

**UNIFORM AND ADAPTIVE (RE)MESHING  
OF SURFACES**

by

John Michael Schreiner

A dissertation submitted to the faculty of  
The University of Utah  
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

School of Computing

The University of Utah

May 2009

Copyright © John Michael Schreiner 2009

All Rights Reserved

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

**SUPERVISORY COMMITTEE APPROVAL**

of a dissertation submitted by

John Michael Schreiner

This dissertation has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

---

---

Chair: Cláudio T. Silva

---

---

Robert M. Kirby II

---

---

Valerio Pascucci

---

---

Emil Praun

---

---

Peter Shirley

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

**FINAL READING APPROVAL**

To the Graduate Council of the University of Utah:

I have read the dissertation of \_\_\_\_\_ John Michael Schreiner \_\_\_\_\_ in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

\_\_\_\_\_  
Date

\_\_\_\_\_  
Cláudio T. Silva  
Chair: Supervisory Committee

Approved for the Major Department

\_\_\_\_\_  
Martin Berzins  
Chair/Director

Approved for the Graduate Council

\_\_\_\_\_  
David S. Chapman  
Dean of The Graduate School

## ABSTRACT

Triangle meshes are frequently used for computer representations of surfaces, with applications in computer graphics, visualization, and finite element analysis. Many geometry processing operations require meshes with high-quality triangles as input, but tend to degrade the quality of the output. This drives the need for remeshing algorithms, which take a poor mesh as input, and improve the sizes and shapes of the triangles while keeping the geometry the same. While a wide variety of geometry processing operations operate only on meshes, many surfaces are acquired with different underlying representations. For example, isosurfaces are often acquired from CT and MRI scans, and point set surfaces are acquired from laser range scans. This drives the need for generating meshes from other surface representations. This dissertation presents novel methods for meshing and remeshing surfaces in both uniform and adaptive ways.

A novel parameterization-based method for generating highly-uniform remeshes is presented. This method considers the fully general problem of creating a map between two arbitrary triangle meshes. Whereas previous approaches compose parameterizations over a simpler intermediate domain, the presented method directly creates and optimizes a continuous map between the meshes. The distortion of the map is measured with a new symmetric metric, and is minimized during interleaved coarse-to-fine refinement of both meshes. By explicitly favoring low intersurface distortion, maps are obtained that naturally align corresponding shape elements. Typically, the user need only specify a handful of feature correspondences for initial registration, and even these constraints can be removed during optimization. The method robustly satisfies hard constraints if desired. This general intersurface mapping framework can be applied to parameterize surfaces onto simplicial domains, such as coarse meshes, and octahedral and toroidal domains. These parameterizations can be used to create high-quality remeshes.

This dissertation also presents a novel surface meshing algorithm that is closely related to surface reconstruction techniques, and requires no explicit parameterization. This approach is based on the advancing front paradigm, augmented with a *guidance field* to both adapt the triangle sizes to the surface curvature and bound rate at which they can change. Simple and intuitive user controls are given for the size and adaptivity of the triangles. The method can be used to mesh surfaces with a wide variety of underlying definitions, including isosurfaces, point set

surfaces, as well as other meshes. It is accurate, fast, robust, and suitable for use with interactive mesh processing applications that require local remeshing. A number of applications are shown, including Boolean operations between surfaces with different underlying definitions, extraction of large out-of-core isosurfaces that do not fit in working memory, and reconstruction of point set surfaces with sharp features.

To Mom and Dad.

# CONTENTS

<b>ABSTRACT</b> .....	<b>iv</b>
<b>LIST OF FIGURES</b> .....	<b>x</b>
<b>LIST OF TABLES</b> .....	<b>xvii</b>
<b>ACKNOWLEDGEMENTS</b> .....	<b>xviii</b>
<b>CHAPTERS</b>	
<b>1. INTRODUCTION</b> .....	<b>1</b>
1.1 Motivation .....	1
1.2 Contributions .....	3
<b>2. BACKGROUND</b> .....	<b>7</b>
2.1 Parameterization .....	7
2.1.1 Planar Parameterization .....	7
2.1.2 Spherical Parameterization .....	8
2.1.3 Simplicial Parameterization .....	9
2.1.4 Intersurface Mapping .....	9
2.2 Surface Triangulation .....	11
2.2.1 Remeshing .....	11
2.2.2 Isosurface Extraction .....	12
2.2.3 Point Set Surfaces .....	13
2.2.4 Advancing Fronts .....	14
<b>3. UNIFORM REMESHING: INTERSURFACE MAPPING</b> .....	<b>16</b>
3.1 Algorithm Overview .....	16
3.2 Map Representation .....	17
3.3 Initialization of the Coarse Map .....	18
3.3.1 Path Tracing .....	20
3.3.2 Avoiding Separating Cycles .....	21
3.3.3 Avoiding Swirls .....	23
3.3.4 Boundaries and Features .....	24
3.4 Coarse-to-Fine Optimization .....	24
3.4.1 Vertex Optimization .....	25
3.4.2 Boundaries and Features .....	28
3.4.3 Kink Vertices .....	29
3.4.4 Distortion Metric .....	29
3.5 Applications .....	31
3.5.1 Intersurface Mapping .....	31



3.5.2	Simplicial Parameterization	36
3.5.3	Octahedral Parameterization	36
3.5.4	Toroidal Parameterization	37
3.6	Discussion	39
<b>4.</b>	<b>ADAPTIVE MESHING: GUIDANCE FIELD ADVANCING FRONT</b>	<b>42</b>
4.1	Algorithm Overview	43
4.1.1	Curvature Adaptivity	45
4.1.2	Lipschitz Edge Sizing	46
4.2	Surface Definitions	49
4.2.1	Triangle Meshes	50
4.2.2	Isosurfaces	51
4.2.2.1	Structured Grids	53
4.2.2.2	Unstructured Grids	54
4.2.3	Point Set Surfaces	55
4.2.3.1	Nonlinear	56
4.2.3.2	Linear	56
4.3	Guidance Field	57
4.3.1	Constructing the Guidance Field	57
4.3.2	Sampling Condition	59
4.3.3	Trimming	61
4.3.4	Approximation Error	63
4.3.5	Triangle Quality	66
4.4	Surface Triangulation	67
4.4.1	Initialization	68
4.4.2	Advancing Fronts	68
4.4.3	Front Interference Detection	69
4.5	Streaming Output	70
4.6	Implementation	72
4.6.1	Guidance Field Evaluation	72
4.6.2	Bounding Curvature	73
4.6.2.1	Interval Arithmetic	75
4.6.2.2	Tensor Product Splines	76
4.6.3	Isotopic Initial Boundary Fronts	78
4.6.4	Out-of-Core	80
4.6.4.1	Streaming Input	80
4.6.4.2	Incremental Boundaries	81
4.6.4.3	Connected Components	83
4.6.5	Parallelization	84
4.7	Applications	85
4.7.1	Remeshing	85
4.7.1.1	Boolean Operations	85
4.7.1.2	Mixed-Mode	89
4.7.2	Isosurface Extraction	90
4.7.3	Multimaterial Volumes	95
4.7.4	Out-of-Core	98
4.7.5	Point Set Surfaces	100
4.8	Discussion	103

4.8.1 Performance .....	105
4.8.2 User Control .....	106
<b>5. CONCLUSIONS AND FUTURE WORK .....</b>	<b>108</b>
<b>APPENDIX: ADVANCING FRONT SOFTWARE ARCHITECTURE .....</b>	<b>111</b>
<b>REFERENCES .....</b>	<b>118</b>

## LIST OF FIGURES

3.1 Example of the consistent partitioning process. Given a set of corresponding feature points, the meshes $M^1$ and $M^2$ are partitioned into corresponding patches (left). Each mesh is then simplified while preserving the patch boundaries, which produces base meshes with corresponding vertices and identical connectivity. . . . .	18
3.2 Vertex ordering must be consistent when creating a consistent partitioning. If the ordering is inconsistent, it will be impossible to create a bijection. For example, intersecting paths will be required, and the triangles ABD and ACE on the left will overlap in the area shown in yellow on the right. . . . .	19
3.3 If the user does not define enough feature correspondences (green) to resolve the genus of the object, additional correspondences are automatically inserted (blue). These extra correspondences are only used to initialize the map, and are not maintained as hard constraints throughout the optimization. . . . .	21
3.4 If a direct connection between A and B would create a separating cycle, a non-separating cycle is created instead. A breadth first search starting from the path AB creates an expanding front (b). This front splits at a point P (c), and merges back together at a point O (d). The “parent” fields trace two paths that create a nonseparating cycle P and O (e). A nonseparating cycle AX–XB is inserted into the path network. . . . .	22
3.5 There are multiple ways to connect set of points A, B, C, and D. Both ways shown here have the same path ordering around each vertex (e.g., CA, CD, CB appear in the same order in both cases. The paths on the right can only be created by rotating clockwise the points C and D around each other. Hence the right configuration is a <i>swirl</i> . . . . .	23
3.6 The configuration of the neighborhood of a vertex $v$ before optimization. Mesh $M^1$ is shown in green and mesh $M^2$ is shown in black. . . . .	25
3.7 A 2D parameterization of the neighborhood is used to facilitate the optimization. First, the 1-ring neighborhood of the vertex of $M^2$ being optimized is unfolded (a). The corresponding region of $M^1$ is then mapped to the 2D parameterization (b). The kernel of the boundary (c) is used to perform multiple line searches (d) to minimize the distortion. . . . .	26

3.8	Once a new vertex location is found that minimizes the distortion in the 2D parameterization (a), the meta-mesh must be updated. First, the region now containing $\hat{v}$ is triangulated (b). The same triangulation is applied to $M^1$ , and barycentric coordinates are used to find the new location of $\tilde{v}$ on $M^1$ (c). Similarly, the split ratios of the edges of $M^2$ within the 2D neighborhood are found, and mapped to $M^1$ (d). For each vertex of $M^1$ lying in the neighborhood, the face of $M^1$ containing it is found (e), and barycentric coordinates are used to map it back to $M^2$ (f). The meta-mesh update is completed by mapping the split ratios of the edges of $M^1$ back to $M^2$ (g).	27
3.9	To evaluate the distortion, the 2D parameterization of the neighborhood is triangulated, which is then applied to the neighborhoods on both $M^1$ and $M^2$ (top). This gives a common triangulation on both of the meshes, which are used to evaluate the distortion. One such corresponding triangle is highlighted (bottom).	28
3.10	A given location for $\hat{v}$ (a) will sometimes create a fold in the intersurface map because a direct segment $\tilde{v}\tilde{w}$ goes on the wrong side of $\tilde{u}$ (b). A kink vertex is required to redirect the edge around $\tilde{u}$ (c).	30
3.11	The use of a conformal metric results in a poor intersurface map. Instead of bringing the heads of the animals into correspondence, they are each mapped to a small region on the other's neck.	31
3.12	A cow-horse intersurface map using only 4 features.	32
3.13	An intersurface map between two genus-1 objects. (a,b) use fixed constraints while (c,d) drop the constraints after initialization. The cup edges are shown on the teapot in (a,c), and the teapot edges are shown on the cup in (b). A 50% morph between the meshes is shown in (d). (Symmetric stretch efficiencies: (a,b) 0.471, (c,d) 0.598.)	33
3.14	An intersurface map for two objects of genus 2, initialized with 8 user-specified feature points. (Symmetric stretch efficiency 0.311.)	34
3.15	Map between two meshes with boundaries. The close-up on the eye shows low distortion around the feature point ( $M^1$ edges over $M^2$ geometry). The boundary at the necks are treated as path constraints. (Symmetric stretch efficiency 0.967.)	35
3.16	The intersurface map automatically favors shape correspondence, unlike the composition of two separate simplicial parameterizations, as shown in these morphs. The simplicial map uses the 17 feature points shown in Figure 3.1. (Symmetric stretch efficiencies: (a) 0.416, (b) 0.442)	35
3.17	Comparison of semiregular remeshing using GSP (middle) and intersurface mapping (right). Left: base domain patches. (One-way stretch efficiencies: bunny 0.800, 0.915; David 0.761, 0.902.)	37
3.18	Praun and Hoppe [115] use a sphere as an intermediate domain to create geometry images [58] without making cuts on the mesh. Both the mesh and octahedron are mapped to the sphere, which are combined to form an intersurface map. The octahedron is then unfolded to create a geometry image (top). Improved geometry images with better parameterization efficiencies and more accurate remeshes can be created by directly optimizing the intersurface map (bottom).	38

3.19	The toroidal base domain tessellation is made up of nine vertices (A–I) that repeat when unfolded to the plane. . . . .	39
3.20	Examples of toroidal parameterization and remeshing. (One-way stretch efficiencies: teapot 0.458, rocker arm 0.582.) . . . . .	40
4.1	The advancing front algorithm can be divided into six components: the input, seed identification, guidance field construction, initialization of the seed fronts, triangulation of the surface, and finally handling the output stream. . . . .	44
4.2	The ideal edge size function $\iota$ is controlled by a user parameter $\rho$ . The ideal edge subtends the angle $\rho$ on a sphere of radius $r = \kappa_{\max}(\mathbf{x})^{-1}$ . If the curvature changes too quickly, such as between points $A$ and $B$ , poor triangles may be created. When the curvature changes slowly, such as between points $B$ and $C$ , the quality will be good. The Lipschitz property of $g$ will enforce that the edge lengths always change slowly, independent of how fast $\iota$ changes. . . . .	46
4.3	Depending only on the curvature to determine the edge sizes makes advancing front algorithms sensitive to changes in curvature. If the algorithm does not anticipate the need for small triangles early, large triangles will be created next to small ones (left). The expected behavior is shown on the right. . . . .	47
4.4	Bounding the minimum angle $\beta$ in a triangle is equivalent to bounding the ratio of edge lengths to a parameter $\eta$ . When these are used to bound shapes of individual triangles, edges from large triangles can still share vertices with edges from small triangles. This results in situations where poor quality triangles are forced to be created. . . . .	48
4.5	The rate at which edges can change is controlled by a user parameter $\eta$ . No pair of edges incident to any given vertex should have a length ratio greater than $\eta$ or less than $\eta^{-1}$ . This constrains the triangle shape and enforces a smooth grading of the triangle sizes. . . . .	49
4.6	Different projection procedures can produce very different triangle shapes. Closest point projection behaves poorly around sharp corners, which are common in many meshes (middle). Projecting the point in a circular arc maintains the edge lengths of the new triangle (right). . . . .	50
4.7	To estimate the curvature at a vertex of a mesh, a quadratic polynomial is iteratively fit to a neighborhood of the vertex. If the neighborhood is too small, noise in the input mesh will produce inaccurate estimations, leading to artifacts in the output mesh (middle). A larger neighborhood slows the computation, but produces more stable results (right). . . . .	51
4.8	Illustration of $\hat{g}_i(d)$ , the function that defines the correct edge size as a function of the distance to a sample $s_i$ of the surface. . . . .	59
4.9	The guidance field $g(t)$ on a curve $t : \mathbb{R} \rightarrow \mathcal{S}$ . Note that the functions are plotted against the parameter $t$ , not the curve itself. $g(t)$ is the minimum over all $\tilde{g}$ . At each sample point $s_i$ , $\tilde{g}_i$ is at its minimum, and grows linearly as the distance from $s_i$ increases. Since each $\tilde{g}_i$ is Lipschitz, so is $g(t)$ . . . . .	60

4.10	Note that if the sampling $s_i$ of the surface is too coarse, $g(t)$ might not bound $l(t)$ (as shown in the region inside the red circle). Section 4.3.2 shows how to provably prevent this. Additionally, some samples will not influence $g(t)$ (in the figure, the sample shown in blue). Section 4.3.3 shows how to efficiently remove these points.	61
4.11	Pseudo-code for trimming the guidance field.	63
4.12	Two models are rendered using only the points in the guidance field — the black speckles are simply the inside of the surface without any illumination. After trimming, all the tan-colored points are removed from the data structures, leaving only the red points for querying.	65
4.13	A comparison of the predicted error and measured error for a set of remeshes, in percent of the bounding box diagonal. The measured error is less than half the predicted error when the input mesh is sufficiently smooth compared to the output mesh.	66
4.14	Cumulative histograms show the quality distribution of triangles in a mesh. The quality of a triangle is measured by the ratio of its incircle to circumcircle radii, normalized so that the best ratio is 1.0. On the left, input meshes and the results of competing triangulation methods are shown. On the right are histograms from all of the meshes generated by the advancing front algorithm.	67
4.15	Pseudo-code for the triangulator at the core of the advancing front algorithm.	69
4.16	To robustly detect front interference, a set of <i>fences</i> are used. These are extensions of the front curve in the normal direction of the surface. The bound on the Hausdorff error is exploited to determine the correct fence height — the inset on the right shows the argument in two dimensions.	70
4.17	Triangles are kept in memory until all three vertices have no neighbors on any of the advancing fronts. This provides a band (shown in gray) in which edge flips are performed to improve triangle quality.	71
4.18	An efficient algorithm to evaluate the guidance field at a given point makes use of a kd-tree to traverse the guidance field samples $s$ .	73
4.19	The boundaries of the surface are used as initial fronts. The boundaries of the Marching Cubes mesh (left) are not always isotopic to the isosurface, due to the regular structure of the MC mesh. Additionally, MC may produce spurious connected components. These may still be used for seed points for fronts since they will simply merge together. The output of the advancing front algorithm is shown on the right.	79
4.20	The blocks composing the volume are traversed in z-order minimize the shuffling of blocks in and out of memory. When traversed in this order, the active block at any time (red) will border blocks that have already been visited (blue) in the negative axial directions, and blocks that have not yet been visited (white) in the positive directions. A shell of blocks around the active block (bold square) must be in memory to evaluate the implicit function and guidance field near the active block boundaries.	81

4.21	Comparison of memory usage (top) and running time (bottom) between Marching Cubes and the advancing front algorithm. The input volume is a $512^3$ scan of a vertebra, and “# blocks” is the number of blocks it was divided into along each axis. “0” blocks uses the standard in-core codebase instead of the out-of-core version. The running time increases approximately linearly as the block size decreases, so the largest block size that does not cause memory swapping should be used for the best performance. . . . .	82
4.22	As new blocks are loaded into memory, new boundary segments are identified. The partial segments are closed into loops by <i>phantom</i> edges, shown in red. When new boundaries are inserted into the set of active fronts, they are joined at the endpoints of the phantom edges. . . . .	83
4.23	When the triangulation moves to a new active block, new seed points for the surface within that block may need to be created. If the portion of the surface exits the active block to any other blocks that have already been visited, new seeds are not required (a). Each connected component that either lies entirely within the active block, or only exits the block to unvisited neighbors will need seeds (b). Some connected components may enter and leave the active block multiple times. Each distinct part of the component that enters the block is treated independently, some may require seeds and some may not (c). . . . .	84
4.24	A sample of the results of our algorithm. From left to right we show remeshes of the Happy Buddha, the Feline, Pensatore, a Marching Cubes reconstruction of a pelvis bone used for a biomechanics simulation, and the head of Michelangelo’s David. . . . .	86
4.25	Many surface processing tasks require <i>good</i> meshes. At the same time, many meshes created automatically exhibit bad triangulations. The advancing front remeshing algorithm is based on surface reconstruction and requires no parameterization. The top row shows the watertight and manifold input mesh that was created with Reconstruct3D [78], and the bottom row shows a remesh generated by the advancing front algorithm. . . . .	87
4.26	An example of a CSG difference operation. The output generated by Maya is shown in the upper-right corner, while the bottom row shows the results of using advancing fronts. The algorithm only changes the portion shown in blue. . . . .	90
4.27	The generality of the advancing front technique allows meshing of <i>mixed-mode</i> models: CSG operations between a mesh and a point-set, for example. The union of a triangle mesh with genus 48 and a torus defined by a point set is computed, with a final genus of 68. Only a local portion of the input mesh is remeshed, shown in blue. The entire input point set is triangulated, shown in green. Note that triangles meet one another in the intersection curves with the same resolution, yielding high triangle quality. . . . .	91
4.28	Some results of the advancing front algorithm, compared to MC and MT. From left to right: CT scans of an aneurism, a bonsai tree, an engine block, and isopotential surfaces of a human torso simulation. The first three datasets are regular grids, while the last one is a tetrahedral mesh. . . . .	91

4.29	Triangle meshes generated from Marching Cubes have inherently biased sampling, which produces badly shaped triangles (left). A modified MC algorithm that makes use of an extended case table [117] greatly reduces the number of poorly shaped triangles, but incorrectly connects triangles across small gaps, and creates nonmanifold meshes (middle). The advancing front algorithm ensures appropriate mesh grading and curvature adaptation, and generates triangle meshes with excellent triangle shape (right). . . . .	92
4.30	Isosurface extraction from a structured grid of a silicium lattice simulation. From left to right: Marching Cubes output, and the advancing front method for $\rho = 0.3$ , using, respectively, Catmull-Rom and B-splines for reconstruction. . . . .	93
4.31	Isosurface extraction from unstructured grids. From left to right: MT output, and the advancing front method for $\rho = 0.5$ , using, respectively, Nielson interpolation and Moving Least Squares for reconstruction. . . . .	94
4.32	Extraction of material interfaces from a segmented volume. The outer surface mesh is removed to show the internal structures on the right. . . . .	96
4.33	Comparisons of a multimaterial interface extraction between the particle based method of Meyer et al. [100] (left) and the advancing front algorithm (right). Since the curvatures of the material boundaries are directly included in the guidance field, they are more faithfully reconstructed by the advancing front algorithm (a,b,c,d). The particle based method has also created a nonmanifold point where a surface passes close to itself (e,f). The true local feature size at this point is smaller than the grid used to implement the sizing field, resulting in edge sizes larger than expected. . . . .	97
4.34	An isosurface of a simulation of the Richtmyer-Meshkov instability. In these images, the shockwave passed from the top to the bottom. Marching Cubes is shown on the left, and the advancing front algorithm is shown on the right. Each image is a close-up of the region in the red box above it. . . . .	99
4.35	Several example point set surface triangulations are shown. The top row shows the input points. In each of the split views, the left shows the surface extracted with a nonlinear MLS surface definition, and the right shows a linear surface definition. . . . .	100
4.36	Sharp features can be preserved by starting initial fronts along them. The features that have been identified by the method of Daniels et al. [40] are used as input and are shown to the left of the output triangulation. . . . .	102
4.37	A point set composed of multiple laser range scans is used as the input to several meshing algorithms for comparison. There are more than 1.1 million points in the input dataset. . . . .	103
4.38	Several methods of generating meshes from the set of points in Figure 4.37 are shown: (a) a Delaunay based method [16], (b) a volume based method [38], (c) the advancing front algorithm applied to (b), and (d) the advancing front algorithm applied directly to the original point set. . . . .	104
4.39	The effects of $\rho$ and $\eta$ on the output mesh. $\rho$ controls the approximation accuracy: a bigger $\rho$ will result in a coarser triangulation. $\eta$ controls triangle grading: a larger $\eta$ will result in more adaptive triangulations. . . . .	107



A.1 The advancing front diagram is annotated with source code files and lines for implementations of the individual components. . . . . 112

## LIST OF TABLES

3.1 Comparison of octahedral remeshing using spherical parameterization ( $D \rightarrow S \rightarrow M^2$ ) [115], and using the direct map onto the octahedral domain $D$ . . . . .	38
4.1 Sample of results of guidance field culling. “before” refers to running time without culling, and “after” refers to running time after culling. There are more initial samples in the guidance field with smaller values of $\rho$ for isosurface extraction due to the adaptive sampling. The remesh examples have a constant number because only the input vertices are used. . . . .	64
4.2 Summary of results of the advancing front algorithm applied to remeshing. Error is measured as Hausdorff distance, in percent of the bounding box diagonal. Input and output size is measured in thousands of triangles. Quality is shown as a histogram of circle ratios. The three vertical bars show the worst triangle, the first half percentile, and the median. . . . .	88
4.3 A sample of results of isosurface extraction from regular grids, and comparison with MC and MT. The output size is measured in thousands of triangles. Dataset sizes: Aneurism, $256^3$ bytes; Silicium, $98 \times 34 \times 34$ bytes; Engine, $256 \times 128 \times 128$ bytes; Skull, $256 \times 256 \times 226$ bytes. . . . .	94
4.4 A sample of results of isosurface extraction from unstructured grids, and comparison with MC and MT. Dataset sizes: SPX, 13k tetrahedra; Torso, 1.1 million tetrahedra. . . . .	95
4.5 A sample of timing and quality results of triangulating point set surfaces. The input size is the number of points, and the output size is the number of triangles. The linear surface definition tends to produce many fewer triangles, and the running times are accordingly slower. However, the applicability of the linear definition is limited to points with normals. . . . .	101

## ACKNOWLEDGEMENTS

I would like to thank Cláudio Silva and Emil Praun for both serving as excellent advisors, always making themselves available, and guiding me to where I am today. I have also had many interesting and enlightening conversations with my other committee members: Mike Kirby, Valerio Pascucci, and Peter Shirley. I would especially like to thank Cláudio for his understanding and flexibility when I needed to drop everything and attend to personal issues, as well as continuing to support me after I took a job before graduating. I would also like to thank all of my committee members for their helpful comments to improve this dissertation.

While only a portion of the research that I have participated in appears in this dissertation, it has given me the opportunity to ponder about a wide variety of interesting problems. I would like to thank all of my collaborators: Erik Anderson, Arul Asirvatham, Steven Callahan, Hamish Carr, João Comba, Carlos Dietrich, Brian Duffy, Shachar Fleishman, Hugues Hoppe, Yuri Lima, Luciana Nedel, Tilo Ochotta, and Carlos Scheidegger.

I would like to thank my parents, Dennis and Claudia, for always supporting me and instilling curiosity and a desire to learn when I was young. I would especially like to thank Dad for showing me the kind of person I'd like to be, and Mom for being the strongest person I know. I would also like to thank my family in Florida, who always encouraged me to continue when times were tough, and especially aunt Colleen, for always doing absolutely everything she could when we needed it the most.

I would like to thank Christian Rössl, Andrew Anderson, Cyberware, the Stanford 3D Scanning Repository, the Digital Michelangelo Project, VCG\_ISTI, SensAble Technologies, and the AIM@SHAPE shape repository, and the Lawrence Livermore National Laboratory for the 3D models used in this dissertation. I would also like to thank Andrei Khodakovsky for sharing his Globally Smooth Parametrization data, and Miriah Meyer for sharing her multimaterial volume data.

Many pieces of software written by others have been made use of in this dissertation. I would like to thank the following: Jonathan Shewcheck for his Triangle library, used for creating constrained Delaunay triangulations; Hugues Hoppe for his mesh library, used as a basis for the intersurface mapping implementation, and his mesh format, which I have used ever since;

Emil Praun for his spherical parameterization code, providing a starting point for the intersurface mapping code; Michael Kazhdan for Reconstruct3D, used to generate examples; Szymon Rusinkiewicz for his trimesh2 library; and Carlos Scheidegger for his patches to it, used to convert mesh formats and create high-quality screenshots; Martin Isenburg and Peter Lindstrom for their streaming mesh library, used to save meshes in their streaming format and render them; Gordon Kindlmann for his volume file format and manipulation library teem; and Sundaresan Raman and Rephael Wenger for their isosurfacing library IJK, used for comparisons.

The research presented in this dissertation has been partially supported by the following funding sources: the Department of Energy SciDAC (VACET and SDM centers), the National Science Foundation (grants CCF-0401498, CCF-0528201, CNS-0514485, CNS-0551724, CNS-0751152, EIA-0323604, IIS-0513692, OCE-0424602, OISE-0405402), the Department of Energy, Sandia National Laboratories, Lawrence Livermore National Laboratory, an IBM Faculty Awards (2005, 2006, and 2007), and a University of Utah Seed Grant.

# CHAPTER 1

## INTRODUCTION

### 1.1 Motivation

Triangle meshes are ubiquitous as representations of geometric objects in many fields, including computer graphics, visualization, and finite element analysis. They are composed of a set of vertices, and a set of triangles connecting those vertices. This simple and explicit nature makes them attractive for many applications. They can easily be edited, compressed, simplified, and rendered with both scan-line and ray-tracing algorithms.

Digital geometry processing systems are often constructed from a series of individual operations applied in sequence [43, 60, 61, 87]. The efficiency and robustness of each geometric operation often depends directly on the quality of the mesh that it is given as input. Unfortunately, many geometric algorithms also tend to lower triangulation quality after each application. Additionally, certain operations require meshes with roughly the same triangle size and reasonable triangle shape to work well. Mesh-editing systems are the typical example: Boolean operations, such as union and intersection between meshes work much better when the triangle sizes between the meshes are similar. This is particularly true for the recently developed advanced mesh-editing techniques based on Laplacian coordinates [130] or on the Poisson equation [143]. These techniques allow automatic smooth stitching of two object parts. For these techniques to work well, it is necessary to match the resolution of the meshes before the computations.

The problem of improving the size and shape of the triangles in a mesh has driven much of the development of remeshing algorithms. However, many geometric processing applications are not well served by current remeshing techniques. Methods based on local mesh-improving operations (e.g., edge splits, edge collapses, and vertex repositioning) are practical, but need to be performed with care not to modify the overall geometric shape of the input mesh. Parameterization based methods often produce very uniform remeshes, with valence six vertices almost everywhere. This regularity is highly desirable for many geometry processing operations, but poor parameterizations with high distortion can also create poorly shaped triangles. Many of these methods require parameterization of the surface to a planar region, which limits the techniques to surfaces

homeomorphic to disks [52, 113, 138]. If more general surfaces are used, cutting and stitching are necessary, and the quality of the remesh tends to suffer near the seams of the parameterization. Global planar parameterization methods are also available, but contain degeneracies when the input is not genus-1 [59, 73].

Many existing parameterization methods are highly restricted by the topology of the input mesh. A general framework for creating parameterizations with low distortion and no degeneracies could be used to create remeshes for any input mesh topology would be valuable. One method of doing this is through intersurface maps: 1-1 correspondences between two input meshes. By mapping a surface to a simpler domain, represented by another mesh that can easily be resampled, high-quality remeshes can be created. In this way, the same framework could be used to remesh genus-0 surfaces (map to an octahedron), genus-1 surfaces (map to a torus), and higher genus surfaces (map to a simplicial domain). Most existing intersurface mapping methods either use an intermediate domain, increasing the distortion of the overall map, or are not robust for high-genus surfaces.

In addition to improving the quality of existing meshes, many problems can be avoided entirely by creating high-quality meshes directly from the original representations of surfaces. For example, isosurfaces of implicit functions, and point set surfaces are natural ways to represent surfaces resulting from medical and laser range scans. However, many geometric processing operations cannot be applied directly to these alternative representations, so they are often converted to triangle meshes. Existing methods for doing this often produce meshes with poor-quality triangle shapes and point distributions, which then must then be smoothed and simplified.

Implicit functions are a simple and effective technique for defining surfaces, with a wide range of applications in many scientific areas. An *isosurface*  $\mathcal{S}_a$  is defined as the *preimage* of a function  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$  and value  $a$ . The surface is the set of points in the domain that map to  $a$ , i.e.,  $\mathcal{S}_a = \{\mathbf{x} : f(\mathbf{x}) = a\}$  [36].  $f$  is the *implicit function* and  $a$  is the *isovalue*. The popularity of implicit surfaces comes from their representation power, solid theoretical foundation, and relative ease of manipulation [23].

Although a number of techniques have been proposed to solve the isosurface meshing problem, the Marching Cubes (MC) algorithm [92, 108] forms the basis for many widely used methods. MC works by sampling the implicit function at a grid of fixed resolution, and uses a table of possible configurations of range signs to create a triangulated surface from those samples. The main strengths of MC are its generality, simplicity and robustness, which have made it one of the most used meshing algorithms in practice. The main problem with MC is the inherent

bias caused by only placing vertices on grid edges which intersect the surface. This also implies that the sampling density is proportional to the grid resolution, and not to any intrinsic surface properties. Meshes generated by MC (and variants) are typically over-tessellated, and contain many bad triangles. The low quality makes them unsuitable for further geometry processing, and typically they require some form of postprocessing before being used in applications.

Point set surfaces are another surface representation that have become popular as laser range scanners become increasingly available. There are two frequently used approaches to generating triangle meshes from point sets. First are those that take the input points as the vertices of the output mesh, and find a set of triangles to connect them [13, 14, 16, 21]. This often creates dense meshes and poor-quality triangles, especially when there is noise in the input points. Another approach often used is to define an implicit function from the point samples [38, 77, 78]. The mesh can then be generated, for example, with MC. However, the deficiencies of the isosurface extraction method will then be apparent in the output mesh.

Applying remeshing techniques to the output of these alternative techniques can be challenging, since the meshes are often large, complex, and contain many geometric degeneracies. To make the meshes simpler, and overall of better quality, geometric simplification and smoothing steps are applied to them. Note that this is currently the preferred way to deal with MC meshes. Existing simplification techniques are very robust and scalable [56, 89]. However, a major issue with this approach is that the use of multiple processing steps makes it difficult to control the accuracy of the resulting mesh to the original surface.

Rather than creating poor-quality meshes from isosurfaces and point set surfaces and trying to improve them, a more elegant solution is to avoid the intermediate step and create high-quality meshes directly from the alternative surface representation. Such a method for generating meshes that does not depend on the input surface representation would provide a powerful tool that could be applied in a wide variety of situations.

## 1.2 Contributions

This dissertation presents two methods that can be used for meshing and remeshing surfaces. A parameterization method for forming a 1-1 correspondence between two meshes  $M^1$ ,  $M^2$  is presented in Chapter 3. The method never cuts either of the surfaces, directly optimizes the overall map, and is provably robust for arbitrary genus inputs. This mapping can be used to generate both fully regular and semiregular remeshes. Some parameterization schemes may require a large set of manually specified features to guide the parameterization process to a good (or even

valid) solution. As will be shown, the mapping method presented here is robust even with few feature constraints. Moreover, directly minimizing the distortion of the intersurface map tends to naturally align corresponding shape elements. Of course, a few user-specified constraints are helpful for overall registration and for linking semantically related regions.

