Interactive Visualization of Multi-Dimensional Scientific Data (Ph.D. Thesis)

Daniel R. Schikore

Report Number:
97-053
INTERACTIVE VISUALIZATION OF MULTIDIMENSIONAL SCIENTIFIC DATA

Daniel R. Schikore

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD-TR #97-053
December 1997
INTERACTIVE VISUALIZATION OF MULTIDIMENSIONAL SCIENTIFIC DATA

A Thesis
Submitted to the Faculty of Purdue University by Daniel R. Schikore

In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

August 1997
This thesis is dedicated to my parents,
Robert and Barbara Schikore,
and to my fiancée,
Katherine Price.
ACKNOWLEDGMENTS

Throughout the course of my education and research, there have been many individuals who have supported and encouraged my academic development.

I would first like to thank my thesis advisor, Professor Chandrajit Bajaj, for his years of guidance in my research and educational activities. His enthusiasm for learning, teaching, and discovering has been an inspiration.

I thank my advisory committee, Professors Robert Lynch, Robert Oglesby, and Jörg Peters, for discussions and useful comments on my work and dissertation.

I am grateful for the sound educational foundation that I received prior to entering Purdue, during my years at McCluer North Senior High School, in Florissant, Missouri. Above all, the enthusiastic and devoted teachers at McCluer North taught me to enjoy learning. In particular, I thank Paul Schroeder and Dave Reindl for encouraging me and teaching me to strive to better myself.

During my time at Purdue, a number of individuals have taken a role in encouraging my academic development. I thank Professor Burgess Davis, Dr. William Gorman, Professor Eugene Spafford, Dr. Ying Wang, and Professor Robert Zink for their support.

I have had the privilege of working with and learning from many students and associates in my work with the Shastra Lab at Purdue University. I thank Vinod Anupam, Brian Bailey, Fausto Bernardini, Professor Greg Blaisdell, Professor Richard Borgens, Jindon Chen, Steve Cutchin, Brad Duerstock, Susan Evans, Insung Ihm, Hae-Young Lee, Kwen-Nan Lin, Raimund Merkert, Jeff Pajor, Valerio Pascucci, Eric Price, Hongxin Qin, Andrew Royappa, Professor Elisha Sacks, Professor GuoLiang Xu, Peinan Zhang, Weiping Zhang, and Guozhang Zhuang. Portions of the research
described in this dissertation are the result of collaborations with Fausto Bernardini, Jindon Chen, and Valerio Pascucci.

I am grateful to the many staff of the Department of Computer Sciences at Purdue University for providing a stable and supportive working and computing environment.

I will not forget the many friends at Purdue and in the Department of Computer Sciences and the good times we shared.

I was fortunate to have the opportunity to interact with outside researchers and practitioners during two summer internships with B-Division at Lawrence Livermore National Laboratory in Livermore, California. The interaction with individuals at LLNL has been of great value to my understanding of the problems faced in real-world application of my research. I thank Sean Ahern, Barry Becker, Bob Broderick, Eric Brugger, Linnea Cook, Roger Crawfis, and Chris Wynn for many useful discussions during and since my time at LLNL.

My family has been a constant source of support and encouragement. I thank my mother and father, Barbara and Robert Schikore, for encouraging my curiosity and for always being there when I needed them. My sister and brother, Melanie and Matthew Schikore, have always encouraged me.

Finally, I thank my fiancée, Katherine Price, for sharing with me the good and the bad times during the course of this exciting and challenging experience. Her support and encouragement have given me strength.

A portion of this research was sponsored by a Graduate Fellowship from the National Science Foundation under grant GER-9253915-02.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>xv</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.1.1 Sources of Data</td>
<td>2</td>
</tr>
<tr>
<td>1.1.2 Problem Statement and Challenges</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Classification of Interactive Techniques</td>
<td>4</td>
</tr>
<tr>
<td>1.2.1 Approximation</td>
<td>6</td>
</tr>
<tr>
<td>1.2.2 Efficient Data Structures</td>
<td>7</td>
</tr>
<tr>
<td>1.2.3 Coherence</td>
<td>10</td>
</tr>
<tr>
<td>1.2.4 Optimization for Graphics Architecture</td>
<td>11</td>
</tr>
<tr>
<td>1.2.5 Optimization for Parallel Architecture</td>
<td>13</td>
</tr>
<tr>
<td>1.2.6 Hardware Solutions</td>
<td>13</td>
</tr>
<tr>
<td>1.3 Outline of This Thesis</td>
<td>13</td>
</tr>
<tr>
<td>2. PRELIMINARIES</td>
<td>15</td>
</tr>
<tr>
<td>2.1 Mesh Description</td>
<td>15</td>
</tr>
<tr>
<td>2.1.1 Mesh Classification</td>
<td>16</td>
</tr>
<tr>
<td>2.1.2 Adjacency Terminology</td>
<td>18</td>
</tr>
<tr>
<td>2.2 Data Types</td>
<td>19</td>
</tr>
<tr>
<td>2.3 Data and Gradient Interpolation and Approximation</td>
<td>19</td>
</tr>
<tr>
<td>2.4 Scalar Field Topology</td>
<td>20</td>
</tr>
<tr>
<td>2.4.1 Classification of Critical Points</td>
<td>21</td>
</tr>
<tr>
<td>2.4.2 Examples</td>
<td>22</td>
</tr>
<tr>
<td>2.5 Other Definitions and Terms</td>
<td>23</td>
</tr>
<tr>
<td>3. MESH SIMPLIFICATION</td>
<td>25</td>
</tr>
<tr>
<td>3.1 Related Work</td>
<td>26</td>
</tr>
<tr>
<td>3.1.1 Terrain Simplification</td>
<td>27</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.1.2 Geometry Simplification</td>
<td>27</td>
</tr>
<tr>
<td>3.1.3 Hierarchical Representations</td>
<td>31</td>
</tr>
<tr>
<td>3.2 Simplification</td>
<td>34</td>
</tr>
<tr>
<td>3.2.1 Simplifying Terrains</td>
<td>34</td>
</tr>
<tr>
<td>3.2.2 Vector-Valued Images</td>
<td>40</td>
</tr>
<tr>
<td>3.2.3 Surfaces with Multi-valued Functions</td>
<td>44</td>
</tr>
<tr>
<td>3.3 Hierarchical Representations</td>
<td>49</td>
</tr>
<tr>
<td>3.3.1 Simplification Hierarchies</td>
<td>50</td>
</tr>
<tr>
<td>3.4 CAD Model Reconstruction</td>
<td>52</td>
</tr>
<tr>
<td>3.4.1 Problem Statement</td>
<td>53</td>
</tr>
<tr>
<td>3.4.2 Reconstruction Algorithm</td>
<td>54</td>
</tr>
<tr>
<td>3.5 Summary</td>
<td>57</td>
</tr>
<tr>
<td>4. ACCELERATED ISOCONTOURING</td>
<td>60</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>60</td>
</tr>
<tr>
<td>4.1.1 Cell Triangulation</td>
<td>61</td>
</tr>
<tr>
<td>4.1.2 Cell Search</td>
<td>66</td>
</tr>
<tr>
<td>4.1.3 Cell Traversal</td>
<td>71</td>
</tr>
<tr>
<td>4.1.4 Summary of Prior Work</td>
<td>73</td>
</tr>
<tr>
<td>4.2 Seed Set Construction</td>
<td>75</td>
</tr>
<tr>
<td>4.2.1 Cell Connectivity</td>
<td>75</td>
</tr>
<tr>
<td>4.2.2 Seed Sets</td>
<td>78</td>
</tr>
<tr>
<td>4.2.3 Greedy Climbing</td>
<td>80</td>
</tr>
<tr>
<td>4.2.4 Sweep Filtering</td>
<td>84</td>
</tr>
<tr>
<td>4.2.5 Responsibility Propagation</td>
<td>90</td>
</tr>
<tr>
<td>4.2.6 Seed Set Results</td>
<td>97</td>
</tr>
<tr>
<td>4.3 Range Queries</td>
<td>97</td>
</tr>
<tr>
<td>4.3.1 Interval Tree</td>
<td>99</td>
</tr>
<tr>
<td>4.3.2 Segment Tree</td>
<td>103</td>
</tr>
<tr>
<td>4.3.3 Bucket Search</td>
<td>107</td>
</tr>
<tr>
<td>4.3.4 Search Structure Discussion</td>
<td>109</td>
</tr>
<tr>
<td>4.4 The Contour Spectrum</td>
<td>110</td>
</tr>
<tr>
<td>4.4.1 Contour Attributes</td>
<td>119</td>
</tr>
<tr>
<td>4.4.2 Real Time Quantitative Queries</td>
<td>124</td>
</tr>
<tr>
<td>4.5 User Interface</td>
<td>126</td>
</tr>
<tr>
<td>4.5.1 Rule-based Contouring</td>
<td>127</td>
</tr>
<tr>
<td>4.5.2 Future Work</td>
<td>128</td>
</tr>
<tr>
<td>4.6 Summary</td>
<td>129</td>
</tr>
<tr>
<td>5. SUMMARY</td>
<td>131</td>
</tr>
</tbody>
</table>
LIST OF REFERENCES ........................................... 134
VITA ............................................................. 150
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Seed set sizes for the three presented algorithms, compared with the checkerboard approach and the total number of cells.</td>
<td>98</td>
</tr>
<tr>
<td>4.2</td>
<td>Comparison of the theoretical complexities of the three search structures for performing an interval query.</td>
<td>109</td>
</tr>
<tr>
<td>4.3</td>
<td>Comparison of the storage requirements in typical implementation of the three search structures.</td>
<td>109</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>Data acquisition by (a) scanning devices (b) microscopy (c-d) numerical simulation</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Typical set of graphics primitives</td>
<td>12</td>
</tr>
<tr>
<td>2.1</td>
<td>The cell model and the voxel model for data representation</td>
<td>16</td>
</tr>
<tr>
<td>2.2</td>
<td>Type of grids for data representation</td>
<td>17</td>
</tr>
<tr>
<td>2.3</td>
<td>Nested relationships of grid classes</td>
<td>18</td>
</tr>
<tr>
<td>2.4</td>
<td>Scalar critical point classifications</td>
<td>21</td>
</tr>
<tr>
<td>2.5</td>
<td>Isocontours (dotted) of part of a scalar field along with the critical points and integral curves</td>
<td>21</td>
</tr>
<tr>
<td>2.6</td>
<td>A surface representation of the function in Figure 2.5</td>
<td>22</td>
</tr>
<tr>
<td>2.7</td>
<td>Visualization of density in a pion collision simulation. (a) Scalar topology displayed with color-mapping. (b) Isocontours overlayed with scalar topology demonstrate the orthogonal relationship between them.</td>
<td>23</td>
</tr>
<tr>
<td>2.8</td>
<td>(a) Scalar topology diagram of the wave function computed for a high potential iron protein. (b) Close-up of image (a) near critical points.</td>
<td>24</td>
</tr>
<tr>
<td>3.1</td>
<td>The curve simplification problem. Given a dense set of samples along a curve and an error parameter ε, determine a minimal subset of the points such that the tolerance region is not violated.</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>A multiresolution triangulation represented as a directed acyclic graph (DAG). Arcs encode dependency based on overlapping extent. A cut through the graph selects the triangles associated with the intersected arcs as well as the triangles associated with the nodes above the cut.</td>
<td>33</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>General simplification operations applied to a local region of the mesh in (a). In (b), the resulting triangulation is formed by collapsing the deleted point to an adjacent point. In (c), the retriangulation is general and may be chosen to minimize the error introduced.</td>
<td>35</td>
</tr>
<tr>
<td>3.4</td>
<td>Measurement of introduced errors at black vertices and the fragmentation into linear error regions. The introduced error at the white vertices and along the boundary is zero.</td>
<td>36</td>
</tr>
<tr>
<td>3.5</td>
<td>1D illustration of the accumulated error bound representation. The error intervals ([e^-, e^+]) define a tolerance region for each point in the domain.</td>
<td>37</td>
</tr>
<tr>
<td>3.6</td>
<td>Geometric interpretation of the accumulation of error bounds. The introduced error (e_i) contributes to the error interval as illustrated.</td>
<td>38</td>
</tr>
<tr>
<td>3.7</td>
<td>Illustration of propagation of error bounds. As each successive point is deleted, the error bounds on each segment are updated to reflect the minimum bound between the new segment and the original polyline.</td>
<td>39</td>
</tr>
<tr>
<td>3.8</td>
<td>Simplification of a test function sampled on a 64x64 grid.</td>
<td>41</td>
</tr>
<tr>
<td>3.9</td>
<td>MRI density simplification. Original image (a) consists of 130,050 triangles. Simplified image (b) consists of 1,193 triangles with an error bound of roughly 10%. Image (c) shows the coarse triangulation.</td>
<td>42</td>
</tr>
<tr>
<td>3.10</td>
<td>Greyscale image simplification.</td>
<td>43</td>
</tr>
<tr>
<td>3.11</td>
<td>Simplification of a color image.</td>
<td>45</td>
</tr>
<tr>
<td>3.12</td>
<td>Simplification of a color image.</td>
<td>46</td>
</tr>
<tr>
<td>3.13</td>
<td>Problems which can occur in defining mappings between surfaces. (a) a planar projection may not be one-to-one. (b) The minimum distance mapping is not bi-directional (c) Global minimum distance may relate separate connected components or remote regions of the same component.</td>
<td>47</td>
</tr>
<tr>
<td>3.14</td>
<td>Illustration of geometric error. The geometric error interval defines a tolerance volume as well as a tolerance region for each individual point.</td>
<td>48</td>
</tr>
<tr>
<td>3.15</td>
<td>Simplification of an isocontour of an engine.</td>
<td>49</td>
</tr>
<tr>
<td>3.16</td>
<td>Iterative construction of the DAG through edge collapse operations.</td>
<td>51</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>3.17</td>
<td>Adaptive triangulation is performed dynamically by (a) coarsening operation on the cut or (b) refinement of the cut</td>
<td>52</td>
</tr>
<tr>
<td>3.18</td>
<td>Dynamic triangulation of the golf club. Image (a) presents the dynamic triangulation from the perspective of the viewer, while the image in (b) displays the same triangulation with the viewing frustum, illustrating the adaptive nature of the display</td>
<td>53</td>
</tr>
<tr>
<td>3.19</td>
<td>The complete reconstruction process. (a) Point sampling. (b) 3D Delaunay triangulation. (c) Alpha-solid. (d) Simplified mesh. (e) Sharp features. (f) Support mesh. (g) A-patches. (h) Reconstructed model</td>
<td>55</td>
</tr>
<tr>
<td>3.20</td>
<td>Types of candidate vertices. (a) A &quot;smooth&quot; vertex (all dihedral angles are larger than feature angle). (b) A vertex along a feature edge. (c) A corner</td>
<td>56</td>
</tr>
<tr>
<td>3.21</td>
<td>Reconstruction and piecewise-smooth A-patch fitting. (a) Scanned points. (b) α-solid. (c) Simplified mesh. (d) Reconstructed model (different colors identify different surface patches)</td>
<td>58</td>
</tr>
<tr>
<td>4.1</td>
<td>Standard cell representation for contour computation in a structured grid</td>
<td>62</td>
</tr>
<tr>
<td>4.2</td>
<td>15 distinct vertex colorings</td>
<td>63</td>
</tr>
<tr>
<td>4.3</td>
<td>Topological inconsistency associated with the original marching cubes</td>
<td>64</td>
</tr>
<tr>
<td>4.4</td>
<td>A two dimensional bilinear saddle and its contour configurations</td>
<td>65</td>
</tr>
<tr>
<td>4.5</td>
<td>Two topologically consistent triangulations with respect to the shared face. Note that additional distinct topological configurations exist due to additional face saddles on the non-shared faces</td>
<td>66</td>
</tr>
<tr>
<td>4.6</td>
<td>Spatial hierarchical cell decompositions for accelerating the search for isocontours</td>
<td>67</td>
</tr>
<tr>
<td>4.7</td>
<td>The (a) 1D value space and (b) 2D span space representations for range-space searches</td>
<td>68</td>
</tr>
<tr>
<td>4.8</td>
<td>Illustration of contour propagation. The active surface is traced through adjacent cells</td>
<td>72</td>
</tr>
<tr>
<td>4.9</td>
<td>Illustration of the &quot;checkerboard&quot; approach to sufficient seed sampling. Black cells are on the checkerboard, while a number of grey cells are also required in the seed set</td>
<td>73</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>4.10 Illustration of w-connected components. On the left are four contour components for a particular isovalue w. On the right a portion of the graph G is displayed, corresponding to the three w-connected components. Displayed in green is a w-path between two of the nodes.</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>4.11 Greedy climbing approach to seed cell selection. Grey cells represent the selected seed cells. Yellow cells have been processed and removed from consideration, while red cells represent the current front of cells from which the next seed cell will be chosen.</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>4.12 Results of seed selection by contour climbing (76/7938 cells)</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>4.13 Illustration of one-pass contour sweeping. Contour components are computed as they are crossed by the sweep line.</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>4.14 Tangent conditions of contour with sweep direction</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>4.15 One-pass seed selection by forward sweep</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>4.16 Conditions for determining a local maxima along the ( \vec{l}_\perp ) direction. In linear cells (a-b) the maxima lie along cell edges. With regular cells (c) the maxima remain along edges, though the conditions may change along the length of the edge. In non-linear cells as simple as the non-axis-aligned bi-linear cell (d), maxima may occur in cell interiors.</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>4.17 Three dimensional examples of local minima along the sweep direction.</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>4.18 Degenerate minima which occur with greater frequency in grids of regular topology with integer-valued data.</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>4.19 The cells on the current sweep plane are processed in regular order. A bit flag is turned on when a local maximum exists on the top edge of a cell.</td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>4.20 Results of seed selection by directional sweep (296/7938 cells)</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>4.21 Illustration of responsibility propagation. Each cell processes input responsibilities and produces output responsibilities.</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>4.22 Results of seed selection by range propagation (206/3969 cells)</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>4.23 A set of segments representing cell ranges</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>4.24 Interval tree for the intervals given in Figure 4.23</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>4.25</td>
<td>Segment tree for the segments given in Figure 4.23</td>
<td>104</td>
</tr>
<tr>
<td>4.26</td>
<td>Bucket search structure for the intervals given in Figure 4.23</td>
<td>107</td>
</tr>
<tr>
<td>4.27</td>
<td>Number of seeds (a) and search structure storage requirements (b) for the Eagle Pass USGS Terrain Data (1201x1201)</td>
<td>111</td>
</tr>
<tr>
<td>4.28</td>
<td>Number of seeds (a) and search structure storage requirements (b) for a sample function (64x64)</td>
<td>112</td>
</tr>
<tr>
<td>4.29</td>
<td>Number of seeds (a) and search structure storage requirements (b) for the Hipip data (64x64x64)</td>
<td>113</td>
</tr>
<tr>
<td>4.30</td>
<td>Number of seeds (a) and search structure storage requirements (b) for the LAMP Climate Data (35x41x15)</td>
<td>114</td>
</tr>
<tr>
<td>4.31</td>
<td>Number of seeds (a) and search structure storage requirements (b) for the SOD data (97x97x116)</td>
<td>115</td>
</tr>
<tr>
<td>4.32</td>
<td>Comparison of preprocessing time required for the 5 seed cell extraction algorithms. The interval tree is used as the search structure in all cases</td>
<td>116</td>
</tr>
<tr>
<td>4.33</td>
<td>Comparison of preprocessing time required for the 5 seed cell extraction algorithms. The interval tree is used as the search structure in all cases</td>
<td>117</td>
</tr>
<tr>
<td>4.34</td>
<td>Comparison of preprocessing time required for the 5 seed cell extraction algorithms. The interval tree is used as the search structure in all cases</td>
<td>118</td>
</tr>
<tr>
<td>4.35</td>
<td>Comparison of query time for the interval tree, the segment tree, and the bucket search structures. Query time computed as an average over 1000 searches and is plotted as a function of isovalue</td>
<td>118</td>
</tr>
<tr>
<td>4.36</td>
<td>(left) A 2D scalar field displayed as a terrain. (right) The portion of an isocontour contained in a single triangle</td>
<td>120</td>
</tr>
<tr>
<td>4.37</td>
<td>Area computation for the continuous range of isocontours contained in a single tetrahedron</td>
<td>122</td>
</tr>
<tr>
<td>4.38</td>
<td>2D area computation by integration of the length function (L(w)). The shaded region corresponds to the area less than or below the isovalue. The area above the isovalue is computed symmetrically</td>
<td>123</td>
</tr>
<tr>
<td>4.39</td>
<td>Top: MRI scan data of a heart. Bottom: the corresponding contour spectrum with in yellow the normalized gradient</td>
<td>124</td>
</tr>
</tbody>
</table>
4.40 A 2D scalar field (top) with the associated contour spectrum and in white the relative contour tree (bottom). .................................................. 126

4.41 Example 1D contour spectrum interface .................................................. 127

4.42 Example 2D contour spectrum interface .................................................. 128

4.43 Isocontour of a CT scan of an engine (top) automatically selected at the isovalue corresponding to the maximum of the weighted gradient spectrum (bottom). .................................................. 129
Schikore, Daniel R. Ph.D., Purdue University, August 1997. Interactive Visualization of Multidimensional Scientific Data. Major Professor: Chandrajit L. Bajaj.

