasets. The vertices of the Marching Cubes mesh was super-sampled using Monte Carlo point sampling, to generate a point cloud with approximately sixty-five thousand points.

2. The polygonal meshes described in Section 5.1.7 were also super-sampled to generate a point cloud with approximately sixty-five thousand points.

The first experiment measured how well FeatureRecon and SingularCocone could reconstruct an isosurface from scalar data. The second experiment measured how well FeatureRecon and SingularCocone reconstructed surface meshes from point cloud samplings of those meshes.

The two different point clouds are used as input to FeatureRecon to extract sample points on the surface features. The feature sample points generated by FeatureRecon along with the supersampled point cloud are used as input to SingularCocone.

FeatureRecon has nine separate parameters. Experimentally and also noted by the authors Dey et al. [24], we found FeatureRecon to be heavily reliant on parameter fine tuning. We used the following parameter values for the TwoCubes datasets, (as suggest by the authors Dey et al. [24]). $-t = 25, -fl = 0.04, -cl = 0.06, -dc = 0, -\rho 3 = 0.32, -\rho 1 = 0.0, -rc = 3$.

Figure 5.22 shows a TwoCubes mesh reconstructed by SingularCocone. The associated “sharp” edges are also shown. Figure 5.22(a) shows the reconstruction from the point cloud generated from the polygonal mesh. The red “sharp” edges show that the reconstruction has many errors. The magnified regions show some of the errors along the sharp edges and corners. Figure 5.22(b) shows the reconstruction from the point cloud generated from
running Marching Cubes. The magnified regions show the same regions as Figure 5.22(a). The reconstruction is worse than SingularCocone reconstruction from the supersampled polygonal mesh.

Figure 5.21 shows a comparison of SingularCocone and SHREC on 15 different datasets. The inputs to SingularCocone for these tests were generated from the polygonal meshes. The results using sample points created from Marching Cubes are much worse and not shown. SingularCocone does not assume the output is a manifold, so the triangles in the SingularCocone mesh have arbitrary orientations. Thus, the unoriented angle distance was used to compare the SingularCocone and SHREC meshes.

We ran SingularCocone and SHREC on the same 15 datasets, and used both exact and Religrad gradients for SHREC. Figure 5.21(d) shows the unoriented angle distance between the SingularCocone isosurfaces and the corresponding polygonal meshes, and the SHREC isosurfaces based on Religrad gradients and the corresponding polygonal meshes. The maximum angle distance was $43^\circ$ for SHREC with the mean error of $8^\circ$ for the 15 tests. In comparison, the maximum angle distance was $87^\circ$ for SingularCocone, with a mean of $79^\circ$. Figure 5.21(f) shows the number of SingularCocone triangles with normal difference greater than 30, 40 or 50 degrees to the corresponding polygonal mesh normals. On all the test cases there are large numbers of triangles with normal differences more than $40^\circ$ and $50^\circ$.

Figure 5.21(e) shows the degree errors for SingularCocone isosurfaces and SHREC isosurfaces based on Religrad gradients. SHREC with Religrad gradients generates degree errors in only two isosurfaces (t6 and t13) and does not generate more than 4 degree errors per isosurface. SHREC with exact gradients generates no degree errors. In contrast, SingularCocone generates an average of 330 degree errors in each isosurface.
Table 5.6: CT dataset sizes, isovalues, and average statistics on isosurfaces produced by SHREC. Average number of active cubes (Avg Num Act. Cubes), average number of isosurface vertices, average number of isosurface triangles, and average SHREC running time. All isosurface quadrilaterals are triangulated before counting the number of isosurface triangles.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Grid Size</th>
<th>Spacing</th>
<th>Avg Num Isovalue</th>
<th>Avg Num Act. Cubes</th>
<th>Avg Num Iso Vert</th>
<th>Iso Tri</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motorcycle Engine</td>
<td>$200 \times 129 \times 62$</td>
<td>$0.27 \times 0.27 \times 0.68$ (mm)</td>
<td>22000</td>
<td>25k</td>
<td>26k</td>
<td>51k</td>
<td>1.9 sec</td>
</tr>
<tr>
<td>Volt</td>
<td>$411 \times 431 \times 61$</td>
<td>$1 \times 1 \times 1$</td>
<td>4000</td>
<td>519k</td>
<td>527k</td>
<td>1039k</td>
<td>39 sec</td>
</tr>
<tr>
<td>CMM</td>
<td>$500 \times 500 \times 196$</td>
<td>$0.2 \times 0.2 \times 0.31$ (mm)</td>
<td>20000</td>
<td>861k</td>
<td>863k</td>
<td>1720k</td>
<td>60 sec</td>
</tr>
</tbody>
</table>

5.5 Experimental Results on CT Data

We applied the SHREC algorithm on industrial X-ray computed tomography (CT) scans of three objects. We set the singular value threshold $\varepsilon$ to 0.1, and used $9 \times 9 \times 9$ subgrids for gradient selection. The Motorcycle Engine dataset is an industrial CT scan of a motorcycle engine cylinder. (See photo in Figure 5.23(a).) The Volt dataset is an industrial CT scan of a 440 voltage electrical connector. is an industrial CT scan of a solid aluminum shape used to calibrate measurements from CT scans. CMM stands for “coordinate measuring machine”. Since the CT scanner provides only scalar values for each object, we used Religrad to construct gradients at the grid vertices. Dataset sizes, spacing, isovalues and average isosurface sizes are presented in Table 5.6. The full Motorcycle Engine dataset has very large dimensions so we only report on a small $200 \times 129 \times 62$ corner of that dataset depicted in Figures 5.23 and 5.24. Note the non-uniform spacing in the Motorcycle Engine and CMM datasets.
5.5.1 Reliable Gradients from industrial CT data

Figure 5.8 shows the central difference gradients at all grid vertex location around an edge of the *engine cylinder* dataset. The length of the vectors are proportional to the magnitude of the gradients. Note that the gradients quickly fall off away from the isosurface. The problem is more severe along the Z-Axis. The magnified portion shows that there are about two good columns of gradients after which the gradient magnitude becomes too small and the gradient directions meaningless. Figure 5.9(a), 5.9(b) show the results of Algorithm 1 and Algorithm 2 respectively. Unlike the simulated datasets, almost no gradients along the Z axis are marked as reliable. This shows the need for dividing the vertex set into tangential and orthogonal neighbor sets. This lead us to Algorithm 7. Figure 5.9(c) shows the corresponding gradients of the `FINDRELIABLE` technique. Figure 5.9(d) shows the `RELIGRAD` gradients. For reference, Figure 3.3 shows the Central Difference gradients.