The main contributions of this method are:

- The creation of an intersurface mapping without any intermediate domain, to directly measure the distortion of the overall map.
- A symmetric distortion metric, i.e., invariant to the interchange of  $M^1$  and  $M^2$ .
- Initialization of the map to robustly satisfy any user-specified feature correspondences.
- A symmetric coarse-to-fine optimization algorithm to provide robustness and convergence to a good solution.

While the fully general application is the creation of maps between surfaces of comparable complexity, this framework can also be used in cases where  $M^1$  is a simpler mesh, possibly inferred from  $M^2$ :

- Simplicial parameterization (for semiregular remeshing): given a surface  $M^2$  and desired domain vertices on  $M^2$ , automatically create domain  $M^1$  and a parameterization.
- Octahedral parameterization (for geometry-image remeshing):  $M^1$  is a regular octahedron, and feature points are unnecessary.
- Toroidal parameterization (for remeshing of genus-1 shapes).

These parameterization scenarios do not cut the mesh into charts, and enable high-quality remeshing.

An advancing front based approach to meshing arbitrary surfaces is presented in Chapter 4. This method builds a high-quality triangulation by directly resampling the geometry and topology of the input, as though it was performing surface reconstruction. This approach can be applied to surfaces with many different underlying definitions, including triangle meshes, isosurfaces, and point set surfaces. When applied to triangle meshes, it can be restricted to a local region of interest as in the case of editing operations between two meshes. The flexibility also enables extraction of gigantic out-of-core isosurfaces. The output meshes are optimally sampled in terms of triangle quality and Hausdorff distance between the input surface and the output mesh, while giving the user intuitive control over the allowed error.



This work is based on the technique of Scheidegger et al. [122]. The main idea of their advancing-front algorithm is to grow a triangulation over a point set surface using a guidance field that dictates the appropriate triangle size. They use a finite set of samples from the surface curvature that, when queried appropriately, tends to overcome the excessive locality of decision-making by advancing-front methods. This approach is extended in significant ways. First, a more principled way to construct the guidance field is presented, which can be used to prove that the induced edge sizing function is Lipschitz continuous. As a result, fewer and better triangles are generated. The algorithm is also significantly more robust. Additionally, the algorithm works on any surface for which curvature can be computed, and onto which points can be projected. This allows more flexibility when performing geometry processing operations. For example, operations can be performed between different types of surface representations, or users can perform mesh operations which result in a triangle soup (i.e., a mesh that may contain triangle flips, overlaps, holes, etc.), and then sample the affected region with points and simply triangulate that local region using a point set surface definition. Additionally, sharp features that are annotated on the input surface can be preserved.

The key contributions of this method are:

- A simple solution to a problem that is encountered by most implementations of mesh processing algorithms: the requirement of high-quality triangulations and control of their resolutions.
- Simple and intuitive user controls over the triangle size and quality, which also give control over the approximation error.
- A rigorous theoretical foundation for the guidance field that shows the conditions under which the output mesh accurately reflects the input surface.
- An efficient way to cull redundant information from the guidance field, improving memory usage and execution time.
- A fully generalized method that can be applied to many surface definitions.
- Adaptations to allow meshing of gigantic surfaces in an out-of-core manner.
- A set of detailed experimental results of remeshing, isosurface extraction from both regular and unstructured grids, and point set surface reconstruction, that shows the effectiveness of the approach.

- A prototype implementation that is available under the GPL software license.

The work presented in this dissertation has been published in peer-reviewed journals [123–125].

## CHAPTER 2

### BACKGROUND

This chapter provides context and background for the remainder of this dissertation by reviewing relevant related work. Section 2.1 reviews work in the area of surface parameterization, which is relevant for the intersurface mapping algorithm presented in Chapter 3. Section 2.2 reviews surface triangulation techniques for a variety of surface definitions, which are relevant for the advancing front algorithm presented in Chapter 4.

#### 2.1 Parameterization

Surface parameterization refers to mapping a triangle mesh onto a simpler domain such as the plane, the sphere, or a coarse simplicial domain. The parameterization is represented by a map  $\phi_{D \rightarrow M}$  where  $M$  is the mesh and  $D$  is the simpler domain. In computer graphics, parameterization is central to texture mapping, whereby images placed in the domain are sampled on rendered surfaces to provide texture detail, place decals, encode shadows, record radiance transfer coefficients, etc. Surface parameterizations also appear in numerous applications, including digital geometry processing, morphing, surface editing, object recognition. In particular, parameterizations enable the creation of highly regular remeshes. See [1] for an excellent survey of parameterization methods and applications.

##### 2.1.1 Planar Parameterization

The traditional surface parameterization problem considers the case where the domain  $D$  is a planar region  $P \subset \mathbb{R}^2$  (see survey in [54]). The map  $\phi_{D \rightarrow M}$  is represented by the parametric locations of vertices of  $M$  within the plane. Optimization can freely move the vertices within the domain as long as bijectivity is maintained.

There are many planar parameterization methods based on the work of Tutte [138], which are referred to as barycentric methods. These methods map a topological disk to the plane by formulating a linear system where the parametric location of each vertex is written as a convex combination of the locations of its neighbors, with a convex boundary fixed. The method

of choosing the convex combination weights greatly affects the quality and properties of the resulting parameterization. Tutte originally used uniform weights, which do not encode any of the geometry of the 3D mesh. Floater [52] proposed shape preserving weights to improve the quality. Cotangent [113] weights and Floater’s [53] mean value coordinates, which approximate conformal (angle preserving) maps, are also popular.

An important limitation of these barycentric methods is that parameterizing an entire surface requires that it be cut into one or more disk-like charts, where each chart is parametrized independently. Some techniques cut the surface into a single chart (e.g., [58, 126, 129]), while others cut it into an atlas of charts (e.g., [88, 93, 121]). In either case, the cuts tend to increase distortion and break the continuity of the parameterization. This makes it difficult to use a planar parameterization approach to construct a continuous map between two different surfaces since their cut structures will often differ as well.

More recently, attention has been focused on methods which allow global surface parameterizations of arbitrary genus surfaces without cuts. Gu and Yau [59] find a differential 1-form over the surface that can be integrated to form a parameterization. Jin et al. [73] use Ricci flow to deform a distance metric over the surface. Under the deformed metric, the surface has zero curvature, so it determines the lengths of the triangle edges in the plane. These distances can be used to flatten the mesh.

Most parameterization methods either do not allow constraints to be specified, or are not robust when they are. Kraevoy et al. [82] present the Matchmaker scheme for satisfying corresponding feature point constraints in  $D$  and  $M$ . Their method triangulates the feature points in  $D$ , while forming a corresponding partition of  $M$  into triangular patches: edges are created between feature vertices in  $D$  while paths are traced between the corresponding vertices in  $M$ . The interior of each patch is then mapped to its associated triangle in  $D$  to create the complete parameterization.

### 2.1.2 Spherical Parameterization

Letting the surface domain  $D$  be the unit sphere  $S$  allows one to directly parameterize a closed genus-zero surface without any cuts. Simple spherical parameterization methods do not take advantage of this, and use the plane as an intermediate domain. For example, Haker et al. [62] first create a planar conformal map, which is then mapped to the sphere using an inverse stereographic projection. Alexa [5] extends the planar barycentric coordinate based technique to the spherical domain. In this setting, the system is not linear and it is not supported by a convex boundary,

so iterated relaxation is performed and it is not robust. Gotsman et al. [57] rigorously generalize this adaptation of barycentric coordinates to the sphere and prove its correctness. This results in a system of quadratic equations, which Saba et al. [119] show how to solve in an efficient and stable way. Praun and Hoppe [115] take a different approach which inspires this intersurface mapping work. They first simplify the mesh to a tetrahedron, which is trivially mapped to the sphere. This tetrahedron is then refined while performing local optimizations to reduce the distortion.

### 2.1.3 Simplicial Parameterization

Another approach lets the domain  $D$  be a coarse base mesh. The surface  $M$  is partitioned into triangular regions that are mapped, respectively, to faces of  $D$  (e.g., [48, 61, 87]). This has the advantage of being able to choose the parametric domain  $D$  to match the genus of  $M$ .

The challenge in simplicial parameterization is that it is difficult to globally optimize the parameterization. Whereas planar and spherical domains are smooth everywhere, simplicial domains have sharp edges and vertices. Since the whole domain cannot be simultaneously “unfolded,” most methods iteratively apply a linear relaxation to a small group of adjacent faces. For example, Eck et al. [48] iteratively unfold a pair of adjacent domain faces and reparametrize the surface neighborhood over the resulting quadrilateral. Guskov et al. [61] perform local reparameterizations over 1-ring vertex neighborhoods, with the advantage that the images of domain vertices can shift over the surface.

Rather than iteratively optimizing local neighborhoods, Khodakovsky et al. [79] set up a global system where the mesh edges spanning adjacent domain faces are treated as if the two faces were locally unfolded into a plane. Solving the global system provides much faster convergence. Unfortunately, the domain vertices are fixed during the global system, and must be relaxed separately using traditional 1-ring relaxation.

### 2.1.4 Intersurface Mapping

An intersurface map refers to a bijection between two high resolution meshes  $M^1$  and  $M^2$ . The function  $\phi_{M^1 \rightarrow M^2}$  maps points on  $M^1$  to  $M^2$ , while  $\phi_{M^2 \rightarrow M^1}^{-1}$  maps points on  $M^2$  to  $M^1$ , and are treated equally. Since neither mesh need be considered the domain, it is often referred to as cross parameterization. The intersurface mapping problem could be viewed as an instance of simplicial parameterization where the domain  $D$  is an unusually complicated simplicial domain. However, existing simplicial parameterization techniques are not applicable, because:

1. They require an initial correspondence from all vertices of  $D$  to surface  $M$ , obtained by the construction of  $D$  from  $M$ .
2. Even with this initial correspondence, the techniques would converge too slowly due to the high complexity of  $D$ .
3. Simplicial parameterization techniques ignore the geometry of surface  $D$ , since they assume it to be an abstract domain.

Many intersurface mapping techniques use *intermediate* domains to construct the overall map. Lee et al. [86] create an intersurface map between two surfaces  $M^1, M^2$  by first constructing simplicial parameterizations  $\phi_{D^1 \rightarrow M^1}, \phi_{D^2 \rightarrow M^2}$ . Since the domain meshes  $D^1, D^2$  are different, user assistance is required to form a good map between them, and this map construction is not robust. Similarly, Fan et al. [49] define polycubes for each of the meshes to use as intermediate domains. They require the polycubes to be similar so a map can be created between them, and cannot satisfy feature correspondences.

To overcome this drawback, Praun et al. [116] develop a simplicial parameterization method in which the connectivity of the simplicial complex  $D$  can be specified a priori. Given a genus-0 simplicial complex and desired images of each domain vertex on multiple surfaces, they construct consistent parameterizations  $\phi_{D \rightarrow M^1}, \phi_{D \rightarrow M^2}$  over the shared simplicial domain  $D$ .

Both spherical parameterization and consistent simplicial parameterization can be used to create a continuous map between two surfaces  $M^1$  and  $M^2$  by forming the composition  $\phi_{M^1 \rightarrow M^2} = \phi_{D \rightarrow M^2} \circ \phi_{D \rightarrow M^1}^{-1}$  (where  $D$  is the sphere or simplicial domain, respectively). However, using an intermediate domain may result in a poor intersurface map, since each submap *ignores* the nonuniform distortion present in the other. For example, when creating a map between a cow and a horse, the cow legs would not be encouraged to match up with the horse legs. While it is possible to manually force correspondences of constraints on a dense set of domain vertices, a more elegant and flexible solution is to automatically *favor* this correspondence within the distortion metric itself.

Kraevoy and Sheffer [81] use the composition  $\phi_{D \rightarrow M^2} \circ \phi_{D \rightarrow M^1}^{-1}$  to remesh the surface  $M^2$  using the connectivity of  $M^1$  (together with some extra vertices where additional resolution is needed). They smooth the overall map using a spring relaxation where edge weights are related to local remesh error.

An alternative approach can be taken that parameterizes surfaces by a low-dimensional deformation from a canonical template. Joshi et al. [74] use the medial axis of an object to parameterize

its shape. Several of these parameterizations could be used to create correspondences between objects. Wiley et al. [139] use a deformation of the spatial domain to map objects onto each other. While these methods are more simple than generalized intersurface mapping methods, they do not allow optimization of the correspondences that they produce, and require that the input objects be geometrically quite similar.

Most intersurface mapping techniques require that the genus of the two meshes are the same. If they are not, it is not possible to create a bijection between them. Bennett et al. [20] create maps between meshes of different genus by first introducing cuts to remove handles that don't have a corresponding handle on the other mesh. This converts the problem of mapping meshes of different genus to mapping meshes with different sets of holes. They make use of variational implicit functions [137] to robustly handle these holes and the degeneracies they introduce in the map.

## 2.2 Surface Triangulation

The need to generate discrete representations of continuous geometry for computational, imaging and other purposes is widespread, and has generated substantial amount of literature in surface polygonization, meshing, and, more recently, remeshing.

### 2.2.1 Remeshing

The growing field of mesh processing has developed a number of advanced surface processing techniques, e.g., mesh editing [128, 143], mesh deformation [132], cloth simulation and compression. For these, and other applications, good triangle meshes are required since often the mesh is assumed to be a piecewise approximation of a smooth surface. As explained in the excellent survey of Alliez et al. [12], there is a need to improve *raw* meshes with oversampled and redundant geometry. This has given rise to the subfield of *remeshing*. Although no formal definition of remeshing exists, it is roughly the process of creating a mesh with certain *improved* properties from a preexisting mesh. Typically, the goal is to optimize sampling, grading, regularity, size, and shape of elements, while keeping the overall geometry of the model the same. See [12] for thorough coverage of existing remeshing techniques.

A number of remeshing techniques work by first computing a parameterization of the input geometry. Then, the surface is resampled in the parameter domain, and the samples are mapped back into 3D space. While parameterization based methods typically produce very regular remeshes (vertex valence almost always six), the specific choice of parameterization method can have a large effect on the quality of the triangles. The planar barycentric methods are

very simple, but often have significant distortion, particularly around the boundary. Additionally, unless the surface is homeomorphic to a disk, planar parameterization requires cutting of the surface, which introduces more distortion. Surazhsky and Gotsman [134] point out that such remeshing techniques are sensitive to the specific global parameterization used and can be slow. Additionally, robust implementation of these techniques is nontrivial, since numerical precision issues often arise from the distortion induced by the parameterizations. Instead, they advocate iteratively applying local optimizations directly on the existing mesh until some quality criteria are fulfilled. Simplicial parameterization methods can also be used to create remeshes [48, 61, 79]. These methods are typically more difficult to implement, but produce high-quality results because the parameter domain more accurately reflects the geometry of the surface, producing less distortion.

Another class of remeshing algorithms are those that distribute points across the mesh according to some distribution (usually uniform or curvature dependent), and then connect them to create a remesh. Turk [136] was one of the first to uniformly sample a mesh with a set of points that repulse each other. Alliez et al. [11] combine parameterization and point sampling by first creating a planar parameterization of the mesh. A halftoning technique is used to sample the parameterization proportional to the surface curvature to create an adaptive remesh. Followup work [10] additionally applies Lloyd relaxation [47, 91] to the halftoned samples to improve the quality of the sampling. Surazhsky et al. [133] further extend this idea to local overlapping parameterizations.

### 2.2.2 Isosurface Extraction

Since the introduction of the classical Marching Cubes algorithm [92], there has been an immense amount of work in the polygonization of isosurfaces. Isosurfaces are defined as the level sets of implicit functions, and are always smooth manifold surfaces under mild conditions on the implicit function. However, the original MC algorithm contained ambiguities, where some configurations could be interpreted in more than one way, possibly generating non-manifold meshes. Nielson and Hamann's asymptotic decider [108] was among the first to address the topic using bilinear interpolation to resolve face ambiguities, and now the issue of ambiguity in MC has been thoroughly examined [33, 103, 105].

There are many variants of MC. For instance, the Marching Tetrahedra (MT) [109, 135] class of algorithms works well for tetrahedral meshes, and does not have ambiguous cases. Dual methods generate the topological dual of the MC surfaces [107], by placing vertices inside the



grid cells rather than on the edges. Ju et al. [75] use the normals of the implicit function sampled on the edges to reconstruct sharp features in their dual method. Much effort has also been put into making MC more efficient, with particular emphasis on avoiding empty cells [34, 90, 140]. MC has also been adapted to ensure that the extracted mesh is isotopic to the underlying smooth isosurface [24, 114]. Some extensions modify the grid on which MC and MT operate. Dietrich et al. [45] let each edge of the grid float independent of each other. Figueiredo et al. [41] use a physically based method to deform a tetrahedral grid, resulting in higher quality triangles.

An alternative approach to the isosurface extraction problem samples the surface with a set of points which are then triangulated, similar to some remeshing methods. Witkin and Heckbert [141] sample isosurfaces with particles to visualize their evolution. Meyer et al. [98] improve the adaptivity and robustness of this technique. The points can then be triangulated to create high quality meshes [37, 99]. Hybrid approaches are also possible. For example, Wood et al. [142] first create a coarse mesh with the correct topology, then use a physically based repulsion solver to semiregularly tile the surface.

### 2.2.3 Point Set Surfaces

Point sets are becoming an increasingly popular way to define surfaces as laser range scanners become more affordable. Methods for creating triangle meshes from a set of points can be divided into two main categories: interpolating and approximating.

Interpolating methods use the input points as the vertices of the output mesh, and find a set of triangles to connect them. Bernardini et al. [21] conceptually roll a ball across the set of points, creating triangles between any three points that simultaneously touch the ball. There are many methods based on Delaunay tetrahedralization of the input points that assume that the input point set is sampling some underlying surface [13, 14]. Amenta et al. [15] prove that the triangulation is homeomorphic to the underlying surface under certain constraints on the sampling of the input points. The Power Crust algorithm [16] improves the robustness to always produce a valid triangulation. This has further been refined to better handle noise in the input points [96]. Dey and Giesen [44] adapt the technique to prevent triangulation of poorly sampled regions, allowing surfaces with boundaries to be triangulated.

Approximating methods do not use the input points as vertices of the output. Hoppe et al. [69] approximate a point set with a mesh by defining an objective function that considers the distance between the mesh and points, the triangle count, and the shape of the triangles. The function is minimized by adjusting vertex positions and performing edge flips, collapses, and splits. This

method has been extended to fit piecewise smooth subdivision surfaces to a set of unorganized points [67].

Many point set surface reconstruction techniques use the points to define an implicit function, from which an isosurface can be extracted (e.g., with MC). Hoppe et al. [68] define a signed distance function by estimating the tangent plane at each input point. The distance function is then the distance to the plane associated with the closest point. Curless and Levoy [38] reconstruct surfaces from multiple laser range scans with a similar method, but additionally taking advantage of the line of sight information implicit in the scans. Kazhdan et al. [78] use the point samples to construct the Fourier coefficients of the characteristic function (1 inside, 0 outside) of the surface. The characteristic function is then computed by the inverse Fourier transform. Kazhdan et al. [77] show how to cast this as a Poisson problem, which can be solved efficiently [26]. Radial basis functions can also be used to represent an implicit function that approximates the input points [28], though the fast evaluation methods are difficult to implement.

#### 2.2.4 Advancing Fronts

*Advancing fronts* refers to a class of meshing algorithms that begin with a *seed* point, and grow the triangulation across the surface. The *front* is the boundary of the mesh that has already been generated. It is *advanced* by choosing an edge in the front and creating a new triangle attached to it. If the new triangle causes a front to intersect itself, it is split into two separate fronts. If it intersects a different front, the two are merged. This process is continued until the entire surface has been covered.

Advancing front algorithms arose from early work on piecewise-linear continuation algorithms [8, 46], often used in numerical analysis to approximate manifolds. These early methods work by growing a set of tetrahedra that span the surface through a series of reflections. The surface is then be extracted from the tetrahedra using a MT like algorithm. There is little control over the quality of the triangles generated in this way, which led to the development of more sophisticated advancing front algorithms, where the output triangulation is directly grown over the surface.

There has been a substantial amount of work on using advancing front techniques for mesh generation [21, 127], and specifically for isosurface triangulation [22, 63, 65]. Noteworthy advances were made by Karkanis and Stewart [76], who adapt triangle sizes to the curvature of the input surface. This greatly reduces the size of the output mesh, but introduces a new issue of creating high quality triangles. In particular, the work presented here is inspired by the

advancing front algorithm of Scheidegger et al. [122], who extend the work of Karkanis and Stewart. Their key innovation was the use of a global guidance field (also called a sizing field in other works [9, 25]) to restrict how fast edge sizes can change, and avoid missing features of the underlying surface. Even though this was not done rigorously, it greatly improves the robustness of the advancing front algorithm, and improves the quality of the triangles generated.

## CHAPTER 3

# UNIFORM REMESHING: INTERSURFACE MAPPING

This chapter addresses the problem of directly constructing a continuous bijective map  $\phi_{M^1 \rightarrow M^2}$  between two triangle meshes  $M^1$  and  $M^2$  of the same topology. (Continuity precludes maps between surfaces with different genus or number of boundaries.) Unlike previous approaches which compose parameterizations of  $M^1$  and  $M^2$  over some intermediate domain (as reviewed in Section 2.1), the quality of the overall map  $\phi_{M^1 \rightarrow M^2}$  is directly optimized. The method works for arbitrary genus and does not require the user to provide a simplicial complex (e.g., [116]). The user may optionally specify corresponding feature points on  $M^1$  and  $M^2$ , and the construction guarantees that the map satisfies these constraints. While triangle meshes are often used to approximate smooth underlying surfaces, the input meshes  $M^1$  and  $M^2$  are taken to be the actual surfaces. This allows precise control over the correspondence and direct computation of the distortion of the map.

This chapter is organized as follows. Section 3.1 presents an overview of the intersurface mapping algorithm, and the representation of the map is discussed in Section 3.2. The algorithm for initializing and coarse-to-fine optimization of the map are presented in Sections 3.3 and 3.4. Applications of the intersurface map are presented in Section 3.5 and the results are discussed in Section 3.6.

### 3.1 Algorithm Overview

The strategy is to use progressive refinement to robustly create and optimize the intersurface map. Even for planar and spherical parameterizations, which involve smooth domains, coarse-to-fine approaches help parameterizations converge to good solutions [4, 71, 120]. For intersurface maps, the lack of domain smoothness exacerbates the problem of falling into local minima, further motivating progressive refinement.

The method first constructs progressive mesh (PM) representations of both  $M^1$  and  $M^2$  [66]. To simplify the task of initializing the intersurface map (and in fact make this task trivial), the

two progressive meshes are constrained to have base meshes with identical connectivities. To satisfy user-specified correspondences, feature points are retained as vertices in the base meshes. Consequently the algorithm becomes provably robust. A trivial valid map is created initially, and the refinement operations always succeed, so that by induction a valid map between the fully refined surfaces is guaranteed.

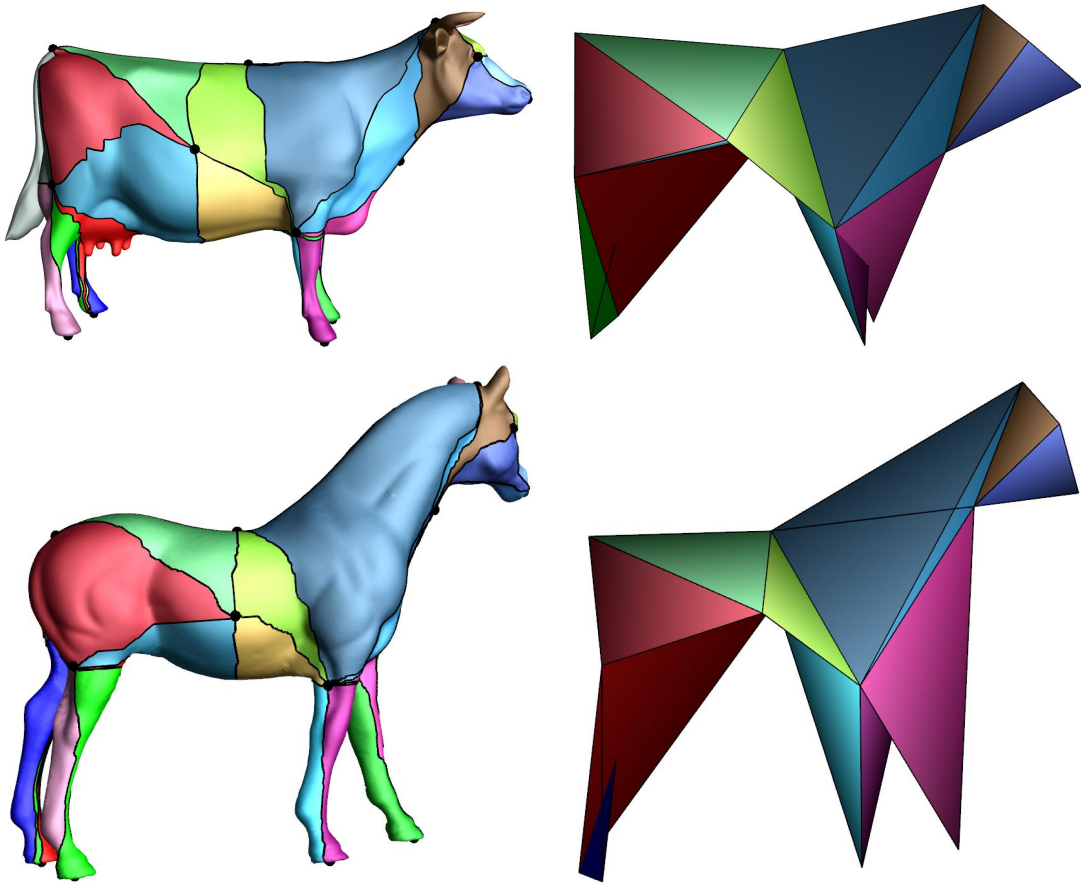
The basic steps of the algorithm are:

1. Partition the surfaces  $M^1$  and  $M^2$  into a corresponding set of triangular patches, by tracing a set of corresponding paths. If user-specified features are provided, these are chosen as path endpoints. (Section 3.3)
2. Create progressive mesh representations of both  $M^1$  and  $M^2$ , using the path networks to constrain the simplifications, resulting in two base meshes with identical connectivities.
3. Establish a trivial map between the two base meshes: a 1-to-1 map on vertices, with no edge-to-edge intersections.
4. Iteratively refine the two progressive meshes. After each vertex split, update the intersurface map and optimize it in the local neighborhood. When both meshes are fully refined, the intersurface map is obtained. (Section 3.4)

Steps 1 and 4 are the most challenging, and are presented in more detail in the next two sections. To create the progressive meshes in Step 2, the edge collapse sequences are constrained to preserve the topology of the paths, as described by Sander et al. [121]. Base domains are thus obtained whose edges correspond to original paths and whose triangles correspond to original patches (see Figure 3.1). Since the two base domains have the same connectivity, the construction of the initial map in Step 3 is trivial.

## 3.2 Map Representation

The goal is to produce a piecewise-linear map between two triangulated surfaces. Unlike in planar parameterization, the linear pieces of the map are finer than the original mesh faces, as they correspond to triangles of a *mutual tessellation* [136] (a.k.a. *meta-mesh* [86]) of the two surfaces. Vertices of this meta-mesh include the vertices of both initial meshes as well as vertices formed by edges of  $M^1$  intersecting those of  $M^2$ . To fully specify the map, for each mesh vertex the face of the other mesh to which it maps is recorded, along with barycentric coordinates within that face, and for each edge-edge intersection, the two ratios formed by the split point on



**Figure 3.1:** Example of the consistent partitioning process. Given a set of corresponding feature points, the meshes  $M^1$  and  $M^2$  are partitioned into corresponding patches (left). Each mesh is then simplified while preserving the patch boundaries, which produces base meshes with corresponding vertices and identical connectivity.

each edge are recorded. Together, the vertex and edge-edge barycentric coordinates define a set of corresponding polygonal subregions on faces of  $M^1$  and  $M^2$ . Further triangulation of these polygonal regions in a prescribed manner defines a unique piecewise-linear map.

On rare occasions, it is necessary to “bend” the image of an edge of  $M^1$  inside a triangle of  $M^2$  and vice versa. This is achieved by introducing special *kink vertices* of valence 2 in mesh  $M^i$  (Section 3.4.3). These vertices have a corresponding face and barycentric coordinates in the other mesh just like regular vertices.

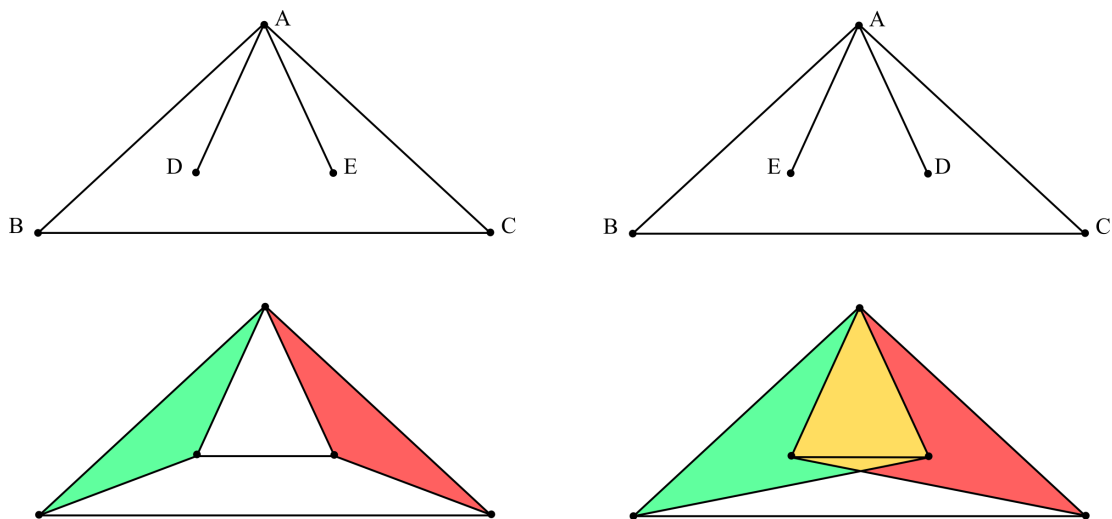
### 3.3 Initialization of the Coarse Map

The first step is to form a consistent partitioning of meshes  $M^1$  and  $M^2$  into corresponding triangular patches. The patch boundaries are defined by path networks linking together feature

vertices. These feature vertices are optionally specified by the user. If their number is insufficient for the given surface genus (e.g., at least 4 features for genus 0, more for higher genus), additional pairs are automatically inserted. The path connectivity can be either specified (as in [116]) or arbitrary (as in [82]), depending on the application scenario.

To guarantee the successful termination of the path insertion process, ordering constraints are imposed on the neighbors of a feature vertex, and a spanning tree and  $2g$  nonseparating cycles are traced before completing the full graph. Consistent neighbor ordering is necessary to avoid partial graphs that are impossible to complete, as shown in Figure 3.2 (if  $D$  and  $E$  link to the same base vertex  $B$  or  $C$ , this will result in flipped triangles; if they link to different ones, edges will cross).

The approach is to link together corresponding feature pairs on both meshes using constrained shortest paths, similarly in spirit to the methods of Praun et al. [116] and Kraevoy et al. [82]. Paths are added in a greedy fashion, subject to constraints that ensure consistent topology, and using heuristics that avoid swirls. When a maximal graph of noncrossing paths has been created, the two surfaces have been partitioned into triangular patches.



**Figure 3.2:** Vertex ordering must be consistent when creating a consistent partitioning. If the ordering is inconsistent, it will be impossible to create a bijection. For example, intersecting paths will be required, and the triangles  $ABD$  and  $ACE$  on the left will overlap in the area shown in yellow on the right.

### 3.3.1 Path Tracing

The shortest path between a pair of feature vertices is traced using a Dijkstra shortest path search. The search is constrained to not intersect with paths already in the network. To obtain path networks with consistent topologies between the two meshes, consistent ordering of the neighbors of each vertex must be maintained. Therefore an additional constraint on the paths is to start and end in corresponding sectors on the two meshes. (The meshes are assumed to be orientable.) When the shortest paths on each mesh are not consistent, two candidate pairs of paths are traced while imposing the sectors from  $M^1$  on the path on  $M^2$ , and vice versa, and then pick the best pair.

To allow the creation of a valid path between any pair of features, extra Steiner vertices are lazily added to the meshes, as suggested by Kraevoy et al. [82]. The Dijkstra searches are performed on both the mesh vertices and the edge midpoints. Since using edge midpoints in a path corresponds to adding Steiner vertices, preference is given to paths that do not use them. This may lead to slightly more jagged paths, but the precise geometry of the paths is not critical to the final map, since the paths are not constraints — they only guide the construction of compatible PM sequences.

The greedy path-insertion algorithm selects the best pair of corresponding paths from a priority queue sorted by the sum of path lengths on  $M^1$  and  $M^2$ . The queue is initially populated by tracing paths from each vertex to its 10 closest neighbors. When the best candidate is selected, it is checked whether it is still valid, and if not is recomputed and inserted back into the queue.

To guarantee the success of the algorithm, enclosing any vertex within a path cycle not connected to it must be avoided. Praun et al. [116] observe that for genus-0 surfaces it is sufficient to first build a spanning tree of the feature vertices (before forming any cycles). This approach can be generalized to arbitrary genus. To this end, separating and nonseparating cycles formed by the paths must be distinguished. (A separating cycle is one that breaks the surface into two disjoint components.) The strategy is to first build a maximal path network without separating cycles, before adding any paths forming separating cycles.