Scientific data visualization concerns the manipulation of sampled and computed data for comprehensive display. The goal of the visualization is to bring to the user a deeper understanding of the data, as well as any underlying physical laws and properties. In this thesis, we review the techniques contributing to interactive visualization of multidimensional and multivariate data. We describe a set of tools which we have developed for interactive data visualization, exploration, and interrogation. Our work draws on the fundamentals of data field representations and properties as well as efficient hierarchical structures for processing and querying data. We describe a novel approach for simplifying meshes with guaranteed error bounds in both geometry and associated functions, and demonstrate the ability to build multiresolution hierarchical representations using our approach. We introduce a new computational framework for the extraction of isocontours from scalar valued data. The search for cells intersected by an isocontour is accelerated through the use of range query data structures. We present three seed set construction algorithms, of varying complexity and performance, which reduce the storage requirements of the search structure without penalty in the query complexity. We analyze three search structures of varying space and query complexity, demonstrating that our approach of reducing the size of the search structure introduces additional freedom in the overall algorithm architecture, allowing adaptation to application dependent problems. We conclude with a discussion of open problems and extensions.
Visualization of scientific data is critical to the understanding of the physical world around us, from the study of the earth, sky and sea to the construction of bridges, airplanes, and space vehicles. In the continuing quest for understanding of the human body and molecular interactions, visualization plays an increasingly important role in our ability to uncover the underlying physical laws and properties governing the behaviors that we observe.

Visualization is inherently linked to human perceptual issues. The visual senses are extremely adept at taking in large amounts of information and abstracting from this the patterns and shapes which are present. The ability to create meaningful visual representations of scientific data is crucial to furthering our ability to interpret and understand the large amounts of data we are capable of measuring and computing today, and the even greater size of the data that we will examine in the future.

Interactive visualization of data has the potential to bring the user to a higher level of understanding. Static images can be misleading and mask important features of the data, while motion in visualization brings out hidden features which are inherently dynamic. Interactive manipulation and control of visualization is an important tool which allows scientists to explore more quickly and focus on the region of interest. Delays or inconsistencies in interactive environments tend to be distracting and cause the user to lose focus. In environments which are immersive, motion is critical, to the point that delays or inconsistencies prevent the "suspension of disbelief." In this case there is a desire to bound response time using time-critical response techniques.

In this chapter, we give an overview of interactive visualization paradigms, briefly review prior work in the area and outline our contributions and the organization of the remainder of this thesis.
1.1 Motivation

Measurement, simulation and experimentation occur across the sciences, generating a constantly growing stream of scientific data for interpretation. Interactive visualization accelerates the understanding process, in addition to providing the tools for new applications such as virtual and enhanced surgery simulations and display of real-time data such as current global weather conditions.

1.1.1 Sources of Data

The sources of scientific data are wide and varied. Figure 1.1 illustrates several common data acquisition techniques including both measurement and simulation. Examples used throughout this thesis draw on several of the data types shown. The method of data acquisition introduces several parameters which may affect visualization algorithms, including error bounds or estimates, organization of data, and registration of multiple samples.

1.1.2 Problem Statement and Challenges

The goal of our research is to develop data structures and algorithms which support the development of interactive visualization systems.

We are driven both in consideration of the theoretical time and space complexity of the computational techniques as well as the practice of efficient implementation. Prior work in interactive graphics and visualization systems can be classified in a number of ways, which we review in Section 1.2.

1Visible Female data courtesy the National Library of Medicine, Dr. Michael J. Ackerman, program director
2Histological section of a rat spine courtesy Professor Richard Borgens, Director of the Center for Paralysis Research at Purdue University
3Vortex simulation courtesy Professor Greg Blaisdell, Purdue University School of Aeronautical Engineering
4Global climate simulation courtesy Professor Robert Oglesby, Purdue University Department of Earth and Atmospheric Science
In this thesis, we address two primary components affecting interactive visualization environments. Efficient hierarchical data representation and efficient computational techniques are required to support interactive visualization with large scientific data. Our contributions include:

- Development of practical techniques for measuring the local errors introduced by simplification operations and bounding the global error accumulated by multiple applications. Our techniques begin with simple scalar fields and extend easily to multi-valued fields.

- Extension of our simplification technique to arbitrary surfaces with multi-valued functions defined on them. Geometric error in the surface as well as functional error in the data are bounded in a uniform manner.
• Construction and traversal of a multiresolution data structure encoded during the simplification process. By varying the extraction criteria, the dynamic triangulation can adapt to changes in viewing direction, lighting conditions, and other viewing parameters.

• Acceleration techniques for multidimensional isocontouring of both structured and unstructured data. We present three algorithms for the construction of seed sets, a subset of the cells of a mesh from which all intersected cells can be efficiently located.

• An analysis of three data structures for performing fast range queries on sets of cells, quantifying the complexity tradeoffs for building and storing the search structures and performing queries for intersected cells.

1.2 Classification of Interactive Techniques

It is important to define the aspects of visualization which must be addressed by interactive visualization systems. Interactive visualization can be broken down into three major components [Baj95]:

Computation – the ability to rapidly perform computations associated with visualization. This may include computing a polygonal approximation to an isosurface of a scalar function, or the computation of a particle trace through a time-dependent vector field, or other computation which involves extracting an abstract object or representation from the data being examined.

Display – the ability to quickly display or render the visualization. Display encompasses both computed visualizations as listed above, as well as direct scene display methods such as volume ray-casting and image-based methods such as light fields.
**Interrogation** – the ability to interactively query a displayed visualization for the purpose of extracting quantitative information from what is inherently a qualitative visual analysis.

Many approaches for interactive visualization have been developed which address one or more of the above aspects. Below we present a general classification of tools and techniques for interactive visualization. This classification is not meant to be exhaustive or with hard boundaries, but it provides a convenient mechanism for presentation of a large number of works in the area of interactive visualization.

**Approximation** – Perhaps the most widely used optimization is approximation of exact results. While a degree of approximation is inherent in most of the computational techniques used in visualization, it is common practice to further approximate results, either with or without bound on introduced error, to increase interactivity.

**Data Structures** – Through efficient and hierarchical structuring of regular and irregular data, computation, display, and querying can all be accelerated.

**Coherence** – Exploiting coherence is key to efficiency across the field of computation, and is used in several ways to permit interactive visualization.

**Optimization for Graphics Architecture** – Modern graphics workstations are optimized for display of certain primitives. Therefore significant advantage can be gained by transforming data into the optimal format for rendering before display or otherwise tailoring visualization techniques to take advantage of specialized hardware.

**Optimization for Parallel Architecture** – Data partitioning and parallel design permit acceleration of traditional sequential algorithms on massively parallel architectures and clusters of workstations.
**Hardware Solutions** - Generic graphics workstations provide a set of primitives which can be rendered quickly. Custom solutions facilitate interactive visualization by designing the hardware to suit the visualization.

In the following subsections we further expand on the classifications above and present a partial collection of the prior work in each area.

### 1.2.1 Approximation

Interactive visualization is often impeded by the sheer size of the data being visualized. One approach for reducing this stress is through reduction in the number of primitives used to represent the data with a specified level of fidelity. Reduction of detail can refer to elimination of geometric detail, data detail, or both.

Simplification of geometric detail is commonly used in real-time environments and simulations. Interactive frame rates are critical in such applications as flight simulators, Virtual Reality walk-throughs, etc. The goal of geometry simplification is to produce a reduced detail (reduced number of primitives such as triangles or points) representation for a given object, based on certain criteria which are designed to provide a measure of the error between the two representations.

Geometric simplification techniques [SZL92, HH93, Tur92, HDD+93, CVM+96] and terrain simplification techniques [DFNP84, DFP85, PDDT94, SP92, Tsa93, WJ92, FL79] will be discussed in more detail in Chapter 3.

Wavelets [Mal89, DL92] have been utilized for their multiresolution applications in many areas of computer graphics and visualization [SDS95a, SDS95b, SDS96], including image compression [Luc92b, DJL92a], surface description [DJL92b, EDD+95, CPD96], tiling of contours [Mey94] and curve and surface editing [FS94].

A number of multiresolution volume hierarchies have been proposed for developing adaptive volume rendering and isocontouring [Mur92, Mur95, CDM+94, WV94].

Funkhouser et al. describe adaptive display algorithms for rendering complex environments at a sustained frame rate using multiple levels of detail [FS93].
In visualization, there is a need to accurately represent and display data. In some cases, high accuracy in the simulation of a physical model has a large cost in computation time. Small changes in integration and interpolation techniques, for example, may provide significant speedups in software, and may even allow acceleration using existing hardware. It has been demonstrated that the high computational cost of volume rendering in the spatial domain can be replaced by an asymptotically faster computation in the frequency domain [Lev92, Mal93, TL93].

Several researchers have examined interfaces which allow for selective specification of regions of interest, which may be rendered at a higher resolution while adopting a lower resolution or approximation for the remaining data [BSP+93, CMS94, LW90].

1.2.2 Efficient Data Structures

The application of special data structures has been used to accelerate all aspects of interactive visualization. Well designed data structures allow rapid traversal of data for visualization computations and direct visualization, faster display of general scenes, and faster resolution of queries on the data.

In particular, three data structures are commonly used to aid visualization:

- **Octree** — a uniform space subdivision
- **BSP Tree** — BSP (binary space partition) trees subdivide space into regions based on binary classification
- **Topology Graph** — Topology graphs partition data into regions in which the behavior is simply described

1.2.2.1 Octrees

Octrees (and their counterpart the Quadtree) are used to successively subdivide a 3D (2D) space by three (two) non-parallel planes. Although not necessary for the general definition, the planes are generally parallel to each of the three coordinate axes, and thus perpendicular to each other. A typical representation for an octree
would be a tree with leaves of order 0 and nodes of order 3. Non-leaves can store additional information which summarizes the information contained in its subtree on a large scale. Octrees simplify the traversal of 3D data by allowing large regions of the data to be skipped based on information stored at higher levels.

Isosurfacing is a common visualization task. It is also a task for which brute force processing of dense data is generally extremely wasteful. Wilhelms and Van Gelder [WV90b] estimate that 30% - 70% of the total time in isosurfacing is wasted in visiting cells which have no intersection with the surface of interest. In [WV90a], they apply the use of octrees to isosurface generation by storing minimum and maximum density values at non-leaves in the tree. In this way, large portions of data can be skipped during traversal of the tree by determining whether the surface of interest is between the minimum and maximum density for all cells represented by a non-leaf. Experimental results demonstrate a speedup of 25% - 75% over the brute-force marching method in which all cells are visited.

Octrees have also been used in accelerated volume rendering [Lev90]. Image-order ray casting of dense data is costly due to the large numbers of voxels intersected by each individual ray. By imposing an octree structure on dense rectilinear data, spatial coherence can be exploited in order to skip large regions which are nearly transparent and contribute little to the final image. Levoy reports rendering times 50% - 80% faster than the brute force method by using a hierarchical enumeration.

1.2.2.2 BSP Trees

Binary space partition (BSP) trees are a data structure which provides a simple method for binary classification in space [FKN79, FKN80]. Based on splitting planes, all objects are classified depending on which side of the plane they lie. Objects which span the plane must be split or otherwise dealt with. The result can be represented as a binary tree. Objects lying in the split plane may be stored at the node, while the left child is a BSP tree for one half-space of the space which was split, and the right child is a BSP tree for the remaining half-space. The hierarchical subdivision of space
can then be exploited in several ways. One advantage is that the BSP tree provides a natural method for rendering certain scenes from back-to-front. Through traversal of the tree, one can determine at each node which half-space is facing the user, and render the furthest half-space first, followed by the node, followed by the remaining half-space. Such orderings provide a hidden surface ordering, as well as a correct ordering for proper display and accumulation of transparent surfaces. In addition, the binary classification provides a method for determining spatial relationships between objects. For example, determining if a given point is inside or outside of a closed object, determining if two objects have collided, and determining intersection of a ray with an object are all performed more efficiently through traversal of the tree. Naylor describes methods for using such traits to speed up the computation of ray tracing [NT86]. A drawback of the method is the splitting aspect. Naylor has also addressed the issue of how to efficiently compute partitioning trees without incurring many splits [Nay93], however the method still increases the number of primitives, which may be costly in certain applications. BSP trees are also applied to compression of discrete representations of data [SN92].

1.2.2.3 Topology Graphs

Topology graphs of a set of data provide a partitioning of the domain such that the behavior of a function within the subset is more simple than the behavior of the data as a whole. One such useful partitioning results in regions in which the data is monotonic. One application of this technique is rapid extraction of isocontours [IK95]. Given a decomposition of a set of data which is monotonic, all contours of a given threshold must cross one of the edges of the graph at some point. Searching the graph for intersections provides a rapid method for isolating seed points from which isocontours can be generated. Results have shown an speedup of between 2.5 and 9.5 depending on the particular data.
1.2.3 Coherence

Mesh simplification techniques take advantage of coherence in object space to represent meshes at varying resolutions. Another class of algorithms for interactive visualization take advantage of coherence in the images produced through visualization. There is typically little change between frames of an interactive navigation of an environment. Exploiting this coherence provides another method for reducing computation time in generating visualizations.

1.2.3.1 View Interpolation

"In-betweening" has become a popular technique in approximating intermediate results given a starting and ending reference. View interpolation applies this to images [CW93]. In navigation of a data set, one would expect there to be little change between two consecutive scenes. By generating a mapping from one image to a future image, interpolation provides a continuous transition between two relatively similar projections. If the cost of producing a visualization is large compared to the cost of interpolating between images, we have a situation in which we could increase the interactivity by precomputing a future scene and interpolating toward the new projection. This technique can be likened to that of keyframing, in which distinct positions and orientations of an object are specified, and defined interpolations are performed in order to provide a smooth mapping between the key frames. In the context of image interpolation in real-time, key frames would be generated on the fly, based on the current direction of motion. Changes in the direction may be delayed slightly due to the continuation of the interpolation in one direction.

1.2.3.2 Frameless Rendering

Another approach, which exploits the image coherence in interactive display, is frameless rendering [BFMS94]. Typically, a graphical display is double-buffered. An entire scene is rendered in an off-screen buffer, and then quickly mapped to the screen in order to eliminate any discontinuities in the sequence of frames. The following
frame is then rendered off-screen, from scratch, and when the rendering is complete, it replaces the old image being displayed. Again, if we achieve interactivity, there is little difference expected between two consecutive frames. Frameless rendering eliminates the popular notion of off-screen rendering and double buffering altogether. Instead, pixels on the screen are continually updated to reflect the current state of a scene. In order to combat aliasing, the pixels are updated stochastically. When movement is slow, the result may be that all pixels reflect the current view with very little error. When movement is fast, there is a motion-blurring effect followed by a gradual resolution to a correct image when the movement slows again. In such a system, there is never a lag in response, but only a level of accuracy cost in the image display.

1.2.3.3 Rendering Coherence

Rather than using image space coherence to actually eliminate the traditional rendering stage, Yagel et al. use the temporal coherence in the change of view-position to feed rendering information from one frame to the next [YS93]. In the application of volume visualization [Kau91], it is common to exploit spatial coherence by “space-leaping” through empty regions [YS93]. In image space, there is coherence in the locations of these empty spaces, which can be used effectively to speed up space-leaping algorithms during interaction with a volume.

1.2.4 Optimization for Graphics Architecture

Current computer graphics technology allows rapid rendering of certain geometric primitives. By converting the data to be visualized into a format which is efficient for hardware rendering, significant speedups can be realized. In certain cases, the format of the data is inherently organized for rapid display (see Figure 1.2.4. For example, a typical set of geometric primitives is a strip of quadrilateral elements, such that each quadrilateral shares two vertices with the previous quadrilateral, and
the other two with the next quadrilateral. Graphics hardware is able to take advantage of such a configuration when projecting 3d graphics and rendering them to the screen. Another common primitive is the triangle strip, which consists of two line segments connected by an alternating diagonal. For arbitrary triangular meshes, such a representation must be extracted. While simple brute-force methods generate reasonable results, improved methods have been developed which address both the display efficiency [ESV96] and communication efficiency [TR96].

Another feature of high-end graphics hardware, which is beginning to standardize in commodity hardware, is texture-mapping [BN76]. Texture mapping is a general pixel-level operation for adding fine detail to simple geometry. The use of textures allows gross specification of geometry, with fine detail represented in the texture. In its most common form, the specified detail determines the color of the object at a given texture coordinate. The same pixel-level operations may be applied to other surface attributes, such as normals, to simulate wrinkles or bumps on a surface without the excessive fragmentation of primitives required to actually detail the complex geometry [Bli78]. While initially utilized to increase interactivity in the visual simulation arena through realistic rendering of roads, buildings, furniture, etc., texture mapping is being leveraged in new ways to accelerate and create innovative visualization techniques [CM93, CCF94, VK96].
Splatting techniques for rendering scalar and vector volumes use an object space reconstruction kernel which can be efficiently approximated and composited in the image space [Wes90, CM93, MY96, IL95, MHK95].

Computationally intensive visualization, such as volume rendering, can be approximated by primitives which can be rendered directly by common graphics hardware [Luc92a, SH92, ST90, SBM94, WV91, Wil92, YRL+96]. Techniques have also been developed for real-time previewing using coarse approximations of volumes using only lines and points [Sai94].

1.2.5 Optimization for Parallel Architecture

Vector and parallel architectures are designed specifically to address large scale problems and have been applied successfully to several visualization techniques. Volume rendering is a particularly intensive computational task which is amenable to parallel computation [AGS95, ABSS94, Elv92, Lac95, Ma95, SK94, Sil96]. Significant speedups have been reported for performing isocontouring on massively parallel architectures as well [HH92, SHL+96].

1.2.6 Hardware Solutions

While a majority of the work we have discussed is geared toward tuning computations for the widely available graphics workstation, there is an obvious alternative: tune the graphics workstation for the widely desired visualization. Scalar volume rendering is a particular task for which specialized hardware architectures have been designed and implemented [Ben95, KS94, PKC94, PK96].

1.3 Outline of This Thesis

The remainder of this thesis is organized as follows. In Chapter 2 we review the concepts and definitions which are used in describing and manipulating scientific data. In Chapter 3 we discuss mesh simplification and present an efficient mechanism for global bounding of error in simplification of terrains, images, and surfaces. We
describe how these techniques can be used to generate hierarchical representations of meshes, and also present a modification of our simplification technique which aids in the reconstruction of smooth CAD models from unorganized points. In Chapter 4 we discuss the problem of isocontouring and how the fundamental process of searching for intersected cells may be accelerated. We present three seed cell generation algorithms and analyze three data structures for performing the associated range query. We also introduce the use of a new quantitative user interface component for isovalue selection. Chapter 5 summarizes our results and presents a set of open problems and directions for future work.
2. PRELIMINARIES

In this chapter, we introduce the nomenclature used in scientific visualization and throughout this thesis, including representations for describing scientific data and terminology used in computation and manipulation of scientific data.

2.1 Mesh Description

Scientific data frequently consist of a number of discrete samples which are measured by a data acquisition device or computed through numerical simulation. A grid or mesh may be implicit in the organization of the discrete samples or it may be computed, based on the positions of the samples, either prior to simulation or subsequent to the measurement of data. This grid serves as a basis for interpolation of data values over the data domain.

A variety of terms are used in describing a mesh. The basic terminology that we use is described below:

- **Node/Point** – The coordinates of a sample (independent variables).
- **Cell** – A region of the data domain (convex or non-convex) bounded by two or more points.
- **Data** – Measured or computed dependent variables associated with the points or cells of a mesh.
- **Meta-Data** – Auxiliary information about a mesh, such as timestep of the simulation, associated error values or data confidence, date of creation, units of measure, etc.
Figure 2.1: The cell model and the voxel model for data representation

Data are usually associated with the points or cells of a mesh. We refer to these two representations as the cell model and the voxel model, as illustrated in Figure 2.1.

2.1.1 Mesh Classification

The characteristics of the points and cells of a mesh lead to general classifications as presented by Speray and Kennon [SK90] and others. In the remainder of this thesis we use the following terminology in describing mesh types (illustrated in Figure 2.2):

- **Regular** – An axis-aligned lattice of evenly-spaced points with an implicit regular cell topology.