5.5.2 Reconstruction from industrial CT data

Figures 5.23 and 5.24 show a small corner of the motorcycle engine cylinders and the results of reconstruction of that corner. The magnified regions (with black border) in Figure 5.24 show that the sharp edges are well reconstructed. In yellow border boxes, we see magnified parts of the reconstructed mesh along with the sharp and non-sharp edges generated by SHREC.

Figure 5.25 shows part of the 440 voltage connector (in red) and the SHREC reconstruction of the Volt dataset. To aid in visualization of the sharp features, the figure displays a cropped image of the the reconstructed isosurface. Once again we see that SHREC is able to reconstruct the sharp (flange-like) curves accurately.
Figure 5.26 shows the SHREC reconstruction of the CMM dataset. Again, the reconstructed isosurface is cropped to better display the sharp features. Figure 5.26(b) shows a single slice of the CT scan, with scalar values mapped to the “heat” color map.

SHREC does a good job of reproducing the 0 and 1 dimensional features from each of these data sets and in producing meshes which accurately reflect those features.

5.6 Timings

<table>
<thead>
<tr>
<th>Software</th>
<th>Num Iso Tri</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marching Cubes</td>
<td>41K</td>
<td>0.4 sec</td>
</tr>
<tr>
<td>EMCpoly</td>
<td>46K</td>
<td>0.8 sec</td>
</tr>
<tr>
<td>PolyMender</td>
<td>85K</td>
<td>0.7 sec</td>
</tr>
<tr>
<td>MergeSharp</td>
<td>35K</td>
<td>0.6 sec</td>
</tr>
<tr>
<td>SHREC grad3</td>
<td>34K</td>
<td>1.0 sec</td>
</tr>
<tr>
<td>SHREC grad9</td>
<td>34K</td>
<td>1.4 sec</td>
</tr>
<tr>
<td>Religrad</td>
<td>34K</td>
<td>30 sec</td>
</tr>
<tr>
<td>SingularCocone</td>
<td>125K</td>
<td>1.8 sec</td>
</tr>
<tr>
<td>FeatureRecon</td>
<td>125K</td>
<td>344 sec</td>
</tr>
</tbody>
</table>

Table 5.7: Timings. SHREC grad3 and SHREC grad9 is SHREC using gradients from $3 \times 3 \times 3$ and $9 \times 9 \times 9$ subgrids, respectively, around selected vertices.

Table 5.7 gives the running time of the various algorithms on a TwoCubes dataset. The running time was measured on a Windows PC with an Intel Core i7-4770 processor (3.4 GHz) and 16 GB RAM.

Inputs to Marching Cubes, EMCpoly, MergeSharp and SHREC were grids of size $150 \times 150 \times 150$. PolyMender constructed an oct tree with depth 7, with a full resolution grid of size $128 \times 128 \times 128$. Inputs to SingularCocone and FeatureRecon had 60K sample points.
The point cloud reconstruction software SingularCocone and FeatureRecon are very different from the isosurface reconstruction software, so one should be extremely careful in comparing its timings to the others. SingularCocone receives a set of points on 0 and 1-dimensional features as part of its input, while EMCpoly, PolyMender, MergeSharp and SHREC must spend time computing such points from gradients or surface normals. On the other hand, FeatureRecon receives a set of sample points without any gradient or surface normal information, while EMCpoly, PolyMender, MergeSharp and SHREC receive gradient or surface normals as part of their input. Computing points on 0 and 1-dimensional features is much easier if gradient or surface normals are provided.

EMCpoly, PolyMender, MergeSharp and SHREC take comparable times. SHREC takes the longest and the time increases if gradients are selected from larger subgrids around each cube. The extra time taken by SHREC is spent selecting gradients around each cube. The time for merging cubes in MergeSharp and SHREC is a small part of the total, since merging is only performed around selected grid cubes covering sharp features.

If a gradient grid file is not available, then it must be constructed from the scalar grid. The Religrad time (30 second) to construct the gradient grid should be added to the SHREC time. Note that EMCpoly, PolyMender and MergeSharp would also require Religrad or some similar program to construct a gradient grid or set of surface normals, so the programs still take comparable times. FeatureRecon takes considerably longer than Religrad but input to FeatureRecon is set of surface sample points, not a scalar grid.

5.7 Software and Datasets

Software and datasets used in this paper can be downloaded from:

web.cse.ohio-state.edu/research/graphics/isotable.
In particular, the web site contains source code for SHREC, Religrad, MergeSharp and EMCpoly. The web site also contains programs to generate regular grid samplings of scalar and gradient fields (ijkgenscalar), generate polygonal meshes of annuli, flanges, two cubes, cones, frustra, smooth tip cones and cannons (ijkgenmesh), compute angle distances (angle_dist), find sharp edges (findsharp) and count the vertex degree in the graph formed by the sharp edges (countdegree). Finally, the web site contains the Cannon and Cone scalar and gradient datasets and some of the Flange and TwoCubes scalar and gradient datasets. (Because of restricted space, we were not able to include all 39 Flange datasets and 34 TwoCube datasets in the web site.)

5.8 Assumptions about extracted “sharp” features

We make the following implicit assumptions about the “sharp” features which the algorithm SHREC along with RELIGRAD reconstructs.

- *Sharp features are not very close to each other.*

RELIGRAD uses agree-ability between central difference gradients to find reliable gradients. As discussed in Section 3.2 algorithm RELIGRAD uses gradients two grid vertices away to approximate gradients and the vertex which is two away itself uses vertices one grid edge away from it to compute its central difference gradients. Thus in total RELGIRAD utilizes vertices which are three grid edge distance away. Features (gradient discontinuities) very close to each other (within 6 voxels) might not produce reliable gradients.

SHREC also uses a number of steps to merge isosurface vertices. The underlying assumption being the “sharp” isosurface vertices are part of a single feature. Two or more sharp feature very close to each other, might cause the features to be mistakenly
joined by merging isosurface vertices lying on separate sharp features. Two separate features must be at-least separated by 6 grid voxels.

For both RELIGRAD and SHREC sharp features which are very close (within 6 voxels) of each other are thus problematic for reconstruction.