For a surface of genus  $g$  with  $k$  feature vertices, the maximal nonseparating graph is the union of a tree spanning all feature points and  $2g$  nonseparating cycles, and thus has exactly  $k - 1 + 2g$  paths. This maximal nonseparating graph topologically cuts the surface into a disc [58], with all the sectors around feature vertices as vertices on the boundary of the disc. The neighbor ordering constraint ensures that the ordering of the disc vertices is the same for both  $M^1$  and  $M^2$ . In such a configuration, there always exists a unique way to link any two vertices (sectors adjacent to a



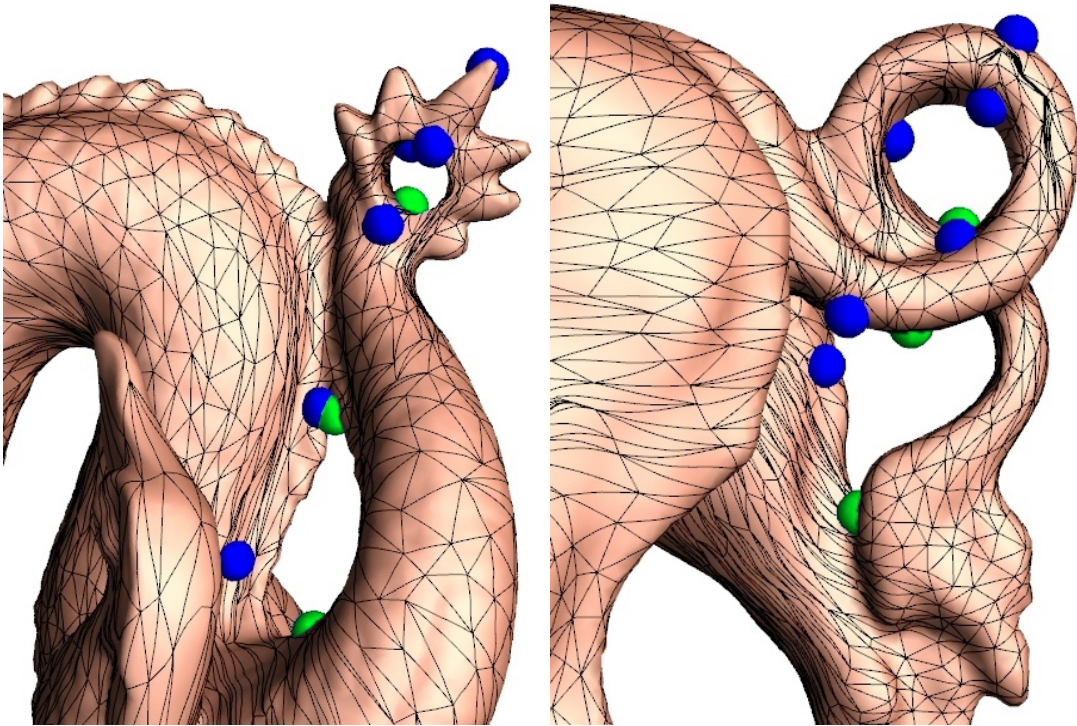
feature). Once such a path is added, each of the two topological discs representing  $M^1$  and  $M^2$  is further split into two discs. This recursive decomposition is continued until each disc has only three feature vertices on its boundary.

The new path that needs to be added to split the discs may sometimes link two features that are already connected (by a path in different sectors, going across a handle of the objects). In such cases additional feature points are automatically introduced to support the new path. As an example, for the pair of genus-2 surfaces in Figure 3.3, eight features are specified by the user, and seven additional ones are automatically introduced.

There are two issues related to building the maximal nonseparating graph: avoiding separating cycles, and avoiding swirls.

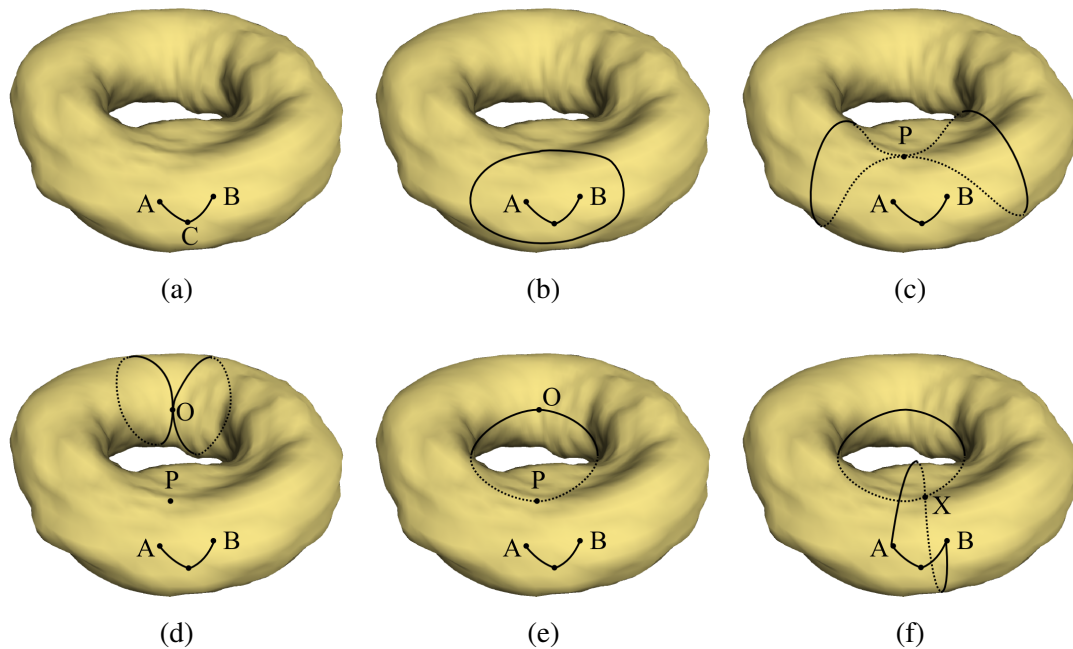
### 3.3.2 Avoiding Separating Cycles

If a newly introduced path between vertices A and B forms a cycle, it is tested whether it is separating, and if so, the path is replaced with one forming a nonseparating cycle using



**Figure 3.3:** If the user does not define enough feature correspondences (green) to resolve the genus of the object, additional correspondences are automatically inserted (blue). These extra correspondences are only used to initialize the map, and are not maintained as hard constraints throughout the optimization.

an algorithm similar to that of Lazarus et al. [84]. Specifically, two simultaneous breadth-first searches are performed starting from the vertices incident to the path  $AB$ , on its two sides. The searches are constrained by the existing path network and by the candidate path  $AB$ . Each visited vertex is tagged with its parent (the vertex visited previously to get to it) and with the left/right side of  $AB$  it connects to. If a “left” vertex is ever reached from a “right” tagged one, then the cycle is nonseparating and is added to the path network. Otherwise, a new nonseparating cycle is formed as follows (see Figure 3.4). The boundary of the region visited in the search at a certain time is in general composed of several contours that can subsequently split, merge, or contract to a point. When contours merge (say at a point  $O$ ), two paths are traced back to the previous split event  $P$ , using the “parent” fields. From this nonseparating cycle between  $P$  and  $O$  the vertex  $X$  closest to  $A$  and  $B$  is selected. Distances are measured by tracing paths  $XA$ ,  $XB$  that (1) do not cross the cycle at points other than  $X$ , (2) meet the cycle from opposite sides, and (3) end at  $A$  and  $B$  on the same side of the temporary  $AB$  path. The path  $AX$ – $XB$  forms the final path.



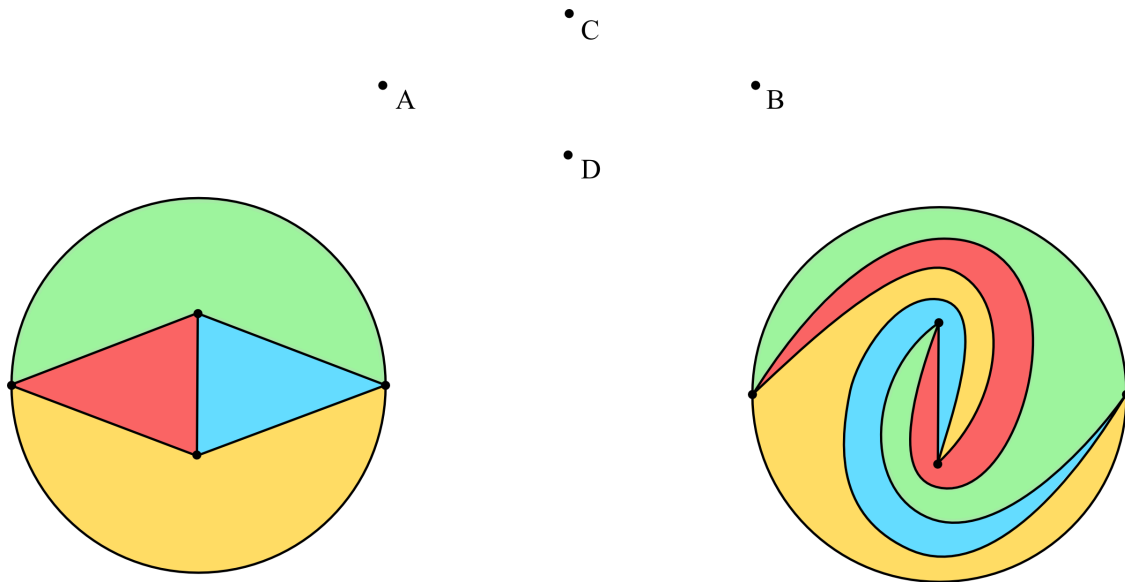
**Figure 3.4:** If a direct connection between  $A$  and  $B$  would create a separating cycle, a nonseparating cycle is created instead. A breadth first search starting from the path  $AB$  creates an expanding front (b). This front splits at a point  $P$  (c), and merges back together at a point  $O$  (d). The “parent” fields trace two paths that create a nonseparating cycle  $P$  and  $O$  (e). A nonseparating cycle  $AX$ – $XB$  is inserted into the path network.

If there are not enough user-provided features to resolve the genus of the object, non-separating cycles are traced connecting to one of the existing features using a procedure similar to the one above (with  $A=B$  and no “left”/“right” tags). A pair of corresponding feature constraints are then automatically created at the X point on each mesh to support the cycle.

### 3.3.3 Avoiding Swirls

A *swirl* is an awkward geometric configuration in which paths between feature vertices take unnecessarily long routes around other existing paths (see Figure 3.5). More precisely, the presence of corresponding feature constraints establishes homotopy classes on the set of intersurface maps. Two maps belong to the same class if there exists a continuous deformation between them that maintains the constraints. Since swirls correspond to “poor” homotopy classes, they cannot be fixed using local continuous relaxation [116]. Two heuristics have been found to be effective at avoiding swirls.

The first heuristic is to prefer early connection of feature points at mesh extremities. To identify mesh extremities, the average distance from each feature vertex to the closest set of 8 neighboring features is computed. Vertices with a high distance (top 25%) are considered extrema.



**Figure 3.5:** There are multiple ways to connect set of points A, B, C, and D. Both ways shown here have the same path ordering around each vertex (e.g., CA, CD, CB appear in the same order in both cases). The paths on the right can only be created by rotating clockwise the points C and D around each other. Hence the right configuration is a *swirl*.

The second heuristic is to delay insertion of paths that pass on the “wrong side” of neighboring features [116], and when forced to choose such a path, to reroute it on the correct side. For each candidate path, a set of neighboring feature vertices is gathered (the  $k$ -nearest neighbors of the two endpoints on the two meshes). For each of these neighbors it is determined on which side of the path it lies by computing the side on which the shortest route from the neighbor to the path meets the path. If the side is different between the two meshes, then the path is likely to cause a swirl, so it is penalized in the pool of candidate paths. If only penalized paths are left, the lowest-cost path is re-routed to the correct side of the offending neighbor vertex as follows. Shortest paths between the offending neighbor and the candidate path endpoints are computed (under normal constraints) and temporarily added to the path network. The new path is thus forced to go on the correct side of the connected component of the offending neighbor.

### 3.3.4 Boundaries and Features

The map initialization algorithm is easily extended to meshes with boundaries. Each boundary contour is triangulated using a single central point. The point is treated as a feature vertex, and must be associated with a corresponding boundary-center vertex on the other mesh. Once the two path networks are computed, these boundary-center vertices are removed along with the faces used to triangulate the boundaries. The paths connecting to the boundary centers are clipped to the boundary, and these clip points become new feature vertices. The resulting nontriangular patches are then consistently triangulated, and the remaining steps proceed as before.

User specified path constraints can also be accommodated. The user can trace corresponding paths on the two surfaces that will be constrained to map to each other. Each path must either terminate at feature points, or it is a cycle and temporary correspondences can be automatically inserted along the path. These paths are then inserted into the network before any other paths. If they form cycles, care must be taken to ensure that they are either both nonseparating, or they do not partition the two meshes into topologically different regions. In this case, they split the partitioning process into two independent partitioning problems.

## 3.4 Coarse-to-Fine Optimization

Like previous work (e.g., [61, 121]), the map is optimized by moving one vertex at a time within its one-ring neighborhood to decrease the distortion metric. During the refinement of the progressive meshes, this optimization is performed after each vertex split for the new vertex and each of its neighbors, and for all mesh vertices when their total number has increased by a factor of 1.5.

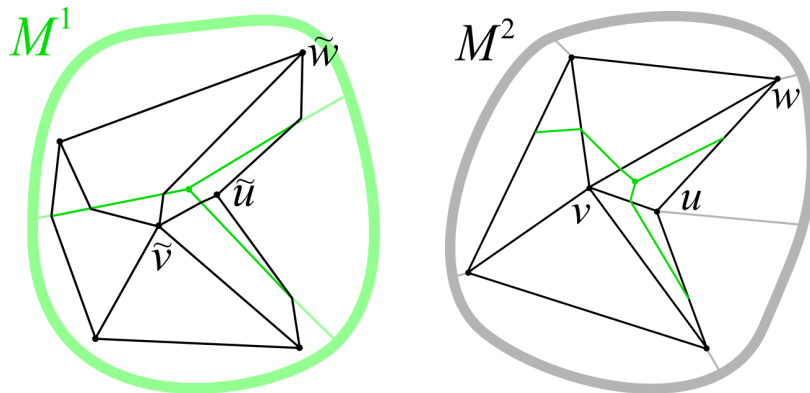
Unlike previous methods, the optimization is performed for vertex neighborhoods not just of  $M^1$  but also of  $M^2$ . This convenient symmetry is necessary since neither mesh is a special “domain.” It also provides finer grain optimization than previous simplicial parameterization methods.

In the current implementation, only  $M^2$  is refined for a number of steps, while  $M^1$  is held at constant resolution. Their roles are then swapped and  $M^1$  is refined, and then the process is repeated. Keeping track of only one refining mesh at a time while the other is static results in lighter-weight data structures and more manageable code. For the scenarios where one of the meshes is very simple (octahedral and simplicial parameterizations, where each vertex of the “domain” mesh has a specified feature correspondence), the swaps are unnecessary.

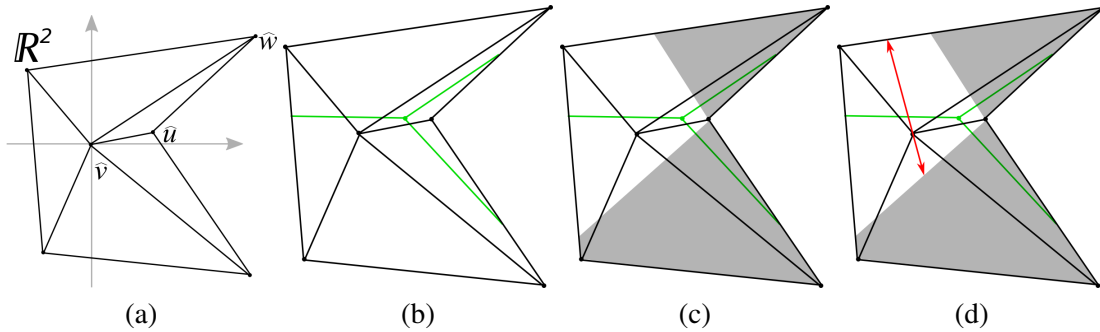
### 3.4.1 Vertex Optimization

The main operation considers a vertex  $v$  of  $M^2$  and optimizes its location  $\tilde{v}$  on  $M^1$ . Let  $\mathcal{N}(v)$  be the 1-ring neighborhood of  $v$  in  $M^2$ , and  $\mathcal{N}(\tilde{v})$  be the preimage of this neighborhood in  $M^1$  under the intersurface map (Figure 3.6). The optimization only modifies the map inside these corresponding neighborhoods, i.e., by regenerating barycentric coordinates for all meta-mesh vertices within the interior. Therefore the change in overall distortion can be exactly computed.

To perform the relaxation, a temporary 2D parameterization of the neighborhood  $\mathcal{N}(v)$  onto a planar polygon  $\mathcal{N}(\hat{v})$  is constructed as follows (see Figure 3.7). A one-ring unfolding is used, where  $v$  is initially mapped to the origin  $\hat{v} = (0,0)$ , each neighbor  $w$  of  $v$  is mapped to a point  $\hat{w}$  at a radius equal to the path length  $\tilde{v}\tilde{w}$ , and the angle  $\angle \hat{u}\hat{v}\hat{w}$  between successive neighbors  $\hat{u}$ ,  $\hat{w}$  is proportional to  $\angle \tilde{u}\tilde{v}\tilde{w}$  (scaled such that their sum equals  $2\pi$ ). The angle  $\angle \tilde{u}\tilde{v}\tilde{w}$  on  $M^1$  is computed



**Figure 3.6:** The configuration of the neighborhood of a vertex  $v$  before optimization. Mesh  $M^1$  is shown in green and mesh  $M^2$  is shown in black.



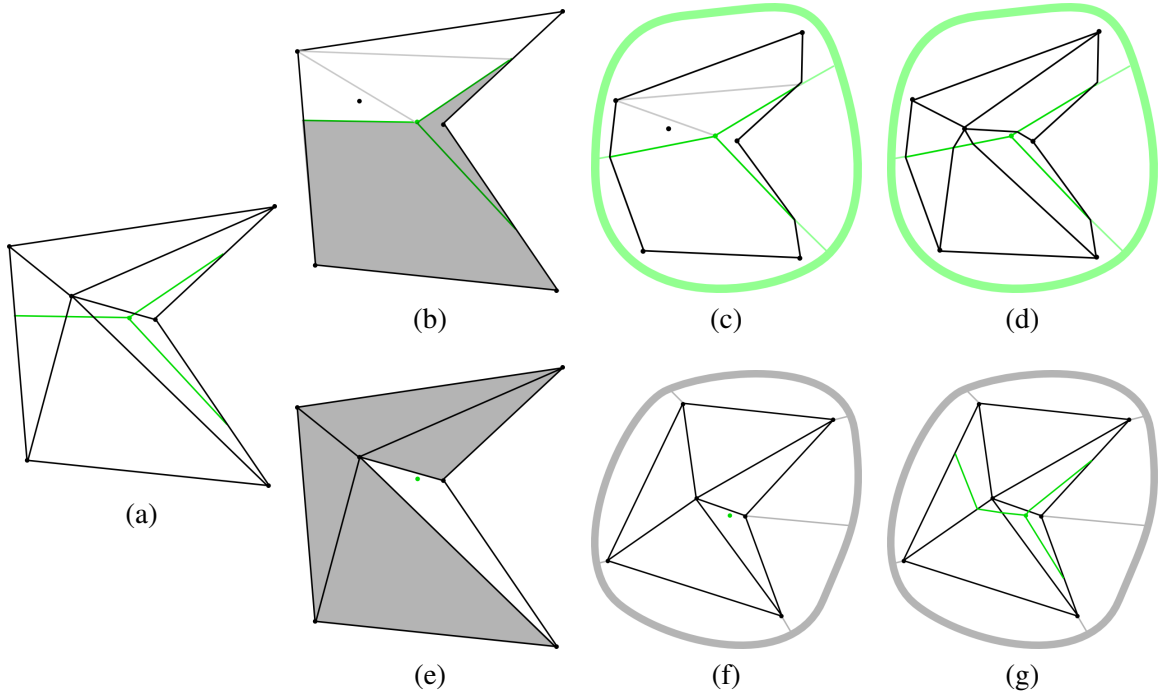
**Figure 3.7:** A 2D parameterization of the neighborhood is used to facilitate the optimization. First, the 1-ring neighborhood of the vertex of  $M^2$  being optimized is unfolded (a). The corresponding region of  $M^1$  is then mapped to the 2D parameterization (b). The kernel of the boundary (c) is used to perform multiple line searches (d) to minimize the distortion.

using the law of cosines applied to the path lengths  $\tilde{u}\tilde{v}$ ,  $\tilde{v}\tilde{w}$ , and  $\tilde{w}\tilde{u}$  (or to the respective Euclidean distances between endpoints if the path lengths do not obey the triangle inequality). Importantly, when  $\mathcal{N}(\tilde{v})$  is entirely contained inside a single face of  $M^1$ , the map from  $\mathcal{N}(\tilde{v})$  to  $\mathcal{N}(\hat{v})$  is an isometry.

Once the 2D parameterization of  $M^2$  has been created, the region of  $M^1$  within  $\mathcal{N}(\tilde{v})$  is mapped to  $\mathcal{N}(\hat{v})$ . Since the edge-edge crossings will be updated during the optimization, the current crossings in the interior of the neighborhood are removed. The mean-value parameterization scheme of Floater [53] is then used to relax the 2D locations inside  $\mathcal{N}(\hat{v})$  of the vertices of  $M^1$  contained within  $\mathcal{N}(\tilde{v})$ . The edge crossings of  $M^1$  and the boundary of the neighborhood are used as constraints during this relaxation. Since the boundary of  $\mathcal{N}(\hat{v})$  can be concave, some interior pieces can be nonconvex, or flips can occur. In those rare cases, the boundary is remapped to a convex circle-inscribed polygon [61], and the relaxation is repeated, this time guaranteeing no folds. The result of this mapping is a 2D parameterization of the local neighborhood of the vertex being optimized where edges of both meshes are straight (Figure 3.7b).

The intersurface map is then optimized by searching for a new location for  $\hat{v}$  in the 2D parameterization. This search is restricted to the kernel of the boundary of  $\mathcal{N}(\hat{v})$  (Figure 3.7c) to prevent flips in the parameterization, thus preserving the bijectivity of the inter-surface map. Since the energy is nonlinear, it is minimized through repeated line searches within the kernel (Figure 3.7d) as in [121].

For each potential location of  $\hat{v}$ , the intersurface map within  $\mathcal{N}(\hat{v})$  must be reconstructed to evaluate the distortion (see Figure 3.8). First, the image of  $\hat{v}$  on  $M^1$  must be found. This is done by



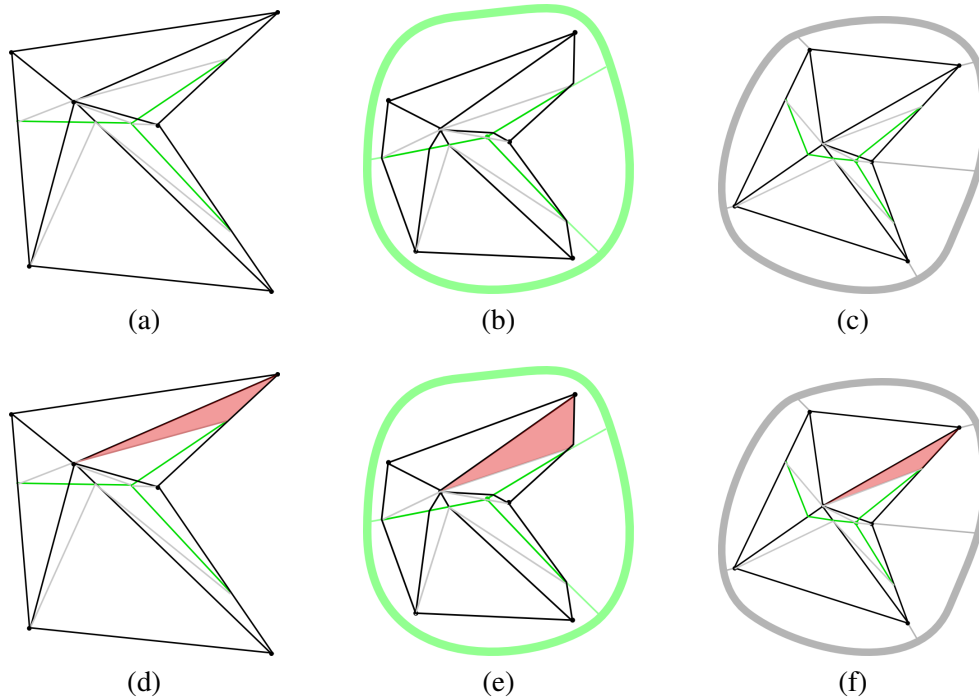
**Figure 3.8:** Once a new vertex location is found that minimizes the distortion in the 2D parameterization (a), the meta-mesh must be updated. First, the region now containing  $\hat{v}$  is triangulated (b). The same triangulation is applied to  $M^1$ , and barycentric coordinates are used to find the new location of  $\tilde{v}$  on  $M^1$  (c). Similarly, the split ratios of the edges of  $M^2$  within the 2D neighborhood are found, and mapped to  $M^1$  (d). For each vertex of  $M^1$  lying in the neighborhood, the face of  $M^1$  containing it is found (e), and barycentric coordinates are used to map it back to  $M^2$  (f). The meta-mesh update is completed by mapping the split ratios of the edges of  $M^1$  back to  $M^2$  (g).

finding which polygon of the 2D parameterization of  $M^1$   $\hat{v}$  lies in (Figure 3.8b). This polygon may be nontriangular when it is on the boundary of  $\mathcal{N}(\hat{v})$ , which corresponds to a subset of a triangle of  $M^1$ . If this is the case, Constrained Delaunay Triangulation (CDT) is applied to the polygon. The new location of  $\tilde{v}$  on  $M^1$  is then found through the barycentric coordinates of the triangular piece that  $\hat{v}$  is in (Figure 3.8c). The new edge-edge intersection ratios are then computed in the 2D parameterization and give the images of  $M^2$  as they appear on  $M^1$  (Figure 3.8d). Similarly, each vertex of  $M^1$  in  $\mathcal{N}(\tilde{v})$  must be updated. The triangle of  $M^1$  containing each vertex in the 2D parameterization is found (Figure 3.8e). Since  $\mathcal{N}(\hat{v})$  contains a complete 1-ring neighborhood of  $v$ , it will always be a full triangle of  $M^2$ . The vertex of  $M^1$  is mapped to  $M^2$  through the barycentric coordinates within this triangle (Figure 3.8f). Finally, the edge split ratios are used to complete the intersurface map (Figure 3.8g).

After the intersurface map is reconstructed for a potential new location of  $\hat{v}$ , the distortion of the map over the neighborhood is evaluated. CDT is applied to the nontriangular regions of  $\mathcal{N}(\hat{v})$  (see Figure 3.9). This triangulation is then applied to both  $M^1$  and  $M^2$ . Each resulting triangle in the 2D parameterization then has an associated pair of corresponding triangles on each mesh, representing the piecewise linear map. The total distortion for the neighborhood is then simply the sum of the distortions over each triangle. The location of  $\hat{v}$  that achieves lowest distortion is retained. Note that due to the initial relaxation and deletion of edge-edge crossings when constructing the neighborhood  $\mathcal{N}(\hat{v})$ , the final distortion may be larger than that before the optimization. In this case, the whole operation is discarded.

### 3.4.2 Boundaries and Features

As with the consistent partitioning process, both point and path constraints can easily be accommodated within this framework. When the user specifies these constraints, the consistent partitioning process ensures that they appear in the coarse base meshes. This ensures that the con-



**Figure 3.9:** To evaluate the distortion, the 2D parameterization of the neighborhood is triangulated, which is then applied to the neighborhoods on both  $M^1$  and  $M^2$  (top). This gives a common triangulation on both of the meshes, which are used to evaluate the distortion. One such corresponding triangle is highlighted (bottom).



straints are satisfied on the coarsest level. During the refinement, the points that are constrained are never optimized to prevent the correspondence from being broken.

For path constraints, more care must be taken in the optimization to preserve them. Consider a vertex  $v$  on  $M^2$  that is constrained to lie on a path of edges of  $M^1$ . The vertex  $v$  must have two neighbors that are constrained to the same path (otherwise it is a path endpoint, in which case it is not optimized). When creating the 2D parameterization of  $\mathcal{N}(v)$ , it is ensured that these three vertices form a straight line. When mapping the vertices of  $M^2$  that lie within  $\mathcal{N}(\tilde{v})$  to  $\mathcal{N}(\hat{v})$ , it is similarly ensured that the vertices along the path are mapped to the same line. Any location along this line for  $\hat{v}$  will then maintain the correspondence between the paths. A single line search is done in this direction to minimize the distortion while maintaining the constraint.

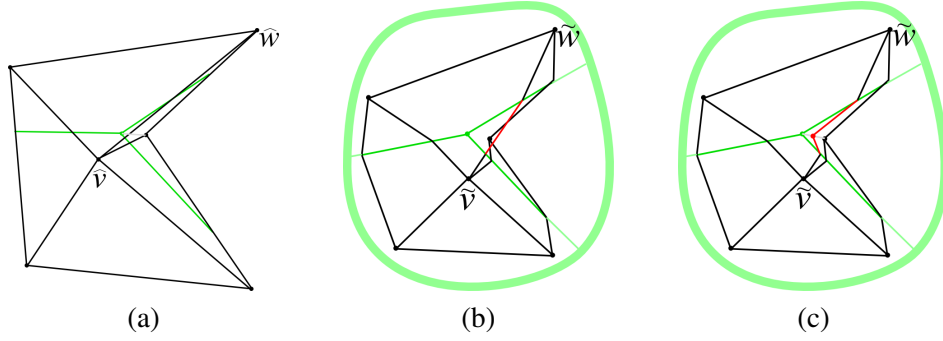
The consistent partitioning process handles boundaries by creating a set of corresponding feature points along them, and joining them with paths along the boundaries. During the vertex optimization, these paths are treated similar to any other path constraints. The 2D parameterization during the optimization is forced to map the path to a straight line, but rather than the line running through the center of  $\mathcal{N}(\hat{v})$ , it runs along the edge. Again, the optimization performs only a single line search along this line.

### 3.4.3 Kink Vertices

Just as Steiner vertices are sometimes necessary to create a valid bijection, in rare cases the image of an edge of  $M^2$  on  $M^1$  must be “kinked” by breaking it at points other than intersections with edges of  $M^1$  (see Figure 3.10). After optimizing the 2D location  $\hat{v}$  of a vertex, its incident edges must be mapped back to  $M^1$ . An edge  $\hat{v}\hat{w}$  is mapped to a path  $\tilde{v}\tilde{w}$  by finding its intersections with pieces of  $\mathcal{N}(\hat{v})$  in 2D, and mapping these intersection points to  $M^1$  using the split ratios on their supporting segments. Since the pieces of  $\mathcal{N}(\tilde{v})$  on  $M^1$  may have concave vertices, the straight-line segment between the two mapped intersection points may not be contained inside the piece. In these rare cases, the CDT diagonals of the concave piece are used to support additional break points in the path  $\tilde{v}\tilde{w}$ . These kinks are represented as temporary vertices of  $M^2$  with valence 2, and are removed when next optimizing  $\tilde{v}$  or  $\tilde{w}$ . (When swapping  $M^2$  and  $M^1$ , one of these optimizations is forced, to remove the kinks.)

### 3.4.4 Distortion Metric

Many parameterization distortion measures have been proposed, including angle-preservation (conformal map) [42, 48, 70, 88], area-preservation (authalic map) [42], and stretch minimization [121]. Often, these metrics can be expressed in terms of the singular values  $\Gamma, \gamma$  of the map



**Figure 3.10:** A given location for  $\hat{v}$  (a) will sometimes create a fold in the intersurface map because a direct segment  $\hat{v}\hat{w}$  goes on the wrong side of  $\tilde{u}$  (b). A kink vertex is required to redirect the edge around  $\tilde{u}$  (c).

Jacobian  $J$  from the parameter domain to the mesh (i.e.,  $\Gamma^2$  and  $\gamma^2$  are eigenvalues of the metric tensor  $J^T J$ ). Since  $\phi$  is piecewise linear, computing the overall distortion is straightforward. For each pair of corresponding triangles, the map is linear, so the Jacobian is constant. Integrating the distortion over the whole map is then simplified to a weighted sum of the constant distortions over each triangle.

Most previous distortion metrics are asymmetric, in the sense that optimizing  $\phi$  and optimizing  $\phi^{-1}$  would not result in the same map. Two exceptions are the  $(\frac{\Gamma}{\gamma} + \frac{\gamma}{\Gamma})$  metric of Hormann et al. [70] and the  $\max(\frac{1}{\gamma}, \Gamma)$  metric of Sorkine et al. [129], which have the key property that they are invariant to the substitution  $(\Gamma, \gamma) \leftrightarrow (\frac{1}{\gamma}, \frac{1}{\Gamma})$ .

It is likely feasible to create symmetrized versions of many prior metrics, including the popular discrete conformal map. This work symmetrizes the  $L^2$  stretch of [121] because it smoothly penalizes scale distortion. This is done by summing direct and inverse  $L^2$  stretch:

$$L^2(T)^2 = L_{stretch}^2(M^1 \rightarrow M^2)^2 + L_{stretch}^2(M^2 \rightarrow M^1)^2 \quad (3.1)$$

$$= A_{\tilde{T}} \frac{A_{M^2}}{(A_{M^1})^2} \left( \frac{1}{\gamma^2} + \frac{1}{\Gamma^2} \right) + A_T \frac{A_{M^1}}{(A_{M^2})^2} (\gamma^2 + \Gamma^2), \quad (3.2)$$

where  $A$  denotes area,  $\tilde{T}$  is a triangular piece of  $\mathcal{N}(\hat{v})$ ,  $T$  is a triangular piece of  $\mathcal{N}(v)$ , and  $\Gamma$  and  $\gamma$  are the singular values of the Jacobian  $J$  of the composed map between  $\tilde{T}$  and  $T$ . This particular definition has the key property of being invariant to the scale of either model. First, let  $U^1$  be the units of mesh  $M^1$ , and  $U^2$  be the units of mesh  $M^2$ . The scale invariance is because  $A_T$  and  $A_{M^1}$  are in units of  $(U^1)^2$ ,  $A_{\tilde{T}}$  and  $A_{M^2}$  are measured in units of  $(U^2)^2$ , and  $\Gamma, \gamma$  are measured in units of  $U^2/U^1$ . This results in Equation 3.2 being a unit-less quantity. Note that the symmetric formulation obviates the need for a regularizing term as was used in [115]. The map quality is

reported with the symmetric stretch efficiency, which is defined simply as  $2/\sum_T L^2(T)^2$  and has an upper bound of 1.