- **Rectilinear** – An axis-aligned lattice of variable-spaced points with an implicit regular cell topology.

- **Curvilinear** – An lattice of points with an implicit regular cell topology.

- **Simplicial** – A set of points with an associated simplicial decomposition.

- **Unstructured** – A set of points and an associated complex of convex cells.
Regular, rectilinear and curvilinear meshes can be grouped by the feature that the topology of the points and cells is implied by a regular ordering of the data. Such a *structured* mesh can be described by the number of points $n_i$ in each dimension. The total number of points is

$$\prod_{i=0}^{d} n_i$$

while the total number of cells is

$$\prod_{i=0}^{d} (n_i - 1)$$

The three structured mesh types form a nested set of classes in that regular $\subseteq$ rectilinear $\subseteq$ curvilinear, hence algorithms which are suitable for curvilinear meshes are suitable for all structured meshes described here.

Likewise the unstructured complex of convex cells is both a generalization of the simplicial mesh as well as all structured meshes, leading to the generalized hierarchy shown in Figure 2.3.
2.1.2 Adjacency Terminology

A variety of terms have been introduced for describing the topological connectivity relations between points and cells in a mesh [LVG80, Sri81, USH82, VW94a]. We adopt a variation of the terminology of [VW94a] which is generalized for arbitrary grids and dimension.

**Definition 1** Two cells of a $d$-dimensional mesh are said to be $k$-adjacent for $1 < k \leq d$ if the dimension of their common boundary is at most $k$.

The complete topological connectivity of a mesh can be determined from the $(d-1)$-adjacencies of the mesh. For convenience we use the general term *face-adjacent* or simply *adjacent* as a dimension independent equivalent.
2.2 Data Types

Data refers to the measured or computed values at the points or cells of the mesh. The terms function or variable may be used equivalently. Following are the types of data which are frequently used in scientific visualization and the representations which are used in this thesis.

- **Scalar** – single-valued function consisting only of a magnitude (such as pressure, temperature, or helical density)

\[ w = \mathcal{F}(x) \]

- **Vector** – multi-valued function consisting of several components (such as velocity)

\[
\begin{pmatrix}
  w_1 \\
  w_2 \\
  \vdots \\
  w_n
\end{pmatrix} =
\begin{pmatrix}
  \mathcal{F}_1(x) \\
  \mathcal{F}_2(x) \\
  \vdots \\
  \mathcal{F}_n(x)
\end{pmatrix}
\]

- **Tensor** – (such as stress or strain)

\[
\begin{pmatrix}
  w_{11} & w_{12} & \cdots & w_{1n} \\
  w_{21} & w_{22} & \cdots & w_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  w_{m1} & \cdots & w_{mn}
\end{pmatrix} =
\begin{pmatrix}
  \mathcal{F}_{11}(x) & \mathcal{F}_{12}(x) & \cdots & \mathcal{F}_{1n}(x) \\
  \mathcal{F}_{21}(x) & \mathcal{F}_{22}(x) & \cdots & \mathcal{F}_{2n}(x) \\
  \vdots & \vdots & \ddots & \vdots \\
  \mathcal{F}_{m1}(x) & \cdots & \mathcal{F}_{mn}(x)
\end{pmatrix}
\]

- **Multivariate** – a collection of scalar quantities

\[
\left\{ \mathcal{F}(x), \mathcal{G}(x), \cdots, \mathcal{K}(x) \right\}
\]

2.3 Data and Gradient Interpolation and Approximation

An important aspect of data representation and computation is the method for approximating function values and gradients within a computational cell. Piecewise
linear interpolation is widely used in visualization techniques, but suffers due to the inability to model smooth curves and surfaces with a small number of cells.

Higher degree piecewise polynomial interpolation methods are attractive for their improved modelling capabilities but often introduce complexity in the computation and rendering.

Most of the analysis presented in this thesis is based on the use of a piecewise linear interpolation model. In certain cases we explicitly state that no restriction is made in order to generalize to other data approximation methods.

2.4 Scalar Field Topology

Scalar field topology describes a function in terms of its criticalities and the relations between them, providing a foundation for describing the structure of a scalar field [BS96b]. In this section, we give a brief review of the definitions and outline an algorithm for computation of scalar topology of scalar fields defined over a 2D domain.

A point \( x \) is a critical point of the function \( F(x) \) if all first-order partial derivatives of \( F \) are zero [MC69]. Critical points play an important role in describing the structure of a scalar field, and hence provide a theoretical framework for visualization operations which process such data.

Critical points may be further classified by the second-order derivative information which describes the change of the function in the local region. Figure 2.4 illustrates some of the non-degenerate situations which can occur in two dimensional functions.

Integral curves are defined as curves which are everywhere tangent to the gradient field of the scalar field \( S \). Integral curves computed near critical points have special significance in segmenting the domain into basic regions. In Figure 2.5 we illustrate the display of integral curves for the surface in Figure 2.6.

The procedure for computing the topology of a scalar field \( S \) can be outlined as follows:

1. Detect critical points of \( S \).
2. Classify points by the Hessian.

3. Integrate critical curves in the gradient field.

2.4.1 Classification of Critical Points

Information about the behavior of the gradient field near a critical point is obtained by analysis of the Hessian of $S$:

$$
\begin{pmatrix}
\frac{\partial^2 S}{\partial x^2} & \frac{\partial^2 S}{\partial x \partial y} \\
\frac{\partial^2 S}{\partial y \partial x} & \frac{\partial^2 S}{\partial y^2}
\end{pmatrix}
$$

The eigenvalues and eigenvectors of this matrix determine the behavior of the gradient field and hence the scalar field near the critical point, much the same as for the behavior of a general vector field [BD92, HH91]. One notable difference is that for a gradient field, the matrix of derivatives is symmetric ($\frac{\partial^2 S}{\partial x \partial y} = \frac{\partial^2 S}{\partial y \partial x}$).
and therefore the eigenvalues will all be real. This is to be expected, as imaginary eigenvalues indicate rotation about the critical point, and a gradient field is an irrotational vector field. This observation allows us to simplify the classification of critical points for computation of scalar topology.

A positive eigenvalue corresponds to gradient flow away from the critical point, while a negative eigenvalue indicates gradient flow toward the critical point. In the case of a saddle point, there is gradient flow toward and away from the critical point, distinguishing them from the field behavior near other critical points. In this case, the eigenvectors, corresponding to the positive and negative eigenvalues, define the separatrices of the saddle in the directions of flow toward and away from the critical point, respectively. Integral curves computed in the directions of the separatrices are used to segment the field in a meaningful way.

2.4.2 Examples

The properties of scalar topology diagrams are illustrated through 2D and 3D examples. Figure 2.7 displays the scalar topology diagram for a 2D density field from a simulation of a subatomic particle collision. Figure 2.8 displays the integral curves of the wave function of a high potential iron protein calculation.
2.5 Other Definitions and Terms

Other definitions for terms which are commonly used in graphics and visualization include:

- *Transfer function* – A function which describes how data are mapped to display parameters. A transfer function usually maps a scalar value $w$ to color and opacity values for display:

\[
\begin{pmatrix}
  r \\
g \\
b \\
a
\end{pmatrix} =
\begin{pmatrix}
  R(w) \\
  G(w) \\
  B(w) \\
  A(w)
\end{pmatrix}
\]

- *Computational coordinates* – Refers to the parameter space of a regular mesh. Many visualization techniques can take advantage of the regular topology of curvilinear meshes by operating in computational coordinates.
Figure 2.8: (a) Scalar topology diagram of the wave function computed for a high potential iron protein. (b) Close-up of image (a) near critical points.
3. MESH SIMPLIFICATION

There are many tradeoffs in visualizing scientific data. Accuracy of representation and display can be critically important. This factor tends to cause scientific meshes to become very large, in order to accurately represent the complexity of the underlying data. Interactivity in visualization can greatly enhance the user experience, however real-time interaction with large meshes designed for accuracy is often not possible. It is often the case that only a small amount of accuracy can be sacrificed for the sake of increased interactivity with the data, without rendering the visualization useless for interpretation.

The need for accuracy and fine detail, combined with our ability to sample and compute extremely dense data, has long competed with our ability to display or transmit geometric information at interactive rates.

Two primary approaches to improving the performance of interactive 3D systems have emerged:

- **Simplification** – It is not uncommon for a polygonal database to contain millions or even tens of millions of polygons, far more than the number of pixels being displayed. For objects (or parts of objects) which are distant from the viewer, a coarse representation of the object may be sufficient.

- **Visibility** – Equally important is the notion that, for many applications, a small portion of the entire database is visible from any given view. Visibility analysis goes beyond view-frustum culling to handle occlusion, pruning portions of the scene which are in the view frustum but occluded by objects near to the viewer.

Simplification has a long history of usage. Early systems used hand-crafted models at varying levels of detail, rendering the appropriate level at run-time based on the
position of the viewer [Cla76]. Automatic methods for generating multiple levels of
detail have now begun to give way to hierarchical representations of objects which
permit extraction of appropriate levels-of-detail on the fly [Lue96]. In this dynamic
triangulation, the level-of-detail may be constant over the entire object, or it may be
adaptive, giving high-resolution detail in regions of interest or in regions closer to the
viewer, and varying continuously to lower resolution in regions of less importance or
at distances further from the viewer.

In this chapter, we present our results in simplification of multi-valued terrains
and surface meshes. In Section 3.2.1 we introduce the case of scalar-valued terrains
and our method for bounding approximation errors. In Sections 3.2.2 and 3.2.3 we
extend this method to multi-valued terrains and then to surfaces with multi-valued
functions defined on them. In Section 3.3 we demonstrate how these techniques have
been used to generate hierarchies suitable for dynamic triangulation with bounded
error in both geometry and functions. In Section 3.4 we demonstrate how our simpli-
fication techniques have been modified and applied to the problem of reconstructing
CAD models from unorganized point scans [BBCS96], with explicit detection and
preservation of sharp features.

3.1 Related Work

Mesh reduction, or simplification, refers to a broad category of techniques de-
dsigned to trade space and complexity for accuracy in representation of a triangulated
planar, surface, or volume mesh. Related work comes from several research com-
munities, including Geographical Information Systems (GIS), Virtual Reality (VR),
and Scientific Visualization. Each community has a similar goal for achieving inter-
activity with large datasets. In the following sections we review the work in terrain
simplification, geometry simplification, and extensions of simplification techniques to
hierarchical representations.
3.1.1 Terrain Simplification

A driving application for reduction of height-fields has been GIS. A simple 1D version of the terrain approximation problem is illustrated in Figure 3.1. A wide range of techniques are based on extraction of key points or edges from the originally dense set of points, followed by a constrained Delaunay triangulation [DFP85, DFNP84, FL79, PDDT94, Tsa93, WJ92]. Silva et al. introduce a greedy method for inserting points into an initially sparse mesh, reporting both better and faster reduction compared to a public domain implementation of a terrain reduction tool [SMK95]. A survey by Lee reviews methods for computing reduced meshes by both point insertion and point deletion [Lee91].

Figure 3.1: The curve simplification problem. Given a dense set of samples along a curve and an error parameter $\epsilon$, determine a minimal subset of the points such that the tolerance region is not violated.

3.1.2 Geometry Simplification

A wide variety of algorithms have been developed for the simplification of geometry. Geometry simplification can be seen as an extension of terrain simplification, in that terrains may be viewed as surfaces in three dimensions as well as functions over a two dimensional domain. The primary difference in applying general geometry
simplification techniques to terrain is in the definition of *error*, which we will discuss in more detail in Section 3.2.3.1. In many cases, geometry simplification is restricted to *manifolds*, though in some cases arbitrary complexes are allowed. We review a collection of prior work which are clustered into one of several categories based on the overall simplification approach.

**Vertex insertion/deletion**

A large class of geometry simplification algorithms are based on successive application of one or more topological mesh operators, such as the *edge collapse*, which contracts an edge of the mesh to a point, or *vertex deletion*, in which a vertex and adjacent triangles are removed and replaced with a covering of the resulting hole.

Schroeder et al. compute reduced representations for dense triangular surface meshes such as those computed by Marching Cubes [LC87] or similar isosurfacing algorithms [SZL92]. Vertices in the dense mesh are examined and classified based on geometric features in the triangulation surrounding the vertex. If error criteria are satisfied, the vertex is deleted and the resulting hole is retriangulated. Retriangulation is guided by local edges detected in the classification stage and aspect ratios of new triangles. Several passes over the object successively remove vertices until no vertex satisfies the criteria for removal. There is no error propagation, and therefore no guarantee on the amount of accumulated error in the final representation.

Hamann applies a similar technique in which triangles are considered for deletion based on curvature estimates at the vertices [Ham94]. Reduction may be driven by mesh resolution or, in the case of functional surfaces, root-mean-square error.

Ronfard et al. apply successive edge collapse operations to compute a wide range of levels-of-detail for triangulated polyhedra [RR96]. Edges are extracted from a priority queue based on a computed *edge cost* such that edges of lesser significance are removed first.
Cohen et al. introduce *Simplification Envelopes* to guide mesh simplification with global error bounds [CVM+96]. Envelopes are an extension of *offset surfaces* which serve as an extreme boundary for the desired simplified surface.

**Region merging**

Hinker et al. perform "geometric optimization" on triangular surface meshes by grouping faces into contiguous sets which are nearly co-planar [HH93]. Points interior to a region and points along nearly linear boundaries of regions are deleted, and the resulting hole is retriangulated. Kalvin et al. cluster mesh faces into *superfaces*, triangulating the resulting polygons for a simplified representation [KT96].

**Filtering**

Filtering techniques are capable of producing a large range of simplified models through application grouping and merging rules. Filtering techniques are capable of reducing objects to a minimal representation through successive applications.

Subsampling is a simple type of filtering which is easily applied to subdivision meshes for which there exists a natural remeshing when nested sets of vertices are successively deleted. The major drawback to subsampling is that there is no bound on the error which is introduced through its application.

Rossignac et al. use clustering and merging of features of an object based on a regular spatial subdivision [RB93]. Clustering approaches have the advantage that small features which are geometrically close but not topologically connected can be grouped and merged for higher rates of simplification. In this scheme long, thin objects may collapse to an edge and small objects may collapse to a point.

He et al. provide more control over subsampling of regular grids by filtering the simplified mesh at each step [HHK+95]. The regular grid corresponds to a sampling of the signed-distance function of a 3d surface. A multi-resolution triangle mesh is
extracted from the resulting multi-resolution volume buffer using traditional isosur-
fac ing techniques.

Optimization

Optimization methods define measures of energies for point sets or triangulations based on an original mesh, and use interactive optimization to minimize these energies in forming a simplified mesh.

Turk computes simplified polygonal surfaces at a desired number of vertices [Tur92]. Contrast this with the point insertion and deletion methods which are usually driven by error computations rather than desired resolution. Given the desired number of vertices, point repulsion on the polygonal surface spreads the points out. A mutual tessellation of the original triangulation and the introduced points followed by deletion of the original vertices guarantees that the topology of the polygonal surface is maintained. Point repulsion is adjusted based on estimated curvature of the surface, providing an adaptive triangulation which maintains geometric features.

Hoppe et al. perform time-intensive mesh optimization based on the definition of an energy function which balances the need for accurate geometry with the desire for compactness in representation [HDD+93]. The level of mesh reduction is controlled by a parameter in the energy function which penalizes meshes with large numbers of vertices, as well as a spring constant which helps guide the energy minimization to a desirable result.

Multi-resolution analysis

Multi-resolution analysis is a structured mathematical decomposition of functions into multiple levels of representation. Through the use of wavelet transforms [Mal89,
DL92], a hierarchical representation of functions can be obtained by repeatedly break- ing the function into a coarser representation in addition to a set of perturbation co- efficients which allow the full recovery of the original representation from the coarse representation. During reconstruction from the wavelet representation, sufficiently small wavelet coefficients can be omitted, resulting in a coarser approximation to the original data, with a bound on the amount of error [Luc92b, DJL92a, SDS96]. Further extensions have provided similar basis for the decomposition of surfaces [DJL92b]. Muraki [Mur92] applies wavelets in 3D to compute multi-resolution models of 3D volume data. Isosurfaces and planar cross sections of the resulting data show little change in image quality with large reductions in the amount of data representing the volume.

3.1.3 Hierarchical Representations

Recently a great deal of attention has been concentrated in the area of hierar- chical mesh representations and algorithms. Hierarchies for dynamic simplification can be generated using the same techniques which are used in generating discrete levels-of-detail. The entire simplification process which is presented in Section 3.2 can be considered as a preprocessing phase, with an auxiliary multi-resolution triangulation created as a result. We describe several recent approaches for generating and manipulating hierarchical representations of meshes.

Lindstrom et al. adopt a recursive triangulation of a regular terrain and compute preprocessing metrics at various levels of resolution which permits real-time adaptive triangulation for interactive fly-through [LKR+96]. The technique is limited to terrain models represented as regular grids of elevation values. Duchaineau et al. use a split queue and a merge queue to adaptively maintain an optimal triangulation from a recursive subdivision of a regular terrain [DWS+97]. Gross et al. adopt a similar quadtree subdivision and perform adaptive surface meshing by examining the wavelet coefficients at each level of detail [GGS95].
Zhou et al. have extended the subdivision technique to 3D regular grids and demonstrated the ability to extract contours of varying resolution from the multiresolution tetrahedral representation [ZCK97].

Hoppe introduced *Progressive Meshes* (PM), which are created by incrementally applying *edge collapse* operations to a triangle mesh, based on an energy minimization strategy [Hop96]. The final data structure consists of a base mesh and a list of detail records. Scalar attributes are handled by incorporating them into the energy function. The PM representation was extended to impose a hierarchical ordering on edge collapse operations, facilitating the dynamic triangulation for selective refinement based on viewing parameters. More recently Hoppe has constructed hierarchies of edge collapse operations, which allow for adaptive refinement based on viewing direction and related criteria [Hop97].

Multi-resolution analysis encodes a surface as a base triangle mesh and a sequence of correction detail terms [EDD+95]. This type of representation is restricted to meshes with "subdivision connectivity." Meshes with general connectivity can only be approximated, thus limiting their effectiveness when used as a level-of-detail tool for arbitrary meshes.

Hierarchies of Delaunay triangulations are used by de Berg and Dobrindt to generate dynamic triangulations of terrain [dBD95]. The multi-resolution data structure is represented as a directed acyclic graph (DAG). The root node contains the lowest resolution triangulation, while interior and leaf nodes represent refinements to portions of their parent node(s). An adaptive triangulation is extracted from the hierarchy by an iterative process of adding triangles (beginning at the root), and iteratively refining those whose error is not satisfied. The error function may be chosen for the particular application needs. Because of the DAG structure, it is possible that triangles which were previously "accepted" (not refined) must subsequently be refined, in order to generate a consistent triangulation.
Figure 3.2: A multiresolution triangulation represented as a directed acyclic graph (DAG). Arcs encode dependency based on overlapping extent. A cut through the graph selects the triangles associated with the intersected arcs as well as the triangles associated with the nodes above the cut.

Puppo describes an improved DAG representation which explicitly encodes the topological overlap between adjacent nodes in the DAG [Pup96]. Adaptive triangulations are extracted from the DAG by performing a cut through the arcs of the DAG. A subset of the triangles above the cut combine to create the adaptive mesh. An improved algorithm for extracting the cut was presented in [DMP97]. A representative DAG created by edge collapse operations is given in Figure 3.2.

Cohen and Levanoni adopt a tree representation for Delaunay triangulated terrain [COL96]. A tree structure is obtained by restricting the selection of points to be deleted to points interior to a group, corresponding to a node in the tree. In order to reduce memory requirements, a tree of smaller height is created by clustering into larger groups, allowing deletion of more vertices. The main drawback of this approach is that vertices along boundaries that separate nodes at low levels of detail are never removed (if not in the root). This can easily create long slivers in coarse models, and seriously constrains the effectiveness of the simplification when applied to meshes which are initially dense.
Schmalstieg and Schaufler propose a tree encoding of levels of detail [SS97], based on clustering of vertices and removal of degenerate triangles. The clustering operations are stored in a tree, which can be encoded in a compact binary format suitable for transmission. The authors briefly describe the possible use of the tree hierarchy for adaptive, viewpoint-dependent rendering.

### 3.2 Simplification

There are a number of considerations affecting the design of simplification techniques suitable for visualization of scientific meshes. The simplification technique presented here has been designed to consider several factors affecting visualization of scientific meshes:

- It extends easily to handle multi-valued functions defined over the mesh
- It maintains strict bounds on the error which is introduced in each function and the geometry (in the case of general surface simplification)
- It is fast enough to be practical for large meshes, on the order of millions of triangles

We begin by describing a simple error representation and accumulation scheme for scalar-valued terrains in Section 3.2.1. This representation is extended to vector-valued functions in Section 3.2.2 and finally to the case of surfaces with multi-valued functions in Section 3.2.3.