- **Piecewise smooth surfaces which create the sharp features have small dihedral angles.**

SHREC might consider sharp features to be smooth if the dihedral angle of the intersecting piecewise smooth surfaces is greater than a threshold. Our experiments with the Cone and Cannon datasets shows the threshold to be 135 degrees. In such cases the surface will still be reconstructed but with the affected sharp features reproduced as smooth. SHREC will also have problems if the dihedral angle is too small (less than forty-five degrees). Features with very small dihedral angles might be very close to each other and have problems discussed above.

- **Features are not very thin.**

SHREC cannot reconstruct features such as thin “walls” created by two surfaces close to each other. Thin “wall” features are generally associated with gradients going in opposite directions perpendicular to the two surfaces creating the thin wall. As mentioned above this makes computing reliable gradients and reconstructing these features in the local neighborhood difficult. In our experiments two surfaces with less than 6 voxels in distance are considered “thin”.

124
Figure 5.21: Comparison of SHREC and SingularCocone on 10 Flange and 15 TwoCubes datasets. Point clouds for SingularCocone (and FeatureRecon) were computed by super-sampling the polygonal meshes. (a) Unoriented angle distances from Singular Cocone (blue) Flange isosurfaces and from SHREC Flange isosurfaces (exact gradients in red, Religrad gradients in green) to polygonal Flange meshes. (b) Degree errors in the 1-skeleton of the sharp edges for SingularCocone Flange isosurfaces (blue) and SHREC isosurfaces (Religrad gradients, red). (SHREC isosurfaces based on exact gradients have no degree errors.) (c) Number of SingularCocone triangles with normal difference to Flange mesh above 30°, 40° and 50°. (d) Unoriented angle distances from Singular Cocone (red) TwoCubes isosurfaces and from SHREC TwoCubes isosurfaces (Religrad gradients, green) to polygonal TwoCubes meshes. (e) Degree errors in the 1-skeleton of the sharp edges for SingularCocone TwoCubes isosurfaces (red) and SHREC isosurfaces based (Religrad gradients, blue). (f) Number of SingularCocone triangles with normal difference to TwoCubes mesh above 30°, 40° and 50°.
Figure 5.22: SingularCocone TwoCubes isosurface. (a) Input cloud is a supersampled polygonal mesh. (b) Input cloud is a supersampled set of isosurface vertices produced by Marching Cubes. The magnified regions show some of the errors. The reconstruction from Marching Cube vertices is worse than the reconstruction from the supersampled polygonal mesh.

Figure 5.23: Motorcycle Engine dataset. (a) a slice of the original CT image, note the streaking artifacts introduced during the scanning process. The inset shows the original machine part. Figure (b) shows the sharp mesh.
Figure 5.24: SHREC with Religrad gradients computed from part of industrial CT data (Motorcycle Engine). Magnified regions show “sharp” edges reconstructed. In red, picture from the original item.

Figure 5.25: SHREC with Religrad gradients computed from part of the VOLT data. Magnified regions show “sharp” reconstructed edges. In red picture of the original item.
Figure 5.26: (a) SHREC with Religrad gradients computed from part of the CMM dataset. Magnified regions show “sharp” reconstructed edges. (b) Single slice of the CT scan, with scalar values mapped to the “heat” color map.
Chapter 6: MetaTracts: Fiber Composite Visualization

6.1 Overview

In this work, we focus on Carbon Fiber Reinforced polymer datasets scanned with 3D X-ray CT. These datasets are associated with larger viewports but lower resolutions where the individual carbon fibers (filaments) are indiscernible or barely visible. Our domain experts are mainly interested in visualizing the geometric structures in the weaving pattern of fiber bundles in endless carbon fiber reinforced composites instead of high resolution studies of individual fibers. (See [10] for extensive details). Figure 6.2 depicts our targeted dataset type. It shows the recurring fiber bundle pattern in the final CFRP laminate, the unit cell.

Figure 6.1: Flow chart of the MetaTracts approach for fiber bundle extraction
Our work is motivated from the recent progress in two interrelated fields: Firstly, CFRP components have gained wide application in industry because of its superior material and physical properties in comparison to conventional materials [43]. Secondly, recent developments of industrial 3D X-ray computed tomography (XCT) with regard to larger detectors, larger field of views, and better resolutions has opened XCT for the new application area of non-destructive testing for fiber reinforced components [66].

While fiber bundles are now understood as highly important in determining component properties, the tools for visualizing the internal structure have not developed at the same pace. To the best of our knowledge, there is no current work that can resolve simple queries such as:

- How to extract and visualize the geometric structure of a particular fiber bundle?
- How to visualize the interaction between a particular pair of fiber bundles (weaving/braiding) or a unit cell?
- Which fiber bundles show a particular orientation?
- Which fiber bundles are of the same type of yarn? I.e. which bundles show similar sizes or diameters, which is the largest or smallest fiber bundle?

We separate the above queries into two major classes: geometric structure and spatial context. Queries associated with attributes of fiber bundles such as shape, size and orientation are grouped under geometric structure. While those referring to how two or more fiber bundles interact are categorized under spatial context. Spatial context answers questions such as: Are these bundles in contact at a particular location in the dataset? What are the relative orientations of the contacting fiber bundles?

Providing answers to the above queries from currently common approaches, such as volume rendering of the XCT datasets, or visual inspection of particular 2D slices is non-trivial, even for experts.
We interpret and advance techniques from diffusion tensor imaging to extract and visualize geometric structures from 3D X-ray computed tomography data of the woven carbon fiber reinforced composites. The main goal of this work is to expand the state of the art in non-destructive testing through visualization of composite structures in complete unit cells of woven fabrics.
6.2 Data Characteristics and Assumptions

Figure 6.2 shows dataset D1 with woven fiber bundles. The size of D1 is $450 \times 300 \times 500$ voxels with isotropic resolution of 2 $\mu m$ and 8 bit unsigned integer scalars. Figure 6.2 clearly shows the recurring fiber bundle weaving pattern of the composite unit cell used for manufacturing fiber composites. Figure 6.2(a) shows a volume rendering of the dataset.
Figure 6.2(b) and 6.2(c) show 2D slices along the X- and the Z-axis respectively. Figure 6.2(d) shows a magnified image of the green region of interest and contains two bundles going in opposite directions, but the low resolution of D1 renders the individual carbon fibers in a fiber bundle hard to resolve. Also the separation between two fiber bundles is barely visible. The fiber bundles may differ in terms of the amount of fibers in the bundle. Figure 6.2(a) shows the large variation in cross section sizes among the bundles. Depending on the weaving pattern, the fiber bundles cross each other in different orientations. The number of orientations is defined by the weaving pattern and typically consists of two main orientations. The weaving pattern may cause individual bundles to be curved. In consequence, individual fibers may be adjacent in Euclidean distance but belong to different bundles. We make the following assumptions on our data:

- The structure embedded in the data contains fiber bundles of indiscernible fibers.
- Local orientation: each point in a fiber bundle has a local orientation parallel to the corresponding area in the fiber bundle.
- Local orientation may gradually change within the fiber bundle.
- Local orientation may be noisy and not reliable.
- Connectivity: moving along the direction of a non-noisy local orientation in small increments, we will reach another neighborhood with similar local orientation.
- Fiber bundles going in different directions only interact near the surface of contact.
6.3 Extracting MetaTracts

Extracting MetaTracts consists of two main steps. Initially, a local orientation direction (represented as a unit vector) is computed at each grid location. Next, we compute a coarse set of poly-cylinders called MetaTracts which traverses the data constrained by the local orientation directions. Section 6.3.1 describes the procedure to generate Reliable Hessians. Section 6.3.2 describes the MetaTracts and their properties. Section 6.3.3 details the creation of MetaTracts.

6.3.1 Computing Reliable Hessians

The objective of this step is to associate each grid location with a unit vector which represents the orientation in the local neighborhood and a real value [0,1] that represents a measure of reliability of the orientation calculated at the grid location. A reliability score of 0 means the local orientation is not credible and a score of 1 means the local orientation is noisy. The input to this stage is the original scalar data, which is an uniform lattice grid in $\mathbb{R}^3$. We approximate the local orientation by eigenvalue analysis of the Hessian matrix applied locally to each voxel. The Hessian matrix captures the local second-order structures inherent in the intensity (scalar values at grid vertex) variations around each grid location. The eigen decomposition of the Hessian matrix gives the eigenvectors which represent the local curvature of the image. The eigenvector corresponding the smallest eigenvalue gives the direction along which the curvature is smallest. This direction also termed as the principal direction, coincides with the direction of the tubular structure.

Frangi et al. [28] introduced a process that searches for geometric structures which are tubular. They defined a “vesselines” criterion based on the geometric ratios of the second order ellipsoid given by the local Hessian matrix. In order to determine reliable Hessians, we compute the same metric. We include their work here for completeness and direct
the reader to [28] for details. Let \( \lambda_k \) be the eigenvalue with the \( k^{th} \) smallest magnitude. Here, \( |\lambda_1| \leq |\lambda_2| \leq |\lambda_3| \) are the eigenvalues of the Hessian matrix. Specifically, a pixel belonging to a vessel region will have small \( \lambda_1 \) (\(|\lambda_1| \approx 0\)) and \( \lambda_2, \lambda_3 \) of large magnitude and of equal sign (\(|\lambda_1| \ll |\lambda_2|\) and \(|\lambda_2| \approx |\lambda_3|\)). The sign indicates if the vessel is bright in a dark background or dark in a bright background. In our case the individual fibers are bright \((\lambda_2, \lambda_3 < 0)\). The following measures are defined in [28].

\[
\mathcal{R}_A = \frac{\text{Largest Cross Section} / \pi}{\text{Largest Axis Semi-length}^2} = \frac{|\lambda_2|}{|\lambda_3|} \quad (6.1)
\]

\[
\mathcal{R}_B = \frac{\text{Volume} / (4\pi / 3)}{(\text{Largest Cross Section Area} / \pi)^{\frac{3}{2}}} = \frac{|\lambda_1|}{\sqrt{|\lambda_2 \lambda_3|}} \quad (6.2)
\]

In Equation 6.2, \( \mathcal{R}_B \) provides a measure of deviation from a blob like structure while in Equation 6.1, \( \mathcal{R}_A \) distinguishes between plate-like and line-like structure. Grey-scale variations and close proximity of the fibers in our data make the Hessians computed at each voxels susceptible to errors. Thus, we compute reliable Hessians \( (R_H) \) to determine which locations in the volume provide reliable local orientation.

\[
R_H = \begin{cases} 
0 & \text{if } \lambda_2 > 0 \text{ or } \lambda_3 > 0 \\
(1 - e^{-\frac{\mathcal{R}_A^2}{2\alpha^2}})(e^{-\frac{\mathcal{R}_B^2}{2\beta^2}})(1 - e^{-\frac{s^2}{2\gamma^2}}) & \text{otherwise}
\end{cases}
\]

Variable \( s \) is the Frobenius norm of the Hessian matrix. The value of \( (1 - e^{-\frac{s^2}{2\gamma^2}}) \) will be low in regions with no structure. The utility of the vesselness is a little different in our framework than in the work of Frangi et al. [28]. First, vesselness in biology is computed for different scales because the vessels can be of different sizes. In our case, usually the widths of individual fibers are known a priori. Secondly, we do not have clear tubular structures embedded in a dark contrast matrix such as in blood vessels. Instead, we are trying to associate each grid location with a probable orientation based on its local second
Figure 6.3: Reliable Hessians. (a) MetaTracts colored according to the local orientation vector mapped to RGB color scale. (b,c) 2D slices along Z- and X- axis. (d) Magnified region marked in (c).

order structure. The $R_H$ is interpreted as a reliability measure of the local orientation. Grid locations where the $R_H$ is above a cutoff threshold are marked as regions with reliable orientations (see Section 6.7).

Figure 6.3 shows the intermediate results of local orientation computation. The principal direction represented as a unit vector has been mapped to RGB color space. Figure 6.3(a) shows the entire data set. Figure 6.3(b),(c) shows 2D slices along the Z- and X-axes respectively. Regions with X- and Z-axis local orientation show predominant red
and blue color respectively. Figure 6.3(d) shows a magnified region of interest. The black regions within bundles are regions where the $R_H$ is less than the threshold and have unreliable local orientation. The bundles are also not uniformly colored as the Hessians and the corresponding principal directions are noisy.