A symmetric conformal metric was also experimented with. However, such a distortion metric is less sensitive to changes in geometry, and therefore does not lead to natural correspondence of major geometric features. As an example, in Figure 3.11 the whole head of each animal is mapped to a small disc on the neck of the other.

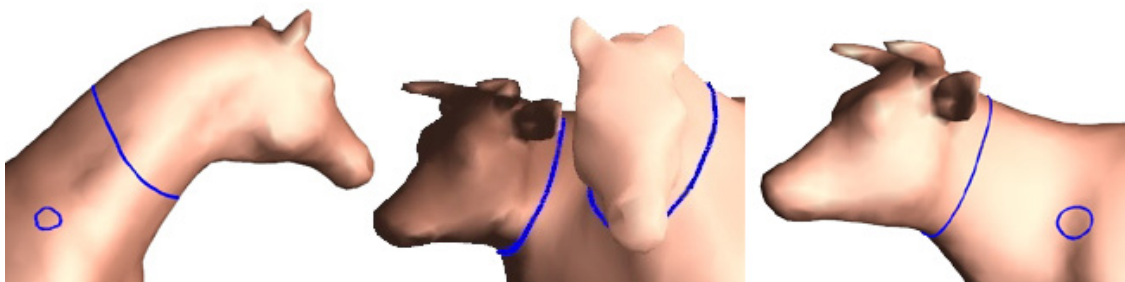
## 3.5 Applications

The availability of an intersurface map enables many applications. When both meshes are high resolution geometric models, the map can be used for morphing between them, detail transfer, and deformation transfer [131]. One of the meshes can also be constructed to represent the domain of a more typical simplicial or planar parameterization, allowing highly regular remeshes to be created.

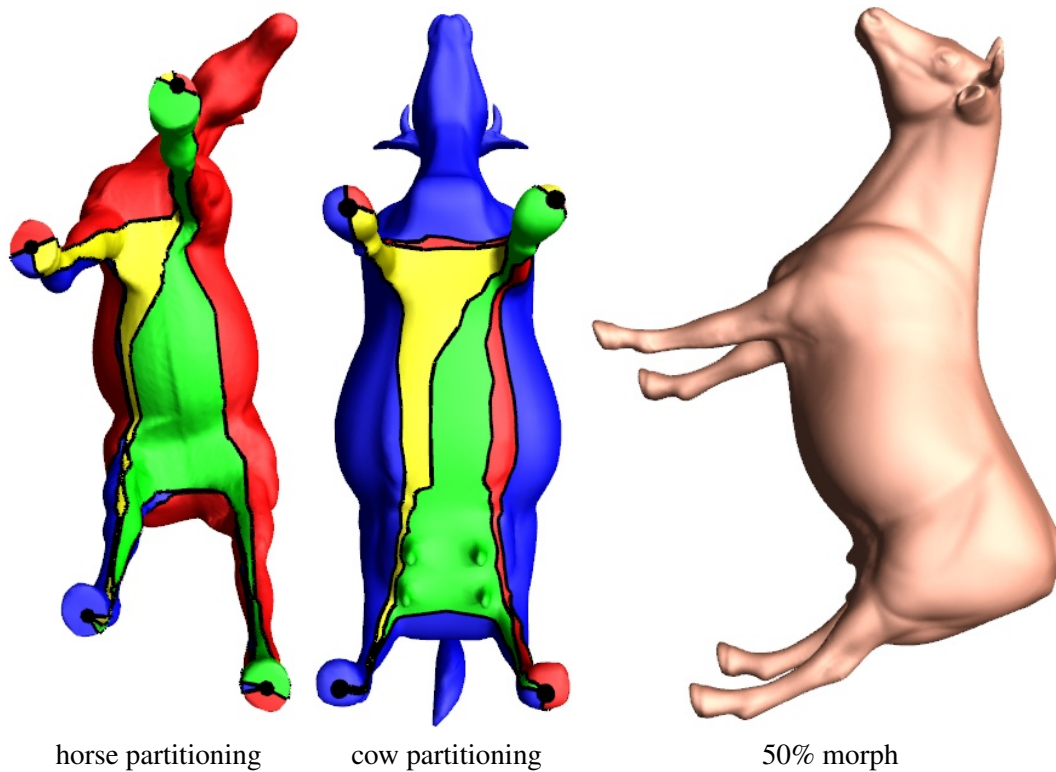
### 3.5.1 Intersurface Mapping

Figures 3.12, 3.13, and 3.14 show mappings between pairs of surfaces of genus 0, 1, and 2 respectively. The horse-cow map in Figure 3.12 uses 4 feature correspondences, the teapot-cup map in Figure 3.13 uses 22 feature points (red dots), and the dragon-feline map in Figure 3.14 uses 8 user-specified points and 7 automatically added.

Generally, the constraint points are used to initialize the map, and are then dropped during the coarse-to-fine optimization to improve the smoothness of the map. To see what can happen when the constraints are held fixed, consider the teapot-cup map in Figure 3.13. Since the interior of the cup has much more surface area than the teapot lid, it tries to flow out around the feature constraints located on the teapot rim, causing significant distortion (Figure 3.13a). In contrast,



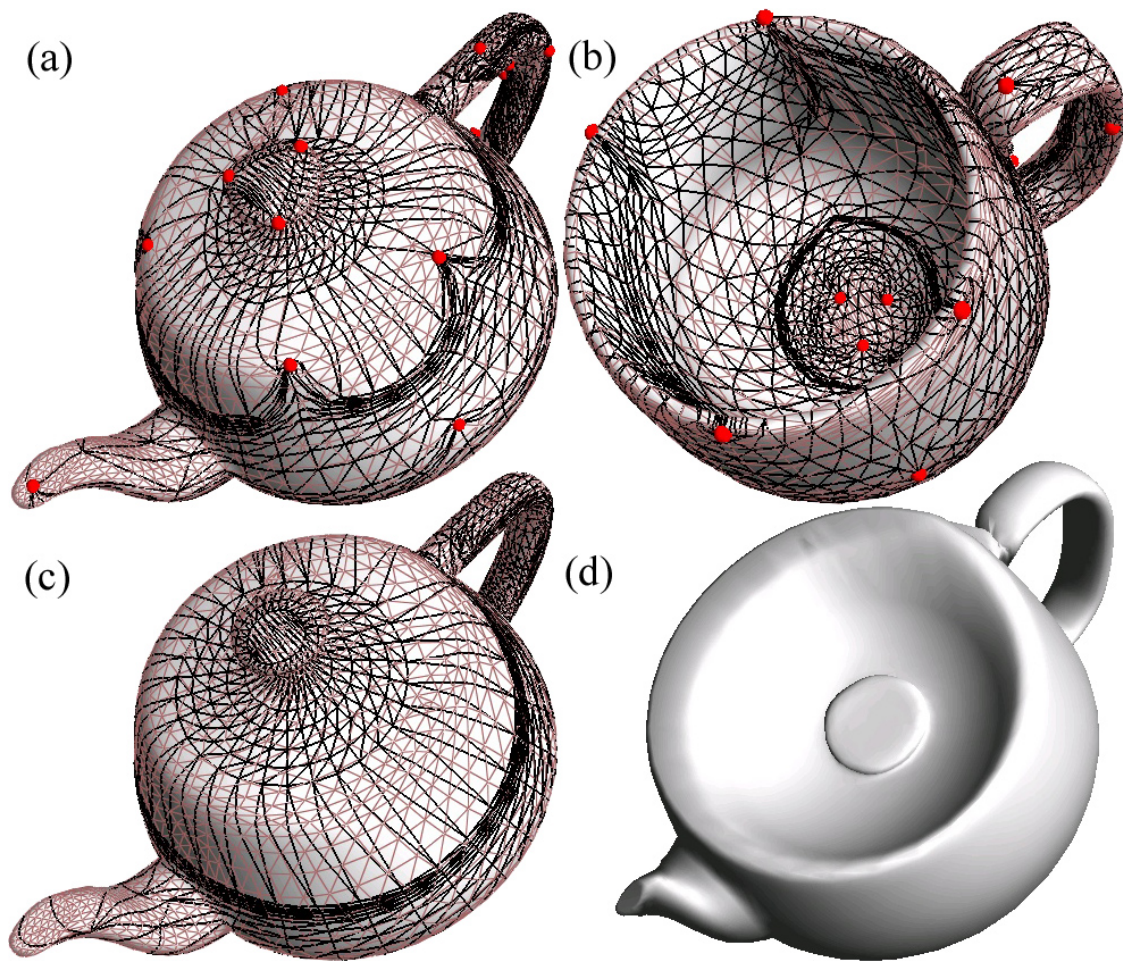
**Figure 3.11:** The use of a conformal metric results in a poor intersurface map. Instead of bringing the heads of the animals into correspondence, they are each mapped to a small region on the other's neck.



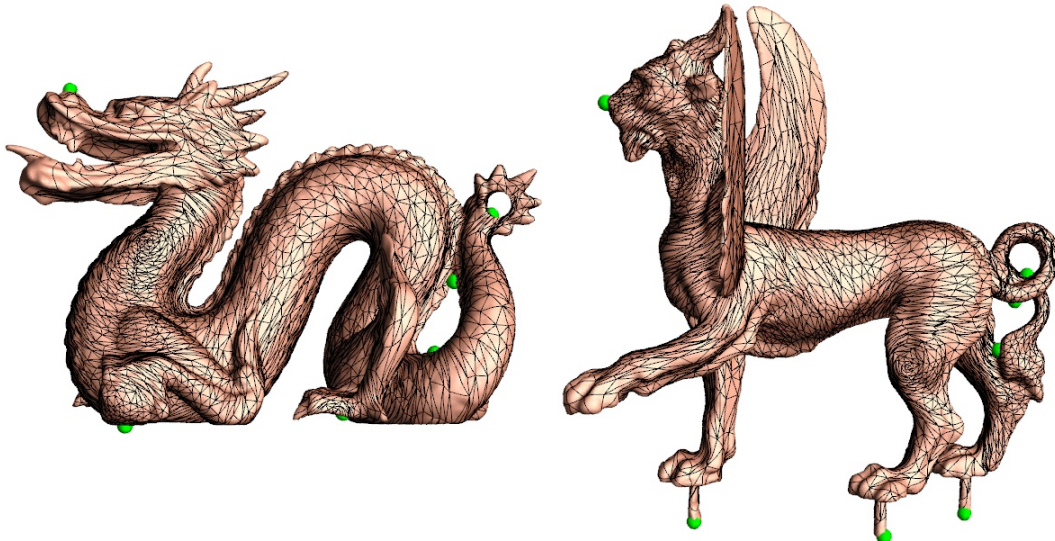
**Figure 3.12:** A cow-horse intersurface map using only 4 features.

it unfolds nicely when the constraints are relaxed (Figure 3.13c). If one did desire the cup and teapot rims to remain in correspondence, it would be best achieved by introducing constraint paths (instead of constraint points). For objects that are geometrically similar, such as the heads in Figure 3.15, point features introduce little distortion.

Figure 3.12 shows that with only four feature points placed on the hooves of the cow and horse models, a map is obtained where all the important features correspond to each other, as demonstrated by the morph. (If features did not match, they would appear doubled.) Not only did the optimization automatically match the two heads without any user-provided features in their vicinity, but it also matched smaller features such as the horses ears to the cows horns. Maps obtained by composing two separate parameterizations to simple domains (planar, spherical, or simplicial) cannot easily match features in the absence of user constraints, since this information is only available in the combined map. Figure 3.16a shows that even with 17 feature points (two on the eyes) the composed map does not achieve the quality of the intersurface map. Notice the presence of doubled features, such as nostrils, both pairs of ears and the cows horns.



**Figure 3.13:** An intersurface map between two genus-1 objects. (a,b) use fixed constraints while (c,d) drop the constraints after initialization. The cup edges are shown on the teapot in (a,c), and the teapot edges are shown on the cup in (b). A 50% morph between the meshes is shown in (d). (Symmetric stretch efficiencies: (a,b) 0.471, (c,d) 0.598.)



(a) Surface  $M^1$  with edges from  $M^2$ .  
(Notice density of edges from left wing.)

(b) Surface  $M^2$  with edges from  $M^1$ .  
(See spike flattened on rear left knee.)

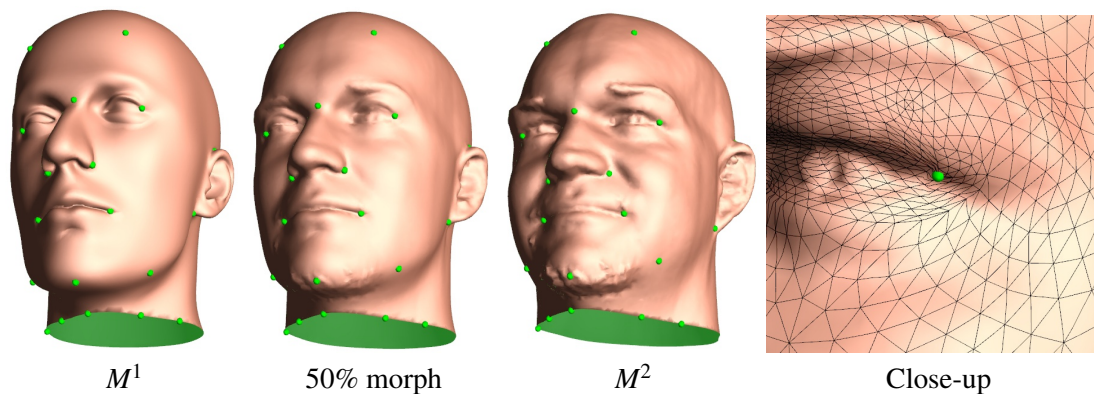


(c)  $M^1$  normals mapped onto  $M^2$ .  
(lit using 2 antipodal light sources.)

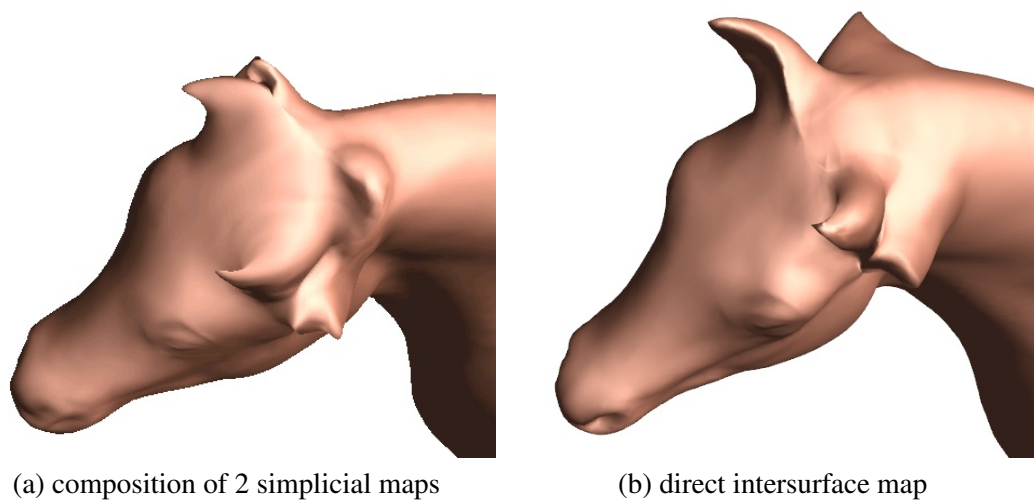


(d) A 50% morph.

**Figure 3.14:** An intersurface map for two objects of genus 2, initialized with 8 user-specified feature points. (Symmetric stretch efficiency 0.311.)



**Figure 3.15:** Map between two meshes with boundaries. The close-up on the eye shows low distortion around the feature point ( $M^1$  edges over  $M^2$  geometry). The boundary at the necks are treated as path constraints. (Symmetric stretch efficiency 0.967.)



(a) composition of 2 simplicial maps

(b) direct intersurface map

**Figure 3.16:** The intersurface map automatically favors shape correspondence, unlike the composition of two separate simplicial parameterizations, as shown in these morphs. The simplicial map uses the 17 feature points shown in Figure 3.1. (Symmetric stretch efficiencies: (a) 0.416, (b) 0.442)

### 3.5.2 Simplicial Parameterization

Simplicial parameterization is the process of mapping a mesh to an abstract simplicial complex whose triangle faces are conceptually all equilateral. The connectivity of this simplicial complex is often represented by a simplified version of the mesh being parameterized. In this scenario,  $M^1$  is the abstract simplicial complex. Although such a domain lacks an isometric embedding in  $\mathbb{R}^3$ , this is not a problem for the intersurface mapping algorithm. Each triangle piece of the map  $\phi$  is a subregion of a single triangle on each of the meshes. When computing the distortion between these triangles, the region of the triangle on  $M^1$  is further mapped to a corresponding region on an equilateral triangle. Treating the geometry of  $M^1$  as equilateral in this local way precludes the need for an isometric embedding in  $\mathbb{R}^3$ , which is in general not possible.

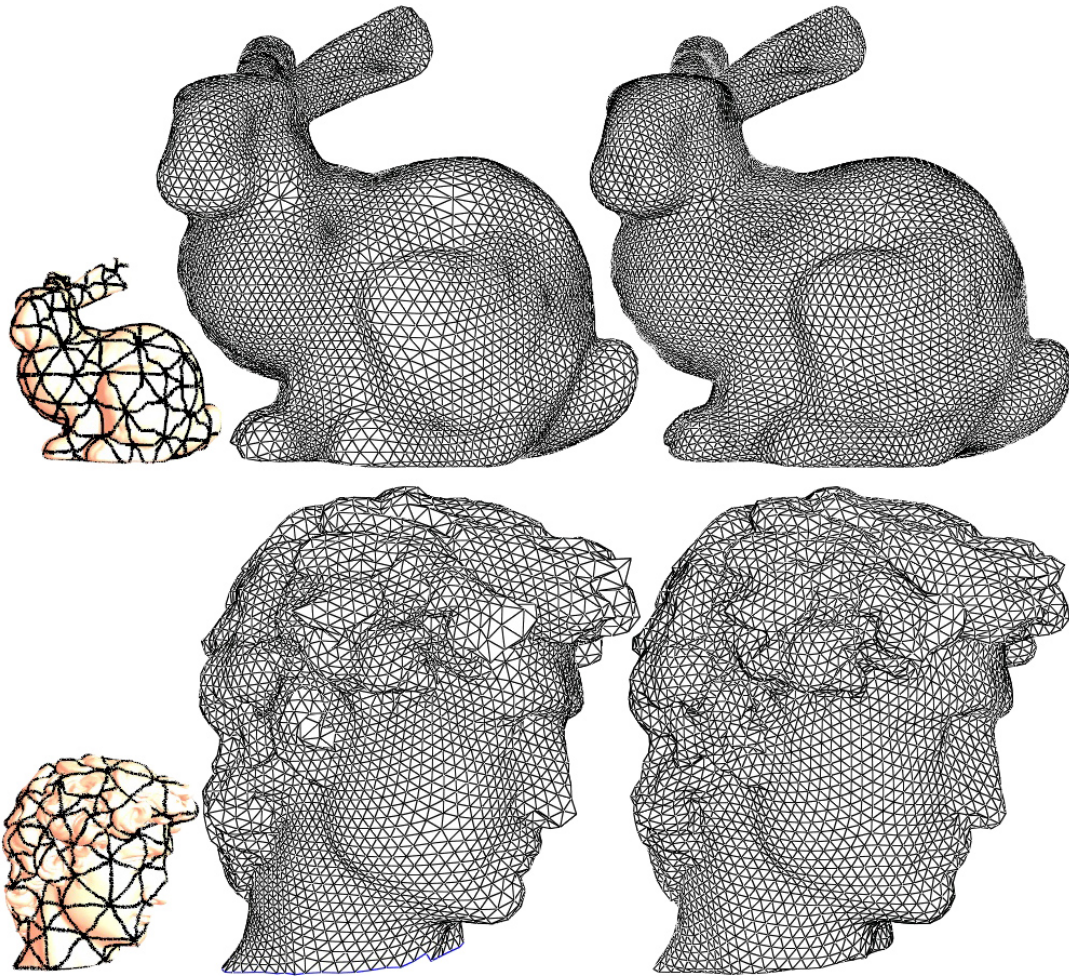
Once a simplicial parameterization has been created, a semiregular remesh of the surface can be generated by regularly resampling each face of the simplicial complex and mapping the samples back to the mesh. This results in a remesh with valence six vertices everywhere, except at the vertices of the base domain.

Among previous simplicial parameterization methods, the most advanced is the Globally Smooth Parameterization (GSP) work of Khodakovsky et al. [79], which attains smoothness across domain edges. However, it compresses the parameterization in the vicinity of low-valence irregular vertices, and stretches it near high-valence irregular vertices. As Figure 3.17 shows, the intersurface maps are visually smooth everywhere, and the extraordinary domain vertices have much less influence on the parameterization uniformity.

### 3.5.3 Octahedral Parameterization

Praun and Hoppe [115] use a sphere as an intermediate domain to parametrize a surface onto an octahedron for subsequent geometry image remeshing. Improved results are obtained by directly optimizing the octahedron-to-surface map. Figure 3.18 demonstrates the unfolding of an octahedral parameterization to create a geometry image for the Venus head. As shown in Table 3.1, the parameterization stretch efficiency is improved in all cases, and the geometric accuracy of the remeshes is also improved for models with many extremities. PSNR is measured as in [115] as  $20 \log_{10}(s/n)$ , where  $s$  is the length of the bounding box diagonal, and  $n$  is the rms Hausdorff error between the original mesh and the remesh.

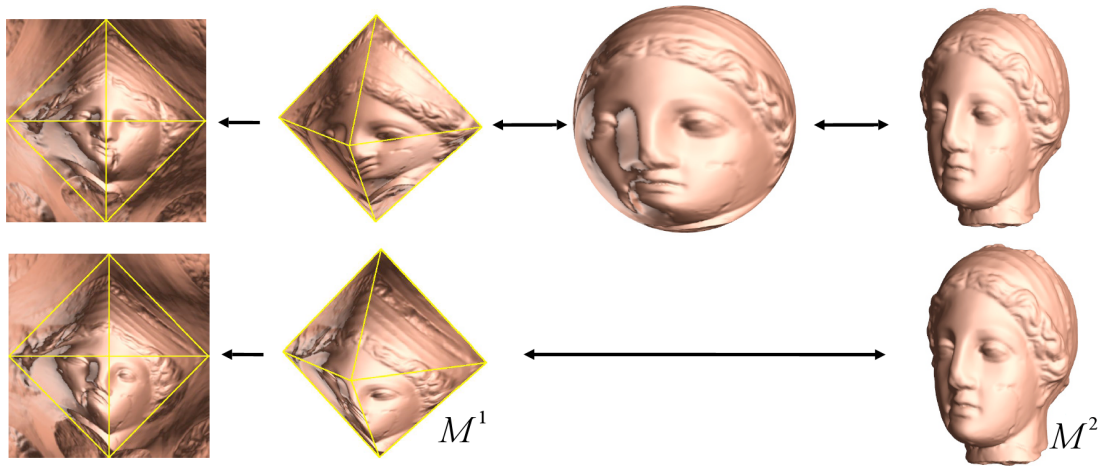




**Figure 3.17:** Comparison of semiregular remeshing using GSP (middle) and intersurface mapping (right). Left: base domain patches. (One-way stretch efficiencies: bunny 0.800, 0.915; David 0.761, 0.902.)

### 3.5.4 Toroidal Parameterization

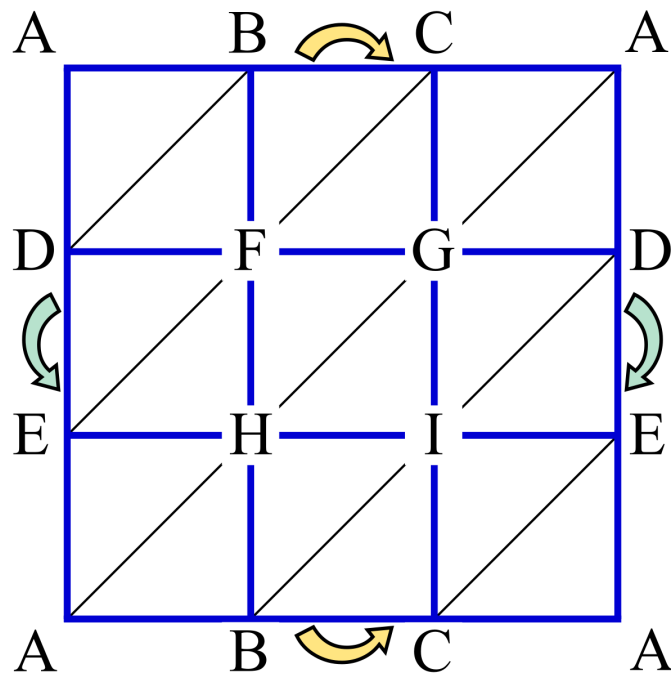
A natural domain for genus-1 surfaces is the toroidal unit square. It is formed by identifying the squares boundaries left-to-right and top-to-bottom. To apply the intersurface mapping framework to this scenario, the toroidal domain can be represented by a mesh  $M^1$  with 9 vertices and 18 triangles (see Figure 3.19). As in simplicial parameterization, the domain  $M^1$  does not have a *global* isometric embedding in  $\mathbb{R}^3$ , but again the local geometry of the domain can be used when evaluating the distortion of the map. In this case, the triangular regions on  $M^1$  are further mapped to right isosceles triangles. The result of treating the domain triangles in this way is that the *local* map  $\mathcal{N}(\tilde{v}) \rightarrow \mathcal{N}(\hat{v})$  is always an isometry.



**Figure 3.18:** Praun and Hoppe [115] use a sphere as an intermediate domain to create geometry images [58] without making cuts on the mesh. Both the mesh and octahedron are mapped to the sphere, which are combined to form an intersurface map. The octahedron is then unfolded to create a geometry image (top). Improved geometry images with better parameterization efficiencies and more accurate remeshes can be created by directly optimizing the intersurface map (bottom).

**Table 3.1:** Comparison of octahedral remeshing using spherical parameterization ( $D \rightarrow S \rightarrow M^2$ ) [115], and using the direct map onto the octahedral domain  $D$ .

Model	One-way $L^2$ Stretch Efficiency		Remesh PSNR		Max Remesh Hausdorff Error (% Diagonal)	
	$D \rightarrow S \rightarrow M^2$	$D \rightarrow M^2$	$D \rightarrow S \rightarrow M^2$	$D \rightarrow M^2$	$D \rightarrow S \rightarrow M^2$	$D \rightarrow M^2$
Venus	0.943	0.947	83.4	83.2	0.194	0.197
Bunny	0.706	0.717	80.0	79.9	0.232	0.193
Gargoyle	0.643	0.679	79.2	79.3	0.233	0.202
Armadillo	0.454	0.528	72.0	73.0	0.520	0.320
Horse	0.363	0.398	76.9	77.7	0.344	0.304
Cow	0.405	0.440	74.9	77.0	1.042	0.412
Tyrannosaurus	0.360	0.418	73.6	74.5	0.497	0.436



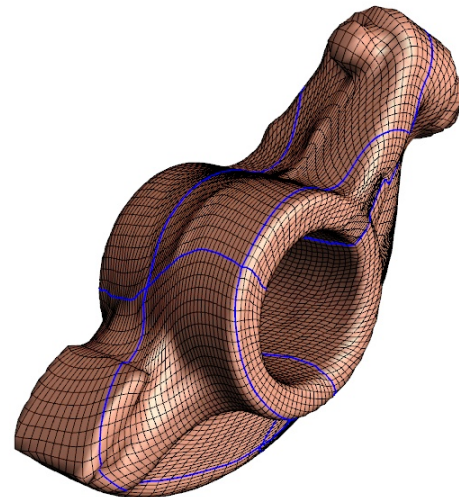
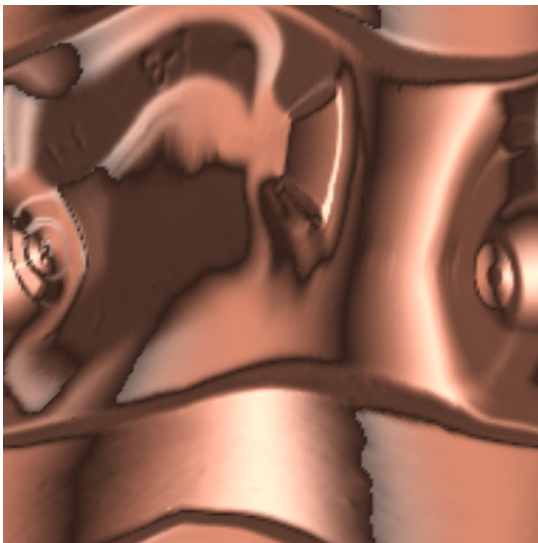
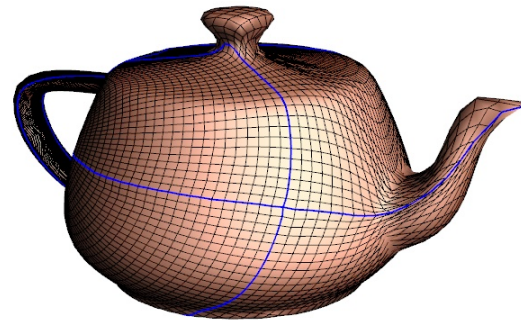
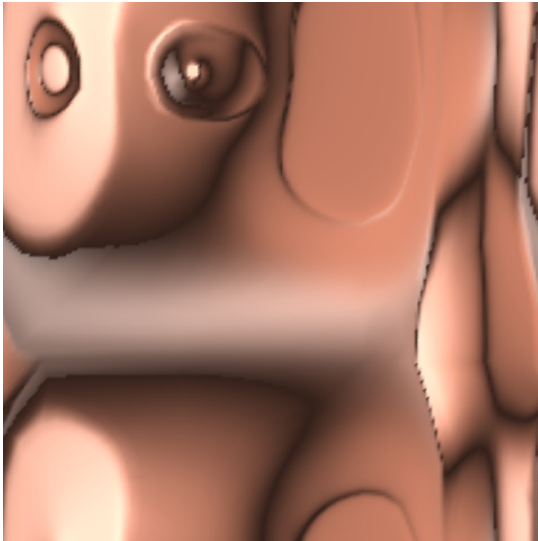
**Figure 3.19:** The toroidal base domain tessellation is made up of nine vertices (A–I) that repeat when unfolded to the plane.

To initialize the parameterization, the user specifies 9 feature points on the input mesh  $M^2$ , to correspond with the domain mesh vertices. To allow maximum freedom for the map, these feature points do not act as constraints during coarse-to-fine optimization. After the intersurface map has been created, the abstract right isosceles domain can be isometrically unfolded to the plane. This parameterization can then be used to create a fully regular quad mesh (i.e., valence-4 vertices everywhere), or geometry images with no singularities. Figure 3.20 shows some example results.

There has been little work on toroidal parameterizations of arbitrary genus-1 surfaces, which is surprising since the domain is the most “Euclidean” of all closed surface topologies. Gu and Yau [59] demonstrate their global conformal approach on genus-1 surfaces. The results shown here exhibit less scale-distortion compared to theirs, due to the use of a stretch functional.

### 3.6 Discussion

An earlier implementation of this method followed a more traditional parameterization approach, with a static domain and only one mesh being optimized using a coarse-to-fine algorithm. The map was initialized by using conformal maps to establish correspondences between the



Surfaces mapped into toroidal domain  
(with 2-sided lighting)

Remeshed surfaces  
(all vertices have valence exactly 4)

**Figure 3.20:** Examples of toroidal parameterization and remeshing. (One-way stretch efficiencies: teapot 0.458, rocker arm 0.582.)

domain vertices and the large base domain faces of the progressive mesh. This method presented two difficulties: (1) some patches were too large to robustly parametrize using a single linear system and (2) having formed this initial map, there was no way to effectively improve it (since it was “stuck” at a fine level). The symmetric coarse-to-fine approach overcomes both these difficulties.

An important property of directly optimizing the map between two surfaces is that the correspondence of geometrically similar features is encouraged within the distortion metric itself, thereby requiring fewer manually specified features. For example, only 4 features are sufficient to obtain a good map between the cow and the horse. These 4 features on the hooves are needed to prevent a combinatorial optimization, i.e., which cow leg corresponds to which horse leg. This mapping problem shares similarities with the problem of obtaining a rigid correspondence between two objects. Mesh registration energy functionals typically have many local minima and thus require initial user guidance, but importantly they have a deep energy well near the global solution.

The major difference between the symmetric coarse-to-fine refinement process and previous simplicial parameterization methods is the opportunity for fine-grain optimization. Simplicial parameterization methods apply linear relaxation operations across coarse domain faces. Using a nonlinear optimization on individual vertices of both meshes coupled with progressive refinement produces improved results.

While this method achieves impressive results for a large class of applications, its main current limitation is execution time. The mutual tessellation is more complex than either of the input meshes, and managing it during optimization is time-consuming. The current implementation takes a couple of hours to create intersurface maps between meshes of approximately 64K faces. For the simplicial, octahedral, and toroidal parameterization scenarios, where  $M^1$  is coarse, it takes about 20 minutes to create the map. The space complexity of the mutual tessellation could theoretically be  $O(n^2)$  for a pathological worst case, but for ordinary models it is about  $8n$ , i.e., a small factor more than the  $2n$  vertices from the two meshes. In practice, memory usage has not been an issue.

Another conceptual drawback of the current implementation (though not of the method in general) is the asymmetry of the data structure, which only allows one of the meshes to be refined at a time. A truly symmetrical implementation allowing fine-grain interleaved refinement of both meshes would be more elegant.