#### 3.2.1 Simplifying Terrains

A terrain is simply a scalar-valued function defined over a 2D domain. We will refer to the function value $H(x)$ as the height, although the underlying data may represent many measures, such as CT density in a 2D slice of medical data or velocity magnitude of a laminar flow.
We assume that the domain is discretized by a triangular mesh $M$ of $n_t$ triangles $T_i$ and $n_v$ vertices $V_j$. The function $H(x)$ is defined for $x = V_j$ and is linearly interpolated over each triangle. $H(T_i)$ is used to represent the interpolant over a triangle.

![Figure 3.3: General simplification operations applied to a local region of the mesh in (a). In (b), the resulting triangulation is formed by collapsing the deleted point to an adjacent point. In (c), the retriangulation is general and may be chosen to minimize the error introduced.](image)

We restrict our simplification to use the original vertices and to local operations such as vertex deletion and vertex collapse, as illustrated in Figure 3.3. The new mesh which results from the $k$th step of simplification is called $M^k$, and its triangles are denoted $T_i^k$. We further define $H^k(x)$ to represent the piecewise defined height function associated with the mesh $M^k$. Note that for all triangles not involved in the operation at step $k$, $T_i^{k-1} = T_i^k$, and thus $H^k(T_i^k) = H^{k-1}(T_i^{k-1})$. Initially, we have $M^0 = M$ and $T_i^0 = T_i$.

### 3.2.1.1 Local Measurement of Error

The first step in maintaining an error-bound on the simplification is to measure the error introduced by deleting a vertex or collapsing an edge. Computing the introduced error is particularly simple due to the limited local influence of the operation.

As illustrated in Figure 3.4, a local retriangulation due to removal of a vertex replaces a set of triangles $T_i^{k-1}$ with triangles $T_i^k$. The error incurred at any point $x$ in the local domain is quantified by the difference between the interpolated function value in the two triangulations, defined as $e^k(x) = H^{k-1}(x) - H^k(x)$. As the difference of two linear functions is again linear, the introduced error is defined as a piecewise
linear function over the decomposition implied by the intersections of the triangulations, as illustrated in Figure 3.4. The regions of the decomposition are called $R_i^k$ and the (continuous) range of introduced error within the region is $[\varepsilon_i^-(R_i^k), \varepsilon_i^+(R_i^k)]$. Note that the $\varepsilon_i^+/-(R_i^k)$ represent signed error values, and that the minimum and maximum errors occur at the extreme points of the convex regions, and thus can be determined by computing the errors at only a few points in the domain.

3.2.1.2 Global Errors from Local Errors

More important than the measurement of error introduced through one simplification operation is the measurement or approximation of errors which accumulate through successive operations. Features which are desirable in, or even required for, an error approximation strategy include:

- The estimation of error must be strict (must not underestimate the actual error).
- The estimation should be tight, such that the estimate provides a useful measure of the actual error.

At the same time, our desire for scalability and efficiency for extremely large scientific datasets requires that the strategy for bounding errors be simple and not
impose a significant cost in addition to computing the introduced errors. We have designed a simple error representation and accumulation scheme which fulfills these criteria.

**Error representation**

We associate with each triangle two floating-point errors bounds $\epsilon^{-}(T_{i}^{k})$ and $\epsilon^{+}(T_{i}^{k})$ (illustrated in Figure 3.5), such that for $x \in T_{i}^{k}$,

$$
\mathcal{H}^{k}(x) - \epsilon^{-}(T_{i}^{k}) \leqslant \mathcal{H}(x) \leqslant \mathcal{H}^{k}(x) + \epsilon^{+}(T_{i}^{k})
$$

[3.1]

Initially, $\epsilon^{-}(T_{i}^{0}) = \epsilon^{+}(T_{i}^{0}) = 0$, indicating that the original representation is exact. Note that one can easily incorporate input data with known source error bounds into this framework by initializing the error bounds appropriately.

![Figure 3.5: 1D illustration of the accumulated error bound representation. The error intervals $[\epsilon^{-}, \epsilon^{+}]$ define a tolerance region for each point in the domain.](image)

With such a representation in hand, we address the problem of computing the errors $\epsilon^{+/\!-}(T_{i}^{k})$, given $\epsilon^{+/\!-}(T_{i}^{k-1})$, such that inequality 3.1 is maintained as an invariant. We noted previously that the local domain is segmented into regions in which the introduced error varies linearly. Each introduced triangle $T_{i}^{k}$ maps to a
set of regions \( \{R_{i,0}^k \cdots R_{i,r}^k\} \). For each region \( R_{i,j}^k \) we determine the minimal sufficient error range which bounds the existing error as well as the introduced error. Each region \( R_{i,j}^k \) corresponds to a portion of a triangle \( T_{i}^{k-1} \) with known error interval \([\epsilon^-(T_{i}^{k-1}), \epsilon^+(T_{i}^{k-1})]\), and by interval arithmetic we arrive at the following inequality which bounds the error over the region \( R_{i,j}^k \):

\[
\mathcal{H}^k(R_{i,j}^k) - \epsilon^-(T_{i}^{k-1}) - \epsilon_j^-(R_{i,j}^k) \leq \mathcal{H}(R_{i,j}^k) \leq \mathcal{H}^k(R_{i,j}^k) + \epsilon^+(T_{i}^{k-1}) + \epsilon_j^+(R_{i,j}^k)
\]

which leads to the accumulated error interval for the region \( R_{i,j}^k \):

\[
\begin{align*}
\epsilon^-(R_{i,j}^k) &= \epsilon^-(T_{i}^{k-1}) + \epsilon_j^-(R_{i,j}^k) \\
\epsilon^+(R_{i,j}^k) &= \epsilon^+(T_{i}^{k-1}) + \epsilon_j^+(R_{i,j}^k)
\end{align*}
\]

In order to bound the error for the triangle \( T_{i}^k \) it is sufficient to take the maximum of the \( \epsilon^+/-(R_{i,j}^k) \) for all regions \( R_{i,j}^k \) which contribute to triangle \( T_{i}^k \). This process is illustrated geometrically for a simple 1D example in Figure 3.6.

Figure 3.6: Geometric interpretation of the accumulation of error bounds. The introduced error \( \epsilon_j \) contributes to the error interval as illustrated.
3.2.1.3 Ordering of Operations

Critical to the success of an error approximation approach is the order in which vertices are selected for removal. Vertex selection by priority queue approaches have demonstrated simplification performance which spans wide ranges of resolution [RR96] while maintaining quality approximations of the original triangulation.

We apply a priority queue driven approach which initially measures the error introduced by removal of each vertex. At each simplification step, the vertex which introduces the minimum error is extracted and the simplification operation is applied. Subsequent to the deletion of the chosen vertex, neighboring vertices which are affected by the result are reprioritized based on the accumulated error which results from their removal.

3.2.1.4 Results

Our results are illustrated for a variety of scalar valued functions. In Figure 3.8, a sampled test function is displayed at several resolutions. Error bounds and actual
errors are presented for the varying resolutions. Note that for moderate levels of simplification, the error bound remains a good approximation of the actual error.

Figure 3.9 applies the simplification to a 2D MRI of a heart. Despite the presence of significant noise, the algorithm is able to achieve over 99% simplification while maintaining many of the visibly interesting features.

In Figure 3.10 we apply the simplification to a scalar-valued image. Important features such as the eyes and mouth are faithfully represented until the very latest stages of the iterative simplification.

3.2.2 Vector-Valued Images

Scientific data often consists of more than just a single scalar function. In this section we extend the scalar-valued techniques developed in Section 3.2.1 to allow simplification of multi-valued images, defined by a vector-valued function

\[ \{ \mathcal{H}_0(x) \cdots \mathcal{H}_{n_f-1}(x) \} \]

where \( n_f \) is the number of functions defined over the domain. The notation for scalar-valued errors naturally extends to an error-vector representation which quantifies the errors for multi-valued functions. The introduced error-vector for a set of \( n_f \) functions is defined as \( (e_0(x) \cdots e_{n_f-1}(x)) \) where \( e_k(x) = \mathcal{H}_k^{-1}(x) - \mathcal{H}_k(x) \), while the accumulated error interval for function \( i \) over a triangle \( T_j^k \) is \([e_i^-(T_j^k), e_i^+(T_j^k)]\), where \( k \) indicates the state of the mesh after the \( k \)th simplification operation.

3.2.2.1 Prioritizing by Error Norms

The error representation introduced in Section 3.2.1 consists of scalar quantities which are easily compared and ordered. With vector-valued functions we have an introduced error and accumulated error for each component of the vector. Depending on the relation between the components (positive correlation, etc), these independent errors may or may not affect the ordering in a consistent manner. A natural approach to balancing the desire to preserve all components with the need to represent all components over the same triangulation is by defining error norms. In practice, we have
Figure 3.8: Simplification of a test function sampled on a 64x64 grid

(a) 7938 triangles
range of function values: 28.08

(b) 6,267 triangles
error bound: 0.013176
actual error: 0.013176

c) 2,368 triangles
(d) 420 triangles

error bound: 0.101465
actual error: 0.100125
0.8666628
0.486738

(e) 6,267 triangles
(f) 2,368 triangles
(g) 420 triangles
Figure 3.9: MRI density simplification. Original image (a) consists of 130,050 triangles. Simplified image (b) consists of 1,193 triangles with an error bound of roughly 10%. Image (c) shows the coarse triangulation.
Figure 3.10: Greyscale image simplification
experimented with using both the \textit{max-norm} and the \textit{2-norm} of the error vector in order to define a priority ordering of the vertices. The resulting simplification percentages have not differed significantly between these two. The actual effect of the error norm depends highly on the manner in which the data is being visualized. In the case of an \((r, g, b)\) vector-valued full color image, there may be perceptual differences depending on the error norm which is used. For vector-valued data representing velocity of a flow, it may be more meaningful to use a \textit{direction/magnitude} representation. These and other data-dependent error measures require further study.

3.2.2.2 Results

The vector-valued simplification techniques described in this section are applicable to a wide range of input data. Figure 3.11 illustrates the technique on a color image. The image is described by a \((\text{red, green, blue})\) vector-valued function. Priority ordering of vertex deletions is based on the 2-norm of the resulting accumulated error vector.

In Figure 3.12, the multi-valued simplification is applied to a second color image. Again we see that high rates of simplification are achieved with little degradation in image quality.

3.2.3 Surfaces with Multi-valued Functions

Surfaces in both scientific and many other applications domains are frequently rendered with triangles. In order to capture details at widely varying scales, it may be necessary to sample the surface at a high sampling density [Ber96]. In this section we describe how the methods for bounding error in terrains and images can be extended to bound geometric error in surfaces. In visualizing scientific data, it is quite often not only geometry that the user is interested in, but data values defined on the surface as well [BX94, MCG94, Nic93]. We demonstrate that the error bounding strategy for terrains and multi-valued functions carries over to arbitrary surface triangulations provided that an appropriate mapping strategy is used.
Figure 3.11: Simplification of a color image
Figure 3.12: Simplification of a color image
3.2.3.1 Local Correspondence

In functional simplification, as described in Sections 3.2.1 and 3.2.2, the notion of error is well-defined. Because the domain remains the same, there is an implied correspondence or mapping between the data values at a particular sample point.

In the simplification of geometry, the domain itself is being deformed, and it is the error associated with this deformation that we are interested in bounding, in addition to any functions or properties which are associated with the geometry. Measuring the error which is introduced through simplification operations is accomplished by establishing a mapping between consecutive triangulations.

Figure 3.13: Problems which can occur in defining mappings between surfaces. (a) a planar projection may not be one-to-one. (b) The minimum distance mapping is not bi-directional (c) Global minimum distance may relate separate connected components or remote regions of the same component.

A number of mappings have been used to measure local and global changes in simplified surfaces. Mappings such as planar projections and minimum-distance mappings may exhibit undesirable singularities and intersections, as illustrated in Figure 3.13.

Rocchini et al. review many aspects of measuring differences in surfaces and describe a general framework for measuring distances between surfaces [CRS96]. The adoption of a general approach permits the comparison of various simplification techniques in a quantitative manner.
In defining a mapping between surfaces we must bear in mind that quantification of the errors is of utmost importance. Because of our desire to deal with geometry with functions as well as simple surfaces, it is important that we retain a one-to-one mapping between points on the original surface and corresponding points on the simplified surface. The use of a one-to-one piecewise linear mapping will enable us to apply error bounding techniques to surfaces much in the same way that they have been applied to scalar-valued and multi-valued functions.

A simple one-to-one mapping may be established by considering a planar projection direction and performing checks to determine whether the mapping is one-to-one. Cohen et al. recently introduced a method for determining whether such a planar projection exists [CMO97].

### 3.2.3.2 Geometric Error

Geometric error can be quantified with an approach which closely parallels that for terrain and image simplification. We associate an error radius with the geometry, in addition to an error interval for each function. The introduced error-vector which is defined in Section 3.2.2 is augmented to \((\epsilon_k^0(x) \cdots \epsilon_k^{n_f-1}(x) \epsilon_k^g(x))\) where \(\epsilon_k^0(x) \cdots \epsilon_k^{n_f-1}(x)\) represent the errors for the \(n_f\) functions defined on the mesh, and \(\epsilon_k^g(x)\) represents the error introduced in the geometry.

![Figure 3.14](image_url)

**Figure 3.14**: Illustration of geometric error. The geometric error interval defines a tolerance volume as well as a tolerance region for each individual point.
Geometrically, the error radius defines a tolerance volume as illustrated in Figure 3.14. Note than the error interval is not merely a tolerance volume; through the one-to-one mapping between the original and simplified surface, the accumulated errors provide an upper-bound on the distance from each point in the simplified mesh to the unique corresponding point in the original mesh.

3.2.3.3 Results

Figure 3.15 demonstrates simplification of an isosurface extracted from a CT scan of an engine. The original isosurface contained over 144000 triangles, and has been simplified to three discrete resolutions.

![Images showing simplification of an isocontour of an engine](a) 32000 triangles (b) 8000 triangles (c) 4000 triangles

Figure 3.15: Simplification of an isocontour of an engine

3.3 Hierarchical Representations

In this section we address the recent trend toward hierarchical representations for dynamic simplification, and describe how our simplification strategy can be applied in this setting.

Hierarchical representations are useful for several reasons:

- They integrate many levels of detail into a single representation
• They may permit adaptive triangulation to account for viewing direction, clipping planes, lighting conditions, etc.

• They provide a basis for transmission of low levels of detail followed by transmission of additional detail

• They may be used to accelerate operations such as contouring

• Hierarchies may be used to resolve spatial queries quickly

3.3.1 Simplification Hierarchies

The iterative process of simplification by point deletion and/or edge collapse implicitly creates a triangulation hierarchy as described by de Berg and Dobrindt [dBD95] and Puppo et al. [Pup96, DMP97].

We have extended our simplification to construct a directed acyclic graph during the simplification process.

3.3.1.1 Constructing the DAG

Construction of the DAG takes place synchronously with the edge collapse operations. Each edge-collapse operation creates a new node of the DAG. Each node in the DAG consists of:

• a set of triangles segmented into groups

• a maximum error bound associated with each group

• a set of incoming arcs $a^i_n$

• a set of outgoing arcs $a^o_n$

Figure 3.16 illustrates the process of creating a DAG from edge collapse operations.
Figure 3.16: Iterative construction of the DAG through edge collapse operations.
3.3.1.2 Moving the Cut

Dynamic updates on the current triangulation defined by the cut can be performed by applying coarsening and refinement operations to the current cut. Figure 3.17 illustrates the coarsening and refinement operations.

Figure 3.17: Adaptive triangulation is performed dynamically by (a) coarsening operation on the cut or (b) refinement of the cut.

3.3.1.3 Examples

Dynamic triangulation based on view-dependent criteria is demonstrated in Figure 3.18. The screen space error bound of 2 pixels is mapped into world space to determine an allowable adaptive geometric error over the surface of the golf club. Figure 3.18(b) demonstrates that portions of the club furthest from the viewer are simplified more than portions near to the viewer.

3.4 CAD Model Reconstruction

In addition to the interactive performance to be gained from simplified meshes and hierarchical representations, the simplification techniques described here have
practical application for other problems. In this section we briefly describe the application of our simplification techniques to the problem of reconstructing smooth CAD models with sharp features from dense unorganized point scans [BBCS97, BBCS96].

3.4.1 Problem Statement

We are given a collection of points \( P = \{x_i, y_i, z_i\} \). The points are assumed sampled from an unknown surface (an orientable 2-manifold embedded in \( \mathbb{R}^3 \)), the domain \( D \).

We will assume that the sampling is dense and uniform, in the sense that everywhere on the surface the distance between two sampled points is much smaller than the size of the smallest geometric feature of interest. This assumption is usually satisfied by real range data, except for the fact that some parts of the surface might lack data points because they were not accessible by the scanning head.

In some cases one or more scalar fields, defined on the scanned surface, are also sampled at the same points, for example the R, G and B components of color, or
temperature, or some other physical properties. All such fields are reconstructed simultaneously with the geometry.

We have developed a fully automated reconstruction technique capable of producing a model with the following characteristics: (a) It matches the topological characteristics of the object; (b) It is geometrically accurate; (c) It can represent both smooth, curvature continuous surfaces and sharp features such as corners and edges, common in manufactured parts; (d) It is suitable to be used in successive phases of the design and simulation process.

3.4.2 Reconstruction Algorithm

Our algorithm consists of three steps. An initial reconstruction step determines the connectivity between the input points, constructs an initial manifold, and fixes the topology of the reconstructed surface. A simplification step follows to approximate the densely sampled surface by fewer triangles. Finally, a patch-fitting scheme is applied to approximate the original points with a smooth surface fit over the simplified triangle mesh. The entire process is illustrated in Figure 3.19.

3.4.2.1 Reconstruction

Connectivity relationships between points are inferred by first computing a 3D Delaunay triangulation of the set of data points. We then compute the associated family of alpha-shapes [Ede92, EKS83, EM94]. An optimal value of the parameter $\alpha$ is selected and the corresponding alpha-shape, suitably regularized, used as a starting point for our reconstruction.

In [BB97] Bernardini et al. provide a formal proof of sufficient conditions on the sampling to guarantee a homeomorphic and error-bounded reconstruction using alpha-shapes, in addition to presenting an automatic selection strategy for an optimal alpha value.
Figure 3.19: The complete reconstruction process. (a) Point sampling. (b) 3D Delaunay triangulation. (c) Alpha-solid. (d) Simplified mesh. (e) Sharp features. (f) Support mesh. (g) A-patches. (h) Reconstructed model.
3.4.2.2 Surface Simplification

Given a reconstructed triangular mesh (as in Figure 3.19(c)), we next apply a modified simplification algorithm with explicit edge preservation and marking.

Prior to simplification, edges are classified as smooth or sharp based on the dihedral angle the two adjacent triangles. Edges whose angle is less than a user specified threshold are marked as sharp. During simplification, each vertex is classified based on the smoothness criteria for each adjacent edge. The vertex classifications, illustrated in Figure 3.20, are similar to those used by Schroeder et al. [SZL92].

![Figure 3.20: Types of candidate vertices. (a) A "smooth" vertex (all dihedral angles are larger than feature angle). (b) A vertex along a feature edge. (c) A corner.](image)

In order to preserve features, a vertex which lies on a feature edge is constrained to be collapsed along the edge, and edge classifications are propagated to the introduced edges. Note that by propagating feature classifications, we avoid a situation in which repeated coarsening of an object introduces features which were not present originally. This is very important in our application, as the feature classifications will be used not only in the simplification, but also in the patch-fitting stage described in the following section. Vertices which lie at a corner are restricted to remain in the simplified mesh, while smooth vertices are unrestricted and can be removed as described previously. All edge collapse operations use a priority-queue ordering and error propagation mechanism as described in Section 3.2. Vertices which are removed during the simplification are collected and used in the patch-fitting step for least squares approximation.
3.4.2.3 Patch-fitting

Our patch-fitting scheme is based on A-patches of degree three. An A-patch is the zero-set of a polynomial, defined inside a tetrahedron. The polynomial can be expressed in Bernstein-Bézier form, so that its 20 coefficients (or weights) are associated with a regular lattice of control points within the tetrahedron. Constraints on the sign changes of these weights along certain directions guarantee smoothness and single-sheeted properties for the surface patches. A-patches allow an easy formulation of the continuity constraints for objects of general topological genus. They provide more degrees of freedom for data fitting and local shape control than parametric patches of the same degree [BCX95].