We indicate some intrinsic differences between DTI and our XCT data. Fiber traces can be generated in DTI using a standard fiber tracking algorithms following the principal direction of diffusion by employing a fourth order Runge-Kutta method [17]. The principal direction based on the Hessian matrix works best when the tubular structures in the data are well separated from the background. This is not the case for our data. The local orientations at each voxel are inherently noisier.

### 6.3.2 MetaTracts Properties

Conventional integral curve based techniques cannot be directly used to extract fiber bundle traces from reliable Hessians because of the spurious nature of the Hessian based local orientations. Thus instead of building fiber traces, we define an abstract representation of the fibers. We start from two key assumptions on the data, specifically “local orientation” and “connectivity”, while taking into account the adverse effects of noise and low resolution.

This is achieved by interpreting the underlying geometric structure of the fibers as a set of connected cylinders. Hence, we formulate MetaTracts as a coarse and simple approximation of integral curves. In the form of a continuous chain of cylindrical tubes in $\mathbb{R}^3$. MetaTracts traverse the fiber bundles embedded in the original data. Extending the intuition developed above, we devise the following *properties* that the MetaTracts share;

1. MetaTracts are associated with a continuous set of cylinders.
2. MetaTracts are associated with a start point at a grid vertex.
3. Individual cylinders in MetaTracts have constant lengths, radii, and start points (which are also grid’s vertices).

4. Individual cylinders in MetaTracts (except the first one) are connected to their previous cylinders at their start point.

5. Individual cylinders are parallel to the local orientation vector at their start points.

### 6.3.3 MetaTracts Generation

In this section we detail the process of generating MetaTracts. Here we explain the process in $\mathbb{R}^2$, the procedure extends to $\mathbb{R}^3$ trivially. The output of the last (reliable Hessian) stage is input to the current step. In $\mathbb{R}^2$, each grid location is associated with a unit vector (representing the local orientation) and a reliability measure $R_H[0,1]$. Figure 6.4 shows some key features of MetaTracts. In Figure 6.4(a),(b) the regions with unreliable Hessians
are marked in blue. In $\mathbb{R}^2$ the individual components of MetaTracts are rectangles. Figure 6.4(c) shows an individual rectangle, let the seed point associated with the MetaTract be grid point $C_p$ (property 2 in Section 6.3.2). The local orientation at $C_p$ as computed in the reliable Hessian stage is $N_p$ and is given by the dark green arrow. The rectangle itself is of length $L$ and radius $R$ (properties 3 and 5 in Section 6.3.2).

We start with an individual rectangle with seed point $C_p$, local orientation $N_p$ and dimensions $L, R$. The set of grid vertices which are located within the rectangle (green region) in Figure 6.4(a),(b) but are possible start points for the next cylinder. Among these we first discard the grid points which have unreliable Hessians (marked in blue). These set of grid points are called “candidate vertices” and are potential start points. Based on the “local orientation” and “connectivity” assumptions on our data, we rank all the candidate points. The rank of each candidate point is based on the following characteristics.

- Orientation similarity: The local orientation of the start points ($N_p$) for the consecutive cylinders should be similar.

- Large distance: The MetaTracts should traverse the data using as few cylinders as possible. Thus the distance between start point ($C_p$) of one cylinder and the start point for the next cylinder should be as large as possible. We measure the distance of each candidate vertex from $C_p$ by projecting the Euclidean distance between them onto $N_p$. For example, in Figure 6.4(b) the distance is measured as the Euclidean distance between the green and the orange vertices projected on to $N_p$. We refer to this perpendicular projection distance as ‘projected dist’.

We next define a “priority” for each candidate vertex, based on the above characteristics. For each cylinder in a MetaTract, we put it’s “candidate vertices” in a priority queue based
on Equation 6.3.

\[ \text{Priority} = \gamma_1 e^{-\text{angle}^2/\alpha^2} + \gamma_2 e^{-\text{projected dist}^2/\beta^2} \] (6.3)

\( \gamma_1, \gamma_2 \) are the weights \((\mathbb{R}_{\geq 0})\) which decide how the “priority” depends on the affine combination of the two factors. For all our cases, we use \( \gamma_1 = 1/3 \) and \( \gamma_2 = 2/3 \). In general, we suggest \( \gamma_1 + \gamma_2 = 1 \) and \( \gamma_1 \leq \gamma_2 \). At each iteration, we pick the top element in the priority queue to generate the corresponding cylinder, and repeat the steps. Essentially, Equation 6.3 selects a grid point which is farthest from the current start point and is going in a similar direction. This approach tackles noise/errors in local orientation better than integral curves by looking at multiple choices for vertex candidates and avoiding intra-cell interpolation in an already noisy environment.

In Figure 6.4(b) the orange grid vertex is selected next and process repeated. The purple vertex is in a region of unreliable Hessian and is not selected even though it is further away from the seed point in terms of euclidean distance. If we generate MetaTracts that have erroneous local orientations the size of the candidate vertex set will be small. This will create MetaTracts of small length which are then removed.

The MetaTracts generated by the above procedure are shown in Figure 6.5. The MetaTracts are colored with the mean orientation direction mapped to the RGB space. Figure 6.5 also shows a single slice of YZ plane and the XY plane. Consistent orientation is a key intrinsic feature in our data which was not obvious from the original gray scale images, but becomes visually pronounced in the generated MetaTracts.

Figure 6.10 shows the MetaTracts of a particular bundle colored according to their individual length. MetaTracts in a given fiber bundle may extend the full length of the bundle or have different lengths and partial overlaps.
Figure 6.5: MetaTracts color-coded according to their mean orientations, along with slices of the YZ, XY-plane.
6.4 Fiber Bundle Generation

The MetaTracts generated in the previous section represent fragments of fiber bundles. The MetaTracts are clustered in order to extract the final fiber bundles. The clustering process benefits from both the orientation and geometric proximity information inherent in the carbon fiber bundles. Experimentally we found that “orientation” information was more reliable, while partially overlapping fibers (Figure 6.10) created problems for “geometric” proximity based approaches. We also observed that different clustering techniques performed preferably for different measures. Instead of creating a heuristic and artificially combining the orientation and geometric proximity measures, we separated the clustering process. Specifically, we first cluster based on “orientation”. Each orientation cluster is then further subdivided based on “geometric” proximity. The two-step clustering process, helps making the problem more tractable, and the parameters more intuitive to the end users.