## CHAPTER 4

### ADAPTIVE MESHING: GUIDANCE FIELD ADVANCING FRONT

The meshing research community has for a long time used advancing fronts for generating high-quality meshes suitable for numerical simulation [110]. In that context, a meshing algorithm starts with well-defined domain representations and creates a piecewise linear representation, progressively increasing the dimension of the primitive: edges, triangles, and then tetrahedra. The essential difference in the setting here is the absence of a well-defined description of the components of the volume. This work assumes a much more general model — any surface for which curvature can be computed, and onto which points can be projected. In this case, it would be a challenge in itself to find *any* charts for the reconstructed surface, let alone one that is suitable for use in meshing. Some works have tried to create a mesh under more general conditions. These methods, however, seem limited to *ad hoc* steps like smoothing a mesh generated from preclassified images [30].

Given a two-dimensional orientable manifold surface  $\mathcal{S}$  embedded in  $\mathbb{R}^3$  that supports a projection operator and curvature computation, the work presented here produces a high quality triangulation that accurately captures details. The final reconstruction has bounded error from the original surface, and most of the triangles exhibit excellent quality — the user defines how close to equilateral they must be.

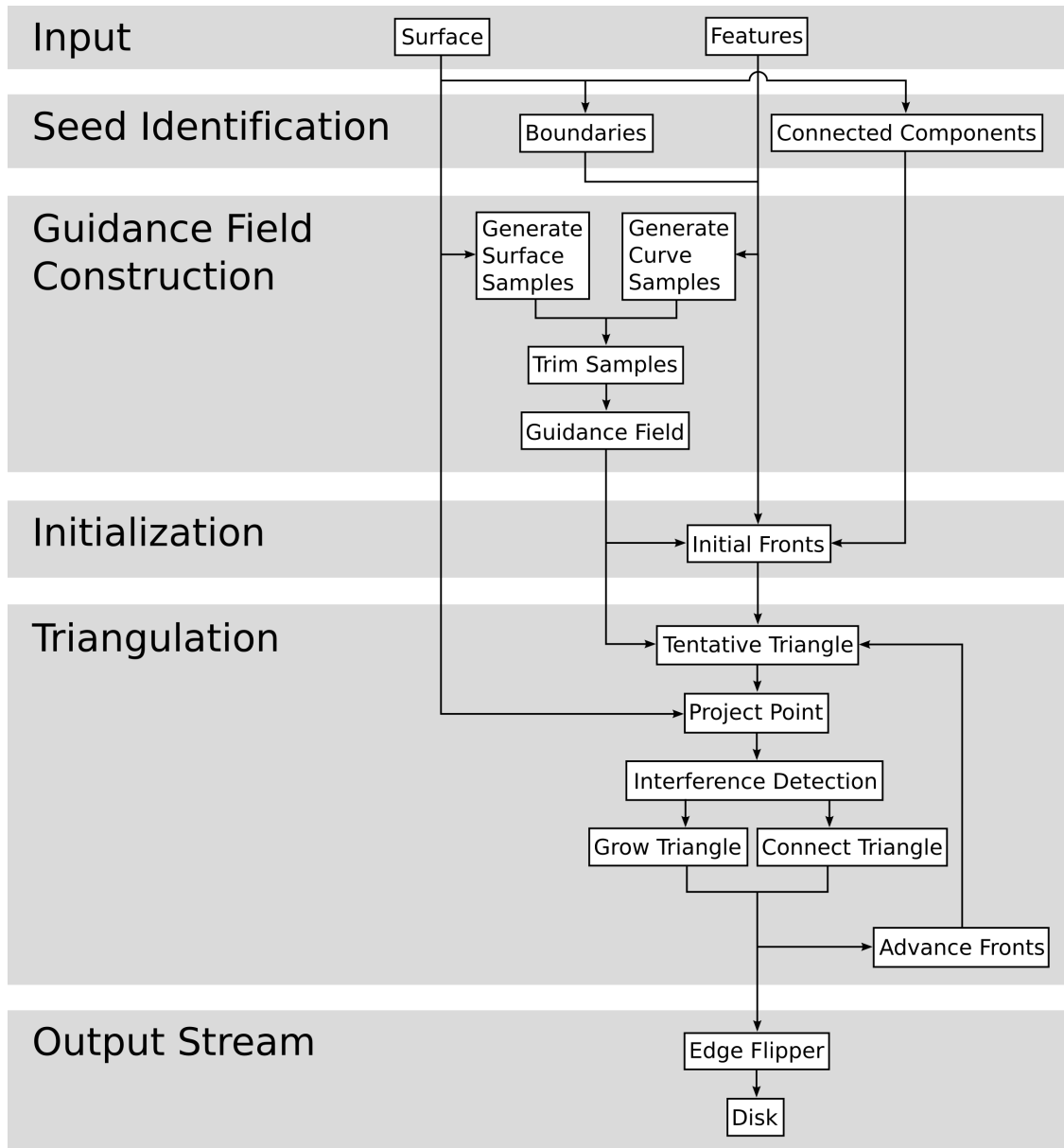
This chapter is organized as follows. An overview of the algorithm is given in Section 4.1, and discusses edge sizing considerations. A variety of surface definitions that the algorithm can be applied to are discussed in Section 4.2. The construction of the guidance field used to control the sizes of the edges in the triangulation, and the consequences of adhering to it, are presented in Section 4.3. The core triangulation algorithm is described in Section 4.4, and handling of the output stream is discussed in Section 4.5. Further implementation details are discussed in Section 4.6, and applications of the algorithm are presented in Section 4.7. Finally, the results are discussed in Section 4.8.

## 4.1 Algorithm Overview

The basic surface triangulation problem is stated as follows. Given a surface  $\mathcal{S}$  as input, defined as a projection operator  $\mathcal{P} : \mathbb{R}^3 \rightarrow \mathcal{S}$ , construct a triangulation such that the distance between  $\mathcal{S}$  and the triangulation is bounded. The number and quality of the generated triangles should be easily and intuitively controlled. There are two user-defined parameters  $\rho$  and  $\eta$  which will control approximation accuracy and triangle quality, respectively.

The advancing front algorithm can be summarized with the following components (see also Figure 4.1):

1. **Input:** The inputs to the algorithm, in addition to the user parameters  $\rho$  and  $\eta$ , are a surface  $\mathcal{S}$  and a description of any features that must be preserved in the output mesh. These features may be the intersection curves computed from a *constructive solid geometry* (CSG) operation, creases identified on a surface, or simply a curve drawn by a user.
2. **Seed identification:** Each connected component of the surface requires a *seed* placed on it to start the triangulation. Boundaries and features are used for these seeds when they exist. Otherwise, a single point on each connected component is found.
3. **Guidance field construction:** The guidance field is a function  $g$  that determines the edge sizes of the output mesh, and is represented by a set of sample points  $s$  on the surface and along the feature and boundary curves. Each sample imposes a constraint on the value of  $g$ . Taking the curvature of the one-dimensional features and boundaries into account in the guidance field guarantees that they will be accurately represented in the output, even when they make sharp bends in otherwise flat regions of the surface. To remove redundant information and improve the efficiency of queries, the samples are trimmed into a minimal representation.
4. **Initialization:** Any boundaries and features are resampled with edges sized according to the guidance field, and used as initial fronts for the triangulation. This prevents triangles from growing across the features, or off the surface. For each connected component without any boundaries or features, a single seed edge is used. Each front is represented by a series of vertices and the normals of the surfaces at those points.
5. **Triangulation:** The triangulation proceeds with a series of *edge growth* operations, where an edge from one of the fronts is chosen to create a new triangle from. The guidance field is queried to determine the edge lengths of the new triangle, and the new point is projected



**Figure 4.1:** The advancing front algorithm can be divided into six components: the input, seed identification, guidance field construction, initialization of the seed fronts, triangulation of the surface, and finally handling the output stream.



onto the surface with  $\mathcal{P}$ . This new triangle is then checked for interference with any of the other front edges. If there is no interference, a *free* triangle is created with the new vertex. If there is interference, a *connection* triangle is created between the growing edge and an already existing vertex. If the edge and the connection vertex are part of the same front, the front gets split into two. Otherwise, the two different fronts are merged into one. This process continues until the entire input surface has been triangulated.

6. **Output stream:** As the fronts advance and new vertices and triangles are created, they are output in a streaming manner. This allows streaming mesh processing techniques to be applied to the generated mesh, before it is written to disk.

Note that there is a correspondence between splitting and merging of fronts when connection triangles are created, and handlebody decompositions [95]. Therefore, advancing front techniques naturally cope with high-genus manifold reconstruction.

The problem of choosing edge sizes is a fundamental issue for advancing front algorithms. The user parameters  $\rho$  and  $\eta$  will be used to define guidance field edge sizing function  $g : \mathcal{S} \rightarrow \mathbb{R}^+$ . It will be specifically constructed so that it creates triangles that adapt to the surface curvature, but also do not change too rapidly. Note that the domain of  $g$  is  $\mathcal{S}$ , but the construction will define it over all of  $\mathbb{R}^3$  for simplicity. This restriction does not affect any of the properties of  $g$ .

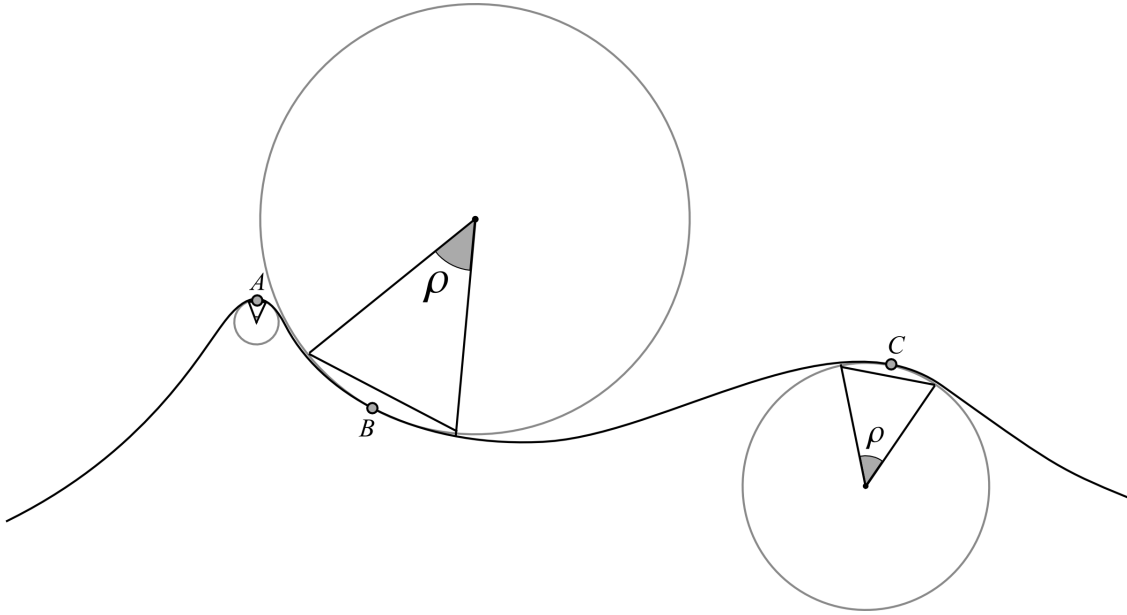
#### 4.1.1 Curvature Adaptivity

To accurately mesh a surface, triangle sizing should be inversely proportional to the maximum curvature of the surface. This adaptivity to the local surface also prevents the output mesh from becoming unreasonably dense. Most early work did not adapt triangle size to geometric features, and thus depended on an *a priori* compromise between the size of the resulting mesh and the representation of small features in the output. These algorithms were quickly followed by adaptive variants [3, 31, 76].

Following in this direction, for each surface, an ideal edge size function  $\iota : \mathcal{S} \rightarrow \mathbb{R}^+$  is defined to be the length of the edge that subtends an angle  $\rho$  on the osculating circle of minimum radius situated at  $\mathbf{x}$ :

$$\iota(\mathbf{x}) = \frac{2 \sin(\rho/2)}{\kappa_{\max}(\mathbf{x})}, \quad (4.1)$$

where  $\kappa_{\max}(\mathbf{x})$  is the maximum absolute curvature of  $\mathcal{S}$  at  $\mathbf{x}$  (see Figure 4.2). This is called the *ideal* edge size, because it is the maximum edge size allowed that ensures that all of the features in the surface are captured accurately to the degree defined by  $\rho$ , independently of their scale.

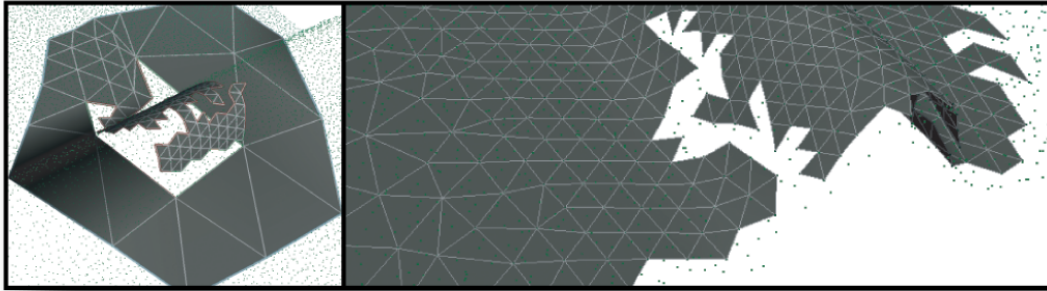


**Figure 4.2:** The ideal edge size function  $\iota$  is controlled by a user parameter  $\rho$ . The ideal edge subtends the angle  $\rho$  on a sphere of radius  $r = \kappa_{\max}(\mathbf{x})^{-1}$ . If the curvature changes too quickly, such as between points  $A$  and  $B$ , poor triangles may be created. When the curvature changes slowly, such as between points  $B$  and  $C$ , the quality will be good. The Lipschitz property of  $g$  will enforce that the edge lengths always change slowly, independent of how fast  $\iota$  changes.

While adapting edge sizes to the surface curvature helps to reduce the size of the resulting mesh, it still has a major drawback. Since curvature is purely local, it is still possible to step over important geometrical and topological features of the surface. Figure 4.3 shows a common geometric situation where typical adaptive advancing front algorithms fail to reconstruct the surface correctly. The inset on the left shows the consequences of determining triangle sizes completely locally, while the right inset shows the desired correct behavior: triangles need to be made smaller *before* they reach the geometric feature.

### 4.1.2 Lipschitz Edge Sizing

In and of itself, adapting the surface to curvature introduces another problem: abrupt changes in curvature across the surface lead to abrupt changes in triangle size and poorly shaped triangles [27, 118]. The assumption of maximum curvature (embodied by a predetermined triangle size) is essentially replaced by an assumption of *small curvature changes* across the surface. Vastly different triangle sizes make robust front interference tests highly nontrivial, and instead heuristics have been used [76]. This is important, since the test is critical to the correct behavior of the



**Figure 4.3:** Depending only on the curvature to determine the edge sizes makes advancing front algorithms sensitive to changes in curvature. If the algorithm does not anticipate the need for small triangles early, large triangles will be created next to small ones (left). The expected behavior is shown on the right.

algorithm. If the test fails to detect front interference, two or more sheets of triangles will cover portions of the surface.

This problem of abrupt changes in triangle size have previously been noticed in the community as well. Borouchaki et al. [27] discuss methods for achieving gradation control: enforcing that the length ratio of two consecutive edges is bounded. This constraint on the edge sizes can be encapsulated by an edge sizing function  $g$  that is Lipschitz continuous, with the Lipschitz constant reflecting the allowed rate of change of the edge sizes.

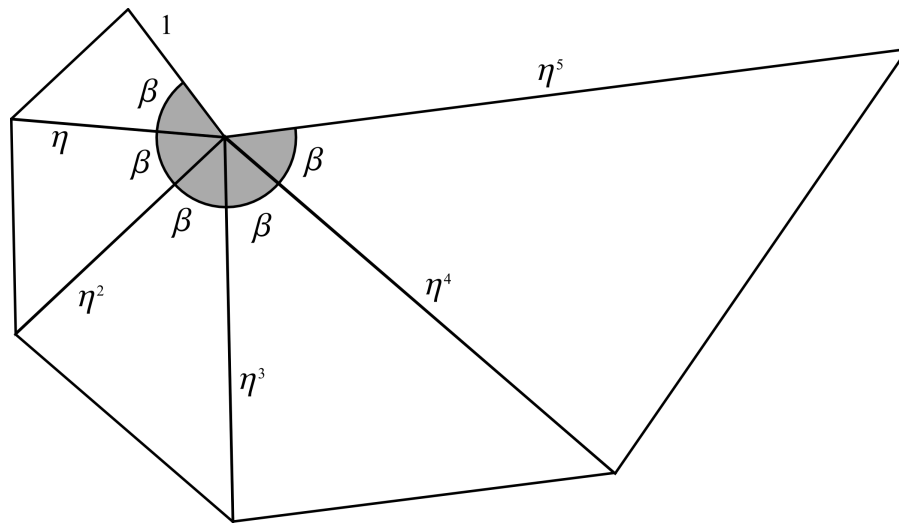
Persson [111] evolves a given ideal edge sizing function by a PDE to limit the gradient magnitude of the resulting  $g$ , and thus enforce the Lipschitz property. However, since this requires a discretization of the embedding space to solve the PDE over, with the size of the elements directly affecting the accuracy of the result, it may be difficult or slow to generate a suitable background grid for arbitrary input surfaces. The approach taken here is not to compute and represent the value of  $g$  directly, but instead to sample  $S$  and  $t$  and show how these samples can be used to efficiently evaluate  $g$  at any given point.

Scheidegger et al. [122] introduced an advancing front algorithm for triangulating point set surfaces which made use of an edge sizing function  $g$  to both provide curvature adaptability and restrict the changes in triangle sizes. Since the goal of  $g$  is to *guide* the advancing front algorithm, they termed it a *guidance field*. Although their goal was to create an edge sizing function that was Lipschitz, it was not strictly enforced. They use a parameter  $\beta$  to control the minimum and maximum angles of any triangles they create. This has the effect of bounding the quality of

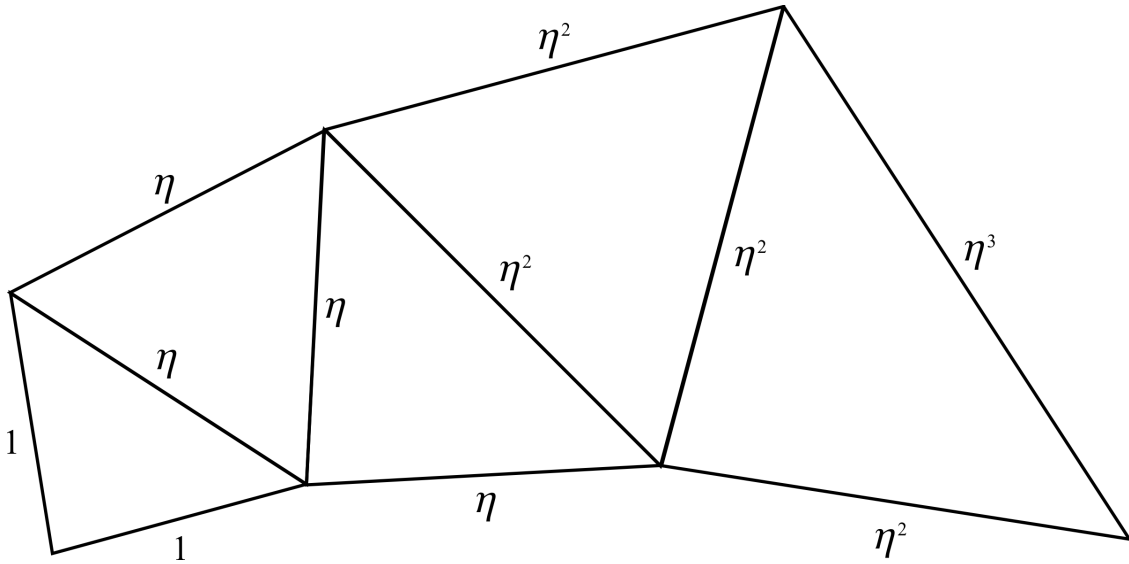
the triangles. However, when considering each triangle independently, the ratio of edge lengths incident to any given vertex is not bounded, as illustrated in Figure 4.4.

Some meshing algorithms use the local feature size of Ruppert [118] to control the size of the elements (e.g., [9, 99]). This has the significant advantage that the local feature size is itself a Lipschitz continuous function. The disadvantage is that it can be very difficult to evaluate, especially in the fully generalized context taken here, where the underlying surface is treated in an abstract way. Instead, the approach is to take an arbitrary ideal edge size function  $\iota$ , and find a suitable guidance field  $g$  that closely bounds  $\iota$  below, but is Lipschitz by construction. This prevents edges from being created that are too large to capture the curvature of the surface, while also restricting them to change in a controlled manner.

While the user parameter  $\rho$  is used to define the ideal edge size function, the parameter  $\eta$  is used to determine the Lipschitz constant of  $g$ . The guidance field  $g$  is constructed to have  $\iota$  as an upper bound while having a gradient magnitude never greater than  $1 - \eta^{-1}$ . This prevents the triangle sizes from changing too rapidly and poorly shaped triangles from being created (see Figure 4.5).



**Figure 4.4:** Bounding the minimum angle  $\beta$  in a triangle is equivalent to bounding the ratio of edge lengths to a parameter  $\eta$ . When these are used to bound shapes of individual triangles, edges from large triangles can still share vertices with edges from small triangles. This results in situations where poor quality triangles are forced to be created.



**Figure 4.5:** The rate at which edges can change is controlled by a user parameter  $\eta$ . No pair of edges incident to any given vertex should have a length ratio greater than  $\eta$  or less than  $\eta^{-1}$ . This constrains the triangle shape and enforces a smooth grading of the triangle sizes.

## 4.2 Surface Definitions

An important advantage of the advancing front technique is its generality. The algorithm itself is entirely oblivious to how the surface is defined, and only requires a few properties of  $\mathcal{S}$ . It must be possible to find the boundaries of  $\mathcal{S}$ , and a point on each connected component that will be triangulated. These will be used as *seeds* for triangulation to grow from. There must be a way of computing the surface normal and curvature at any given point  $\mathbf{x} \in \mathcal{S}$  (the surface must be orientable and smooth), and a way of generating a set of samples  $s$  on the surface. The normal is used to perform robust front interference detection. The curvature is used for computing  $\iota$ , which is combined with surface samples to create the guidance field.

The surface must also admit a projection operator  $\mathcal{P}$ , for projecting the new points of free triangles onto the surface. When creating free triangles, the guidance field  $g$  is queried at the two vertices of the edge being grown. A *tentative point* for the new vertex is found that respects these edge lengths, and is then projected to  $\mathcal{S}$  with  $\mathcal{P}$ . In general, the projection will modify the lengths of the edges of the new triangle. Note, however, that if  $\mathcal{P}$  is continuous and has fixed points for all points in  $\mathcal{S}$ , the edge lengths can be preserved in the triangle that is created. Consider creating a free triangle from a front edge with vertices  $\mathbf{v}_1$  and  $\mathbf{v}_2$  and new vertex  $\mathbf{x}$ . Ideally,  $\|\mathbf{x} - \mathbf{v}_1\| = g(\mathbf{v}_1)$  and  $\|\mathbf{x} - \mathbf{v}_2\| = g(\mathbf{v}_2)$ . Each of these equations defines a sphere, and the points satisfying both equations defines a ring. The goal then is to find a fixed point of  $\mathcal{P}$  that is on that ring (e.g., by

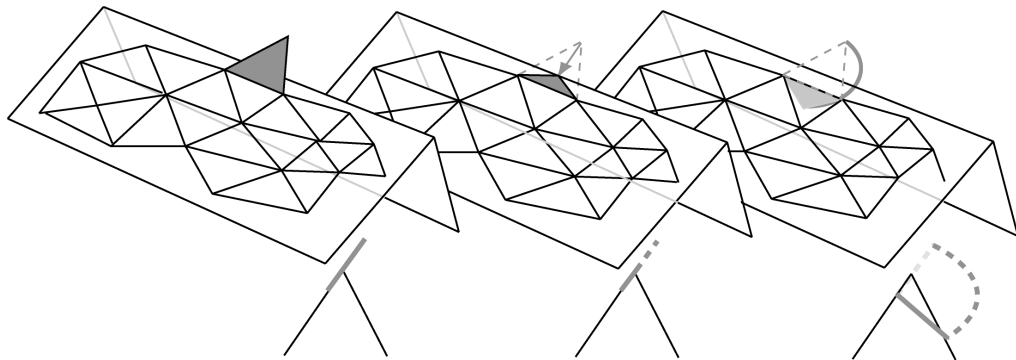
parameterizing the ring and minimizing  $\|\mathbf{x} - \mathcal{P}(\mathbf{x})\|$ ). This, however, may be very inefficient even for a simple  $\mathcal{P}$ . Instead, projection procedures for most surface definitions can be modified to use the extra endpoint and edge length information to do this efficiently.

These requirements can be satisfied by a wide range of surface definitions, including triangle meshes, isosurfaces, and point set surfaces.

### 4.2.1 Triangle Meshes

Since triangle meshes are ubiquitous in all areas of computer graphics, remeshing is an obvious target for the advancing front triangulator. One possible definition of  $\mathcal{P}$  for mesh surfaces is simply a nearest point projection. This can be performed efficiently with a *kd*-tree, but since triangle meshes are not smooth, the problem of changing edge lengths is remarkably apparent. Points tend to cluster on parts of the surface that are under represented in the input mesh, leading to artifacts, as illustrated in Figure 4.6. Instead, the exact point on the surface that is the correct distance from the existing edge endpoints can easily be found. This is done by finding the ring of points that are the correct distance from each of the endpoints, and traversing a *kd*-tree of the mesh faces to find the exact intersection of this ring with the mesh.

Though triangle meshes are clearly not smooth, they can be interpreted as an approximation of an underlying smooth surface. The normal and curvature of this underlying surface are estimated at a point by fitting a quadratic polynomial to a neighborhood of the mesh vertices [55], though other estimations are possible [97]. The normal and maximum curvature can then be extracted from the polynomial. Using the 3-ring neighborhood of a vertex to compute the curvature works well for all the meshes that have been experimented with. Using a smaller neighborhood will



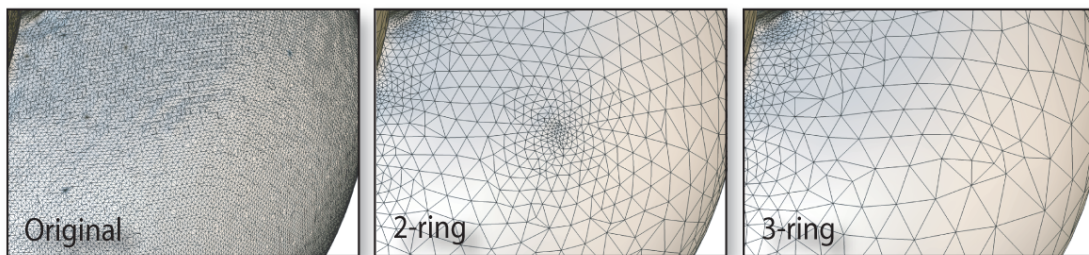
**Figure 4.6:** Different projection procedures can produce very different triangle shapes. Closest point projection behaves poorly around sharp corners, which are common in many meshes (middle). Projecting the point in a circular arc maintains the edge lengths of the new triangle (right).

typically lead to erroneous curvature estimations as most meshes have some poorly represented areas. Just a single overestimation of the curvature can lead to significant artifacts in the output mesh, where the generated triangles are much too small, as shown in Figure 4.7.

The seed points for the advancing front algorithm can easily be found for triangle meshes by performing flood fills across the faces the mesh. A random face is chose as the source, and the flood is continued until no new faces are found. Since the flood will not cross connected components of the mesh, a random vertex from the flooded region is used as a seed. A face that has not been flooded yet is then chosen as a new source, and the process is repeated until each of the connected components has a seed point. Boundaries can be found in a similar way. The surface samples  $s$  are simply taken to be the mesh vertices.

### 4.2.2 Isosurfaces

Isosurfaces are defined as the preimage of an implicit function  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ . Specifically, given an isovalue  $a$ , the isosurface  $\mathcal{S}_a$  is the set of points  $\{\mathbf{x} : f(\mathbf{x}) = a\}$ . The implicit function  $f$  can be represented in a variety of ways. It is often defined by a set of scalar samples on either a regular or unstructured grid, acquired through medical scanning techniques (e.g., CT, MRI), or as the output of a simulation. A smooth implicit function is then reconstructed from these samples. The method used to reconstruct the implicit function from the set of data samples has a direct impact on the resulting isosurface. A common trade-off when choosing a reconstruction method is between high-order continuity and interpolation of the data samples. Simple reconstruction methods can achieve either of these, but often not both. The choice of the implicit function reconstruction method is therefore left to the user.



**Figure 4.7:** To estimate the curvature at a vertex of a mesh, a quadratic polynomial is iteratively fit to a neighborhood of the vertex. If the neighborhood is too small, noise in the input mesh will produce inaccurate estimations, leading to artifacts in the output mesh (middle). A larger neighborhood slows the computation, but produces more stable results (right).

There are only a few requirements on the implicit function to provide a suitable isosurface for the advancing front algorithm. Within a band of the desired isosurface, the function must be continuous (at least  $C^1$ , preferably higher order continuity), and its value, first, and second order partial derivatives must be easily evaluable. The gradient of  $f$  must also be nonzero at all points in the isosurface. These minor requirements are quite mild and allow the advancing front algorithm to be applied to many different implicit function definitions. Choosing the definition can be left up to the user. This allows the implicit function being used by the advancing front algorithm to exactly match the function used to generate the data. For example, if the user wishes to extract the isosurfaces of a high order finite element simulation which assumes a specific polynomial interpolation scheme, the same interpolation could be used when extracting a surface for analysis or visualization.

A simple projection  $\mathcal{P}$  for an isosurface can use Newton iteration to follow  $\nabla f$  to the isosurface. A more stable projection, with the added benefit of preserving the edge lengths of the new triangle, can also be created. This is done by parameterizing the ring of points satisfying the edge length requirements with a single variable  $\theta$  (i.e., parameterized by some function  $r$  such that the ring is the set of points  $\{r(\theta) : 0 < \theta < 2\pi\}$ ). A simple root finding method is then applied to the function  $(f(r(\theta)) - a)$  to find the point on the surface. This is similar to the technique described by Cermák and Skala [31].

Since first-order and second-order partial derivatives of  $f$  are assumed to be available, the normal and curvature of the isosurface can be computed at a given point. The normal can easily be computed from the gradient, and Kindlmann et al. [80] show how to compute the geometry tensor  $G$  for isosurfaces.  $G$  encodes all curvature information, and so it can be used to compute the curvature. Let  $n = (\nabla f)/|\nabla f|$ . Then,

$$P = I - nn^T \quad (4.2)$$

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial^2 f}{\partial x \partial z} & \frac{\partial^2 f}{\partial y \partial z} & \frac{\partial^2 f}{\partial z^2} \end{bmatrix} \quad (4.3)$$

$$G = PHP/|\nabla f|. \quad (4.4)$$

The curvatures are easily computed from matrix invariants of  $G$ , and the maximum curvature  $\kappa_{\max}$  is given by the absolute value of the largest eigenvalue of  $G$ , or equivalently, the spectral radius  $r(G)$ .



#### 4.2.2.1 Structured Grids

To define an implicit function over a regular grid of sample points, piecewise cubic trivariate polynomials can be used to reconstruct the continuous function. To generate an implicit function that interpolates the data values, Catmull-Rom splines [29] are used. These splines can be thought of as using finite differences at each data point to generate gradients. The function is then extended to the interior of each cell by Hermite interpolation. This results in cubic polynomials which are  $C^1$  continuous across cell boundaries. B-splines [112] are another popular spline. These produce polynomials in each cell that maintain  $C^2$  continuity across the boundaries, but do not interpolate the input data. This may be desirable if the data are noisy. Since the Catmull-Rom splines interpolate, they tend to produce isosurfaces with high curvature when noise is present. Though these surfaces can be triangulated without problem, the high curvature isosurfaces require more triangles to be accurately captured. These spline representations are ideal because they define analytic, piecewise polynomials, which can easily be differentiated to compute gradient and curvature information.

There are efficient algorithms for computing seed points on the connected components of isosurfaces that would be suitable for initializing the advancing front algorithm (e.g., [19]). An isosurface will often exit the domain on which it is defined, creating a boundary. These boundaries can be extracted by an algorithm similar to MC, but applied to the quadrilaterals on the domain boundaries. This works for many volumes, but may not be robust since the topology of an MC mesh may not match the topology of the smooth isosurface. It is extremely important that the boundaries are correct initially. The advancing front algorithm is more sensitive to the topology of the boundaries than the connected components since the seeds on the connected components can merge together, even if they are incorrect. Since the algorithm uses the boundaries as initial fronts, they will appear in the output. A variant of MC that creates a mesh isotopic to the isosurface can be used (e.g., [24, 114]) to guarantee the correct topology of the initial fronts.

The sample points  $s$  for the guidance field can be created by generating a jittered random set of sample points inside cells that neighbor the surface and then project each of them onto the surface with Newton's method. Isosurfaces often have some areas with very high curvature, so uniformly sampling the surface with enough point to create an accurate mesh of the surface often requires extremely dense samplings. Instead, an adaptive approach can be taken, which is described in Section 4.6.2.

### 4.2.2.2 Unstructured Grids

Extending the implicit function from the data points to the interior of the cells of unstructured meshes is more challenging. Here, there is a mesh defined by a set of connected tetrahedra with data values assigned to each vertex. A scheme for defining the function on the interior of each tetrahedra that is at least  $C^1$  continuous across faces must be created. Splines cannot easily be applied to this situation because of the irregular nature of tetrahedral meshes. Two methods for defining the function have been experimented with in this work. Similar to regular grids, one definition is  $C^1$  and interpolates the data values, and the other has higher order continuity but only approximates the data.