Our A-patch fitting scheme interpolates the vertices (and estimated surface normals) of the simplified mesh computed as described above, and approximates the remaining data points. Features tagged as *sharp* during mesh simplification are retained in the resulting piecewise-smooth model. The fitting process begins with the construction of a tetrahedral mesh to act as support for the A-patches. Then, weights for each patch are set to interpolate vertices and sharp features, and least-squares approximate the remaining point. Finally, a fairing and fitting optimization can be applied to improve the quality of the reconstructed model. More details can be found in [BCX95].

Figure 3.21 illustrates the reconstruction process for a dense collection of points scanned from a golf club.

3.5 Summary

We have presented an efficient technique for bounding the error introduced through successive local simplification operations. The error bounding mechanism extends easily from scalar-valued terrain simplification to simplification of multi-valued images and to surfaces with associated functions. The use of local simplification operations leads to the construction of multiresolution hierarchies, which are used to perform
Figure 3.21: Reconstruction and piecewise-smooth A-patch fitting. (a) Scanned points. (b) α-solid. (c) Simplified mesh. (d) Reconstructed model (different colors identify different surface patches).
dynamic triangulation based on view-dependent criteria. The error-bounded geometry simplification has also been applied to the reconstruction of smooth models from unorganized points.
4. ACCELERATED ISOCONTOURING

In addition to efficient and hierarchical representations of data, efficient computational methods are crucial for interactive visualization. In this chapter we address the computational aspect of one of the most common visualization techniques for scalar fields, isocontouring.

A number of contouring surveys exist in the literature, covering both implicit surface triangulation and cell-based approaches [Sab86, NB]. We focus on the case of a scalar function which is piecewise-defined over a d-dimensional mesh. In Section 4.1 we discuss the problems faced in isocontouring and the prior approaches which address these problems. In Section 4.2 we present three seed set construction algorithms which reduce the amount of memory required for search structures which accelerate isocontouring, while maintaining output-sensitive time complexity for computing multiple isocontours. We also discuss how our techniques can be applied to isocontouring of extremely large datasets which cannot be stored in main memory. In Section 4.3 we discuss the use of three alternative search structures, with special consideration for the effects of the preprocessing on the search structures and for the types of data which are frequently encountered. A novel user interface for isocontouring is presented in Section 4.4. The contour spectrum provides quantitative user feedback of metric information such as surface area and volume, in addition to providing a visual display which aids the visualization user in selecting meaningful isovalues.

4.1 Introduction

Isocontouring is a widely used approach to the visualization of scalar data and an integral component of almost every visualization environment. Computation of isocontours has applications in visualization ranging from extraction of surfaces
from medical volume data [Lor95] to computation of stream surfaces for flow visualization [van93]. Inherent in the selection of an isocontour, defined by \( C(w) : \{ x | \mathcal{F}(x) - w = 0 \} \), is that only a selected subset of the data is represented in the result. In many applications, the ability to interactively modify the isovalue \( w \) while viewing the computed result is of great value in exploring the global scalar field structure. In fact, it has been observed in user studies that the majority of the time spent interacting with a visualization is in modifying the visualization parameters, not in changing the viewing parameters [Hai91]. Hence there has been great interest in improving the computational efficiency of contouring algorithms.

In the following subsections, we divide the principal components of cell-based isocontouring algorithms in the following three stages:

- **Cell Triangulation** – Method of computation for determining the component of a contour which intersects a single cell.

- **Cell Search** – Method for finding all cells which contain components of the contour

- **Cell Traversal** – Order of cell visitation may be integrated with (or decided by) the cell search technique, however it nevertheless affects the performance of the isocontour extraction algorithm

In Section 4.1.1 we discuss the problem of cell triangulation and review several approaches which have been presented in the literature. In Sections 4.1.2 and 4.1.3 we present the prior work on search and traversal schemes which provides motivation for our work. We demonstrate that combinations of the approaches in each of these three areas yield dramatic improvements in the interactivity of isocontouring with a small overhead in the required data structures.

### 4.1.1 Cell Triangulation

Cell triangulation concerns the approximation of the component of a contour which is interior to the cell. Triangulation has two distinct components, *interpolation*
to determine a set of points and normals, and *connectivity* to determine the local topology of the contour. While the use of linear interpolation edges of cells is a widely accepted approach, other strategies have been developed to reduce this computational portion of isocontour approximation, such as selecting midpoints along intersected edges [MSS94]. Here we present a summary of the work on the topological aspect of contour triangulation.

![Standard cell representation for contour computation in a structured grid](image)

**Figure 4.1**: Standard cell representation for contour computation in a structured grid

Cell-based contouring algorithms generally begin with a classification of each vertex of a given cell as *positive* (if greater than the isovalue) or *negative* (if less than or equal to the isovalue), which we will refer to as *black* and *white*, respectively. Such a binary classification of the 8 vertices of a regular cell (as in Figure 4.1) leads to a total of $2^8$ or 256 possible configurations. Taking rotational symmetry into account, this can been reduced to 22 distinct cases [LVG80, Sri81]. Based on linear interpolation, cell edges are called *intersecting* if the colors of the endpoints differ, or *non-intersecting* if they are colored the same.

Marching Cubes [LC87] further reduces the number of base cases by assigning complementary triangulation for complementary vertex configurations (*black to white*), resulting in 15 distinct colorings, as shown in Figure 4.2. The full table of the 256 possible vertex configurations can easily be generated from this table of 15 cases.
The use of complementary triangulations reduces the number of base cases, but also introduces a well-known topological inconsistency on certain configurations of shared faces between cubes [Dur88], one case of which is illustrated in Figure 4.3. A number of techniques have been proposed which offer solutions to this inconsistency, which we group into two classes. The first class attempts only to provide consistency along all cell faces, while then second class provides correctness with respect to a chosen model.

Consistency may be achieved simply by subdividing each cell into tetrahedra and using a linear interpolant within each tetrahedron [DK91]. An efficient approach to consistency is to adopt a consistent decision rule, such as sampling the function at the center of the ambiguous face to determine the local topology [WMW86].

Zhou et al. make the point that the tetrahedral decomposition and linear approximation change the function and may still result in incorrect, though consistent, topology [ZCT95]. They propose that the tetrahedral decomposition may be used,
Figure 4.3: Topological inconsistency associated with the original marching cubes provided that intersections along the introduced diagonal are computed for the cubic function which results from sampling the trilinear function across the diagonal of a cell, rather than applying linear interpolation along all edges.

Matveyev sorts all intersections on a regular cell face with respect to an axial direction [Mat94]. The nature of the bilinear interpolant ensures that pairs in the sorting will be connected, as the asymptotes at a saddle point for a bilinear function over an axis-aligned regular cell are parallel to the axes.

The core of the problem along shared cell faces lies in determining the topological connectivity of vertices which are colored the same but which lie diagonally across a face of a cell. Nielson and Hamann propose generating a consistent decision on connectivity by enforcing a topology which is correct with respect to the bilinear interpolant along the face [NH91]. Kenwright derives a similar condition for disambiguating the connectivity on the faces in terms of the gradient of the bilinear interpolant [Ken93]. Natarajan further enforces consistency with the trilinear interpolant for the case of ambiguities which are interior to a cell, which occur when diagonal vertices across the body of the cell are similarly colored but have no edge-connected path of vertices of the same color between them [Nat94].

Karron et al. further discuss the proper treatment of criticalities in isocontouring, proposing a digital morse theory for describing scalar fields [KCM94].
Wilhelms and Van Gelder provide a comprehensive review of the topological considerations in extracting isosurfaces, and demonstrate that gradient heuristics applied at the vertices of a cell are necessary and sufficient to disambiguate the topology of functions which are quadratic [WV90b, VW94b, VW94a].

![Saddle configuration](image)

**Figure 4.4**: A two-dimensional bilinear saddle and its contour configurations

The solution suggested by Natarajan [Nat94] is particularly attractive due to its design to enforce consistency with the trilinear interpolant, a commonly used interpolant for 3d reconstruction and visualization. The situation on faces with colored vertices which are diagonally adjacent can be viewed in two dimensions as in Figure 4.4. The unique saddle point at coordinate $x_s$ of the bilinear interpolant lies interior to the face, and the correct topology can be determined by evaluating the function at the saddle point and comparing it with the isovalue as shown. This topological consistency is carried out further by considering the unique saddle point of the full trilinear interpolant in addition to the six possible face saddles. A simple extension to the marching cubes case table requires sub-cases only for configurations which contain saddles. The sub-cases are indexed by the saddle point evaluations in order to determine a triangulation which is topologically consistent with the trilinear interpolant [Nat94].

For the inconsistent case illustrated in Figure 4.3, several distinct topological triangulations are possible, two of which are illustrated in Figure 4.5.
4.1.2 Cell Search

Because a contour only passes through a fraction of the cells of a mesh on average, algorithms which perform an exhaustive covering of cells are found to be inefficient, spending a large portion of time traversing cells which do not contribute to the contour.

The straightforward approach of enumerating all cells to extract a contour leads to a high overhead cost when the surface being sought intersects only a small number of the cells.

Preprocessing of the scalar field permits the construction of search structures which accelerate the repeated action of isocontouring, allowing for increased interactivity during modification of the isovalue. Many preprocessing approaches and search structures have been presented, which are conveniently classified (similar to the classification presented in [LSJ96]) based on whether the search is in domain space or range space.

4.1.2.1 Domain Search

• Octree search – A spatial hierarchy for accelerating the search process is a natural approach which has been explored by Wilhelms and Van Gelder [WV92, WV90a]. For space efficiency considerations, a partial octree decomposition was developed which groups all cells at the highest level and adaptively approximates
the data through axis-aligned subdivisions which better approximate the data. At each level in the tree, \( \min \) and \( \max \) values for the cells contained in the subtree are stored, providing a means to efficiently discard large spatial regions in the search phase. An analysis presented in [LSJ96] suggests a worst-case computational complexity of \( O(k + k \log \frac{n_c}{k}) \), where \( k \) is the size of the output and \( n_c \) is the number of cells.

![Figure 4.6: Spatial hierarchical cell decompositions for accelerating the search for isocontours.](image)

### 4.1.2.2 Range Search

A large number of search techniques in the recent literature perform the search for intersected cells in the range space of the function. As we are dealing only with scalar-valued functions, range space search techniques have the advantage of being independent of the dimension of the domain. In range space, each cell \( c \) is associated with the continuous set of values taken on by the function over the domain:

\[
R(c) = [\min_{x \in c} F(x), \max_{x \in c} F(x)]
\]

There are two approaches for representing the range space, the 1D \textit{value-space}, in which each range \( R(c) \) is considered as a segment or interval along the real line, and the 2D \textit{span-space}, in which each range \( R(c) \) is considered as a point in 2D [LSJ96],
as illustrated in Figure 4.7. While certain search structures are motivated by one geometric representation or another, others may be effectively visualized in either representation.

![Diagram](image_url)

Figure 4.7: The (a) 1D value space and (b) 2D span space representations for range-space searches

We present a brief summary of the range space approaches which have been proposed in the literature.

- **Min-Max lists** – Giles and Haimes introduce the use of min-max sorted lists of cells to accelerate searching. In addition to forming two sorted lists of cells, the maximum cell range, $\Delta w$, is determined. Cells containing an isosurface of value $w$ must have minimum value in the range $[w - \Delta w, w]$, which may be determined by binary search in the min-sorted array. This active set of cells is purged of cells whose range does not contain $w$. For small changes in $w$, the active list can be updated, rather than wholly recomputed, by adding and purging new candidate cells to the active list. In the worst case, complexity remains $O(n_c)$. 

- **Span filtering** – Gallagher describes an algorithm called span filtering [Gal91], in which the entire range space of the scalar function is divided into a fixed number of *buckets*. Cells are grouped into buckets based on the minimum value taken on by the function over the cell. Within each bucket, cells are classified into one of several lists, based on the number of buckets which are spanned by the range of the cell. For an individual isovalue, cells which fall into a given bucket need only be examined if their span extents to the bucket which contains the isovalue. In the worst case, complexity remains $O(n_c)$.

- **Sweeping simplices** – Shen and Johnson describe a *Sweeping Simplices* algorithm [SJ95], which builds on the min-max lists of Giles and Haimes and augments the approach with a hierarchical decomposition of the value-space. The min-sorted list is augmented by pointers to the associated cell in the max-sorted list, and the max-sorted list is augmented by a "dirty bit." For a given isovalue, a binary search in the min-sorted list determines all cells with minimum value below the isovalue. Pointers from the minimum value list to the maximum value list are followed to set the corresponding dirty bit for each candidate cell. At the same time, the candidate cell with the largest maximum value which is less than the isovalue is determined. As a result, all marked (candidate) cells to the right of this cell in the maximum list must intersect the contour, as they have minimum value below the isovalue and maximum value above the isovalue. Optimizations may be performed when the isovalue is changed by a small delta. One min-max list is created for each level of a hierarchical decomposition of the min-max search space. The overall complexity remains $O(n_c)$ in the worst case analysis.

- **Extrema graphs** – Itoh and Koyamada compute a graph of the extrema values in the scalar field [IK95]. Every connected component of an isocontour is guaranteed to intersect at least one arc in the graph. Isocontours are generated by propagating contours from a seed point detected along these arcs. Noisy
data with many extrema will reduce the performance of such a strategy. Liv­­nat et al. note that in the worst case the number of arcs will be $O(n_e)$, and hence straightforward enumeration of the arcs is equivalent in complexity to enumeration of the cells.

- **Kd-tree** – Livnat, Shen, and Johnson describe a new approach which operates on the 2D min-max span space [LSJ96]. Cells are preprocessed into a Kd-tree which allows $O(k + \sqrt{n_e})$ worst case query time to determine the cells which intersect the contour, where $k$ is the size of the output. It is reported that in the average case, $k$ is the dominant factor, providing optimal average complexity.

- **Lattice search** – The same authors, with Hansen, have described a technique which demonstrates improved empirical results by using an $L \times L$ lattice search decomposition in span space, in addition to allowing for parallel implementation on a distributed memory architecture [SHLJ96]. With certain assumptions on the distributions of points in the span space, the worst-case query time improves to $O(k + n_e \log L).

- **Segment tree, interval tree** – Several authors have recently developed improved worst-case performance bounds with the use of the *interval tree* and *segment tree* data structures. Both structures provide a search complexity of $O(k + \log n_u)$, where $n_u$ is the number of unique extreme values of the segments which define the tree and $k$ is the number of reported segments intersected. In Bajaj et al. a segment tree is constructed for a reduced set of seed cells which are extracted in a preprocessing stage [BPS96]. van Kreveld also developed seed sets for the specialized case of a triangular mesh in two dimensions [van96]. The *interval tree* used by van Kreveld provides the same search complexity with lower worst-case storage overhead, which we will examine in Section 4.3. Cignoni et al. use an interval tree constructed for the entire set of cells in a
tetrahedral complex [CMPS96]. More recently Cignoni et al. extend their approach to efficiently handle large regular grids by building an interval tree for a specialized subset of the cells [CMPS97].

4.1.3 Cell Traversal

The order in which cells are visited can impact the efficiency of contouring algorithms in several ways. In the algorithms described above, cells may be traversed in marching order, through contour propagation (breadth first in a connected component), or in random order. One issue is the efficiency of avoiding recomputation (recomputing intersection along shared edges of cells). Through marching order and contour propagation, information can be saved more efficiently than in a random order visitation which is caused by some search techniques.

Contour propagation [AFH80, HB94, IK95, BPS96] is a surface tracking method which is based on continuity of the scalar field, and hence of the isocontours derived from the field. Given a single seed cell on a connected component of a contour, the entire component is traced by breadth-first traversal through the face-adjacencies. The traversal is terminated when a cell which has already been processed is met again, which is usually determined by a set of mark bits, which indicate for each cell whether processing has taken place. The procedure is illustrated in Figure 4.8. In a contour propagation framework, as in a marching order traversal, optimization can be performed based on the fact that with each step, information from adjacent cells is available which can be used to avoid recomputation. In addition, the extracted contours are more easily transformed into representations such as triangle strips for efficient storage and rendering.

Several of the cell search techniques presented above depend upon a subsequent cell traversal algorithm such as contour propagation. The use of a subsequent cell traversal algorithm allows a reduction in the size of the search structure, because a cell which will be processed by traversal need not be entered into the primary search structure. The traversal stage can be considered a secondary search phase.
Itoh and Koyamada compute extrema graphs of the scalar field which, combined with a search of boundary cells, guarantees that each contour component will intersect at least one seed cell [IK95]. More recently the same authors describe a volume thinning approach to computing a seed set, which reportedly results in smaller seed sets [IYK96].

In [vvOB+97], the theory of optimal seed sets is discussed, which suggests that optimal (minimal) seed sets can be constructed in time which is polynomial in the number of cells, though the cost for minimal seed sets remains prohibitive for most cases.

Cignoni et al. introduce a limited propagation scheme for regular grids based on a "checkerboard" seed set, as illustrated in Figure 4.9. By selecting a regular pattern of cells, it is guaranteed that all contours will intersect a black or grey cell. Modified contour propagation rules are applied to reach white cells from the selected black or grey cells. Determining the seed set requires very little computation, thus preprocessing is essentially limited to building the range search structure, in this case an interval tree. In Section 4.2.6 we will contrast this approach with the seed selection algorithms presented here.
4.1.4 Summary of Prior Work

A key to efficient computation is in exploiting coherence. The isocontouring approaches described above can be loosely classified and analyzed based on the coherence which is exploited.

Spatial Coherence – We assume a minimum of $C^0$ continuity in our scalar field. Continuity along shared cell faces is exploited by many contouring approaches described above. The octree decomposition exploits spatial coherence in a hierarchical manner. As should be expected, the analysis in [LSJ96] reveals that the complexity gain breaks down when the spatial frequency is high, forcing large portions of the octree to be traversed.

Range-Space Coherence – Searches in range-space have demonstrated improved worst-case query complexity with performance which is independent of spatial frequency. Such advances, however, come at the cost of decreased ability to exploit spatial coherence. Assuming a continuous scalar field over a cell representation, cells which are spatially adjacent also overlap in the value space for
the range of the shared face. However, the construction of value-space search structures such as the interval tree and segment tree are completely independent of assumptions such as scalar field continuity. While this may be an advantage in the case that discontinuous fields or disjoint groups of cells are considered, for our purposes it usually means that spatial coherence is under-utilized.

We see above that spatial and value-space searches exploit coherence in one sense by sacrificing coherence in another. Our approach is best understood as a hybrid of spatial and value-space approaches, with the goal and result of exploiting both value-space and spatial coherence.

Our approach is based on a fragmentation of the search for intersected cells into range-space and geometric phases, taking advantage of coherence in both. Range-space searches exhibit improved worst-case complexity bounds due primarily to the fact that intersection of a contour with a cell is determined by range-space properties, as opposed to geometric properties. The output, however, is geometric in nature. By adopting contour propagation to compute each connected component, we take advantage of spatial coherence during cell traversal. Contour propagation also has the advantage of requiring only one seed cell for each connected component from which to begin tracing the contour. In our approach, preprocessing determines a subset $S$ of the cells which are maintained as candidate seed cells. For an arbitrary input isovalue, it is guaranteed that every connected component of the isocontour will intersect at least one cell in $S$. A second preprocessing step constructs a range query structure for the cells in $S$. In the contour extraction phase, the contour propagation algorithm sweeps out the contour from each selected cell in the seed set. Thus, the search for intersected cells takes advantage of spatial coherence in the use of contour propagation, and range-space coherence through the construction of a range-space search structure for seed cells.

In Section 4.2 we present three approaches to the construction of seed sets. In Section 4.3 we describe three alternative data structures for performing fast queries for intersected seed cells. Together the presented algorithms and data structures
provide an array of possible combinations which vary in usefulness based on the relative importance of computational, space, and query complexity.

4.2 Seed Set Construction

We introduce three alternatives for the construction of seed sets. Our primary concern is efficient approximation algorithms for computing "good" seed sets. The problem of optimal seed sets is considered and shown to have polynomial time complexity in [vvOB+97], however the complexity may be considered excessive for many applications. As with many approximation algorithms, we find that the performance in terms of the size of the seed set can be balanced with the competing desire for low time/space complexity, resulting in three approaches which are useful in a variety of settings and applications.

In sections 4.2.1 and 4.2.2 we provide some preliminary definitions which are derived from contour propagation and give a more formal definition of seed sets, providing a foundation for seed set generation.

4.2.1 Cell Connectivity

We begin by extending the definitions of cell connectivity based on adjacency to encompass connectivity with respect to a given scalar value of a function defined over the domain. In this way we can identify the cells which can be reached in the cell traversal stage (i.e. by contour propagation) from those which must be part of the seed set.