6.4.1 Orientation Based Clustering

Before extracting individual fiber bundles, we first separate the MetaTracts into classes based on their major local orientations. To cluster MetaTracts going along similar local directions, we use a spectral embedding technique called Laplacian eigenmaps (originally introduced by Belkin and Niyogi [8]). An eigenvalue problem is solved to map the manifold embedded in a graph into a lower dimensional space, while preserving the graph structure. Let $G$ be the graph, we compute the eigenvalues and eigenvectors for the generalized eigenvector problem $Lf = \lambda Df$, where $D$ is the diagonal weight matrix and $L$ is the Laplacian matrix. The eigenvector $f_0$ corresponding to the eigenvalue 0 is left out and the next $m$, $f_1$ through $f_m$ eigenvectors are used to embed in an $m$-dimensional space (see Sec. 6.7 for values of $m$).
In our problem, each MetaTract is a node in the graph. We adopt a simple orientation based measure to define the weight of the edges. Given a pair of MetaTract, the edge weight between two nodes is defined as the cosine of the maximum angle between the local orientations ($N_P$) of all pairs of start points ($C_P$) between the two MetaTracts. The edge weights give a “distance matrix” representing the distance between each pair of nodes. Using the Belkin and Niyogi algorithm we “embed” these nodes in a low dimensional space where the Euclidean distance between nodes, approximates the distance between nodes given by the original “distance matrix”. Following this, K-means clustering is employed in the lower dimensional space. Where $K$ is the number of major fiber bundle directions in the woven structure. $K$ is derived from domain knowledge, for all our test cases, there are two major fiber bundle directions. Dimensionality reduction provides us some interesting advantages, by handling the case of curved bundles. Figure 6.6(b) shows the result of the K-means clustering with the nodes (MetaTracts) projected to the top three eigenvectors as the major axes. As expected there is a clear distinction based on fiber bundle orientation. Figure 6.6(a),(c) shows the MetaTracts colored according to orientation clustering results, in blue and red respectively. Each distinct cluster represents MetaTracts belonging to all fiber bundles, along an individual orientation.

6.4.2 Distance Based Clustering

To subdivide the oriented clusters into individual fiber bundles, we include further information about the geometric proximity between MetaTracts. We use the directed Hausdorff distance for distance based clustering. Each MetaTract is represented as a set of points ($C_P$). Formally, the directed Hausdorff distance from point set $P$ to point set $Q$ is defined as $H_{dir}(P, Q) = \max_{p \in P} \min_{q \in Q} d(p, q)$. The Hausdorff distance is defined as $H(P, Q) = \max(H_{dir}(P, Q), H_{dir}(Q, P))$. The Hausdorff distance is a metric so $H(P, Q) \leq H(P, Q') + H(Q', Q)$ but the directed Hausdorff is not. Unfortunately, the Hausdorff distance does not
work well for our application. A single fiber bundle is represented as a set of “overlapping” MetaTracts. For example Figure 6.10 shows the length distribution of MetaTracts which express the fiber bundle. Consequently, if a MetaTract $P$ covers only a part of the fiber bundle covered by $Q$, then $H_{dir}(P,Q)$ will be very small while $H_{dir}(Q,P)$ will be large. Thus, $H(P,Q)$ will be large, even though $P$ and $Q$ are in the same fiber bundle. Instead of using the Hausdorff distance, $\max(H_{dir}(P,Q),H_{dir}(Q,P))$, we use $\min(H_{dir}(P,Q),H_{dir}(Q,P))$. If $P$ covers only a part of the fiber bundle covered by $Q$, then $\min(H_{dir}(P,Q),H_{dir}(Q,P))$ is very small. Note that if $P$ and $Q$ overlap but do not cover the same parts of the fiber bundle, then $H_{dir}(P,Q)$ and $H_{dir}(Q,P)$ and $\min(H_{dir}(P,Q),H_{dir}(Q,P))$ will be large. The directed Hausdorff distance is very sensitive to outliers in the data. However, because MetaTracts after orientation clustering are constructed using cylinders with similar orientations, they are not plagued by outliers. To cluster based on MetaTract proximity, we used single linkage hierarchical clustering. Hierarchical clustering has a single parameter $h$, the desired number of clusters. Clusters are merged until there are only $h$ clusters left. Hierarchical clustering is intuitive since it is easy to trace how clusters are formed and merged. Single linkage clustering finds pairs of objects $p \in P$ and $q \in Q$ where $P \neq Q$ which are closer than other such pairs, and merges the containing clusters $P$ and $Q$. We found that single linkage hierarchical clustering had two major drawbacks:

- The clustering might produce some “small” clusters of just a few MetaTracts. These MetaTracts are anomalies caused by overlapping fibers and did not represent true fiber bundles.

- Second, if two distinct fiber bundles “ran” parallel for some of their length and then separated, they would sometimes be clustered into a single erroneous bundle. This occurs when a short MetaTract which was parallel to both but did not extend into the
separation region forms a link between the two fiber bundles. This causes the fiber bundles to be clustered into a single bundle.

To address the problem of small clusters, we applied hierarchical clustering and then identified small clusters with few MetaTracts. We removed the MetaTracts that were in those clusters from the data set and reapplied hierarchical clustering. To address the problem of short MetaTracts joining different fiber bundles, we applied hierarchical clustering and then removed the shortest tracts (length less than $\eta$ times the median length, set to 0.6) in each bundle. We then reapplied hierarchical clustering. We repeated both steps until a steady state of clusters was reached and no new small fibers can be removed. The result of hierarchically clustering all the MetaTracts in each orientation cluster (Figure 6.6(a),(c)) are shown in Figure 6.6(d),(e) respectively.

After the clustering step, the MetaTracts are separated into well formed fiber bundles. The final result of the clustering process is shown in Figure 6.6(f). Eleven individual fiber bundles have been separated. Five fiber bundles were extracted from orientation cluster 1 (Z-axis) and six from orientation cluster 2 (X-axis).

### 6.4.3 Choice of Clustering Techniques

Before settling on the two-step clustering approach, we experimented with using proximity alone in a single step clustering approach for MetaTracts. We performed two separate tests. First, we used hierarchical clustering to cluster the MetaTracts. Second, we used K-means to cluster the MetaTracts. Figure 6.7 shows the results.