The scheme of Nielson et al. [106] is used to define an implicit function that interpolates the values on an irregular mesh. This method was chosen for its simplicity relative to other techniques available, such as A-patches [18] and DMS-splines [39]. Nielson interpolation requires that the gradients of the implicit function are known at the vertices as input. Since this is generally not the case, an approximation is substituted. To estimate the gradient at vertex, a trivariate quadratic polynomial is fitted to the 2-ring neighborhood of the vertex. The gradient of the implicit function is then defined to be the gradient of this polynomial at the vertex. The Nielson scheme first uses the function and gradient information of the vertices to define the function and gradient along all of the edges in the tetrahedral mesh. It then uses this edge information and extends it to be defined across all of the faces. Finally, the function is extended from the faces to the interiors of the tetrahedra. Each of the extension procedures (vertices to edges to faces to tetrahedra) is based on Hermite interpolation, and is constructed in a way that maintains  $C^1$  continuity across the cell boundaries. The resulting implicit function definition for the interior of the tetrahedra is a complicated rational function that is not practical to differentiate analytically. Instead of resorting to finite differencing, C++ metaprogramming is used to automatically compute the function and its partial derivatives by encoding the chain rule for all of the primitive functions used.

The popular moving least squares (MLS) method [83] can be used as an implicit function definition that approximates the input data. The idea of the MLS method is to compute a low degree polynomial that best approximates the input data, weighted by a function of the distance from the evaluation point to the data points. The function value at the evaluation point is then simply the value of the polynomial at that point. This has a smoothing effect on the data, so it is especially useful when noise is present in the input data. A nice property of MLS is that the smoothness of the resulting function is exactly that of the chosen weighting function. Rapidly decaying gaussians are used as the weighting functions, with widths determined

by the local point density. Though the function is very smooth, it is not possible to directly compute its derivatives because every evaluation involves the solution of a linear least squares problem. Finite differences could be used, but using quadratic polynomials for the fitting and using their differential properties as an approximation of the function performs well in practice. This application of MLS to define an implicit function over a tetrahedral mesh simply discards connectivity information and uses the vertices as scattered data samples. This approach has also been adopted Ledergerber et al. [85] in the context of volume rendering through ray casting. Since MLS does not take advantage of the connectivity, the mesh is used only to define the domain of the implicit function and to find boundaries of the isosurface.

Finding the boundaries and seeds for isosurface defined on unstructured grids can be performed in a way analogous to those defined on regular grids. Rather than MC on regular grids, MT can be used on unstructured grids. The samples  $s$  for the guidance field can be sampled densely.

### 4.2.3 Point Set Surfaces

Defining a smooth surface from a finite set of unorganized points has received much attention recently. Many approaches are based on MLS formulations [2, 7, 17, 51]. Most of these methods work precisely in terms of a projection operator, the fixed points of which are taken to be the resulting surface. These definitions are in general suitable for the advancing front algorithm.

Surfaces defined in this way inherently have no boundaries. However, point set surfaces often have regions that are poorly sampled. These regions are often detrimental to the stability of the projection operator, so they should be avoided if possible. Scheidegger et al. [122] propose a heuristic to prevent their advancing front algorithm from entering these areas by measuring the largest projected angle of the input points onto the tangent plane at the projected point. When the largest such angle is sufficiently large, the samples are not uniformly distributed around the point, so the projection point is considered to be in a hole on the surface. This approach is also taken here. While this may produce jagged boundary curves, it removes the burden of annotating boundaries on the surface from the user.

The set of samples  $s$  used for the guidance field are constructed from the input samples. Since the input points do not lie on the surface in general, they are first projected. The projected points are then used to construct the guidance field. Point set surfaces often use a large number of points to define the surface, so the set of samples used for  $s$  is typically dense.

Two different point set surface definitions have been experimented with in this work. The first is the nonlinear definition of Alexa et al. [7]. The second is the linear formulation of Adamson and Alexa [2].

#### 4.2.3.1 Nonlinear

The original MLS surface of Alexa et al. [7] is defined in terms of a two stage projection procedure. The first stage is a fitting of a hyperplane to the local input points around a point  $\mathbf{p}$  being projected. Since the weights used for this fit are a function of the location of the orthogonal projection of  $\mathbf{p}$  onto the plane, rather than the location of  $\mathbf{p}$  itself, it requires a nonlinear minimization. The second stage of the projection uses a weighted least squares fit of a polynomial defined over the plane to the input points. This is a linear problem that can be solved efficiently. The point  $\mathbf{p}$  is then projected onto this local approximating polynomial to give the final position of the projection. Amenta and Kil [17] show that this procedure does not produce fixed points, but they are very close and iterating it solves the issue.

Surfaces defined in this way are as smooth as the weighting functions used, which is generally  $C^\infty$ . However, since the first stage of the projection involves a nonlinear optimization, and in fact the entire procedure may need to be iterated, it is nontrivial to compute any differential properties of the surface. Instead, the polynomial in the second stage can be used as an approximation, as done by Scheidegger et al. [122].

#### 4.2.3.2 Linear

Linear MLS formulations, such as that of Adamson and Alexa [2], provide a way of defining a smooth surface from a set of points by using the of points to define an implicit function. The isosurface associated with the isovalue zero is then taken to be the surface. The implicit function is defined to be

$$f(\mathbf{x}) = \langle n(\mathbf{x}), a(\mathbf{x}) - \mathbf{x} \rangle, \quad (4.5)$$

where  $n(\mathbf{x})$  is the direction of smallest weighted covariance of the input points and estimates the normal of the point set, and  $a(\mathbf{x})$  is the weighted average of the input points. If normals are available with the input points, this definition can be made much more stable and efficient by letting  $n(\mathbf{x})$  be the weighted average of the input normals. Using this definition, the set of points in the surface are those where the locally estimated normal direction is orthogonal to the vector from the point to the weighted average of points. Though Alexa and Adamson [6] show how to compute the gradient of  $f$  exactly, it is quite complicated. Additionally, the second-order

derivatives are also required. Again, C++ metaprogramming can be used to easily evaluate any derivatives. Once the derivatives have been computed, the framework of Kindlmann et al. [80] can be applied to compute the curvature of the isosurface.

### 4.3 Guidance Field

At the heart of the advancing front algorithm described in this chapter is the guidance field  $g$ . The guidance field determines the choice of edge size when creating new triangles on the surface. It prevents large triangles from being created near small ones by “looking ahead” and gradually shrinking edges before getting to a detailed area. Limiting the rate of edge length change also bounds the aspect ratio of free triangles. When placing a free triangle, the guidance field is queried at each of the existing edge vertex locations. These values, combined with the normals and ordering of the two front vertices, determine a tentative location for the new vertex. This vertex is then projected onto the surface and inserted into the front.

#### 4.3.1 Constructing the Guidance Field

The guidance field  $g$  is constructed from a finite set of samples  $s$  taken from  $\mathcal{S}$ . Each sample  $s_i$  is associated with an ideal length  $\iota_i = \iota(s_i)$ , as defined by Equation 4.1 at that point. When constructing the guidance field for surfaces with features or boundaries, samples and ideal edge sizes from those one-dimensional space curves are included as well. The ideal edge size for a point on a space curve is easily computed since it is only a function of maximum curvature, which is well-defined. The curvature of the features and boundaries is estimated by fitting a parametric polynomial to samples along the curves. To estimate the curvature at a given point,  $k$  samples neighboring the point are found. Three quadratic polynomials are then fit to those points in a least squares sense – one for each coordinate function. This gives a parametric approximation of the curve:

$$f(t) = (f_x(t), f_y(t), f_z(t)). \quad (4.6)$$

The curvature of  $f$  can then be computed at a given parameter value  $t$  with:

$$\kappa_{\max}(t) = \frac{|\dot{f} \times \ddot{f}|}{|\dot{f}|^3}, \quad (4.7)$$

where  $\dot{f}$  and  $\ddot{f}$  are the vectors of the first-order and second-order derivatives of the component functions taken at  $t$ , respectively. This value for  $\kappa_{\max}$  can then be used directly to compute the ideal edge size for a point on the curve.

The ideal edge sizes of these lower dimensional features are included because their curvature may be much higher than the maximum curvature of the surface they lie in. If only the ideal edge

size, and thus the curvature, of the surface were considered, the output mesh would contain edges too large to accurately capture the details of the feature and boundary curves. Also, the minimum ideal edge size for each of the half-disks is used at points on sharp features. The construction of the guidance field will be discussed for meshing a single surface, but can easily be extended for more advanced operations. The guidance field will be used to impose two conditions on the size of triangles:

1. The triangles placed over a patch of surface must be a good approximation for the local surface features. In other words, all triangle edges incident to a point  $\mathbf{x} \in \mathcal{S}$  should be at most  $\iota(\mathbf{x})$ .
2. Triangle quality throughout the triangulation must be adequate. Specifically, any two edges  $e_i$  and  $e_j$  incident to a common vertex must have a ratio bounded by a user-defined parameter  $\eta$ :

$$\eta^{-1} \leq |e_i|/|e_j| \leq \eta. \quad (4.8)$$

Each of the surface samples in  $s$  will define a constraint on  $g$ . Namely, each  $(s_i, \iota_i)$  pair will specify a function  $\tilde{g}_i : \mathcal{S} \rightarrow \mathbb{R}^+$ , that will constrain  $g$  such that  $g(\mathbf{x}) \leq \tilde{g}_i(\mathbf{x})$  for all  $\mathbf{x} \in \mathcal{S}$ . The guidance field will, at each point, be the maximum value that satisfies all such constraints.

Each constraint function  $\tilde{g}_i$  will be constructed by making use of a simpler function.  $\hat{g}_i : \mathbb{R}^+ \rightarrow \mathbb{R}^+$  will define the constraint on edge size as a function of the distance to the sample:  $\tilde{g}_i(\mathbf{x}) = \hat{g}_i(|s_i - \mathbf{x}|)$ . To use as few triangles as possible,  $\hat{g}_i$  should be as large as possible. Furthermore,  $\hat{g}_i$  should be monotonically increasing so that triangle edge sizes always increase as the edges move away from  $s_i$ . To satisfy Condition 1 above,  $\hat{g}_i$  must satisfy

$$\hat{g}_i(\iota_i) = \iota_i. \quad (4.9)$$

In other words, edges close to the sample must be sufficiently small. While Condition 1 restricts the correct size for the closest edge to the sample, Condition 2 will restrict the remaining edge sizes. Maximizing  $\hat{g}_i(d)$  along with Equation 4.9 directly implies

$$\begin{aligned} \hat{g}_i(\iota_i(1 + \eta)) &= \eta \iota_i \\ \hat{g}_i(\iota_i(1 + \eta + \eta^2)) &= \eta^2 \iota_i \\ &\vdots \\ \hat{g}_i\left(\iota_i \frac{1 - \eta^k}{1 - \eta}\right) &= \eta^{k-1} \iota_i, k > 0. \end{aligned} \quad (4.10)$$

Equation 4.10 defines  $\hat{g}_i(d)$  at a set of discrete values. Since it will be evaluated at any distance  $d \in \mathbb{R}^+$ , its definition must be extended appropriately. There are many such functions. Since  $\hat{g}_i$  will directly determine edge sizes, ideally it will minimize

$$\int_0^\infty (\hat{g}_i''(x))^2 dx. \quad (4.11)$$

This will minimize the change in the grading determined by  $\hat{g}_i$ . Consider the following expression for  $\hat{g}_i(d)$ :

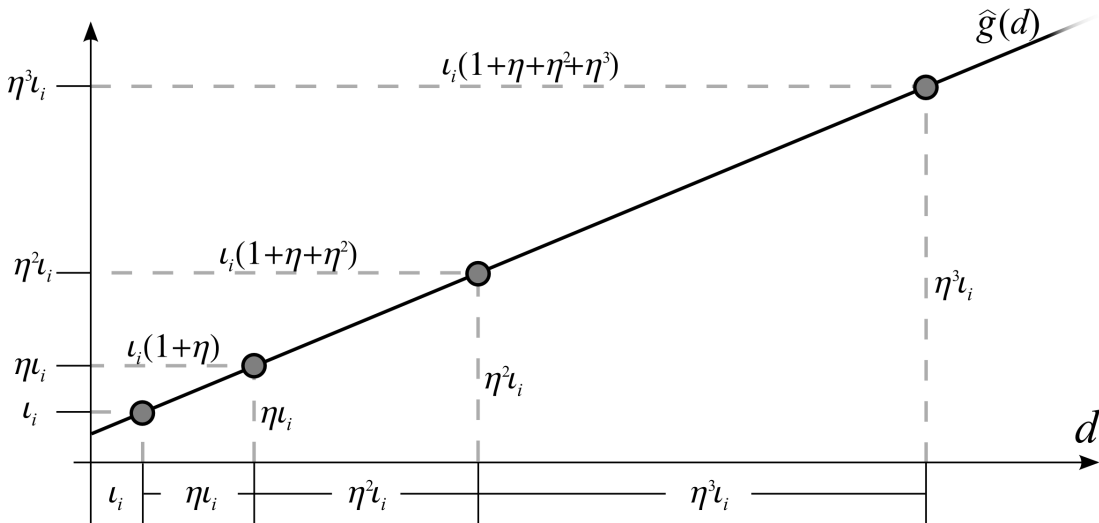
$$\hat{g}_i(d) = (1 - \eta^{-1})d + \eta^{-1}t_i. \quad (4.12)$$

This expression interpolates all values given by Equation 4.10. Since its second derivative is zero everywhere in the open interval  $(0, \infty)$ , the integral is zero, and so it is the global minimizer of Equation 4.11. Figure 4.8 illustrates the situation.

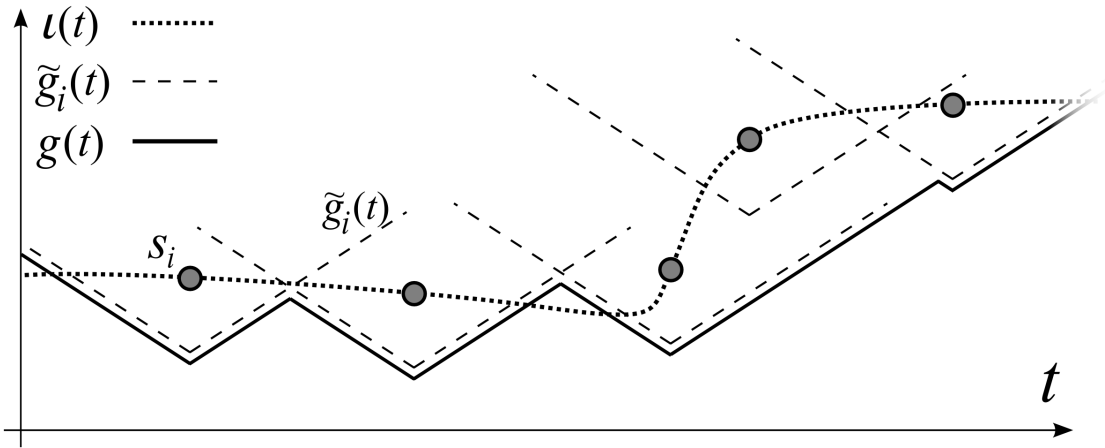
Finally, let  $g(\mathbf{x}) = \min_i \tilde{g}_i(\mathbf{x})$ . Each  $\hat{g}_i$  is clearly Lipschitz, and so are the  $\tilde{g}_i$ . Since the minimum of a set of Lipschitz functions is Lipschitz, a  $g$  constructed in this way will be Lipschitz, regardless of the values of  $t_i$  or the sampling density of  $s$ . Notice that the Lipschitz order is directly related to the allowed rate of change of triangle edges. Figure 4.9 illustrates a plot of an example guidance field.

### 4.3.2 Sampling Condition

The guidance field construction described in the previous section offers an ideal way to control the mesh gradation of the output. A fundamental problem, though, is how to determine whether



**Figure 4.8:** Illustration of  $\hat{g}_i(d)$ , the function that defines the correct edge size as a function of the distance to a sample  $s_i$  of the surface.

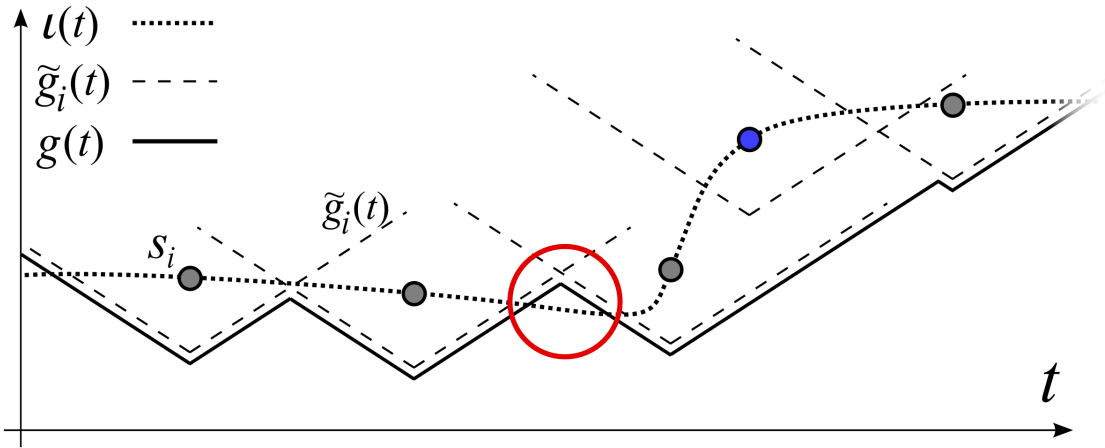


**Figure 4.9:** The guidance field  $g(t)$  on a curve  $t : \mathbb{R} \rightarrow S$ . Note that the functions are plotted against the parameter  $t$ , not the curve itself.  $g(t)$  is the minimum over all  $\tilde{g}$ . At each sample point  $s_i$ ,  $\tilde{g}_i$  is at its minimum, and grows linearly as the distance from  $s_i$  increases. Since each  $\tilde{g}_i$  is Lipschitz, so is  $g(t)$ .

a surface has been appropriately sampled. The guidance field  $g$  is required to bound  $\iota$  below, over all of the points on the surface. However, with an arbitrarily sampled set  $s$ , this may not be the case as illustrated in Figure 4.10. The consequence of  $g$  not bounding  $\iota$  is that there will be areas on the surface where the edge length created by the advancing front will be larger than the curvature of the surface allows, preventing the local features from being captured accurately in the output mesh. Fortunately, since  $\tilde{g}_i(s_i)$  conservatively bounds  $\iota(s_i)$ , an infinite sampling density is not required to enforce a bound of  $\iota$  on  $g$ . In this section, it is shown how a sufficient sampling density can be computed. A guidance field is called *sufficient* when  $g$  is less than  $\iota$  at every point on the surface.

There are two aspects of the guidance field to take note of. First, note that it can be made more conservative by simply decreasing the ideal edge size for a sample point when it is inserted into the guidance field. This will never cause a sufficient guidance field to become insufficient, but it may cause an insufficient guidance field to become sufficient. Second, note that given a guidance field sample  $s_i$ , and another point  $\mathbf{x}$  on the surface where  $|\mathbf{x} - s_i| \leq \iota(s_i)$ , then  $\tilde{g}_i(\mathbf{x}) \leq \iota(s_i)$ . Assume that the minimum of  $\iota$  can be computed over all of the points in the surface, and call it  $\lfloor \iota \rfloor$ . The guidance field samples can now be redefined to be in terms of  $\lfloor \iota \rfloor$ , rather than the  $\iota_i$  originally associated with the samples (i.e., set  $\iota_i = \lfloor \iota \rfloor$  for all  $i$ ). This has the effect of making the guidance field more conservative, as in the first note above. It is now clear that the guidance field is sufficient if there are no points  $\mathbf{x}$  in the surface such that there are no guidance field samples  $s_i$





**Figure 4.10:** Note that if the sampling  $s_i$  of the surface is too coarse,  $g(t)$  might not bound  $l(t)$  (as shown in the region inside the red circle). Section 4.3.2 shows how to provably prevent this. Additionally, some samples will not influence  $g(t)$  (in the figure, the sample shown in blue). Section 4.3.3 shows how to efficiently remove these points.

with  $|\mathbf{x} - s_i| \leq \lfloor \iota \rfloor$ . Stated conversely, the guidance field might not be sufficient if there exists a point  $\mathbf{x}$  such that  $|\mathbf{x} - s_i| > \lfloor \iota \rfloor$  for all guidance field samples  $s$ .

The final remaining issue is how to compute  $\lfloor \iota \rfloor$ . Since  $\iota$  is inversely proportional to  $\kappa_{max}$ , this means that an upper bound on  $\kappa_{max}$  must be found. How this is done depends greatly on the underlying surface definition of  $\mathcal{S}$ . Section 4.6.2 demonstrates how this may be done for isosurfaces defined over regular grids.

Creating a sufficient guidance field in the way described above would have the effect of making  $g$  flat — the edge lengths of the triangulation would be almost uniform over the entire surface. However, the purpose of the guidance field is to allow adaptivity in a controlled way. This issue can be addressed by adaptively subdividing the spatial domain of the surface, and creating sufficient guidance fields in the separate regions independently. The guidance fields are merged by using the new sample points from all separate regions: this ensures that the resulting guidance field is sufficient and Lipschitz-continuous, while still providing adaptivity.

### 4.3.3 Trimming

Constructing the guidance field with sufficiently many samples to capture all of the details in the surface can produce a very large number of points. The size of the set of samples  $s$  directly affects the memory usage and running time of the advancing front algorithm, so it is desirable to remove as many irrelevant samples as possible. Though many of the samples are necessary,

typically a large portion of them provide no information to the guidance field. This happens when the edge length required for the ideal size at a sample  $s_1$  in the guidance field is always smaller than that of another sample  $s_2$ . In this situation,  $s_1$  *dominates*  $s_2$ , as illustrated by the blue point in Figure 4.10. A guidance field is called *minimal* when no samples are dominated by any other samples.

A naïve procedure for culling the unnecessary samples is to compare each sample to every other sample of the guidance field. This is simple since determining if a single point dominates another is straightforward. However, since the guidance fields often initially contain millions of samples, this  $O(n^2)$  algorithm is not practical.

As Figure 4.10 depicts, each of the samples defines a cone in  $\mathbb{R}^{n+1}$ , where  $n$  is the dimension of the embedding space. This analogy translates perfectly into the case of 2-manifolds embedded in  $\mathbb{R}^3$ : each  $\tilde{g}$  defines a right cone in  $[x, y, z, r]$  space (i.e., the cone grows isotropically in  $x, y, z$ , with its axis aligned with the  $r$  axis). By constructing a hierarchical data structure in this space, carefully constructed range queries can be performed, and entire sets of unnecessary samples can be removed in a single query. The most important observation is this: if a point  $s_1$  dominates a point  $s_2$ , then  $s_2$  lies inside the cone defined by  $s_1$ . Also, the *dominates* relation is transitive: if  $s_1$  dominates  $s_2$  and  $s_2$  dominates  $s_3$ , then  $s_1$  dominates  $s_3$ . This means that if  $s_1$  culls a set of points  $t \subset s$ , the cones induced by all  $t_i$  need not be checked for domination of others.

All of the points in the guidance field are initially inserted into a 4-dimensional *kd-tree*. The coordinates for each sample  $s_i$  are  $[s_x, s_y, s_z, s_r]$ , where  $[s_x, s_y, s_z]$  is the sample's location in  $R^3$ , and  $s_r$  is the value of the function  $\tilde{g}_i$  at  $s_i$ . Note now that the set of points  $\{[x, y, z, \tilde{g}_i([x, y, z])] : x, y, z \in R^3\}$ , can be represented by a 4-dimensional cone with apex  $[s_x, s_y, s_z, s_r]$ , axis  $[0, 0, 0, 1]$ , and angle  $\tan^{-1}(\eta/\eta - 1)$ . This cone is completely defined by the sample's location and the user parameters  $\rho$  and  $\eta$ . Finding all of the samples that are dominated by  $s_i$  is now reduced to a *kd-tree* query to find all of the points that lie inside of this cone. Such a query relies on bounding box / cone intersection and point-in-cone tests, both of which are quite straightforward given that the cones are always aligned with the  $r$  axis. When doing the *kd-tree* query, all of the samples that lie inside of the cone are marked. If all of the children of a node have been marked, the node is also marked. This effectively prunes off branches of the *kd-tree* from subsequent queries.

Ideally, the cones of the samples that dominate the most samples would be queried first, thus pruning off large parts of the tree early in the process. However, since this information is not known or easily computed, a heuristic is used instead. Note that a given sample will never be dominated by another sample with a larger ideal edge size. By sorting the samples by their

associated ideal sizes and doing the queries in ascending order, culling can be performed in about 10 seconds for a million samples. When more than this many samples are present in the original guidance field, it can be recursively subdivided until the culling can be performed on a smaller subset. The pseudocode in Figure 4.11 summarizes the algorithm.

This trimming procedure typically removes a large percentage of the sample points, resulting in much lower memory usage and significantly faster guidance field queries. Table 4.1 shows some of the results of culling the guidance field. Notice that even though culling requires additional processing, the total running time improves in all cases. Also, notice how in some cases, less than 2% of the points remain in the guidance field after culling. Figure 4.12 shows the location of the guidance field samples in space before and after culling. It is clear that the samples in the high curvature areas are the most significant, and tend to dominate the low curvature samples in their vicinity. Surprisingly, this effect is quite nonlocal, which accounts for the drastic reduction in the total sample count.

#### 4.3.4 Approximation Error

An important consequence of adhering to the edge sizes specified by the guidance field is that the error between  $S$  and the output mesh can be bounded. The maximum Hausdorff error between the two surfaces can be bounded by

```

CULL( $s, \rho, \eta$ )
1   $tree \leftarrow kd-tree(s)$ 
2  Sort( $s, t(s)$ )
3  for  $0 \leq i \leq |s|$ :
4      do if Not(Marked( $s_i$ ))
5          then  $c \leftarrow Cone(s_i, \rho, \eta)$ 
6              MarkIfInside( $tree, c$ )
7  for  $0 \leq i \leq |s|$ :
8      do if Marked( $s_i$ )
9          then Discard( $s_i$ )

```

**Figure 4.11:** Pseudo-code for trimming the guidance field.

**Table 4.1:** Sample of results of guidance field culling. “before” refers to running time without culling, and “after” refers to running time after culling. There are more initial samples in the guidance field with smaller values of  $\rho$  for isosurface extraction due to the adaptive sampling. The remesh examples have a constant number because only the input vertices are used.

	$\rho$	# Samples	# Remaining	Before	After
Silicium (Isosurface)	0.8	122K	2.52K	0:54	0:44
	0.5	143K	5.62K	1:25	1:05
	0.3	179K	11.8K	2:14	1:37
	0.2	228K	20.4K	3:15	2:46
Skull (Isosurface)	0.8	362K	6.17K	2:45	2:01
	0.5	430K	12.5K	3:39	3:00
	0.3	554K	28.8K	5:34	4:54
	0.2	719K	58.5K	7:44	7:18
Bunny (Remesh)	0.8	34.8K	1.69K	0:15	0:12
	0.5	34.8K	3.19K	0:22	0:20
	0.3	34.8K	6.12K	0:42	0:35
	0.2	34.8K	9.46K	1:00	0:56
Pensatore (Remesh)	0.8	997K	12.2K	4:36	2:58
	0.5	997K	21.4K	5:38	4:10
	0.3	997K	39.6K	7:40	6:50
	0.2	997K	63.4K	11:24	10:28

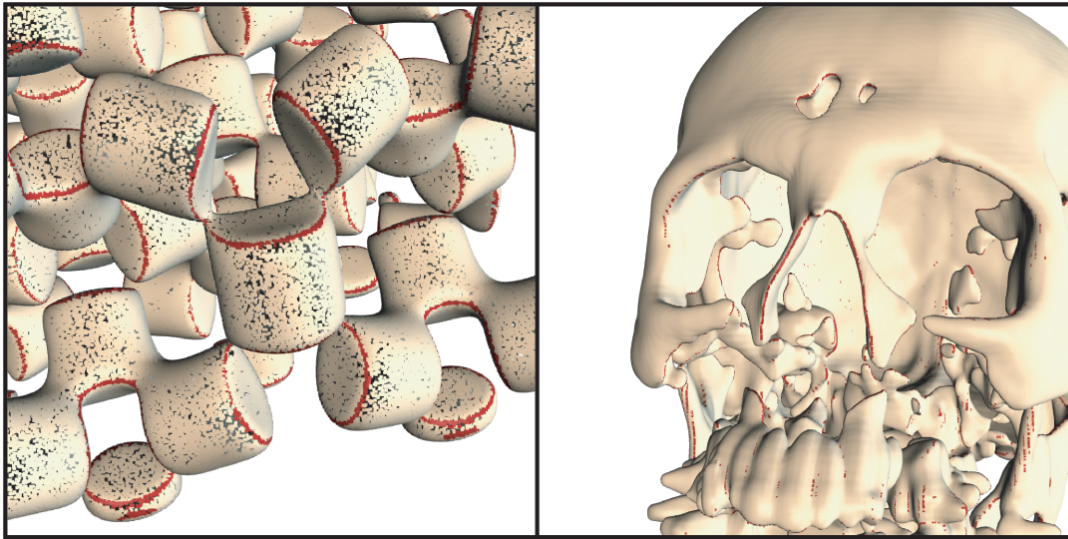
$$\xi = \left(1 - \sqrt{\frac{1 + 2 \cos \rho}{3}}\right) \frac{1}{2 \sin(\rho/2)} \quad (4.13)$$

$$\varepsilon(\rho) = e_{max} \xi, \quad (4.14)$$

where  $e_{max}$  is the largest edge in the generated triangulation. Note that the bound is directly controlled by the user parameter  $\rho$ . The proof of this follows that of Scheidegger et al. [122], fixing a small mistake. The derivation relates an edge length  $e$  to the radius of curvature  $r$  that would generate such a edge when sized by the ideal edge size function  $\iota$ . The error is then the maximum distance between a sphere of radius  $r$  and an equilateral triangle with vertices on the sphere and edge lengths  $e$ .

This bound can be used to compute the maximum distance between any point  $\mathbf{x} \in \mathcal{S}$  and the output mesh. Since the guidance field has been constructed such that  $\mathbf{x}$  is never crossed by a triangle with edges longer than  $\iota(\mathbf{x})$ , Equation 4.14 can be extended to be defined over  $\mathcal{S}$  with

$$\varepsilon(\rho, \mathbf{x}) = \iota(\mathbf{x}) \xi. \quad (4.15)$$

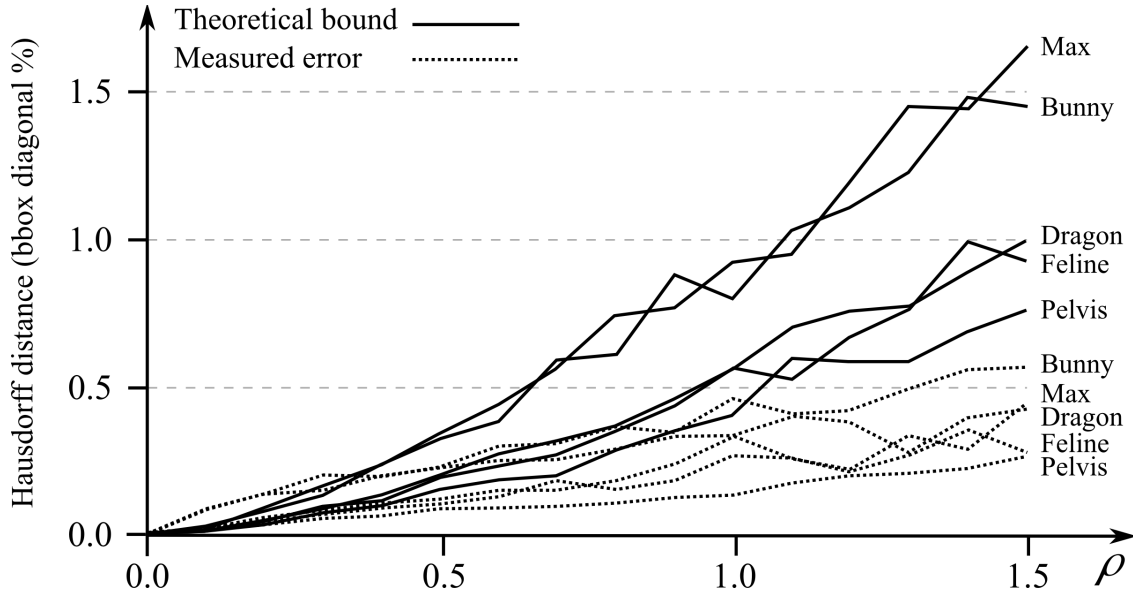


**Figure 4.12:** Two models are rendered using only the points in the guidance field — the black speckles are simply the inside of the surface without any illumination. After trimming, all the tan-colored points are removed from the data structures, leaving only the red points for querying.

This local error bound will be used to make the front interference detection robust.

There are two assumptions in these bounds. First, it is assumed that the triangle edges are always less than or equal to the size specified by the guidance field. This may not hold for connection triangles, where the edge lengths are not precisely controlled. To address this, edges of connection triangles could be split until they are sufficiently small. However, this creates poor triangles where the advancing fronts meet. In practice, the edge size requirement is relaxed to improve the quality, and the bound has still been satisfied in experiments. Second, the input surface is assumed to be smooth, which is not the case for triangle meshes. However, when the input mesh is composed of sufficiently small triangles relative to the output mesh, they again hold in practice.

Figure 4.13 shows the predicted error and the measured error for a number of remeshes. Only triangle meshes are considered as input because their explicit nature allows simpler evaluation of the error than other surface definitions [35]. When larger  $\rho$ s are used, and thus larger triangles are created, the measured error is about half of the predicted error. However, for small  $\rho$ s, the measured error is greater than the theoretical bound. This happens because of the misalignment of a remeshed triangle across edges of the input triangles. Most of these issues can be circumvented by identifying creases and triangulating the piecewise smooth patches of the surface.