Based on propagation of contours through cell adjacencies (as presented in Section 4.1.3), the connectivity is simply described by a labeled adjacency graph of the mesh cells. The use of a different propagation scheme would require the construction of a connectivity graph different from the adjacency graph. In general, to define the connectivity graph we assume:

1. The function $F(x)$ defining the scalar field of our $d$-dimensional mesh is continuous.
2. All the cells of the mesh are connected.

3. A function \( R(c) \) is given which, for any given cell \( c \) of the mesh, returns the range of values assumed by \( \mathcal{F} \) over the domain of \( c \). Note that, since \( \mathcal{F} \) is continuous, the range returned is always an interval \([\text{min}_c, \text{max}_c]\).

4. For each pair of adjacent cells \((c_i, c_j)\), let

\[
f_{ij} = \{x | x \in c_i, x \in c_j\}
\]

and define the connecting interval:

\[
R(f_{ij}) = \left[ \min_{x \in c_i, x \in c_j} \mathcal{F}(x), \max_{x \in c_i, x \in c_j} \mathcal{F}(x) \right] \subseteq R(c_i) \cap R(c_j)
\]

such that if the cell \( c_i(c_j) \) is processed for a value \( w \in R(f_{ij}) \), then the cell \( c_j(c_i) \) will be also processed for the same value \( w \). This is essentially the information we get from the contour propagation scheme.

Based on the above assumptions, we construct a labeled graph \( G \). Note that this graph need not be constructed explicitly in practice. For each cell \( c \) in the mesh, we have a node \( n_c \) in \( G \) which is labeled \( T(n_c) = R(c) \). For each pair of adjacent cells \((c_i, c_j)\), there is an arc \( f_{ij} \) in \( G \) connecting \( n_{c_i} \) to \( n_{c_j} \) which is labeled \( T(f_{ij}) = R(f_{ij}) \). The arc \( f_{ij} \) corresponds to the face which is shared by cells \( c_i \) and \( c_j \).

Connectivity relations between nodes in the graph \( G \) are transferred to relations between the corresponding cells of the underlying mesh. Based on propagation of contours through cell adjacencies we have the following definition:

**Definition 2** Consider a scalar value \( w \) and a connected sequence of nodes

\[
P = \{n_{i_1}, \ldots, n_{i_k}\}
\]

\( P \) is called a \( w \)-path if

\[
w \in R(f_{i_j, i_{j+1}}), \forall j \in [1, \ldots, k-1]
\]
A \textit{w-path} represents a cell traversal sequence based on application of a contour propagation algorithm for a given iso-value \( w \). We further define:

\textbf{Definition 3} Consider a scalar value \( w \) and two nodes \( n_{ci}, n_{cj} \) of \( G \). \( n_{ci} \) and \( n_{cj} \) are said to be \textit{\( w \)-connected} if there exists a \( w \)-path connecting them.

Note that Definition 3 is a transitive relation, and we can define:

\textbf{Definition 4} A maximal set of nodes \( \{n_{i1}, \ldots, n_{ik}\} \) which are \( w \)-connected is called a \textit{\( w \)-connected component}.

A \( w \)-connected component defines precisely the set of cells which are processed by contour propagation from a single cell in the set. Note that a \( w \)-connected component differs slightly from a \textit{connected component} of the isocontour, in that two separate connected components may intersect a common cell, forming a single \( w \)-connected component, as illustrated in Figure 4.10.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure4.10.png}
\caption{Illustration of \( w \)-connected components. On the left are four contour components for a particular iso-value \( w \). On the right a portion of the graph \( G \) is displayed, corresponding to the three \( w \)-connected components. Displayed in green are \( w \)-path between two of the nodes.}
\end{figure}
We can extend the concept of \( w \)-connectivity between pairs of cells to the connectivity of a set of cells with respect to a range of values.

**Definition 5** Consider a subset \( S \) of the nodes of \( G \) and a node \( c \in G \). The node \( c \) is connected to \( S \) if, for any \( w \in R(c) \), there exists a node \( c' \in S \) that is \( w \)-connected to \( c \).

### 4.2.2 Seed Sets

We now characterize some particular subsets, called *seed sets*, of the cells of a mesh in terms of the connectivity properties defined in the preceding subsection. Seed sets are important because all connected components of any isocontour of the entire original mesh can be traced by contour propagating from the cells of any seed set.

**Definition 6** A subset \( S \) of the nodes of \( G \) is a seed set of \( G \) if all the nodes of \( G \) are connected to \( S \).

In order to quickly determine all cells whose range contains a particular scalar value \( w \), we can proceed as follows:

1. search for all the cells \( c \in S \) such that \( w \in T(c) \);

2. starting from the cells reported in step 1 and using the \( w \)-connectivity relation on the graph \( G \) (that is the contour propagation scheme), we find all the cells of the mesh whose range contains \( w \).

To reduce the search time and storage requirements it is desirable to reduce the cardinality of the seed set \( S \) as much as possible. Toward this end we will apply the following property:

**Property 1** If \( S \) is a seed set and \( c \in S \) is a cell connected to \( S - \{c\} \), then \( S - \{c\} \) is a seed set.
Proof: By hypothesis we have that $c$ is connected to $S - \{c\}$. Also, from Definition 3, we have that any cell which is $w$-connected to $c$ is also $w$-connected to some cell in $S - \{c\}$. Hence $S - \{c\}$ is a seed set.

Property 1 provides us with a method to reduce the size of a seed set. If we wish to find a small seed set, we can start with the entire set of the cells – that is the largest seed set – and keep removing cells until we achieve a minimal seed set. Note that a minimal seed set is not the seed set with the minimum number of cells but a seed set from which we cannot remove any cell to obtain a new seed set.

The repeated application of Property 1 requires the knowledge at each step of the connectivity relations within the current seed set. Thus, we may start from the initial graph $G$. At each step, we remove a selected cell $c$ along with all its incident arcs and add some new arcs between pairs of cells that were connected to $c$ to take into account the connectivity relations induced by $c$ on $G - \{c\}$. In particular, if two cells $c_i$ and $c_j$ are both connected to $c_k$ with arcs $f_{ik}$ and $f_{jk}$, then the removal of $c_k$ requires also the removal of $f_{ik}$ and $f_{jk}$ and potentially the insertion of a new arc $f_{ij}$ connecting $c_i$ to $c_j$. This new arc $f_{ij}$ needs to be inserted if $R(f_i) \cap R(f_j) \neq \emptyset$ (a case in which the transitivity of Definition 3 applies). If this condition is true, then the new arc is added with label $R(f_{ij}) = R(f_i) \cap R(f_j)$. If we proceed in this way, it becomes simple to determine if Property 1 can be applied. We can remove a cell $c_k$ of the current seed set if:

$$\bigcup_{i=1}^{k} R(f_{ik}) = R(c_k)$$

where $f_1, \ldots, f_j$ are all the arcs incident to the cell $c_k$ in the reduced graph of the current seed set.

Given this general reduction scheme, we still have freedom to select the cells to be removed in any order. We can use a greedy approach, removing first the cells that we consider less likely to belong to a minimal seed set – for example the cells that have narrower range. In this way we can assume that the minimal seed set we achieve is not much larger than the seed set with the minimum number of cells. On the other
hand, we can use this freedom to make the algorithm as simple as possible (a very important property in actual implementations).

In the following subsections we present three seed selection algorithms. In Section 4.2.3 we present a greedy approach for constructing near-optimal seed sets for irregular or regular grids. In Section 4.2.4 we present a simple, fast seed selection approach by a sweep traversal for both irregular and regular grids. In a regular grid the sweep process can be simply implemented as a traversal of the grid by rows using a regular marching scheme. In Section 4.2.5 we examine a modified case for grids of regular topology which achieves smaller seeds sets with slightly larger temporary storage complexity.

In our seed set generation algorithms, we begin by considering the universal seed set, consisting of all cells in the mesh. We associate with each seed cell a computed range \( T(c) \subseteq R(c) \), which represents the range of values for which the given cell is a seed cell. Initially, we have \( T(c) = R(c) \), the entire range of the cell, hence \( S \) is trivially a seed seed set. Algorithms for seed set generation can be viewed as seed set reduction techniques, which iteratively apply Property 1 to the current seed set to reduce the size of \( S \).

4.2.3 Greedy Climbing

For computation of a nearly optimal seed sets we develop a greedy technique which progressively covers the domain with seed cells by explicitly computing the coverage of each seed cell introduced. This \textit{climbing} algorithm can be applied to any complex of cells provided that the appropriate function \( R \) is given which computes the range of a cell or face. The main advantage of the seed cell selection algorithm presented in [vvOB+97] is that it guarantees the computation of a seed set nearly optimal in size (at most twice the size of the optimal seed set). What makes it difficult to achieve such a goal is the problem of selecting the "best" seed cells at the saddle points of the scalar filed. The cost of solving such "difficult" situations is:

- the necessity to build explicitly the contour tree [vvOB+97] of the scalar field;
- the use of complex data structures as the union-find-split (generalization of the standard union-find);
- the use of involved routines like the tandem-search.

As a consequence it becomes difficult to use such an approach in practice even if the results it produces are highly desirable. Moreover we observe that in many scalar fields which arise in practice the number of saddle points is relatively small and, more important, a greedy selection (not based on contour tree and tandem search) gives in practice very good results. As a consequence we have developed a “practical” algorithm which computes nearly optimal seed sets (even if not guaranteed in theory) for general unstructured meshes. The generalization to the unstructured grids is based on the application of:

- greedy seed cell selection
- contour driven advancing front

We begin by considering the universal seed set $S$ with $T(c) = R(c)$ for all cells $c$. In our algorithm processing begins by selecting an arbitrary cell $c$. For the example in Figure 4.11 the first seed cell is the lower left cell of the grid (in practice the quality of the result is not going to be greatly affected by the selection of the first cell). We can trace as in Figure 4.11(a) the isocontour $C(i)$ which bounds the range of the cell $c$. The cells intersected or included within the isocontour $C(i)$ are processed so that their range is reduced only to the portion outside $i$, as in Figure 4.11(b). At each step the two following operations are performed:

(i) Among the cells across the current front $i$ we select as the next cell to process the one with larger residual range $(T(c) - R(c))$.

(ii) The front $i$ is enlarged up to the isocontour that includes the newly selected seed cell.
Figure 4.11: Greedy climbing approach to seed cell selection. Grey cells represent the selected seed cells. Yellow cells have been processed and removed from consideration, while red cells represent the current \textit{front} of cells from which the next seed cell will be chosen.
The geometric interpretation of the algorithm is based on the idea of contour tree [vvOB+97] (note that we use the contour tree as algorithm analysis tool, but we do not need to build it). The greedy choice is equivalent to selecting each time the seed cell that allows to climb (descend) as fast as possible the contour tree (see Figure 4.11(c)(d)). To achieve this without computing the contour tree we need to resort to the contour driven advancing front. The contour front is realized using a priority queue in which we store the cells that are intersected by (but not included in) the current front \( i \). As the front \( i \) advances new cells are inserted in the priority queue while other are removed (those included in the new advanced front). To access and efficiently update the cells within the priority queue we compound a hashing scheme that allows access in average expected \( O(1) \) time to any cell in the queue. The base algorithm consists of:

ContourClimbing(\( \text{mesh} \))

select an initial cell \( c \)
insert \( c \) in the priority queue \( p \) with priority \( R(c) \)
while \( p \) is not empty
  do
    extract the cell \( c \) with highest priority and associated range \( T(c) \)
    \{ cell \( c \) with range \( T(c) \) remains in the seed set \}
    PropagateRegion(\( \text{mesh}, c, T(c), p \))
done

The function PropagateRegion() is an extension of contour propagation to the case of simultaneous propagation of an interval of values. Similar to contour propagation, interval propagation uses a queue of cells and propagates from cell to adjacent cell. Associated with each cell \( c_i \) in the queue is an interval \( P(c_i) \), which represents the interval range which has been propagated to cell \( c_i \). When intervals are passed from a cell \( c_i \) to a neighboring cell \( c_j \), only values in the shared range \( R(f_{ij}) \) can be propagated, as the purpose is to mimic contour propagation for a range of values. The overall interval propagation algorithm is outlined as follows:
PropagateRegion(mesh, c, T(c), p)

insert c in the queue q with associated range T(c)
while q is not empty

do

extract the cell ci and associated range P(ci) from q
if cell ci is in p
then

T(ci) = T(ci) − P(ci)
if T(ci) = ∅
then

remove cell ci from p
else

set priority of ci to the new span of T(ci)
endif
endif

for each cell cj adjacent to ci

do

if cj is in the queue q
then

P(cj) = P(cj) + P(ci) ∩ R(fij)
else

add cj to q with associated range P(cj) = P(ci) ∩ R(fij)
done
done

Figure 4.12 shows a sample seed set computed with the algorithm described above.

4.2.4 Sweep Filtering

Computation of seed sets need only be performed one time for any dataset, and the results can be stored off-line. In many cases a considerable amount of processing can be devoted to generating very small seed sets, and the results can be used over and over again. However, in many practical situations, the maintenance of a priority queue as described in the contour climbing algorithm may be prohibitively expensive. Such situations include:
Figure 4.12: Results of seed selection by contour climbing (76/7938 cells)

1. **time critical** – The time complexity of the contour climbing algorithm may be prohibitive if results are needed very quickly.

2. **dynamic data** – If the data are being collected and analyzed in real-time, contour climbing may be infeasible.

3. **out-of-core** – If the data are too large to be stored in main memory, a memory access pattern which exhibits greater coherence than the random-access pattern of contour climbing would be desirable.

We present a simple seed selection algorithm which is motivated by these practical considerations. The seed selection is conceptually easiest to understand as a sweep of the cells in a particular direction. The algorithm has the property that selected seeds fall on the extrema of the contours in the given sweep direction. Detection of contour extrema is based on a simple comparison of the gradient within each cell and its immediate neighbors. With such a seed set, contouring may be performed coherently and efficiently by executing a contouring sweep, with only a slice of data required to be resident in memory at any given time, resulting in efficient computation for visualization of large out-of-core datasets.
To understand the properties that such a seed set must have, we first consider the one pass contour tracing algorithm. From its analysis we immediately observe the properties that an appropriate seed set must have.

4.2.4.1 One-pass Contour Tracing

Conceptually the one-pass contour tracing is based on a sweep of the cells along a particular direction. As illustrated in Figure 4.13, a sweep line $l$ (sweep plane in 3D) is moved from left to right along the $x$ direction. The isocontour $C(w)$ of height $w$ is built progressively as it is crossed by the sweep line $l$. Each time $l$ is tangent to the isocontour $C(w)$ three situations may arise:

- $C(w)$ attains a local minimum along the direction $\vec{l}_1$ (orthogonal to $l$) as in Figure 4.14(a)-(b). A new portion of $C(w)$ starts to be traced.
- $C(w)$ attains a local maximum along the direction $\vec{l}_1$ as in Figure 4.14(c)-(d). Two separate portions of $C(w)$ may join or a loop may be closed.
- An inflection point is met that does not need any special processing, as in Figure 4.14(e).

To perform the contour sweep operation two conditions suffice:

- The dataset is stored with the cells sorted by maximum $\vec{l}_1$ so that by loading them into memory (from the end to the beginning) we automatically perform a sweep. This means that in the contour tracing stage and in the seed selection stage the sweep algorithm can be performed in linear time.

- Any local maximum along the $\vec{l}_1$ direction of any isocontour can be immediately detected (that is, a proper seed set is precomputed). This allows us to avoid loading in memory sections of the mesh where there is no contour component.

4.2.4.2 Sweeping Seed Selection

The seed selection stage is performed with a forward sweep as illustrated in Figure 4.15. Conceptually, the sweep line $l$ is moved from left to right to determine the
order in which cells are processed. Note that this ordering is not required by the selection algorithm, and so cells which are stored in main memory can be processed in any order, or even in parallel. Since the cells are already sorted in the mesh, the sweep is achieved by simply loading them into memory in the order they are stored. When a cell $c$ is met which contains a local maximum of an isocontour along the sweep direction $l$, the cell $c$ is added to the seed set.

From the geometrical point of view the determination of a local maximum within a cell of an unstructured mesh is based on the normal $\vec{m}_c$ of the contour within the cell $c$. Consider two adjacent cells $c$ and $d$, as shown in Figure 4.16. The normal $\vec{m}_c$ ($\vec{m}_d$) of the contours within $c$ ($d$) is the projection onto the mesh space of the normal of the scalar field in $c$ ($d$). As in Figure 4.16 we can determine a fixed direction $l$ along $l$ and perform the following test:

- if both $\vec{m}_c$ and $\vec{m}_d$ have positive (negative) scalar product with $l$ then neither a maximum nor a minimum is met;

- Assume that $\vec{m}_c \cdot l \geq 0$ and $\vec{m}_d \cdot l \leq 0$ (the opposite case is symmetric). If the cross product $\vec{m}_c \times \vec{m}_d$ has positive $z$ direction then a minimum is met.
Figure 4.14: Tangent conditions of contour with sweep direction

Note that in order to test whether a cell is a seed it is only necessary to examine the cell and its first neighboring cells. For data stored in primary memory, this is not an issue. For the out-of-core extension to this approach, we must ensure that when a cell is tested, all neighboring cells are available in primary memory as well. There are essentially two approaches to solving this. In the first approach, we can determine the maximum difference between the indices of adjacent cells, in order to compute the amount of primary memory necessary for the out-of-core processing. A more attractive solution is to fix the amount of memory which is available for the out-of-core processing, and adapt the seed selection to this limit. In this case, if an adjacent cell is not available in primary memory during the seed selection, the shared face between the cells is treated as a border of the mesh. Applying this approach, we may select more seed cells, with the advantage that the approach can adapt to situations in which only a small fraction of the mesh can be stored in primary memory at any given time.

Sweep filtering requires \( O(n_c) \) time for considering each cell, and no additional storage beyond that of the extracted seed set (and the portion of the mesh kept in memory).

Note that in addition to facilitating out-of-core computation, in general the sweep filtering approach provides an extremely efficient method for computing a small seed
Moreover, due to the local criteria for seed selection, cells may be considered in any order, allowing for parallel implementation with little or no communication overhead during the preprocessing.

4.2.4.3 Special Cases

While the algorithm described above is general and independent of grid topology, special considerations for particular types of data and grids may be worthwhile. The criteria given above for detecting a minima along the sweep direction have a special condition in the case that \( \vec{m}_c \cdot \vec{I} = 0 \) and \( \vec{m}_d \cdot \vec{I} = 0 \), indicating that the contour is perpendicular to the sweep direction. While this special case may not occur frequently in a general data setting, the frequency of occurrence is much greater in particular settings, such as integer-valued data defined over a regular grid, which is often the case for digital terrain data and medical image data.

With such data, it is not uncommon for degenerate situations to occur, as illustrated in Figure 4.18. In this case, a minimum along the sweep direction is an entire line, which may result in a column of seed cells for a particular isovalue, though it is
clear that the seed cells are $w$-connected. In higher dimensions, the problem remains that a large number of cells along the $d - 1$ dimensional hyperplane may be selected. This degenerate situation is easily and efficiently addressed by making slight modifications to the selection criteria. By modifying the minima detection criteria that $\vec{m}_c \cdot \vec{I} \geq 0$ and $\vec{m}_d \cdot \vec{I} < 0$, (one perpendicular condition is removed), only one seed along each flat minima region will be chosen, as illustrated in Figure 4.19. For regular grids of higher dimension, similar consideration of special cases can be made.

A sample seed set computed by sweep filtering is shown in Figure 4.20.

4.2.5 Responsibility Propagation

The special case for regular grids may be extended to provide smaller seed sets with a constant increase in computational complexity and only slightly greater storage. This technique can be viewed as a simplification of the connectivity graph technique described in the Section 4.2.1 for determining a seed set $S$. The algorithm does not require that we store the entire graph, but instead we maintain a subset of the information from the graph which can be locally propagated from cell to cell using

![Figure 4.16: Conditions for determining a local maxima along the $\vec{I}_L$ direction. In linear cells (a-b) the maxima lie along cell edges. With regular cells (c) the maxima remain along edges, though the the conditions may change along the length of the edge. In non-linear cells as simple as the non-axis-aligned bi-linear cell (d), maxima may occur in cell interiors.](image-url)
simple rules when marching in a regular order, with temporary storage complexity of $O(r^{(d-1)/d})$. We again begin with all cells $c$ in the set $S$. We associate with each seed cell a computed range $T(c)$, which represents the range of values for which the given cell is a seed cell. Initially, we have $T(c) = R(c)$, the entire range of the cell, hence $S$ is a trivially a seed seed set. We present an incremental seed elimination technique to reduce the seed set $S$. The reduction and removal of seed cells is based on propagation of responsibility ranges of isovalues. The information propagated from cell to cell in marching order is a range $T$ for each dimension of the regular grid. An incoming range $T$ represents the range of values $w$ for which responsibility has been propagated to the current cell from the neighboring cells. The incoming range is always a subset of the range of the shared face in the direction of propagation. The complement of the incoming range in the direction which varies fastest consists of values $w$ for which the current cell is $w$-connected to either (i) a processed cell which remains in the seed set or (ii) an unprocessed cell to which responsibility for the value $w$ has been propagated. An outgoing range represents the responsibility range which is propagated from the current cell to a neighboring cell. Illustrated for the 2D case in Figure 4.21, the marching order is $Y$ varying fastest, $X$ varying slowest.
We describe the processing of a cell \( c \) at index \((i,j)\) in a topologically regular grid of dimension \((n_x, n_y)\). Boundary conditions are handled directly through the following notation, defined for simplicity:

1. \( T(f_u) \) represents the range of the incoming face in the \( U \) direction, where \( U \) is an arbitrary dimension.