Figure 6.7(a) shows the results of MetaTracts that are hierarchically clustered directly using proximity alone into 10, 15 and finally 20 clusters. As discussed in Sec. 6.4.2, single linkage hierarchical clustering in the presence of overlapping MetaTracts tends to create large erroneous clusters and small (low-cardinality) outliers. When number of clusters (parameter $h$ in hierarchical clustering) is 10, two large incorrect clusters are generated and
the rest are outliers. As $h$ increases, some appropriate bundles start to form. But even at $h = 20$, fiber bundles incorrectly cluster together. For comparison Figure 6.6(f) shows the two-step clustering result.

Figure 6.7(b) shows the MetaTracts, clustered using K-means clustering by proximity alone, after being embedded in a $m$-dimensional (lower) space. We vary $K$ from 10-20. After the dimensionality reduction, the lower dimensional space does not preserve well the spatial context. As a consequence MetaTracts which are in reality far away are grouped together. Even when $k$ is set to twenty, very few correct fiber bundles are identified. In comparison, the two-step approach described above is simple, robust and extracts fiber bundles correctly.

### 6.4.4 Sampling MetaTracts

To ensure that the MetaTracts capture the features of all the fiber bundles correctly we uniformly seed the entire volume, which generates a considerably large number of MetaTracts. This increases the time and space requirements of the technique. Orientation clustering which performs eigenvalue and eigenvector computations on large matrices is particularly resource consuming.

To ensure that we can handle large datasets and at the same time preserve all features, we start by uniformly seeding the entire volume and generating MetaTracts. We then sub-sample the MetaTracts and perform clustering and fiber bundle extraction on these sub-sampled MetaTracts. The sampling algorithm is given in Algo. 8. The algorithm keeps track of a current set of MetaTracts and in each iteration adds the MetaTract furthest from the current set.

Distance between two MetaTracts is measured as the maximum of the minimum distances, between the start points of the cylinders which generate the individual MetaTracts.
Algorithm 8. Algorithm to subsample MetaTracts.

Computing all pairwise distances is an expensive operation. We bound the distances between a MetaTract to others by maintaining a closest set and compute distances only when necessary. Note that in step 16 of the algorithm, we compute the distance from MetaTract $M$ to the newly selected MetaTract $M_i$ only if MetaTract $M_i$ is within distance $2 \times M.dist$ of $M.closest$. The idea is that if the distance from $M.closest$ to $M_i$ is more than $2 \times M.dist$, then $M.closest$ is closer to $M$ than $M_i$ so there is no reason to compute $d(M, M_i)$.

With the sampling step the user can decide the “resolution” of MetaTracts by setting the parameter $n$ based on their requirements and computational constraints (see Sec.6.7).
6.5 Visualization of MetaTracts

The output of the clustering step, can be used to visualize the geometric structure of the fiber bundles and answer the general queries itemized in Sec. 6.1. Apart from the direct MetaTracts visualization, we added three additional extensions which were requested by the domain specialists as highly important and useful. Specifically,

- The first extension is to voxelize the original volume according to the clusters each voxel is associated with.
- The second extension is to extract surfaces (triangle meshes) associated with each fiber bundle.
- Finally, we added an interactive tool which allows a complete visual analysis of the fiber bundles.

6.5.1 Voxelization and surface extraction

To voxelize the entire volume based on the clustering results of the MetaTracts, we take the following “voting” approach. We compute a neighborhood around each voxel. We then create a histogram by enumerating the number of voxels of each class (cluster) in this neighborhood. The voxel is then assigned to the class with the maximum number of elements in the neighborhood. Surface extraction is often a crucial requirement for post processing of the data. While Marching Cubes[50] remains the most popular technique, other methods specific to ICT data such as MergeSharp [12] which focuses on extracting sharp edges and corners are also used. We extract the corresponding surfaces from voxel data by binarizing the volume per cluster and extracting the isosurface associated with the largest connected component in the input binary volume.
Figure 6.8(a) and 6.9(d) show the result of voxelization. Figure 6.8(b) shows a single slice of the volume along the XY-plane. Figure 6.8(c),(d) shows examples of extracted meshes.
Figure 6.6: Clustering Results: Orientation clustering (a,b,c), Distance clustering (d,e). (b) Results of K-means clustering ($K = 2$) with MetaTracts (nodes) projected to the top three eigenvectors as major axes. (a) MetaTracts belonging to orientation cluster 1 in blue. (c) MetaTracts belonging to orientation cluster 2 in red, MetaTracts in gray (a,c) show context. (d,e) Hierarchical (distance based) clustering on each orientation cluster. (f) Combined result showing 11 individual extracted clusters.
Figure 6.7: Applying only (a) Hierarchical Clustering and (b) Dimensionality reduction followed by K-means clustering, methods for various numbers of clusters (distance measure is minimal directed Hausdorffs). For comparison the results using two-step clustering is in Figure 6.6(f).
Figure 6.8: Voxelization and surface extraction. (a) Voxelization of data set 1, the inset shows the MetaTracts after clustering. (b) a single slice along the XY-Plane (c) Two of the extracted meshes together (d) The meshes rendered separately.
6.6 Experimental Results

We tested our technique on data sets with varying characteristics. Dataset D1 is described in Sec. 6.2 (Figure 6.2). Dataset D1 has five fiber bundles along Z-axis and six along X-axis for a total of eleven separate fiber bundles. With different cross-section sizes and varying degree of curvature of the bundles, D1 is a complicated dataset. All of the fiber bundles were identified correctly by our technique. Figure 6.6 and Figure 6.8 show the results of D1 decomposed into two orientation clusters with each orientation cluster further decomposed into $h = 10$ clusters followed by voxelization and mesh extraction. Note how the thin and curved purple cluster bundle 7 is extracted well (Figure 6.6(e), Figure 6.8(d)).

Figure 6.9 shows the results of the our dataset 2 (D2). This dataset is characterized by dense fiber bundle arrangement, with flat and thin bundles. It also consists of two major fiber bundle orientation, there are four fiber bundles along the X-axis and five along the Z-axis. The dimensions of D2 are $300 \times 350 \times 300$ and the data type is uint16 (D1 is uint8). Figure 6.9(a) shows the scalar data along with a single slice. The individual fiber bundles are indistinguishable. A green dotted line shows one of the bundles marked by an expert. Figure 6.9(b) shows the result of computing MetaTracts and clustering using our two step approach. The fiber bundles belonging to the individual orientations are shown separately. Figure 6.9(c) shows the combined results. All the fiber bundles have been separated out successfully. Figure 6.9(d) shows the result of voxelizing the entire volume.