**Figure 4.13:** A comparison of the predicted error and measured error for a set of remeshes, in percent of the bounding box diagonal. The measured error is less than half the predicted error when the input mesh is sufficiently smooth compared to the output mesh.

An explanation for the measured error being significantly smaller than the theoretical bound depends on the relationship between the actual edge sizing function  $g$ , and the ideal edge size function  $\iota$ . The largest edges in the output mesh are in the lowest curvature areas of  $\mathcal{S}$ . Additionally, the guidance field in low curvature areas is typically restricted by the Lipschitz requirement on edge sizes, rather than the ideal edge size at those points. This creates a situation where the largest edge in the output mesh was probably sized by a  $g$  that was significantly smaller than  $\iota$  at that point. Since the bound in Equation 4.14 is derived from the assumption of triangles being sized inversely proportional to the curvature as specified by  $\iota$ , it is not surprising that it is generally conservative.

### 4.3.5 Triangle Quality

Approximately half of the triangles in the output mesh are free triangles. This is because the creation of every vertex of the output coincides with the creation of a free triangle, and connection triangles are delayed until no more free triangles can be created. Strong bounds exist on the aspect ratio of every free triangle in the output mesh. The reason for this is that the Lipschitz property of the guidance field can be used to show that the edge lengths for these triangles also satisfy a Lipschitz condition. For each of the free triangles, the ratio of the longest to shortest edge will

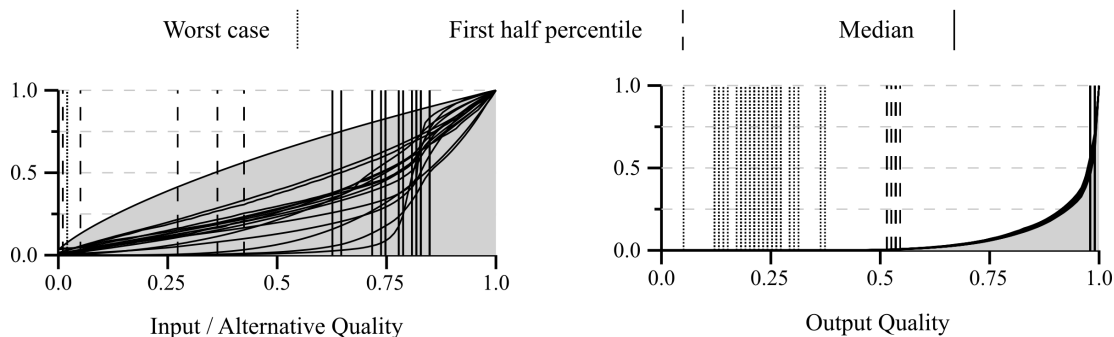
always be less than or equal to the user parameter  $\eta$ . When  $\eta$  is close to one, all free triangles will be close to equilateral.

There are no analytic bounds on the quality of the connection triangles. This is an inherent obstacle for advancing front algorithms. Since an existing vertex must be used, the flexibility required to create exactly the triangle desired is not available. Hence, the error bound for free triangles does not hold for connection triangles, though they typically exhibit very high qualities as well. In the areas with connection triangles, where the fronts merge and split, heuristics are used to determine which vertices to connect [76, 127]. In practice, bad triangles rarely happen.

One remarkable aspect of the algorithm is that it produces a very consistent distribution of triangle qualities, despite not having bounds on quality of the connection triangles. This distribution can be seen in Figure 4.14, which includes all of the meshes generated by the advancing front algorithm with exceedingly similar quality histograms. The histograms are created using the ratio of incircle to circumcircle of a triangle (normalized to  $[0, 1]$ ) as a measure of triangle quality. Notice that the first half percentile of all of the advancing front meshes have ratio 0.5 or better. 99.5% of the triangles have acceptable quality, and all of them have the median ratio within 3% of optimal.

## 4.4 Surface Triangulation

Once the guidance field has been constructed, the triangulation of the surface can begin in earnest. The algorithm must be initialized with at least one front on each connected component



**Figure 4.14:** Cumulative histograms show the quality distribution of triangles in a mesh. The quality of a triangle is measured by the ratio of its incircle to circumcircle radii, normalized so that the best ratio is 1.0. On the left, input meshes and the results of competing triangulation methods are shown. On the right are histograms from all of the meshes generated by the advancing front algorithm.

of  $\mathcal{S}$ . This is required since the fronts are advanced with local operations that restrict them to the connected components on which they are created.

#### 4.4.1 Initialization

The algorithm is initialized by first gathering the boundaries and any features. Identifying features is a hard problem for arbitrary surface definitions, so they are simply treated as input to the algorithm. That is not to say that it could not be automated in some cases. For example, sharp intersection curves from CSG operations between meshes can automatically be identified, and advanced techniques could be used for identifying ridges and valleys in meshes [64], and point-set surfaces [40, 51]. To preserve the features and boundaries accurately, it must be ensured that no triangles are created which cross them. This is done by initializing the advancing fronts *at* the features and boundaries, and growing them away.

After the feature and boundary space curves have been identified, they are each resampled with edges sized by the guidance field. This is done with an algorithm analogous to the advancing front algorithm, albeit in one lower dimension. By using the same guidance field for both resampling the boundaries and features, and for the advancing front triangulation, it is ensured that all of the features are accurately represented while still maintaining the constraints on the triangle quality. The resampled curves then become initial fronts for the triangulation. Since these boundary and feature fronts provide seeds for the connected components, individual seed edges are not required. For any connected components without boundaries or seeds, a single initial edge is created at a random point.

The advancing front algorithm assumes that all of the fronts are closed loops. It is clear that this will be the case for all boundaries of manifold surfaces, but may not be as clear for sets of sharp features. Conceptually, sharp feature curves on the surface can be thought of as cutting the surface, with the initial fronts placed along the boundary of these holes. While the vertices along the feature may be shared by different fronts (or different parts of the same front), they are independent and may have different normals. Treating them in this way allows arbitrary feature networks to be placed on the surface.

#### 4.4.2 Advancing Fronts

The triangulation proceeds by choosing an edge from one of the fronts. The guidance field is evaluated at the endpoints of the edge to determine the shape of the new triangle, and a tentative point is created for the location of the new vertex. This point is then projected onto  $\mathcal{S}$  with  $\mathcal{P}$ . The resulting triangle is then checked for interference with other fronts. When there is no



interference, the new vertex and a *free* triangle are outputted and the front is grown. When there is interference, the new vertex is discarded and a *connection* triangle is outputted that uses a vertex that has already been placed. If the new vertex was part of the same front as the growing edge, that front is split. If it is part of a different front, the two are merged. As the fronts advance, every free triangle is placed before any connection triangles. The triangles are also prioritized within each class, favoring those that would result in larger ratios of the incircle radius to the circumcircle radius. A front is closed when it only contains three vertices, which are used in a single triangle. The triangulation is complete when there are no longer any fronts to advance. This algorithm is illustrated as pseudo-code in Figure 4.15.

### 4.4.3 Front Interference Detection

One of the most challenging aspect of implementing an advancing front algorithm is to determine when a free triangle encroaches on a front in the triangulation. The fronts are only linear approximations of curves on the surface, so they can cross each other without strictly intersecting. If this happens, more than one triangle will cover a patch of the surface, making it

```

TRIANGULATE( $\mathcal{S}, \rho, \eta$ )
1   $g \leftarrow \text{GENERATE-SAMPLES}(\mathcal{S}, \rho, \eta)$ 
2   $\text{CULL}(g, \rho, \eta)$ 
3   $\text{Active} \leftarrow \text{INITIAL-FRONTS}(\mathcal{S}, g)$ 
4  while  $|\text{Active}| > 0$ 
5      do  $\text{edge} \leftarrow \text{GET-BEST-EDGE}(\text{Active})$ 
6           $\text{tri} \leftarrow \text{TRIANGLE-FOR-EDGE}(\text{edge}, g)$ 
7           $\text{front} \leftarrow \text{FRONT-FOR-EDGE}(\text{edge})$ 
8          if  $\text{OK-TO-ADD-TRIANGLE}(\text{tri})$ 
9              then  $\text{ADD-TRIANGLE-TO-FRONT}(\text{front}, \text{tri})$ 
10             else  $\text{other} \leftarrow \text{GET-INTERFERING-FRONT}(\text{tri})$ 
11                 if  $\text{other} = \text{front}$ 
12                     then  $(f1, f2) \leftarrow \text{SPLIT}(\text{front})$ 
13                          $\text{REMOVE-FRONTS}(\text{Active}, \{\text{front}\})$ 
14                          $\text{ADD-FRONTS}(\text{Active}, \{f1, f2\})$ 
15                 else  $\text{new-front} \leftarrow \text{MERGE}(\text{front}, \text{other})$ 
16                      $\text{REMOVE-FRONTS}(\text{Active}, \{\text{front}, \text{other}\})$ 
17                      $\text{ADD-FRONTS}(\text{Active}, \{\text{new-front}\})$ 

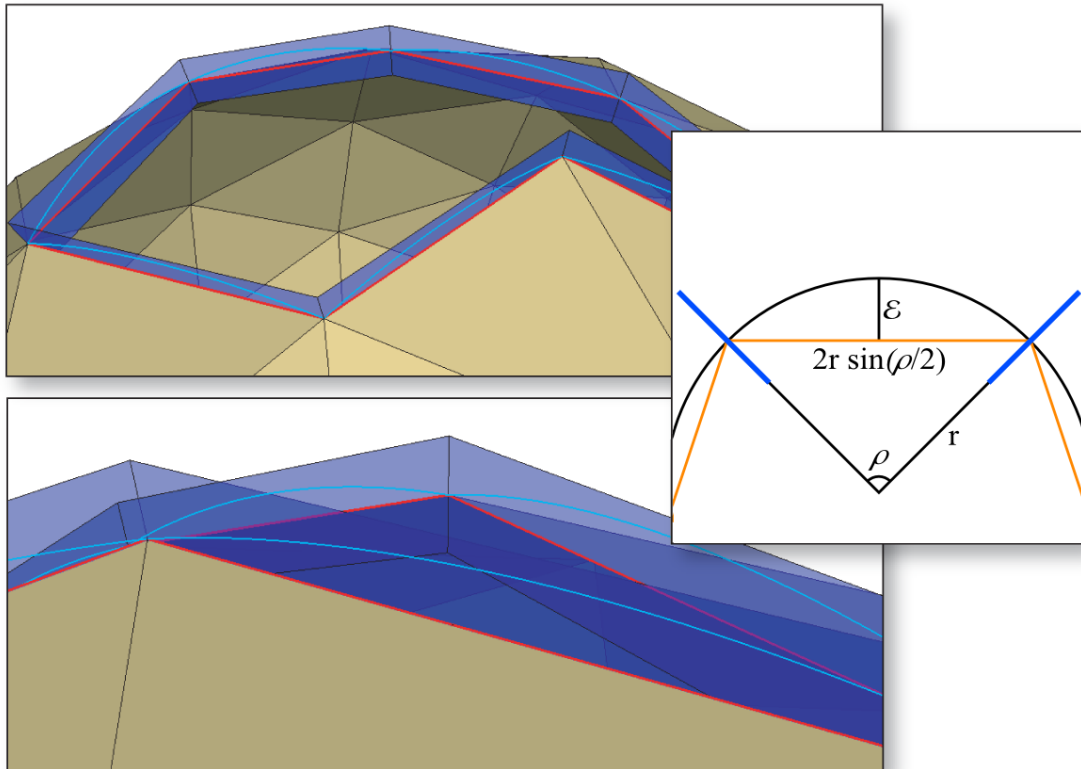
```

**Figure 4.15:** Pseudo-code for the triangulator at the core of the advancing front algorithm.

topologically incorrect. A robust solution to this fundamental issue is to extend the front curve in the surface normal direction, creating *fences* that do not allow any fronts to pass, illustrated in Figure 4.16. To determine if a free triangle is encroaching, it is tested for intersection with the fences. Equation 4.15 gives the maximum distance between the input surface and the output mesh at a point  $\mathbf{x} \in \mathcal{S}$  as  $\varepsilon(\rho, \mathbf{x})$ . To guarantee that the fronts will not cross, the fence heights simply need to be greater than  $\varepsilon(\rho, \mathbf{v})$  at each front vertex  $\mathbf{v}$ .

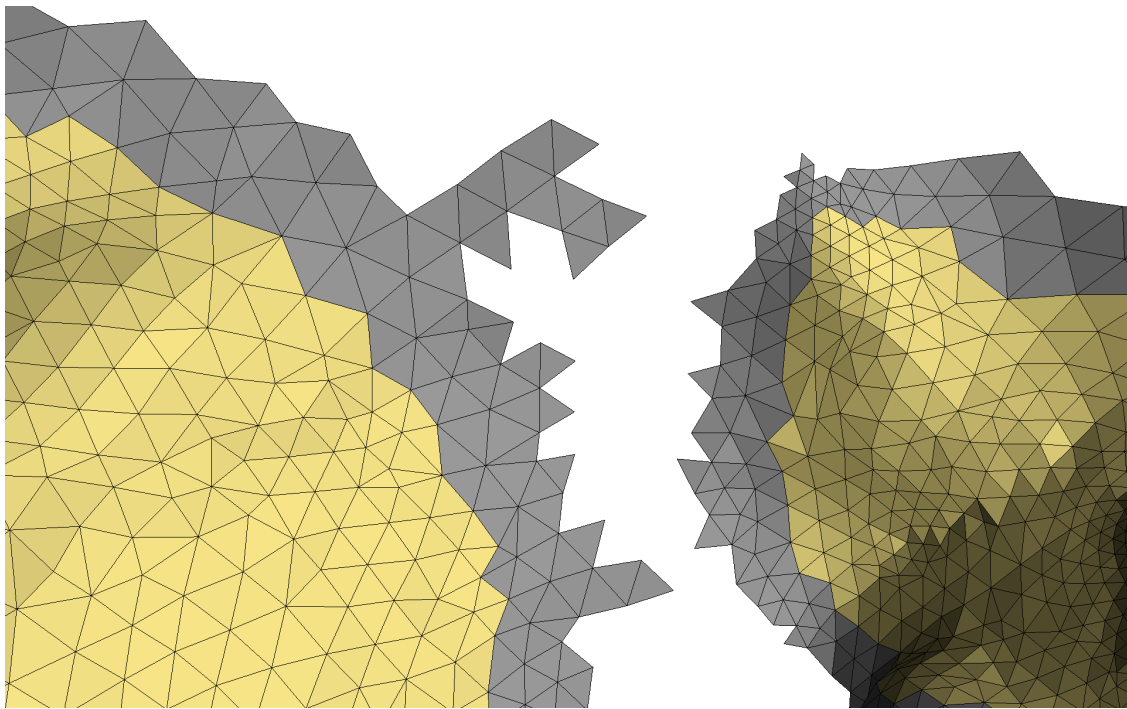
## 4.5 Streaming Output

Since advancing front algorithms generate vertices and triangles that grow across the surface, they can be thought of as creating streaming meshes [72], with vertices being finalized when they are no longer part of any fronts. Streaming the output makes the memory use proportional to the number of vertices in the fronts, rather than proportional to the total size of the output. This allows meshes to be generated that are much larger than the available memory.



**Figure 4.16:** To robustly detect front interference, a set of *fences* are used. These are extensions of the front curve in the normal direction of the surface. The bound on the Hausdorff error is exploited to determine the correct fence height — the inset on the right shows the argument in two dimensions.

In straightforward advancing-front techniques poor connection triangles can be created, which leads to a few bad triangles in an otherwise good mesh. The streaming nature of these algorithms can be used to improve the triangulations. This work adapts the “virtual front” idea of Silva and Mitchell [127] to greatly improve the poorest quality triangles created. This algorithm works by keeping a small band of triangles behind the advancing fronts in memory (see Figure 4.17). As the fronts advance, new vertices and triangles are inserted into the band. When all three vertices of a triangle no longer have any neighbors on a front, the triangle is finalized, at which time it is outputted. This only increase memory usage by a small factor of the length of the fronts. Performing edge flips within this small band of triangles can substantially improve the worst case triangle quality. An edge flip is performed when two conditions are satisfied. First, the two new triangles must have a larger minimum angle than the original two triangles adjacent to the edge. This is the main objective of performing edge flips. Second, the normals of the new triangles must not be in opposition to the normals of the original triangles. This is necessary since flipping edges based solely on the first condition can cause geometric overlaps.



**Figure 4.17:** Triangles are kept in memory until all three vertices have no neighbors on any of the advancing fronts. This provides a band (shown in gray) in which edge flips are performed to improve triangle quality.

Performing edge flips on the triangulation as it is generated typically does not significantly change the histogram of the triangle qualities. However, it usually greatly improves the worst case. This shows that the advancing front algorithm typically produces very good vertex locations and connectivity. Occasionally a poor connection triangle is created, which can usually be fixed with very localized edge flips.

## 4.6 Implementation

The advancing front algorithm was designed to mesh a large class of surfaces, and the design of the prototype implementation reflects this. The main triangulation module handles the front advancement, with all intersection tests, and merging and splitting of fronts. A set of abstract classes defines the interface of the surface definition, and so the triangulation module is unaware of the underlying surface type. This simplifies the implementation of more advanced operations, such as mixed-mode remeshing, where there is not a single input surface type. A more thorough description of the software architecture of the prototype implementation is available in the appendix.

This section discusses details of the implementation of several important aspects of the advancing front algorithm. Since the guidance field may be constructed from a large set of samples  $s$ , and must be evaluated at each point in the output mesh, an efficient procedure the evaluation must be available. Additionally, it may not be clear how to generate the guidance field samples in a way that satisfies the sampling condition. A way of doing this for isosurfaces defined on regular grids is described, which makes use of interval arithmetic. Also, it is very important to give the algorithm a set boundaries for the initial fronts that are consistent with the surface, and a method for doing this in a provable way is described. Adaptations made to the algorithm to accommodate out-of-core surfaces, and ways in which modern multicore CPUs can be taken advantage of are also discussed.

### 4.6.1 Guidance Field Evaluation

Evaluating the guidance field edge sizing function is a procedure that is central to the algorithm. The simple way to do this would be to evaluate  $\tilde{g}_i$  for each sample in  $s$ , and take the minimum. The time complexity of this is  $O(|s|)$ . This is much too slow for a function that must be evaluated once for every vertex in the output mesh. Note that all  $\tilde{g}_i$ s grow at the same rate (i.e.,  $1 - \eta^{-1}$ ). This leads to the following simple and efficient procedure for evaluating  $g$  at any given point  $\mathbf{x}$ . First assume that an upper bound  $\gamma$  of  $g(\mathbf{x})$  is known. Then it can be shown that for all  $i$  such that  $\|s_i - \mathbf{x}\| > \gamma/(1 - \eta^{-1})$ ,  $\tilde{g}_i(\mathbf{x}) > \gamma$ . That is,  $\tilde{g}_i$  will not change the current estimate of

$g(\mathbf{x})$ . This is because if the ideal size at  $s_i$  is zero (the minimum possible), and  $s_i$  is sufficiently far away from  $\mathbf{x}$ , the constant rate of growth of  $\tilde{g}_i$  will force it to be larger than  $\gamma$ . Therefore, there is no need to evaluate  $\tilde{g}_i$  when  $\|s_i - \mathbf{x}\| > \gamma/(1 - \eta^{-1})$ . These observations lead to the procedure outlined in Figure 4.18 for evaluating  $g(\mathbf{x})$ , which examines the fewest points possible by using a *kd*-tree to extract the points  $s_i$  in ordered distance from  $\mathbf{x}$ . The time complexity of this algorithm is  $O(\log(|s| + t))$ , where  $t$  is the number of samples that are actually examined. If all of the points are needed, this will be slightly slower than the straightforward algorithm. However, this situation does not occur in practice, and in general  $t \ll |s|$ .

#### 4.6.2 Bounding Curvature

Creating a sufficient guidance field requires bounding the ideal edge size function over the surface, and hence the curvature. Bounding the curvature of a surface is difficult in general, and depends greatly on the underlying definition of the surface. For example, it would be very difficult for nonlinear MLS surfaces, which define the surface indirectly through a projection procedure, and involves a nonlinear optimization. Since the differential properties are difficult to compute, even at a single point on the surface, it is currently not possible to bound the curvature over a region of the surface.

Isosurfaces of regular grid implicit functions defined by tensor product splines, including B-splines and Catmull-Rom splines, are a much simpler way of defining surfaces. They are simply the preimage of piecewise cubic polynomials, with the added advantage of being defined over regular lattices. This simple nature makes it possible to bound the curvature. However, the curvature is still a complicated function of the first and second order partial derivatives of the implicit function, and finding the maximum involves solving a system of nonlinear equations. The approach taken here is to first derive a simpler expression that gives an upper bound on

```

 $g(\mathbf{x})$ 
1   $\gamma \leftarrow \infty$ 
2  repeat
3       $s_i \leftarrow \text{NEXTCLOSESTPOINT}(\mathbf{x})$ 
4       $\gamma \leftarrow \min(\gamma, \tilde{g}_i(\mathbf{x}))$ 
5  until  $\|s_i - \mathbf{x}\| > \gamma/(1 - \eta^{-1})$ 
6  return  $\gamma$ 

```

**Figure 4.18:** An efficient algorithm to evaluate the guidance field at a given point makes use of a *kd*-tree to traverse the guidance field samples  $s$ .

the curvature. An upper bound is then found on this new expression by making use of *interval arithmetic*. Interval arithmetic provides a generic tool for placing bounds on a mathematical expression, but often produces very loose bounds. The regular structure of the tensor product splines will be taken advantage of to apply interval arithmetic in a hierarchical way, and create tight bounds.

To bound the curvature of an isosurface, first note that  $\kappa_{max}$  is defined as the absolute value of the largest eigenvalue of the geometry tensor  $G$  (Equation 4.4). Notice that this is exactly the spectral radius of  $G$ :  $\kappa_{max} = r(G)$ , so the problem of bounding curvature becomes one of bounding the spectral radius of  $G$ . Now recall the *submultiplicative* property of matrix norms:

$$\|A \cdot B\| \leq \|A\| \cdot \|B\|. \quad (4.16)$$

Since the spectral radius is a consistent matrix norm, it is clear that

$$r(A \cdot B) \leq r(A) \cdot r(B). \quad (4.17)$$

Using this to expand the definition of the maximum curvature  $\kappa_{max}$  gives

$$\begin{aligned} \kappa_{max} &= r(G) \\ &= r(PHP/|\nabla f|) \\ &\leq r(P) \cdot r(H) \cdot r(P) / |\nabla f|. \end{aligned}$$

Since  $P = (I - nn^T)$ , where  $n = \nabla f / |\nabla f|$ , is a projection matrix, it is easily shown that  $r(P) = 1$ . Simple substitution then gives

$$\kappa_{max} \leq \frac{r(H)}{|\nabla f|}. \quad (4.18)$$

This inequality shows that  $\kappa_{max}$  is intimately related to  $|\nabla f|$  and the Hessian  $H$ . More importantly, it shows that to bound  $\kappa_{max}$  above, it is enough to give a lower bound on  $|\nabla f|$  and an upper bound on  $r(H)$ .

Since bounding these values over all of the points in the isosurface is very difficult without a parameterization, a looser bound can be found by taking it over all of the points in the domain of the implicit function. This relaxation is restrained by recursively subdividing the domain, and only considering the regions that the isosurface passes through. Since it is assumed that the implicit function has nonzero gradient at all the points on the surface and the function is at least  $C^1$  continuous, this procedure is guaranteed to find a finite bound: there is always a tubular neighborhood of the surface where the gradient is nonzero.

Restricting this discussion to implicit functions defined by piecewise cubic trivariate polynomials simplifies the problem of finding an upper bound on  $r(H)$  and a lower bound on  $|\nabla f|$  significantly. However, it is still a difficult problem since finding analytic bounds requires solving systems of nonlinear equations. Instead, interval arithmetic can be used to find the bounds.

#### 4.6.2.1 Interval Arithmetic

Interval arithmetic is a simple but powerful method for computing bounds on a mathematical expression [102]. It produces not the value of the expression, but bounds on the value. This is useful for studying the effects of floating point rounding, computing integrals, and differential equation initial value problems. Interval arithmetic can be used to find an upper bound on the curvature of an isosurface. The interval notation  $[x] = [x_0, x_1] = \{x : x_0 \leq x \leq x_1\}$  will be used. The basic arithmetic operations are:

$$\begin{aligned} [x] + [y] &= [x_0 + y_0, x_1 + y_1] \\ [x] - [y] &= [x_0 - y_1, x_1 - y_0] \\ [x] \cdot [y] &= [\min(x_0y_0, x_0y_1, x_1y_0, x_1y_1), \max(x_0y_0, x_0y_1, x_1y_0, x_1y_1)] \\ [x]/[y] &= [\min(x_0/y_0, x_0/y_1, x_1/y_0, x_1/y_1), \max(x_0/y_0, x_0/y_1, x_1/y_0, x_1/y_1)], 0 \notin [y]. \end{aligned}$$

A well known problem with interval arithmetic is that the bounds can expand very quickly, to the point that they provide little use. This is often the result of dependencies between intervals in the computation. For example, if  $f([x]) = [x]^2$ , the straightforward way to evaluate it would be as  $f([x]) = [x] \cdot [x]$ . If this evaluation is applied to the interval  $[x] = [-1, 1]$ , the result is  $[-1, 1]$ . However, it is clear that  $f$  is nonnegative. Just a single multiplication yields a bound interval twice as wide as it should be, so it can easily be seen how more complicated functions will give very loose bounds. The result of the interval arithmetic is conservative because it does not take advantage of the fact that the variables in  $[x] \cdot [x]$  are the same. This can be addressed by evaluating

$$[x]^2 = \begin{cases} [x_0^2, x_1^2], & x_0 > 0 \\ [x_1^2, x_0^2], & x_1 < 0, \\ [0, \max(x_0^2, x_1^2)], & 0 \in [x] \end{cases}$$

which will produce the exact bounds. This idea can be extended to cubic polynomials to give much tighter bounds than a naïve evaluation would. Note that if  $f$  is monotonic,

$$f([x]) = [f(x_0), f(x_1)], f \text{ increasing} \quad (4.19)$$

$$f([x]) = [f(x_1), f(x_0)], f \text{ decreasing}. \quad (4.20)$$

So if the function  $f$  has no local extrema, accurate bounds can be computed simply by evaluating it at the endpoints of the input interval. A cubic polynomial  $f$  can easily be decomposed into piecewise monotonic sections by solving  $f'(x) = 0$  to find the local extrema. This can be done in a numerically stable way since  $f'$  is a quadratic equation. The interval  $[x]$  can then be split into up to 3 subintervals by those points. A bound on each subinterval can then be found, with the final bound being the union of the bounds on each subinterval.

#### 4.6.2.2 Tensor Product Splines

Tensor product splines provide a way of defining a piecewise cubic function over a regular grid. For each cell in the grid, a  $4^3$  grid of neighboring data samples  $p_{ijk}$  are used to define the function  $f$  within that cell.

$$f(x, y, z) = \sum_{ijk} p_{ijk} b_i(x) b_j(y) b_k(z), \quad (4.21)$$

where

$$b_i(x) = ([x^3, x^2, x, 1]M)_i \quad (4.22)$$

are the spline's basis functions defined by the matrix  $M$ . Splines defined in different ways lead to different properties of the function  $f$ . B-splines will be  $C^2$  continuous across adjacent grid cells, but only approximate the data samples  $p$ . Catmull-Rom splines interpolate the data values, but are only  $C^1$  continuous across grid cells. For B-splines, the matrix  $M$  is

$$M = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}. \quad (4.23)$$

For Cardinal splines with tension  $\tau$ , the matrix  $M$  is

$$M = \begin{bmatrix} -\tau & 2-\tau & \tau-2 & \tau \\ 2\tau & \tau-3 & 3-2\tau & -\tau \\ -\tau & 0 & \tau & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \quad (4.24)$$

A Catmull-Rom spline is a Cardinal spline with  $\tau = 0.5$ . Since each  $b_i$  is just a cubic polynomial, Equation 4.21 can be rearranged into the more natural polynomial form

$$f(x, y, z) = \sum_{ijk} c_{ijk} x^i y^j z^k, \quad (4.25)$$

where  $c_{ijk}$  are constants depending on  $p$  and  $M$ .



Evaluating the partial derivatives of  $f$  is straightforward with Equation 4.25. Differentiating Equation 4.21 can also be accomplished by using different  $M$  matrices for the  $x$ ,  $y$ , and  $z$  components, adjusted to correspond to the appropriate derivatives of  $b_i(x)$ ,  $b_i(y)$ , and  $b_i(z)$ . When done this way, there is no need for specialized evaluation procedures for the partial derivatives, which can simply be treated as new splines with different matrices.

To compute a sufficient guidance field for isosurfaces of tensor product splines, each grid cell is treated independently, with an octree subdivision within each cell. The subdivision is terminated when either  $f(\mathbf{x}) \neq a$  in the node (i.e., the surface does not pass through the region), or a lower bound on  $\iota$  is found that is greater than half the node diagonal length (i.e., a single sample  $s_i$  in the node will be a sufficient guidance field for the entire node). The quality of the bound on  $\iota$  directly affects the depth of the recursion, and thus memory usage and running time. To keep these manageable, steps must be taken to keep the bounds tight.

The first step in producing tight bounds on  $\iota$  is to produce tight bounds on  $f$ . If the upper and lower bounds on  $f$  do not span the isovalue  $a$ , the surface does not pass through the node, so subdivision can be terminated. Note that partial derivatives of  $f$ ,  $D_x f$ ,  $D_y f$ , and  $D_z f$  can be used to produce tighter bounds on  $f$  by taking advantage of monotonicity, as in Equations 4.19 and 4.20. For example, if the lower bound on  $D_x f$  is found to be greater than zero, the minimum value of  $f$  is attained when  $x = x_0$ , and the maximum value is attained when  $x = x_1$ , or if the upper bound on  $D_y f$  is found to be less than zero, the minimum value of  $f$  is attained when  $y = y_1$ , and the maximum value is attained when  $y = y_0$ . In this way, the region searched can be restricted to either a face, and edge, or a corner of the node when searching for a bound on  $f$ . This restriction improves the execution time required to evaluate the bound, and the bound will be optimal when the search region is narrowed to a corner.

It can be seen how a bound on the partial derivatives can improve the bounds on  $f$ . Similarly, the second-order partial derivatives can be used to improve the first-order derivatives, and the third-order partials can improve the second. To take advantage of this, first recall that the goal is to bound Equation 4.18, which includes first-order and second-order partial derivatives of  $f$ . Therefore, the rank 3 tensor of third-order partial derivatives is first computed, and used to place bounds on the Hessian matrix. The Hessian is then used to place tight bounds on the gradient, which is in turn used to tightly bound the function value. The additional time used to compute the third-order partials is made up for through earlier termination of the subdivision.

Equations 4.21 and 4.25 will both evaluate to the same value, but when using interval arithmetic they may yield different bounds. To keep the bounds as tight as possible, and thus minimize

the depth of the subdivision, both methods of evaluating  $f$  (or any partial derivative of  $f$ ) are used. The result is then the union of the two intervals.

Once good bounds on the partial derivatives are found, attention can be turned to using them to find a good bound on  $\kappa_{\max}$ . This requires that the spectral radius of  $H$  is bounded. Since the spectral radius satisfies the property:

$$r(H) \leq \|H^k\|^{1/k}, \quad (4.26)$$

for all  $k \in \mathbb{N}$  and all consistent matrix norms  $\|\cdot\|$ ,  $r(H)$  can be bounded by

$$r(H) \leq \min(\|H^2\|_F^{1/2}, \|H^2\|_\infty^{1/2}), \quad (4.27)$$

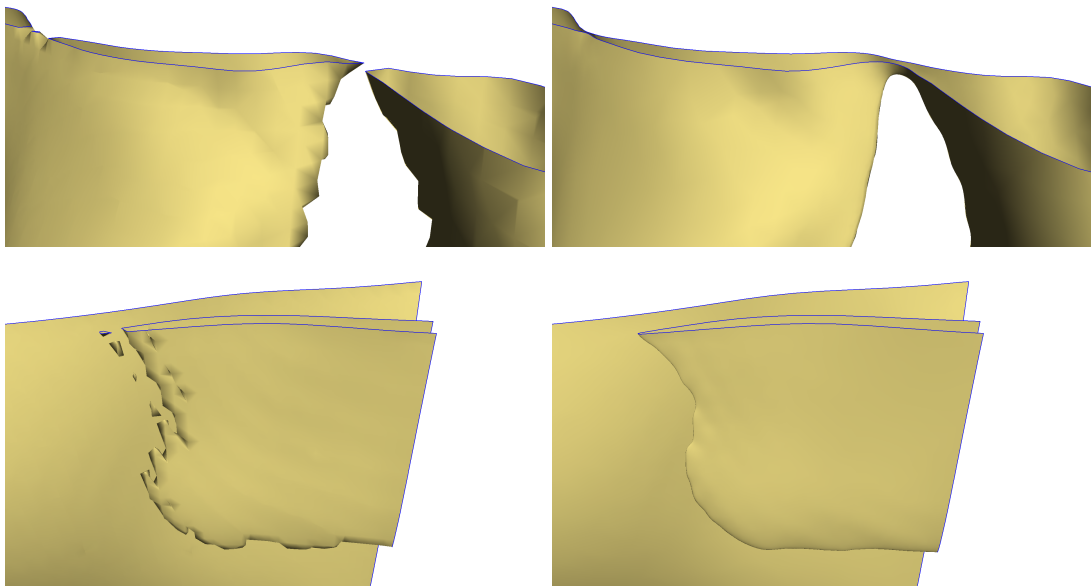
where  $\|A\|_F = \sum_{i,j} a_{ij}^2$  is the Frobenius norm and  $\|A\|_\infty = \max_i \sum_j |a_{ij}|$  is the induced infinity norm. Using  $k = 2$  provides a good trade-off between improved bounds and increased execution time and numerical stability.

Generating a sufficient guidance field in this way greatly improves the execution time and robustness of the algorithm. Adaptively sampling the surface greatly reduces the size of  $s$ , and the number of curvature computations performed. It also removes any a priori requirement on the sampling density of the set  $s$ , so high curvature regions of the surface are always accurately triangulated.