2. \( T(u) \) represents the incoming range propagated in the \( U \) direction. In the case of the boundary condition \( u = 0 \), we take \( T(u) = T(f_u) \).

3. \( \overline{T(u)} \) represents the complement of \( T(u) \) with respect to the range \( T(f_u) \) of the shared face, or \( T(f_u) - T(u) \). Note that the propagated range \( T(u) \subseteq T(f_u) \).

4. \( T(f_{u'}) \) represents the range of the outgoing shared face in the \( U \) direction. In the boundary case when there is no adjacent cell in the outgoing \( U \) direction \((u = n_u - 2)\), we assign \( T(f_{u'}) = \emptyset \), indicating that no propagation may occur in the given direction.
Figure 4.19: The cells on the current sweep plane are processed in regular order. A bit flag is turned on when a local maximum exists on the top edge of a cell.

5. $T(w')$ represents the range propagated from the current cell to the outgoing adjacent cell in the $U$ direction.

We first compute the combined incoming range $T(I)$, and corresponding complement range $\overline{T(I)}$:

$$T(I) = (T(y) \cup T(x)) - T(y)$$  \hspace{1cm} [4.1]$$

$$\overline{T(I)} = (T(f_x) \cup T(f_y)) - T(I)$$ \hspace{1cm} [4.2]

$T(I)$ represents the subset of incoming isovalues which cell $c$ must either account for in the seed set $S$ or defer responsibility for by propagation through $T(x')$ and $T(y')$. The removal of $T(y)$ in Equation 4.1 above is justified based on the algorithm for range propagation presented below. For all $w \in \overline{T(I)}$, there either exists a processed cell in $S$ which is $w$-connected to $c$ or the value $w$ has already been further propagated, and hence $w \in \overline{T(I)}$ need not be considered in processing $c$. This leads to the definition of $T(R)$, representing the entire range of values which make up the responsibility range of cell $c$.

$$T(R) = R(c) - \overline{T(I)}$$ \hspace{1cm} [4.3]
Figure 4.20: Results of seed selection by directional sweep (296/7938 cells)

For $w \in T(R)$, we must take care that $c$ remains $w$-connected to $S$ in order to maintain the property that $S$ is a seed set. We also compute $T(P)$, which represents the combined range of isovalues which may be further propagated through outgoing faces:

$$T(P) = T(f_{x'}) \cup T(f_{y'})$$  \hspace{1cm} [4.4]

We arrive at the following greedy algorithm for deferring seed cell selection through propagation of responsibility. Through the processing of a cell $c$, we maintain the invariant that $S$ is a seed set.

```
if (T(R) \subseteq T(P)) then
    \{ Cell c can be removed from S \}
    S = S - c
    \{ Propagate responsibility ranges \}
    T(x') = T(f_{x'}) \cap T(R)
    T(y') = T(f_{y'}) \cap (T(R) - T(x'))
else
```

Figure 4.21: Illustration of responsibility propagation. Each cell processes input responsibilities and produces output responsibilities

\[
\{ \text{Cell } c \text{ must remain in the seed set} \} \\
T(c) = T(R) \\
T(x') = \emptyset \\
T(y') = \emptyset \\
\text{end}
\]

Proof: (S remains a seed set after processing of cell c)

Case 1 ($T(R) \subseteq T(P)$) - Recall that cell c is $w$-connected to a processed seed cell for $w \in \overline{T(I)}$. Through propagated responsibility ranges, we have that $c$ is $w$-connected to the remaining (unprocessed) seed set for $w \in T(x') \cup T(y') = [T(f_{x'}) \cap T(R)] \cup [T(f_{y'}) \cap (T(R) - T(x'))] = (T(f_{x'}) \cup T(f_{y'})) \cap T(R) = T(P) \cap T(R) = T(R) = R(c) - \overline{T(I)}$. Thus, $c$ is connected to $S - \{c\}$, and by Property 1, $S - \{c\}$ is also a seed set, maintaining the invariant property.

Case 2 (Cell c remains in the seed set) - Cell c is trivially $w$-connected to $S$ for $w \in T(c) = T(R) = R(c) - \overline{T(I)}$. From the input conditions, we have that $c$ is $w$-connected to a processed cell which remains in $S$ for $w \in \overline{T(I)}$. Thus, $c$ is $w$-connected to $S$ for $w \in R(c)$, maintaining the invariant property that $S$ is a seed set.
In the first case, the propagated range $T(P)$ includes the responsibility range $T(R)$ in its entirety, and cell $c$ is removed from the seed set $S$. The responsibility range is propagated through the outgoing faces by the computation of $T(x')$ and $T(y')$. Note that the propagated ranges are disjoint and that the preference is to propagate the range in the $X$ direction. It is this preference which allows us to remove $T(y)$ in equation (4.1). For all $w \in T(y)$, the associated $w$-connected component is either accounted for by a processed cell in the seed set $S$, or responsibility has been propagated to an unprocessed cell, hence $w$ need not be considered for the current cell. The same cannot be said for $T(x)$, because the precedence of propagation indicates that responsibility for values $w \in T(x)$ may, through some path of responsibility propagation, ultimately be propagated through $T(y)$. Consider the case of Figure 4.21, and suppose that the value $A$ is a local minimum. Values $w \in T(x)$ overlap with the range $T(y)$, providing incoming information which appears to conflict. In fact we cannot make use of the range $T(u)$, where $u$ is other than the direction which varies fastest in the marching order.

The second case above occurs when cell $c$ cannot propagate the entire incoming range. Cell $c$ remains in the set $S$, though $T(c)$ is reduced to exclude the complement ranges which have been propagated elsewhere. In this case the empty set is propagated to outgoing edges, indicating that all values on shared faces are accounted for in the seed set $S$.

As described above, the range propagation method for selecting seed cells requires $O(n^{(d-1)/d})$ storage to maintain the propagated ranges for a sweeping line or plane in 2D or 3D. Note that our use of range subtraction may result in ranges with two disconnected components. In practice, disconnected ranges may either be maintained or closed by taking the smallest range which contains the entire disconnected range. Maintaining the disconnected range effectively requires that multiple seeds be processed into the search structure, increasing the number of seeds, while merging disconnected ranges simply means that two or more cells which are $w$-connected may be selected for inclusion in the seed set $S$. Of course, this greedy technique
does not guarantee the selection of a single cell for each connected component in the case that disconnected ranges are maintained. In our implementation, we maintain disconnected ranges through the seed cell selection, closing each range which is ultimately selected to remain in the seed set $S$. In practice the number of seed cells with disconnected ranges does not exceed 10% of the seed cells, and the number of seed cells does not exceed 10% of the data, as presented in the results.

Results for a our 2D sample function are illustrated in Figure 4.22. The relatively smooth function is sampled on a grid of size 64 x 64. Figure 4.22 shows the 206 seed cells chosen by the range propagation seed selection method.

![Figure 4.22: Results of seed selection by range propagation (206/3969 cells)](image)

4.2.6 Seed Set Results

Table 4.1 presents the comparative sizes of seed sets for the three seed selection algorithms applied to a variety of input 2D and 3D meshes.

4.3 Range Queries

The fundamental isocontouring query concerns the enumeration of all cells \( c \) such that \( w \in R(c) \) for the input isovalue \( w \). In this section, we compare the use of three
Table 4.1: Seed set sizes for the three presented algorithms, compared with the checkerboard approach and the total number of cells.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Climbing</th>
<th>Propagation</th>
<th>Sweeping</th>
<th>Checker</th>
<th>Total Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eagle Pass Terrain</td>
<td>1872</td>
<td>7151</td>
<td>29356</td>
<td>720000</td>
<td>1440000</td>
</tr>
<tr>
<td>Sample Function</td>
<td>59</td>
<td>177</td>
<td>238</td>
<td>1985</td>
<td>3969</td>
</tr>
<tr>
<td>Hipip</td>
<td>529</td>
<td>2212</td>
<td>6397</td>
<td>62559</td>
<td>250047</td>
</tr>
<tr>
<td>Climate Data</td>
<td>177</td>
<td>602</td>
<td>1916</td>
<td>4760</td>
<td>19040</td>
</tr>
<tr>
<td>SOD</td>
<td>2308</td>
<td>9944</td>
<td>18608</td>
<td>264960</td>
<td>1059840</td>
</tr>
</tbody>
</table>

Data structures supporting this range query operation in terms of the storage complexity and the time complexity for both creation of the structure and for performing individual queries. While the characteristics of the search structures being studied are easily understood and compared in theory, characteristics of our data and seed sets lead us to examine the practical application considerations. A primary consideration is that of the data type. Note that for integer-valued data, the search structures listed below all simplify to the same complexity, both in space and query time. A second consideration is the size of the seed sets. While in the worst case \( n_s = O(n_c) \) (note that we always have that \( n_s \leq n_c \) as \( S \) is a subset of all cells), we have demonstrated in practice that \( n_s \) is often smaller than \( n_c \) by one or more orders of magnitude. This leads us to consider search structures of greater space complexity, which may lead to improved query complexity or practical demonstrated performance.

In the following sections we review the interval tree, segment tree, and bucket search structures as applied to the contour query problem described. Example search structures are illustrated for the input set of intervals shown in Figure 4.23. For each search structure, we describe:

- Data structures
  Basic C/C++ data structures for representing the search structure.
• Construction
The algorithm for creating a search structure from a set of input intervals. The search structures are not created iteratively, and so balancing of the trees is not an issue.

• Querying
The algorithm for processing an input query for a given isovalue \( w \) is considered.

Our analysis and data structures are based on the general definitions of the data structures, without respect for the data from which they are derived. In particular, the search structures are designed such that queries can be resolved without referencing the original data.

4.3.1 Interval Tree

An interval tree is made up of a binary tree over the set of interval \( \text{min}/\text{max} \) values [McC85]. Each internal node holds a split value \( s \), with which intervals are compared during insertion into the tree. If the interval is entirely less than the split value it is inserted into the left subtree, while intervals greater than the split value are recursively inserted into the right subtree.

In the case that the interval spans the split value (\( \text{min} < s < \text{max} \)), the recursion terminates and the given interval is stored at the current node. Each node maintains
two list of spanning cells. The first list is stored in increasing order by the \textit{min}, the second in decreasing order by the \textit{max} value. Because the intervals are not split in the recursive insertion, each interval is stored only twice, and the storage complexity is $O(n)$. 

4.3.1.1 Interval Tree Data Structure

The data structure for the interval tree is relatively simple. The tree structure is implicit, with no need for pointers (in the case that the intervals are static).

\begin{verbatim}
struct IT_IntervalList {
    int *interval_id;
};

struct IT_Node {
    float split_value;
    int n_intervals;
    IT_IntervalList min_list;
    IT_IntervalList max_list;
};

struct IT_Interval {
    float min;
    float max;
    int cell_id;
};
\end{verbatim}
struct IntervalTree {
    int n_intervals;
    IT_Interval *intervals;

    int n_uniq_val;
    IT_Node *nodes;
};

The total storage requirements can be broken down into per interval and per node costs. For each interval (seed cell), there are two float values for the extrema of the interval, one integer to store the cell identifier, and two indices in the sorted lists, for a total of $5n_s$ words of storage. Each node in the tree contains a split value, an integer number of cells stored at the node, and two pointers to the sorted lists of intervals, for a total of $4n_u$ words of storage. The total storage requirement is $5n_s + 4n_u$.

4.3.1.2 Interval Tree Construction

An interval tree is constructed in three steps. First, a sorted list of unique extreme values of intervals is created. This list forms an implicit binary tree, with the root node for the list $[i_a, i_b]$ taken as $i_{\text{mid}} = \lfloor (i_a + i_b)/2 \rfloor$ with left child $[i_a, i_{\text{mid}} - 1]$ and right child $[i_{\text{mid}} + 1, i_b]$. The second step for interval tree construction requires the iterative insertion of each interval into the tree. Finally, the min-list and max-list associated with each node is sorted as described above. The overall algorithm can be described as follows:

InsertInterval(tree, left, right, interval)

\[
\text{mid} = \lfloor (\text{left} + \text{right})/2 \rfloor
\]

if \(\text{tree.intervals}[\text{interval}].\text{max} < \text{tree.nodes}[\text{mid}].\text{split.value}\)
then

InsertInterval(tree, left, mid-1, interval)

else if \(\text{tree.intervals}[\text{interval}].\text{min} > \text{tree.nodes}[\text{mid}].\text{split.value}\)
then

InsertInterval(tree, mid+1, right, interval)

else
add interval to $tree\text{.}nodes[mid]$
endif

BuildIntervalTree($tree$)
sort list of interval values
store unique sorted list in $split\_value$
for each interval $i$
do
InsertInterval($tree$, 0, $tree\text{.}n\_uniq\_val-1$, $i$)
done
for each node $n$
do
sort $tree\text{.}nodes[n].\text{min\_list}$ by increasing maximum value
sort $tree\text{.}nodes[n].\text{max\_list}$ by decreasing minimum value
done

The overall cost of building the interval tree is $O(n_s \log n_s)$, dominated by the initial cost of sorting the interval values.

4.3.1.3 Interval Tree Queries

For a given query value $w$, the reporting of intersected intervals is performed by a modified binary search for $w$:

QueryIntervalTree($tree$, $w$)
\begin{verbatim}
left = 0
right = $tree\text{.}n\_uniq\_val-1$
while left $<$ right
do
mid = \lfloor(left + right)/2\rfloor
if $w > tree\text{.}nodes[mid].split\_value$
then
    traverse $tree\text{.}nodes[mid].\text{min\_list}$ reporting intervals with $min < w$
    right = mid-1
else
    left = mid+1
\end{verbatim}
traverse tree.nodes[mid].max_list reporting intervals with max > w
left = mid+1
endif
done

The total cost for resolving the query is $O(k + \log n_u)$, where $k$ is the size of the output and $n_u$ is the number of unique values from the set of min/max values.

4.3.2 Segment Tree

A segment tree also consists of a binary search tree over the set of min and max values of all the seed cells [Meh84, Mul94]. The primary difference from the interval tree is the manner in which the segments are stored. Nodes in a segment tree form a multiresolution hierarchy of intervals, with the root representing the infinite line, and with each node dividing the parent interval at a split value (see Figure 4.25). When a segment is inserted into the tree, it is recursively split and propagated downward in the tree to be inserted into the group of nodes whose intervals collectively sum to the entire range of the segment. Each segment will be stored at most $O(I_n \log n_u)$ times, where $\log n_u$ is the height of the tree, resulting in worst case storage complexity of $O(n_s \log n_u)$ in the improbable case that all min-max values are distinct, and all intervals filter all the way down to the leaves. The query complexity for reporting the $k$ intersected cells for a given isovalue $w$ is $O(k + \log n_u)$.

4.3.2.1 Segment Tree Data Structure

The segment tree data structure is similar to that of the interval tree. Note that for the segment tree there is no need to explicitly store the min/max values for each segment. As illustrated in Figure 4.25, there are three principal lists of cells associated with each unique interval value. We group these three lists into one segment tree node, as shown below. The tree structure is again implicit in the sorted ordering of the unique values.

```c
struct ST_CellList {
    int n_cells;
```
In the case of the segment tree, total storage is dependent on the number of times each interval is split. We will introduce \( n_i \), as the total number of cell identifiers stored in all lists of cells. For each node of the tree, we have the float split value, three pointers to cells and three counters for the number of cells in each list, for a total of \( n_i + 6n_u \) words of storage overall.
4.3.2.2 Segment Tree Construction

Construction of a segment tree is very similar to construction of an interval tree. The same binary structure is constructed over the unique extreme values of the seed cells. The primary difference is that each interval is recursively split and propagated down the tree from the root, rather than terminating at the first "split-value" which is spanned by the interval. The algorithm is sketched below:

```
InsertSegment(tree, left, right, cell_id, min, max, imin, imax)

mid = \[\frac{left + right}{2}\]

split_value = tree.nodes[\mid].split_value

if left = right
then
    if min < imax
    then
        add cell to tree.nodes[mid].lt_list
    else
        add cell to tree.nodes[mid].geq_list
    endif
    return
endif

if min <= imin AND max >= imax
then
    add cell to tree.nodes[mid].leq_list
    return
endif

if min < tree.nodes[mid].split_value
then
    InsertSegment(tree, left, mid, cell_id, min, MIN(max, split_value), split_value, imax)
endif

if max > tree.nodes[mid].split_value
then
```


\textbf{InsertSegment}(tree, \textit{mid}+1, \textit{right}, \textit{cell}.\textit{id}, \text{MAX}(\text{min}, \text{split}.\text{value}), \text{max}, \text{imin}, \text{split}.\text{value})

\textbf{BuildSegmentTree}(tree)

\text{sort list of interval values}
\text{store unique sorted list in} \text{split}.\text{value}
\text{for each interval} \text{i}
\text{do}
\hspace{1em} \text{InsertSegment}(tree, 0, \text{tree}.\text{nuniq}.\text{val}-1, \text{cell}.\text{id}, \text{min}, \text{max}, \text{-}\infty, \infty)
\text{done}

4.3.2.3 \textbf{Segment Tree Queries}

Traversal of a segment tree is much like traversal of an interval tree. For a given query value \textit{w}, the reporting of intersected intervals is performed by a modified binary search for \textit{w}. As each node is traversed, the associated list of cells is selected. At the conclusion of the traversal, one or both of the remaining two lists is selected, as outlined below:

\textbf{QuerySegmentTree}(tree, \textit{w})

\hspace{1em} left = 0
\hspace{1em} right = tree.\text{nuniq}.\text{val}-1
\hspace{1em} while left < right
\hspace{2em} do
\hspace{3em} mid = \left\lceil \frac{\text{left} + \text{right}}{2} \right\rceil
\hspace{3em} traverse tree.nodes[mid].leq_list and report all cells
\hspace{3em} if \textit{w} \leq tree.nodes[mid].split_value
\hspace{4em} then
\hspace{5em} right = mid
\hspace{3em} else
\hspace{4em} left = mid+1
\hspace{3em} endif
\hspace{3em} traverse tree.nodes[left].ltb_list and report all cells
\hspace{3em} if \textit{w} = tree.nodes[left].split_value
then

\texttt{traverse tree.nodes[left].geq_list and report all cells}

done

The total cost for resolving the query is $O(k + \log n_u)$, where $k$ is the size of the output and $n_u$ is the number of unique values from the set of \textit{min/max} values.

4.3.3 Bucket Search

Much of the scientific data that we are concerned with comes in the form of integer values in a small range. For example, Computed Tomography (CT) data generally have a 12-bit integer range of values. This regular subdivision allows a simple bucket search strategy with $n_u - 1$ buckets each representing a unit interval $(h, h + 1)$. For each cell, an identifier is stored in each bucket which is spanned by the cell. Clearly, the worst case storage complexity of this strategy is $O(n_u)$, which may be infeasible in the case in which all cells are stored. Given the approach of forming a small set of seed cells, such a technique may prove feasible, with the added benefit of allowing intersected cells to be reported in $O(k)$ time, linear in the number of reported cells.

![Bucket search structure for the intervals given in Figure 4.23](image)

\textbf{Figure 4.26:} Bucket search structure for the intervals given in Figure 4.23

4.3.3.1 Data Structure

\begin{verbatim}
struct B_CellList {
    int n_cells;
    int *cell_list;
};
\end{verbatim}
struct BucketSearch {
    int min, max;
    B_CellList *lists;
};

As in the case of a segment tree, each cell may be stored several times, and so we will use \( n_b \) to represent the total number of cell identifiers stored. In addition, we have one list for each unique extreme value, and so the total measured storage is \( n_b + 2n_u \).