**Implementation Details:** MetaTracts is implemented on a Intel Xeon E5-2667 workstation in C++ using ITK [38]. All statistical computation is done in R (free software environment for statistical computing and graphics) [58]. The MetaTracts generation (Sec. 6.3.3) and fiber bundle extraction (Sec. 6.4) are done as pre-processing.
Figure 6.9: Data set with flat thin and compact bundles. (a) shows the volume rendering and a 2D slice with one of the boundaries marked in green, (b) shows the clusters according to individual orientation. (c) shows the complete result. (d) shows the voxelization of (c).
Figure 6.10: (a) Length distribution of individual MetaTracts for a particular bundle (unit for length is grid cube edge length: $2\mu m$).

### 6.7 Parameter Choices

The critical parameters for MetaTracts are $K$ and $h$, where parameter $K$ is used by the K-means during orientation clustering while parameter $h$ which is used by the hierarchical clustering. $K$ denotes the number of major fiber bundle directions. This is known a priori or can be estimated by considering the weaving pattern. Figure 6.11 shows the number of MetaTracts in each cluster when the orientation cluster 1 (Figure 6.6(a)) is hierarchically clustered into ($h$) 10 clusters. It also shows the minimum, maximum and median length of the MetaTracts in each of the resulting clusters. The ground truth was five separate fiber bundles. We observe that the major fiber bundles remain well separated in accordance with our ground truth and the small clusters (clusters 2, 3, 4, 5 and 7) have very few elements and can be easily discarded. Thus we see that our framework is robust to the choice of parameter $h$ for hierarchical clustering. This is an appealing trait of the proximity based
hierarchical clustering, thus providing good results even when exact $h$ might be unknown. The robustness to parameter variation also reinforces our two-step approach to clustering.

The following parameters were fixed for all the tests. We set the reliable Hessian threshold $R_H$ to be 0.3. A $R_H$ of 0.0 would mean all points have reliable local orientation which would cause spurious MetaTracts detection. A very high $R_H$ would lead to a decline in number of MetaTracts produced. Coefficients $\alpha$ and $\beta$ in $R_H$ are as explained in Frangi et al. [28] and set to 0.5. The length and the radius parameters for the cylinders of MetaTracts decide how coarse our approximation of the fiber bundles are. These are dependent on the underlying fiber characteristic and the weaving pattern. Larger cylinders will handle noisy local orientation better as it inspects a larger number of candidate points to extend the fiber. We used 10 and 2 for length and radius (measured in grid voxel size), respectively for all tests. A simpler geometry (e.g., D2) was experimentally found to handle larger cylinders better. $\eta$ in Sec. 6.4.2 decides how quickly the hierarchical clustering converges, experimentally values 0.3 to 0.6 removed $1.2\% - 5\%$ of fibers (total number of fibers $\sim 10000$)
and gave similar results. We set $n$ the number of sampled MetaTracts to 10,000 for all our test cases. Parameter $n$ intuitively acts as “resolution” for the MetaTracts. Large $n$ captures the features better and generates smoother fiber bundles.

$\alpha$ and $\beta$ in equation 6.3 decide how quickly the value of the factor decays; we have used integer values between [7-10] and half the length of an individual cylinder respectively. Our number of fiber bundle directions is limited. Thus even for small $m$ (cardinality of lower dimension in orientation clustering), the distinction between the orientation clusters is preserved quite well. We compared $m = 3, ..., 7$ experimentally without any change in results.

### 6.8 Limitations

A key assumption of the method is “connectivity” (Sec. 6.2). If the “connectivity” criteria is not fulfilled due to noise in the image, then the generated MetaTracts will be inaccurate. The clustering process also assumes that the fiber bundles have a minimum width.
Chapter 7: Conclusions

This dissertation characterizes some of the problems associated with surface reconstruction from industrial computed tomography data and describes novel methods for extracting isosurfaces and visualizing them.

Broadly two topics have been discussed. The first topic involves the problem of extracting isosurfaces with sharp edges and corners from scalar fields sampled on regular grid, specifically industrial computed tomography data. The key contribution in this regard is the algorithm SHREC.

SHREC provides a technique for computing isosurface vertex location, selecting a well-spaced subset of grid cubes on or near the sharp features, merging grid cubes around selected cubes and constructing isosurface triangles and quadrilaterals. The resulting isosurfaces preserve sharp edges and corners. Through extensive experiments we have shown that resulting isosurfaces outperform existing algorithms.

A complementary problem for sharp isosurface reconstruction is finding correct gradients in the presence of discontinuity and noise. Algorithm RELIGRAD provides one way that the problem of finding correct gradients in the neighborhood of a sharp feature can be approached. Instead of attempting to produce correct gradients at all grid vertices, RELIGRAD identifies correct gradients based on their agreement with neighboring gradients. SHREC along with RELIGRAD provides a complete framework for reconstructing sharp edges and corners, starting directly from industrial computed tomography data.
The second topic involves developing a method to extract and visualize fibers in composites from low resolution image scans, due to the timely need to study and improve these composites for a large variety of engineering applications. MetaTracts can separate fiber bundles and help domain experts interactively visualize the composite. Once again our results show clear advantages over existing techniques.

7.1 Future Directions

While SHREC shows promising results, we do not provide any theoretical guarantees. To this end, a general technique for reliable edge decimation for dual contouring based algorithms which reconstructs provably correct sharp edges and corners remains elusive. Similarly, while we provide certain guarantees in [14], we do not provide theoretical guarantees for RELIGRAD on industrial computed tomography data. One key problem is the fast rate of decay of scalar grid values orthogonal to the embedded surfaces. Once again a unifying theory which selects provably good gradients would be ideal. In a separate direction, with the recent proliferation of rapid prototyping through 3D printing, a compelling direction would be to test the accuracy of 3D printing the extracted sharp meshes compared to the original CAD models.

MetaTracts provide a novel method for carbon composite visualization. Currently, this is restricted to qualitative visualization. The domain experts strongly agree that extending MetaTracts to extract quantitative information would be the next logical next step.

Currently, MetaTracts is restricted to certain composite materials with limited weaving directions. An interesting step would be to extend the algorithm to tackle broader class of composites. In this regard an extension of the MetaTracts approach which is parallelizable, robust and accurate would be interesting and useful direction for future investigation.
Bibliography