### 4.3.3.2 Building a Bucket Structure

The creation of a bucket data structure is straightforward. For each bucket spanned by a cell, it is added to the associated list.

\[
\text{InsertInBuckets}(\text{bucket}, \text{cell.id}, \text{min}, \text{max})
\]

\[
\text{for } b = \text{min} \text{ to } \text{max} - 1 \\
\text{do}
\]

\[
\text{add cell to } \text{bucket.lists}[b].cell_list
\]

\[
\text{done}
\]

\[
\text{BuildBucketSearch}(\text{bucket})
\]

\[
\text{sort list of interval values}
\]

\[
\text{store unique sorted list in } \text{split.value}
\]

\[
\text{for each interval } i \\
\text{do}
\]

\[
\text{InsertInBuckets}(\text{tree, cell.id, min, max})
\]

\[
\text{done}
\]

The time required for building the search structure is proportional to the total number of buckets spanned by all cells, in worst case \( O(n_b n_u) \).

### 4.3.3.3 Bucket Search Queries

The advantage of the bucket search structure is that the range query complexity is entirely output sensitive, \( O(k) \). The procedure is outlined below:
QueryBucket(search, w)

\[ \text{bucket} = w - \text{bucket.min} \]

traverse search.lists[bucket].cell_list and report all cells

### 4.3.4 Search Structure Discussion

In this section we discuss the storage cost of each of the three presented search structures. Table 4.2 summarizes the theoretical space and query complexities.

<table>
<thead>
<tr>
<th>Search Structure</th>
<th>Storage Complexity</th>
<th>Query Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval Tree</td>
<td>( O(n_u) )</td>
<td>( O(k + \log n_u) )</td>
</tr>
<tr>
<td>Segment Tree</td>
<td>( O(n_u \log n_u) )</td>
<td>( O(k + \log n_u) )</td>
</tr>
<tr>
<td>Bucket</td>
<td>( O(n_u n_i) )</td>
<td>( O(k) )</td>
</tr>
</tbody>
</table>

Table 4.2: Comparison of the theoretical complexities of the three search structures for performing an interval query.

In examining the practical considerations, we have measured the storage of each data structure as shown in Table 4.3.

<table>
<thead>
<tr>
<th>Search Structure</th>
<th>Storage Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval Tree</td>
<td>( 5n_s + 4n_u )</td>
</tr>
<tr>
<td>Segment Tree</td>
<td>( n_i + 6n_u )</td>
</tr>
<tr>
<td>Bucket</td>
<td>( n_i + 2n_u )</td>
</tr>
</tbody>
</table>

Table 4.3: Comparison of the storage requirements in typical implementation of the three search structures.

It is clear from both the theoretical complexities and the empirical storage measures that the actual search structure size will depend on certain characteristics of the data. In particular, if \( n_u \) bounded (such as in the case of integer data), the theoretical storage and query complexities are the same for all three search structures.
Empirical results from the three seed set construction algorithms are given in Figures 4.27-4.31. We also compare the results with the size of an interval tree for the checkerboard seed set as well as the entire set of cells. Note that for data with integer values (the terrain data and the SOD data), the size of the segment tree is smaller than that of the interval tree, contrary to what the theoretical complexities might lead one to expect. In Figures 4.32-4.34 we compare the total preprocessing times for each seed selection algorithm. All times in these graphs are computed using the interval tree as a search structure. Note, in particular, that the total preprocessing time for the directional sweep is actually less than using the checkerboard approach or using the entire set of cells, simply because the directional sweep has time complexity $O(n_s)$ and the construction of the interval tree is $O(n_s \log n_u)$. The directional sweep is extremely fast and reduces size of the seed set sufficiently to actually provide an observed time cost advantage over all other approaches tested. Figure 4.35 displays an average cost of performing isovalue queries for an MRI dataset of size 256x256. Note that due to the fast inner loop of the segment tree and bucket search structure query algorithms, both exhibit an advantage in query time over the interval tree.

4.4 The Contour Spectrum

In addition to computational and space complexity issues, user interfaces have a tremendous impact on the interactivity of a visualization environment. In this chapter we describe a novel user interface component developed for enhancing interactive visualization of isocontours.

The *contour spectrum* is a signature consisting of a variety of scalar data and contour attributes, computed over the range of function values [BPS97]. We explore the use of surface area, volume, and gradient integral of the contour. Computed properties are presented to the user as a collection of 1D plots relative to the isovalue, giving the user a quantitative measure of the function to assist in selecting relevant isovalue for informative visualization. For time-varying data, properties can be computed
Figure 4.27: Number of seeds (a) and search structure storage requirements (b) for the Eagle Pass USGS Terrain Data (1201x1201)
Figure 4.28: Number of seeds (a) and search structure storage requirements (b) for a sample function (64x64)
Figure 4.29: Number of seeds (a) and search structure storage requirements (b) for the Hipip data (64x64x64)
Figure 4.30: Number of seeds (a) and search structure storage requirements (b) for the LAMP Climate Data (35x41x15)
Figure 4.31: Number of seeds (a) and search structure storage requirements (b) for the SOD data (97x97x116)
Figure 4.32: Comparison of preprocessing time required for the 5 seed cell extraction algorithms. The interval tree is used as the search structure in all cases.
Figure 4.33: Comparison of preprocessing time required for the 5 seed cell extraction algorithms. The interval tree is used as the search structure in all cases.
Figure 4.34: Comparison of preprocessing time required for the 5 seed cell extraction algorithms. The interval tree is used as the search structure in all cases.

Figure 4.35: Comparison of query time for the interval tree, the segment tree, and the bucket search structures. Query time computed as an average over 1000 searches and is plotted as a function of iso-value.
over time, and displayed using a 2D interface, giving the user a global overview of the time-varying function, allowing interaction in both isovalue and timestep.

4.4.1 Contour Attributes

A contour spectrum consists of computed metrics over the scalar field. On the basis of such metrics we can define a set of functions which provide a useful tool to enhance the interactive query of the dataset. One primary advantage of the contour spectrum interface is that allows one to display in a 2D image a "global" view of the examined scalar field, independent of its dimension. For example, in the display of a 3D isosurface, one contour component may be hidden inside another. If we associate the isocontour display with the contour tree (see below) it becomes immediately clear that the current isosurface is composed of two components and hence we might need a clipping plane to look inside the current isosurface.

Below we report on several examples of contour measures of general utility. The flexibility of the interface allows for numerous enhancements for both general attributes and application specific features.

4.4.1.1 Isoline/Isosurface Length/Area

In this section we introduce the methodology used for efficient exact quantitative queries over the scalar field. In particular we determine a simple spline-based algorithm which allows the exact length (area) computation of an isocontour. The spline approach makes the computed data suitable for direct display in the contour spectrum.

Given a 2D (3D) scalar field we determine the exact length (area) value of any isocontour of height \( w \). Such scalar quantity is represent as a spline function of the isovalue \( w \). The spline functions are easily displayed in the contour spectrum without introducing approximations (with respect to the given sampled data), and at the same time is used to perform interactive quantitative queries. The method generalizes to meshes of higher dimensions, providing a means for analyzing (with the spectrum)
and interactively perform quantitative queries on datasets of any dimension, independently from the ability to display them.

4.4.1.2 2D Contour Length

The two-dimensional case is particularly simple and is treated in detail to introduce the general methodology which becomes increasingly useful for dimensions three or higher.

Consider a 2D scalar field composed of triangles $t_i$ and vertices $v_i$ such as the terrain in Figure 4.36. We build and display the spline function $L(w)$ whose value $L(w_0)$ is the length of the isocontour of height $w_0$. $L(w)$ can be computed as the sum of all the contributions $L_i(w)$ given by each cell $c_i$ to the length of the contours:

$$L(w) = \sum_i L_i(w)$$

Thus, we can concentrate on the computation of the generic term $L_i(w)$ associated with the triangle $t_i$, as illustrated in Figure 4.36. Triangle $t_i$ has vertices $v_1$, $v_2$ and $v_3$ with height values $F(v_1) \leq F(v_2) \leq F(v_3)$. Given the equation $f(x, y, w) = 0$ of the plane containing $t_i$, the value $L_i(w_0)$ is the length of the intersection between $t_i$
(projection of $t_i$ onto the mesh space) and the 2D line of equation $f(x, y, w_0) = 0$ (see figure 4.36). As we change the value of $w_0$ we obtain the measure of all the slices parallel to the line $f(x, y, 0) = 0$. In general it is know from spline theory that given a $d$-simplex in $\mathbb{R}^d$ the function that gives the measure of all the parallel slices of such simplex (that is the measure of the intersection with a set of parallel hyperplanes) is a degree $d - 1$, $C^{d-2}$ continuous, B-spline function [dB78].

In the 2D case the B-spline is simply a piecewise linear $C^0$ function. Hence we need only compute the length of the segment for $w = F(v_2)$ and connect it with the other two extremes for which the length is 0.

Note that the B-spline formulation of the length is also useful to automatically handle the eventual degenerate cases. For example a portion of the terrain at height $w$ can be a flat parallel to the $x, y$ plane (a lake). In this case there occurs a definition problem, in determining the length of an isocontour which is partially a 1-dimensional curve and partially a 2D surface. The natural solution is to remove the flat region to regularize the dimension of the contour. The consequence is that the function that computes the contour length is only $C^{-1}$ at the height $w$. Using the B-spline approach no special care must be taken for this case since the knot vectors of the flat triangles are $F(v_d) = F(v_2) = F(v_3)$ resulting in "valid" splines which shrink to a point (as they should be).

4.4.1.3 3D Contour Area

As already pointed out, the above spline function can be computed for simplices of any dimension. For the 3D case of a tetrahedron $(v_1, v_2, v_3, v_4)$ with scalar function values $(F(v_1) \leq F(v_2) \leq F(v_3) \leq F(v_4))$ we have a degree two $C^1$ B-spline (see Figure 4.37). In this case the determination of the control polygon is as follows:

- First the area $L(v_2)$ of the section of height $F(v_2)$ is computed.

- A straight line from the point $(\frac{F(v_1)+F(v_2)}{2}, 0)$ passes through the point $(F(v_2), L(v_2))$ and continues up to the point $P$ of abscissa $\frac{F(v_2)+F(v_3)}{2}$. The point $P$ is then connected with the point $(\frac{F(v_3)+F(v_4)}{2}, 0)$. 


Again for each cell we obtain a spline function, as illustrated in Figure 4.37. The sum of the splines associated to each cell is a single spline that gives the contour area for any isovalue.

4.4.1.4 Inside Area/Volume Computation

Once the length/area function of the isocontours is given the Area/Volume of the region "below" ("above") the isocontour can be determined by exact integration of the length/area spline function. This gives as a result a new spline function in which degree and continuity are increased by one. In this way we can easily plot the area/volume spectrum. The case for the 2D contour is illustrated in Figure 4.38.

4.4.1.5 Gradient Integral

While length and area are important metrics to report, in many cases they are not sufficient to guide the user in choosing appropriate isovalues. In many situations the user is interested in finding and displaying prominent surfaces in the data. Toward this end we have designed a metric which is based on the slope or gradient of the function. The difficulty with the gradient measure is to define it properly, since
Figure 4.38: 2D area computation by integration of the length function $L(w)$. The shaded region corresponds to the area less than or below the isovalue. The area above the isovalue is computed symmetrically.

along a particular contour the gradient of the scalar field is not (usually) constant. To compute a consistent (single valued) gradient function we resort to the spline decomposition of the contour length/area function. For each triangle/tetrahedron of the mesh we have a spline function which gives the length of any contour within that triangle/tetrahedron. Moreover, by piecewise linear approximation, within each triangle/tetrahedron the gradient of the scalar field is constant. Hence to determine the contribution to the gradient function of the contours within a single triangle we just need to multiply the length function by the absolute value of the (constant) gradient. Again the sum of the splines defined in each triangle/tetrahedron gives a single global spline function which defines the gradient integral of any isocontour in the scalar field. Figure 4.39 shows an MRI scan of a human heart. The maximum of the gradient (yellow function plot) corresponds to the isocontour (red contour on top figure) bounding the relevant portion of the data. Note how the maximum of the contour surface (red function plot) is attained for a lower height value of the field.
Figure 4.39: Top: MRI scan data of a heart. Bottom: the corresponding contour spectrum with in yellow the normalized gradient.

It captures the noisy part of the data that has a large contour length due to the numerous components.

4.4.2 Real Time Quantitative Queries

The 2D plot of each of the above metrics provides a qualitative understanding of their trend. Once an isocontour is selected the user is usually interested in the exact value of each of such metrics. This can be accomplished using the same spline representation. Since the spline defined above are exact representations of the relative metrics for the given piecewise scalar field we need only to search in the knot vector for the interval in which the selected iso-value lies and evaluate the related portion of spline. This would take in the worst case $O(\log n)$ (the evaluation can be considered $O(1)$) time where $n$ is the number of different scalar values at the mesh vertices. Note
that for the MRI data we have \( n = 256 \). In the general case, if \( n \) is too large we can apply any error bounded reduction scheme to keep \( n \) within an acceptable value. In such cases we will not get exact but error bounded results.

4.4.2.1 Contour Tree

While the display of contour metrics is both helpful and informative, there is clearly a lack of global structural information in the metrics described. For example, there is no indication of features such as local maxima and minima of the field. For this purpose we introduce the use of the contour tree as a tool for assisting the user in interaction with complex scalar fields. A contour tree captures the global changes in contour topology of the scalar field defined on the input the mesh. It has been used before in image processing and GIS research [FM67, GC86, KK94, SC86, TIS+95]. Another name in use is the topographic change tree, and it is related to the Reeb graph used in Morse Theory [TIS+95]. Note the difference from the topology graph [BS96b], which remains embedded in the mesh space and hence for 3D meshes is not displayed as a 2D graph.

Figure 4.40 shows a 2D scalar field along with its associated contour tree. For each edge in the contour tree there is a connected component of an isocontour in the scalar field. If, while varying the isovalue, two contour components merge together we have in the contour tree two edges that join. Similarly, if an isocontour splits in two or more components we will have in the contour tree an edge that splits in two or more edges. Moreover the comparison between the contour tree and the spectrum may aid in the selection of interesting contours. Typically an isovalue that has a contour tree with many edges but a relatively small overall contour length/area corresponds to a noisy region. Symmetrically a single component of large length/area correspond to a well defined featured of the scalar field. Computation of the contour tree is discussed in greater detail in [vvOB+97].
4.5 User Interface

The user interface for presenting the contour spectrum takes on two forms. For static data, a window presents a selected subset of the computed data characteristics in 1D plots. The horizontal axis represents the isovalue dimension. The vertical axis represents the range of each function, all of which are normalized for overlapping display. See Figure 4.41 for an example. The user may select a subrange of the isovales for display in order to enhance the local detail in the computed metrics. Vertical bars represent the current isovales, which the user may change with a familiar click-and-drag operation.

Figure 4.40: A 2D scalar field (top) with the associated contour spectrum and in white the relative contour tree (bottom).
With time-varying data, it is desirable that the user have the ability to quickly browse all parameters of the visualization. In this case we use the vertical dimension of the interface as an index into the timestep of the data. Of course, while we use time here as an example, other parameters may be varied similarly, such as input parameters to a numerical simulation. Using this interface, each point in the 2d display maps to a number of functions. We selectively display one function at a time by pseudocoloring of the function values over the 2d grid, as shown in Figure 4.42.

4.5.1 Rule-based Contouring

An interesting and promising pursuit is to develop techniques which strategically choose a set of key isovalues which convey the data most clearly.

An important caveat to rule-based contouring is that users familiar with a particular isovalue selection mechanism, such as the selection of a evenly spaced isovalues, may easily misinterpret the display of a number of contours which are irregularly scattered throughout the range of the function.
The contour spectrum allows the development of an adaptive ability to capture the "interesting" features of a dataset. Figure 4.43 shows the scalar field obtained as a CT scan of an engine. The main component of the engine can be easily determined by selecting the maximum of the gradient integral. Of course this remains simply an aid in the interactive querying stage of the dataset, as the concept of "interesting" feature of a scalar field remains highly dependent on the type of dataset we are dealing with.

4.5.2 Future Work

In addition to increasing user interaction, quantitative interfaces for visualization are a first step to developing the ability to automatically select visualization parameters for effective visualizations. While certain general isovalue selection techniques are discussed here, we propose that application specific rules for isovalue selection based on metric properties be developed.

The measure of visualization effectiveness is the amount of insight gained by the user. For automated visualization and parameter selection to become viable and
effective, it will be necessary for visualization users to understand the implications of the parameter selection techniques which have been applied.

4.6 Summary

In this chapter we have addressed the search and traversal phases of isocontouring. We have presented three algorithms for the construction of seed sets which trade computational and storage complexity in the preprocessing phase for the size of the resulting seed set. Three data structures supporting the range query operation have been compared, both in theory and in practical implementation. We have also described a novel, quantitative user interface for isocontouring. The Contour Spectrum
provides quantitative information and an intuitive interface for assisting user interaction with scalar fields of arbitrary dimension. The scalar field signatures used in the contour spectrum may be used to develop automated isovalue selection techniques in a general or application specific manner.
5. SUMMARY

We have presented a collection of algorithms and data structures for supporting interactive visualization and interrogation of scientific data.

We have described a simplification strategy which has the ability to bound the accumulated global error through error approximation volumes which grow to bound additional introduced error at each step of the iterative simplification. The technique is general and easily extends from terrains to multi-valued functions by adopting an error-vector representation and prioritizing simplification operations based on norms of the accumulated error vector. The technique is further extended to bound error in geometry to allow for simplification of surfaces with functions defined on them. The geometric interpretation of the error is that of a non-directional error value which defines a tolerance sphere at each point on an oriented 2-manifold. We further describe how our simplification techniques can be used to generate continuous level-of-detail hierarchies which are suitable for dynamic triangulation based on a number of criteria, such as viewing direction and lighting conditions. We also describe how modifications to our geometry simplification can explicitly detect and maintain sharp features for the reconstruction of smooth models from unorganized points.

The ability to interactively select isocontour levels permits exploration of scalar fields and reveals global structure. We have presented three algorithms for reducing the amount of storage required by the data structures for resolving contour queries. The first algorithm adopts a more exhaustive approach by explicitly sweeping out the region which is covered by each selected seed cell. The sweep algorithm has the advantage of being extremely low-cost, due to the fact that only first-order adjacency information is used to select seed cells. The propagation algorithm uses slightly more storage and takes advantage of a regular traversal order for grids of regular
topology. Together these algorithms provide an array of performance characteristics which can be balanced with their varying computational complexities in determining which algorithm is appropriate for a particular application.

We have also presented three alternative data structures which permit fast querying for intersected cells. Both the interval tree and the segment tree provide $O(k + \log n_s)$ query complexity in determining the $k$ intervals which are intersected for an arbitrary input value. The $O(n_s \log n_u)$ storage complexity of the segment tree is more costly than the $O(n_s)$ storage complexity of the interval tree in theory, however the segment tree has the advantage that entire lists of cells are selected at once, without the need to traverse the list making comparisons, providing a demonstrated improvement in query time, which is of practical interest. The bucket search structure takes this one step further for a class of data which has an indexed segmentation, such as integer-valued data. With a worst-case storage complexity of $O(n_s n_u)$, the bucket search structure reports in constant time the number of seed cells which are intersected, an enumerates the list in $O(k)$ time, providing optimal enumeration of intersected cells for this special class of data.

The advantage of our seed set based approach to contouring is that it inherits the benefits of a variety of prior contouring approaches. Through the use of contour propagation we can exploit spatial coherence in the computation and data access. The use of value-space search structures for the seed sets permits fast resolution of queries without the storage overhead of techniques which construct search structures over the entire set of cells.

Future extensions of the work described in this thesis will seek to integrate additional visualization techniques with the simplification strategies described within. In addition to representing a mesh in a continuous multi-resolution framework, it is desirable to extend visualization algorithms such as isocontouring and volume rendering to allow for interactive multi-resolution visualization. A number of open problems exist in such work, such as defining the appropriate error measures such that errors in the isocontours or errors in the final image can be efficiently bounded. This
will involve bounding errors introduced by mapping through transfer functions and additional accumulation errors due to compositing of transparent volumes.
LIST OF REFERENCES
LIST OF REFERENCES


VITA
VITA

Daniel R. Schikore was born on the first of July 1970 in Florissant, Missouri, USA. He studied at Purdue University, where he earned the Bachelor of Science degree in Computer Science and in Mathematics in May 1992. He joined the graduate program in the Department of Computer Sciences of Purdue University in May 1992 and was awarded the Ph.D. in August 1997. He is a member of ACM, IEEE Computer Society, and of the Upsilon Pi Epsilon and Phi Kappa Phi honor societies.

His research interests include computer graphics, scientific data visualization, parallel rendering, geometric modeling, computer aided design, and applied computational geometry.