### 4.6.3 Isotopic Initial Boundary Fronts

Initializing the advancing front algorithm with a good set of initial fronts is important for the robustness of the algorithm. If an initial boundary front is used that is not isotopic to the true boundary of the surface, there is no hope for the algorithm to accurately mesh the surface. For input surfaces represented by triangle meshes, the boundaries are explicit, and for point set surfaces, there are no boundaries, so this is not an issue for those representations. For isosurfaces, this is very important. In this case, the algorithm can be initialized by first extracting the surface with Marching Cubes. The seeds for connected components and boundaries can then be found as they would for any other input mesh for remeshing. A significant issue with this is that the MC mesh may have a different topology than the underlying isosurface, as demonstrated in Figure 4.19. For the seed points, this is not a fundamental robustness issue, since fronts from many seed points on a single connected component can simply merge together. However, when the topology of the boundaries is different, the algorithm may fail.

Several extensions to the basic MC algorithm have been proposed to generate meshes that are isotopic to the isosurface [24, 114]. These methods create meshes that can be continuously



**Figure 4.19:** The boundaries of the surface are used as initial fronts. The boundaries of the Marching Cubes mesh (left) are not always isotopic to the isosurface, due to the regular structure of the MC mesh. Additionally, MC may produce spurious connected components. These may still be used for seed points for fronts since they will simply merge together. The output of the advancing front algorithm is shown on the right.

deformed into the isosurface, so the topology is guaranteed to be correct. Here, the method of Plantinga and Vegter [114] is adapted to extract the boundaries of the isosurface, so that the initial fronts for the advancing front algorithm are guaranteed to be correct.

Plantinga and Vegter present an adaptive variant of MC that makes use of interval arithmetic to recursively subdivide the domain until the one of the partial derivatives of  $f$  is monotonic within the node. They show that when the standard MC triangulation is applied to the nodes at this subdivision depth, the resulting mesh is guaranteed to be isotopic to the isosurface. This is done in an adaptive way to prevent an excessively large mesh from being produced.

To extract boundaries isotopic to the boundaries of the surface, each cell face of the boundary regular grid domain can be considered independently. The tensor product spline is restricted to the face in question to create a bivariate cubic polynomial. The problem then becomes one of accurately extracting the isocontours from the cell face. This is done with a two-stage procedure. First, it is determined how many subdivisions are needed at the finest level. A simple quad-tree subdivision is done, using the termination condition of Plantinga and Vegter. Then, the entire

cell face is uniformly divided into a grid corresponding to the deepest level of the quad-tree, and Marching Quads is applied to extract the boundary contour. Uniformly dividing the cell face simplifies the implementation by obviating the need for an adaptive version of Marching Quads. Treating each cell face independently prevents high curvature areas of the boundary from impacting the subdivision required at distant low curvature areas. Since the implicit function is a cubic polynomial, it can have at most three intersections with any edge of the cell face. The result of the contour extraction on each cell face results in a set of contours with up to three endpoints on each edge of cell face, plus additional contours contained entirely within the cell. Any entirely contained contours are added directly to the set of boundaries. Contours that span cell faces are connected to the endpoints of adjacent cells, similar to how the independent triangulations of cells in Marching Cubes are merged together at the shared vertices along the cell edges. Surface boundaries found in this way are guaranteed to maintain the robustness of the advancing front algorithm.

#### **4.6.4 Out-of-Core**

As datasets become increasingly large, algorithms for processing them in an out-of-core way become increasingly important (e.g., [21, 26, 32, 50]). The streaming nature of the advancing front algorithm lends itself well to being adapted to out-of-core input surface formats. As a demonstration, it has been modified to accommodate gigantic isosurfaces defined on regular grids.

Several changes were made to cope with the large amounts of data. The output must be written in a format that can easily be input into other tools designed to work with gigantic meshes. Since the input is too large to fit in physical memory, it must be streamed as well. Having only a portion of the input surface in memory at any given time introduces new issues that must be addressed with the main triangulation component of the algorithm. Specifically, the boundary fronts must be constructed in an incremental way, and the seeding of connected components must be considered.

##### **4.6.4.1 Streaming Input**

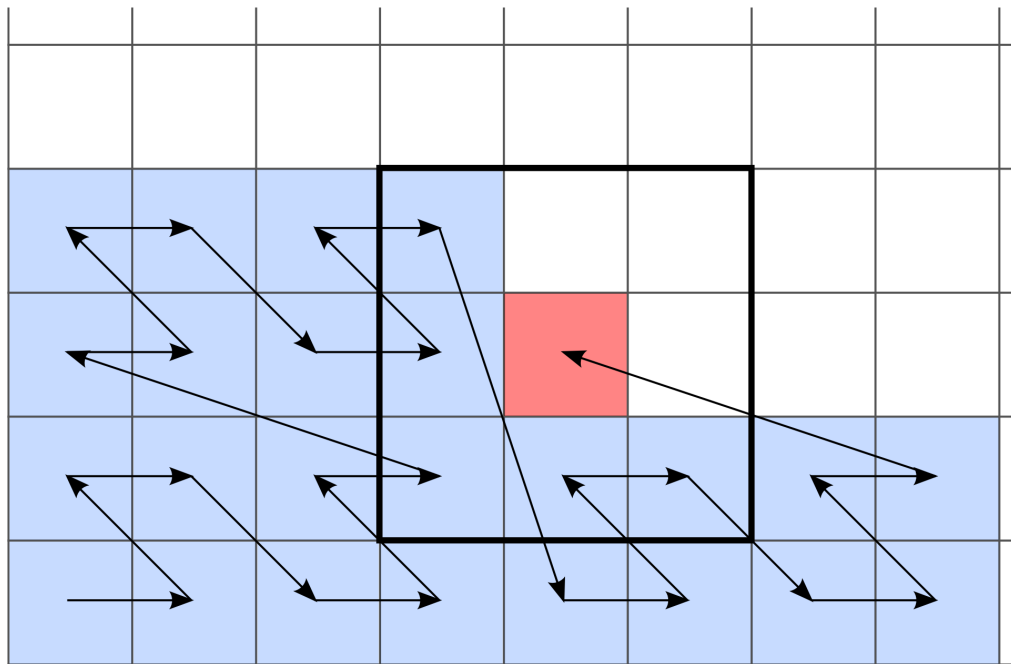
To more easily manage the output mesh, it is written in the binary streaming format proposed by Isenburg and Lindstrom [72]. This enables easy processing such as compression and simplification, as well as interaction and visualization of the output mesh.

Sophisticated methods are available for streaming implicit functions for isosurface extraction, such as that of Mascarenhas et al. [94]. However, the construction of the data structure in their work is performed in-core, and the streaming is quite coupled with the isosurface extraction. Instead, a simpler approach is taken. The data are split up into a number of smaller blocks

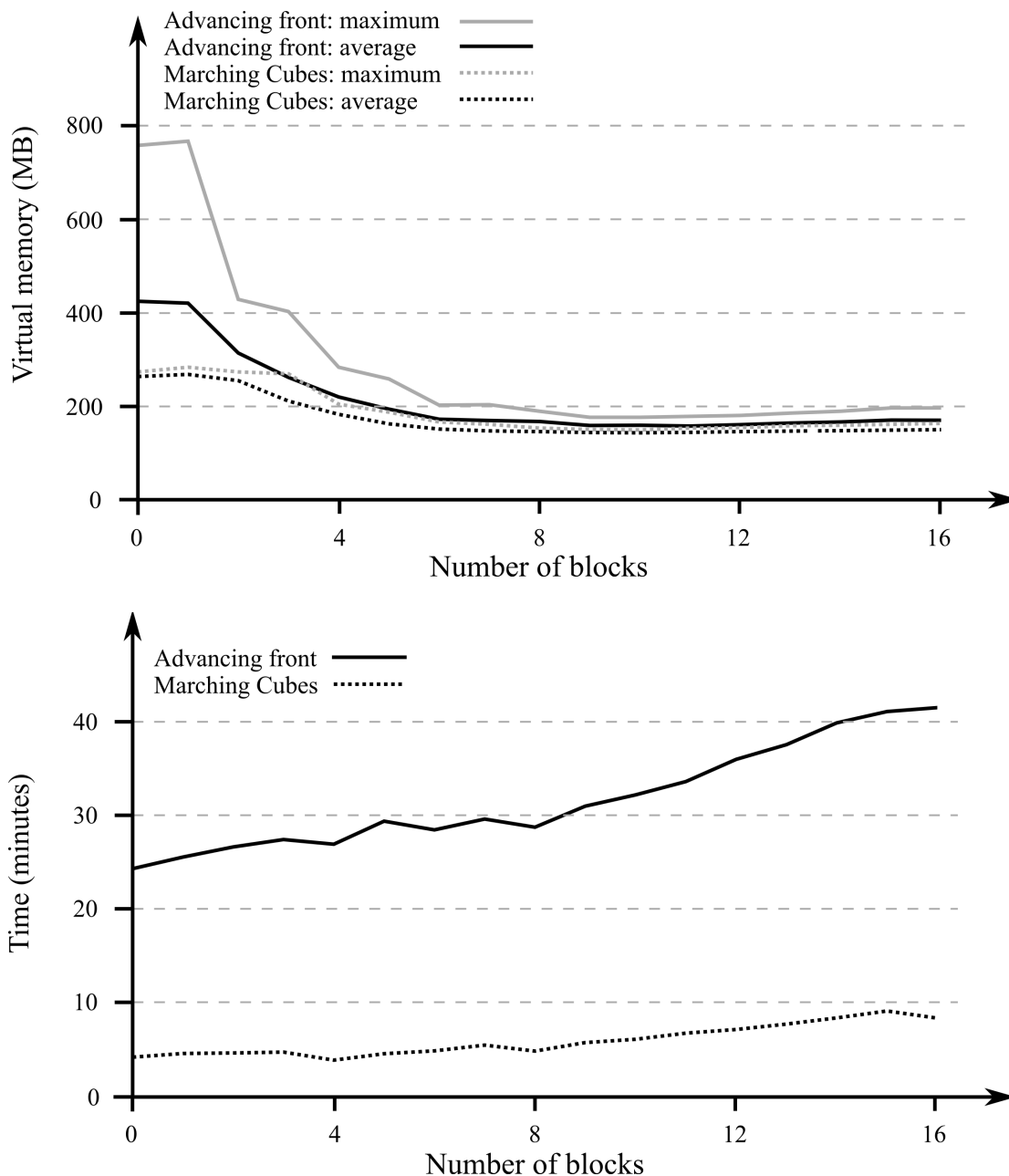
that can fit in-core. The guidance field samples  $s$  are then found for each block. This can be done independently for each block, and can be distributed over a cluster with no need for shared memory. The generated samples are stored on disk for later use during the triangulation. These blocks are then traversed in z-order to extract the isosurface (see Figure 4.20), with the guidance field samples loaded from disk as they are needed. Note also that a shell of blocks around the current one are required to be in-core since evaluating the splines near the block edges requires samples from the neighboring blocks. The effects of the block size on the memory use and running time are summarized in Figure 4.21. The advancing front algorithm is adapted to restrict the triangulation to stay within the current block. The algorithm must also be adapted to handle the boundaries of the surface and the seed points for connected components.

#### 4.6.4.2 Incremental Boundaries

Since a large surface that cannot fit in physical memory will often have a proportionately large set of boundaries, it is not advisable to have them entirely in memory. This presents an issue since



**Figure 4.20:** The blocks composing the volume are traversed in z-order minimize the shuffling of blocks in and out of memory. When traversed in this order, the active block at any time (red) will border blocks that have already been visited (blue) in the negative axial directions, and blocks that have not yet been visited (white) in the positive directions. A shell of blocks around the active block (bold square) must be in memory to evaluate the implicit function and guidance field near the active block boundaries.

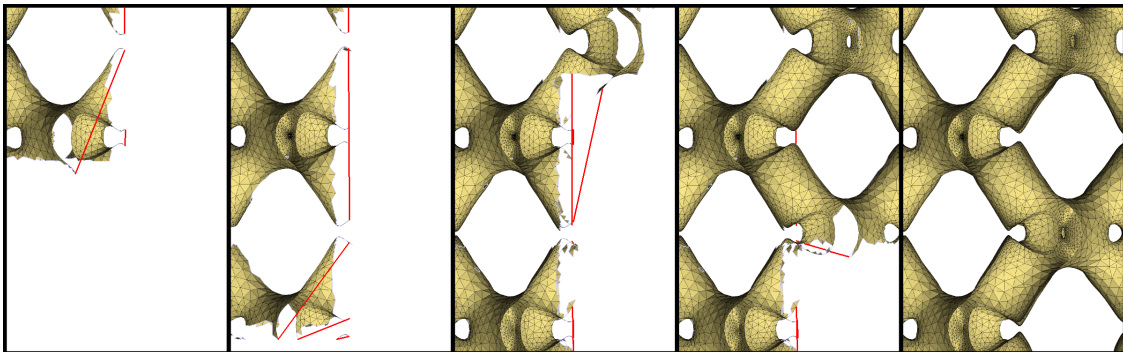


**Figure 4.21:** Comparison of memory usage (top) and running time (bottom) between Marching Cubes and the advancing front algorithm. The input volume is a  $512^3$  scan of a vertebra, and “# blocks” is the number of blocks it was divided into along each axis. “0” blocks uses the standard in-core codebase instead of the out-of-core version. The running time increases approximately linearly as the block size decreases, so the largest block size that does not cause memory swapping should be used for the best performance.

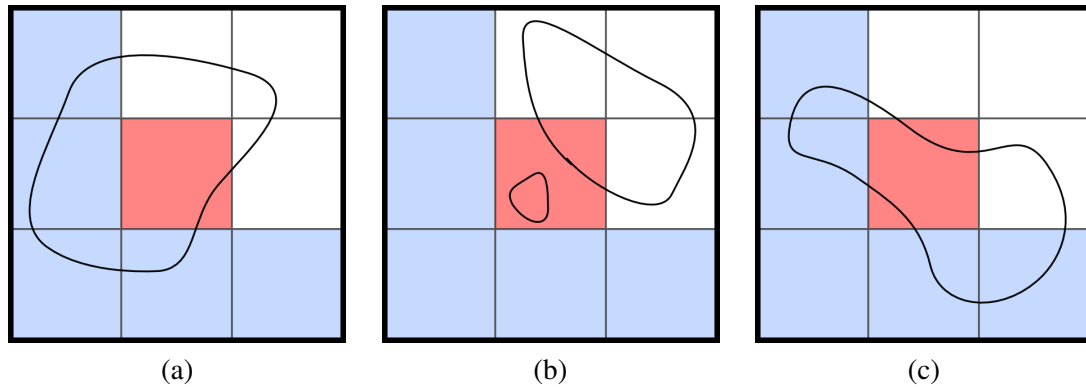
only a portion of the boundary may be available at any given time. The advancing front algorithm assumes that all of the fronts are closed loops, so it must be adapted. This is done by closing any incomplete boundaries with a temporary *phantom* edge (see Figure 4.22). These phantom edges do not have fences associated with them, and are not used in the front interference detection. As the triangulation progresses, new boundary fronts are inserted as new portions of the input surface are read into memory. When a boundary front with a phantom edge is inserted, all of the other phantom edges are searched for an endpoint that can be connected. As more boundaries are found, they are continually connected together until complete loops have been formed.

#### 4.6.4.3 Connected Components

The treatment of connected components must also be modified. Connected components may span many blocks, and they can enter and leave a block many times. To consistently handle the seeding of connected components, first note that during the z-order traversal of the blocks, the current block's neighbors in the negative x,y, and z directions have already been visited, and the neighbors in the positive x,y, and z directions have not. This is taken advantage of when creating the seeds for each connected component within a block, as illustrated in Figure 4.23. First, the connected components are found within the block as usual. Each of these connected components may be a subset of the same larger connected component that spans multiple blocks, but they are treated independently. A seed edge is placed somewhere on the component if it either does not exit the block (i.e., is a connected component entirely contained within the block), or it has boundaries on face in a positive axial direction, which is also not a boundary of the full volume. If the component exits the block in one of the negative axial directions, the triangulation will already



**Figure 4.22:** As new blocks are loaded into memory, new boundary segments are identified. The partial segments are closed into loops by *phantom* edges, shown in red. When new boundaries are inserted into the set of active fronts, they are joined at the endpoints of the phantom edges.



**Figure 4.23:** When the triangulation moves to a new active block, new seed points for the surface within that block may need to be created. If the portion of the surface exits the active block to any other blocks that have already been visited, new seeds are not required (a). Each connected component that either lies entirely within the active block, or only exits the block to unvisited neighbors will need seeds (b). Some connected components may enter and leave the active block multiple times. Each distinct part of the component that enters the block is treated independently, some may require seeds and some may not (c).

have been seeded in that neighboring block and the fronts will grow into the current block. If the component has a boundary that is on the boundary of the volume, it will be seeded as a boundary as described above.

#### 4.6.5 Parallelization

The serial nature of the advancing front algorithm makes it difficult to take advantage of modern multicore CPUs. The inner loop of the algorithm chooses an edge from the current fronts to grow a free triangle from. It must check this triangle against all the other fronts to ensure that no intersections occur. If it is clear, the triangle is created and the fronts are modified. This creates a race condition if two or more threads are performing these operations in parallel. However, much of the computation can be offloaded to worker threads. When a new triangle is created, the new vertex must be projected onto the surface. For some surface definitions, such as nonlinear MLS surfaces, this may be very slow. Additionally, the guidance field must be queried at the new vertex location to determine the sizes of the edges that will be allowed to grow from it. This projection and guidance field evaluation work can be parallelized by forming a pool of worker threads. Whenever a new edge is added to a front by the main thread, it is added to a work queue. When an edge is selected to grow a new triangle from, the result of the projection for that edge is retrieved from the worker threads. Since not all edges will grow free triangles (some are



connection triangles), this will result in work being done that is not used. However, this wasted work is typically made up for by the parallelism.

Another place to introduce parallelism is when constructing the guidance field. This involves finding the curvature of the surface at a dense set of points on the surface. This can be trivially parallelized by decomposing the surface into independent regions (e.g., splitting the points in a point set, or the grid cells of implicit function definitions). Once the samples have been generated, they can be reduced to a minimal guidance field by decomposing the points spatially into bins to be trimmed independently in parallel. Once each bin has been trimmed, a large percentage of the points have been removed, so a second pass with all of the points can be trimmed simultaneously.

## 4.7 Applications

The prototype implementation of the advancing front algorithm presented here has been tested by triangulating a large number of surfaces of different characteristics. This includes remeshing meshes arising from Marching Cubes, 3D photography, etc. It has also been applied to triangulating isosurfaces from medical scans and simulations defined over regular and unstructured grids and point set surfaces.

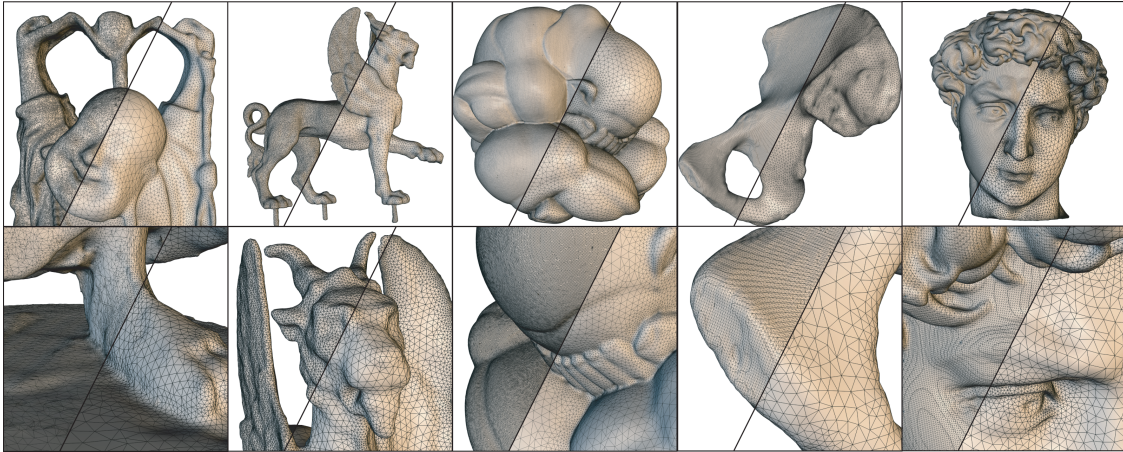
### 4.7.1 Remeshing

Several representative examples of the algorithm applied to meshes are shown in Figure 4.24. A detailed view of Michelangelo’s David model is shown in Figure 4.25, where the input surface is a MC mesh generated by the surface reconstruction method of Kazhdan et al. [78]. Statistics regarding these remeshes are listed in Table 4.2, relating  $\rho$  to the number of output triangles, the Hausdorff error, and the execution time. The quality histograms illustrate the consistently good triangle qualities produced by the algorithm, independent of the quality of the triangles of the input mesh.

#### 4.7.1.1 Boolean Operations

Many interactive tools require local remeshing after an editing operation has been performed. One such example is CSG operations on two meshes  $\mathcal{S}_1$  and  $\mathcal{S}_2$ . Since these operations need to stitch two incompatible triangulations together, they typically create many thin triangles and high valence vertices. A local remeshing is sufficient since only the triangles involved in the intersection are affected.

The procedure for CSG operations begins by finding the intersections of  $\mathcal{S}_1$  and  $\mathcal{S}_2$ , which can be done robustly without user input. Let  $\Lambda$  be the set of intersection loops of  $\mathcal{S}_1$  and  $\mathcal{S}_2$ . The



**Figure 4.24:** A sample of the results of our algorithm. From left to right we show remeshes of the Happy Buddha, the Feline, Pensatore, a Marching Cubes reconstruction of a pelvis bone used for a biomechanics simulation, and the head of Michelangelo’s David.

local remeshing then begins by marking all triangles within a user defined distance to  $\Lambda$ . These are the triangles that will be deleted and replaced by the advancing front triangulation. Let  $\Omega$  be the boundaries between the marked triangles and the unchanged portions of the input meshes. To remesh the marked area, initial fronts are created for each loop in  $\Omega$ , and two fronts with opposing orientation for each loop in  $\Lambda$ . The combination of these fronts surround the area to be remeshed to ensure that only the local region is covered. Since the surface  $\mathcal{S}$  now being triangulated is defined by  $\mathcal{S}_1 \oplus \mathcal{S}_2$ , where  $\oplus$  is a CSG operation, the guidance field must be modified in several ways.

































In addition to bounding the rate of change of the edge lengths within a single input surface, the guidance field must also ensure a smooth gradation between  $\mathcal{S}_1$  and  $\mathcal{S}_2$ . This is achieved by simply using  $s \in \mathcal{S}_1 \cup \mathcal{S}_2$  during the construction. This allows the edge lengths in one surface to be constrained by the ideal lengths of the other. Another issue to address is that the curvature of  $\Lambda$  may be greater than the maximum curvature of either mesh. Since these intersection curves must be represented accurately, they are treated as features and their curvature is included when constructing the guidance field. Finally, the edge lengths of the remesh should also blend into those of the unchanged portion of the input meshes. Since the edge lengths of the original mesh will not necessarily conform to the ideal lengths that the user specifies with  $\rho$ , a transition must be created. This is done by solving the following Laplacian system across the marked triangles where  $i \in \Lambda, j \in \Omega$ :

$$\Delta f = 0 \quad f_i = 0 \quad f_j = 1.$$



**Figure 4.25:** Many surface processing tasks require *good* meshes. At the same time, many meshes created automatically exhibit bad triangulations. The advancing front remeshing algorithm is based on surface reconstruction and requires no parameterization. The top row shows the watertight and manifold input mesh that was created with Reconstruct3D [78], and the bottom row shows a remesh generated by the advancing front algorithm.

**Table 4.2:** Summary of results of the advancing front algorithm applied to remeshing. Error is measured as Hausdorff distance, in percent of the bounding box diagonal. Input and output size is measured in thousands of triangles. Quality is shown as a histogram of circle ratios. The three vertical bars show the worst triangle, the first half percentile, and the median.

Mesh	$\rho$	$\eta$	Time	Error	Quality	Min Angle	In #	Out #
Bunny						0.5°	69.4K	
	0.5	1.25	0:22	0.25%		14.3°		34.5K
	1.0	1.25	0:08	0.37%		14.8°		14.8K
	1.5	1.25	0:07	0.49%		13.7°		9.4K
Dragon						0.6°	100.0K	
	0.5	1.25	1:05	0.20%		12.3°		121.4K
	1.0	1.25	0:31	0.37%		13.9°		53.5K
	1.5	1.25	0:24	0.36%		14.4°		34.8K
Pelvis						0.6°	529.8K	
	0.5	1.25	0:53	0.08%		13.9°		68.2K
	1.0	1.25	0:33	0.13%		14.6°		26.5K
	1.5	1.25	0:30	0.21%		14.5°		16.2K
Max						1.2°	200.0K	
	0.5	1.25	0:34	0.13%		9.4°		45.1K
	1.0	1.25	0:16	0.20%		14.9°		19.5K
	1.5	1.25	0:12	0.39%		16.7°		12.6K
Feline						1.8°	151.0K	
	0.5	1.25	1:11	0.10%		13.2°		130.0K
	1.0	1.25	0:34	0.22%		15.3°		57.2K
	1.5	1.25	0:25	0.33%		13.6°		37.1K
David						0.0°	1300.8K	
	0.5	1.25	5:20	0.08%		8.5°		472.3K
	1.0	1.25	3:06	0.59%		13.5°		203.5K
	1.5	1.25	2:07	0.59%		13.8°		132.4K
Pensatore						0.0°	1995.7K	
	0.5	1.25	4:22	0.07%		12.6°		77.1K
	1.0	1.25	2:42	0.11%		11.8°		77.1K
	1.5	1.25	2:12	0.16%		13.1°		77.1K
Buddha						0.0°	1087.7K	
	0.5	1.25	3:50	0.89%		9.4°		336.2K
	1.0	1.25	2:06	0.90%		4.8°		137.0K
	1.5	1.25	1:23	0.88%		9.2°		84.3K

This is a sparse linear system that can be efficiently solved. The solution will give a smooth transition between the boundary constraints. The scalar field  $f$  is then used to blend between the user defined ideal step length and the lengths of the original edges at a given point.

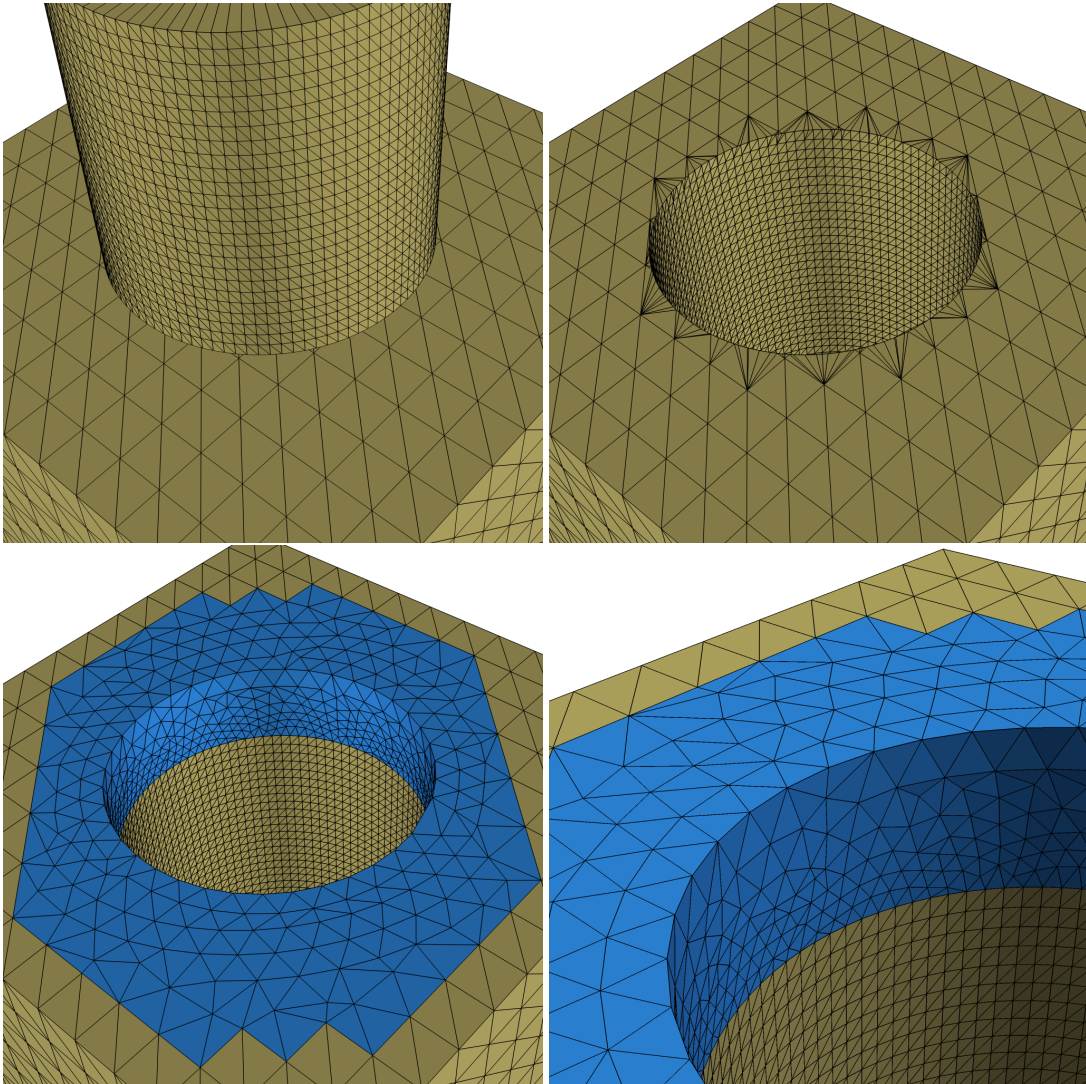
The example shown in Figure 4.26 demonstrates this local remeshing after a CSG difference operation has been performed. While most modeling packages will introduce very poor triangles to join the two meshes, the advancing front method accurately remeshes the intersection and blends the edge lengths into those of the input meshes. Only the portion shown in blue has been triangulated by the advancing fronts. The time required to perform this CSG operation, including finding the intersections, solving the linear system, computing the guidance field for both surfaces, then finally triangulating the local region, was less than 4 seconds.

#### 4.7.1.2 Mixed-Mode

The generalized advancing front algorithm allows a new application, termed *mixed-mode* Boolean operations. Since the core of the algorithm does not depend on the underlying surface definition, CSG operations between different surface types can be performed, for example, between a point-set surface  $\mathcal{S}_1$  and a mesh  $\mathcal{S}_2$ . One way to do this would be to triangulate both of the surfaces, then use the procedure outlined above on the resulting meshes. This approach, however, is not taken here since it introduces an additional error component in the result. Instead, the operation is performed directly on  $\mathcal{S}_1$  and  $\mathcal{S}_2$ .

Finding the intersection curves of two arbitrary surfaces is a much more difficult problem than when dealing solely with meshes. In this case, a small amount of user input is required to select a point close to  $\Lambda$ . This point is then iteratively projected onto each surface until it converges to the intersection. The entire intersection curve is then traversed by moving parallel to both surfaces and reprojecting. This allows the discovery of the intersections of “black box” surfaces with a minimum of user input. If either of the surfaces is a triangle mesh, it will be locally remeshed as described above. Other surface types require that the entire surface be triangulated with initial fronts at the intersection curves.

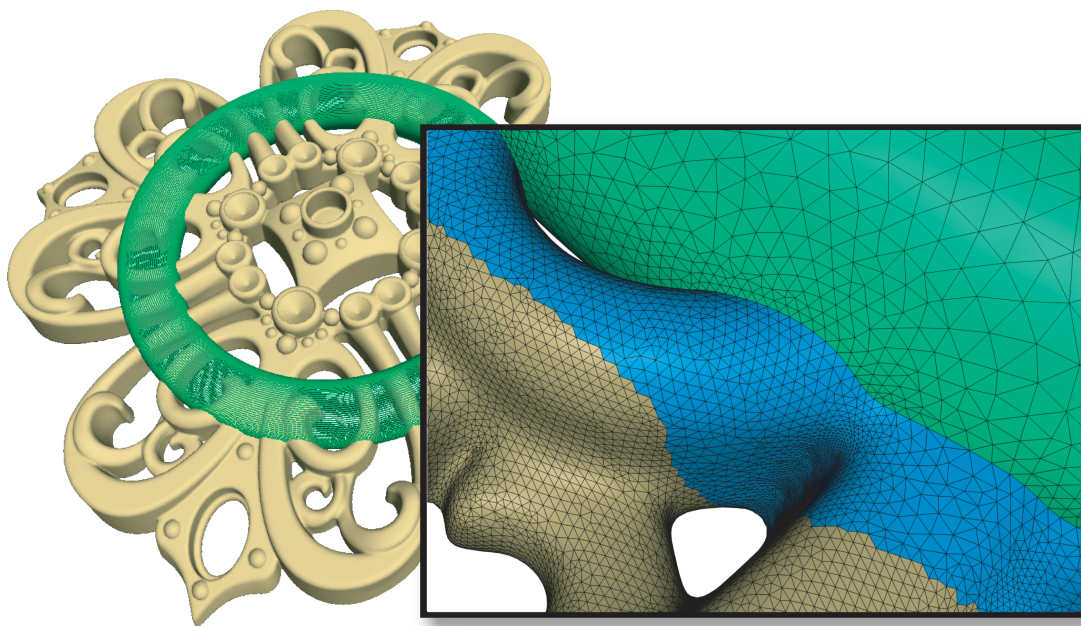
An example of a mixed-mode union between a mesh and a point set surface is shown in Figure 4.27. The triangulations of the two surfaces clearly meet to form a watertight mesh, and the edge lengths of the locally remeshed area smoothly blend into the original part of the mesh. This example also demonstrates that the algorithm is oblivious to the genus of both the input surfaces and the output of the CSG operation.



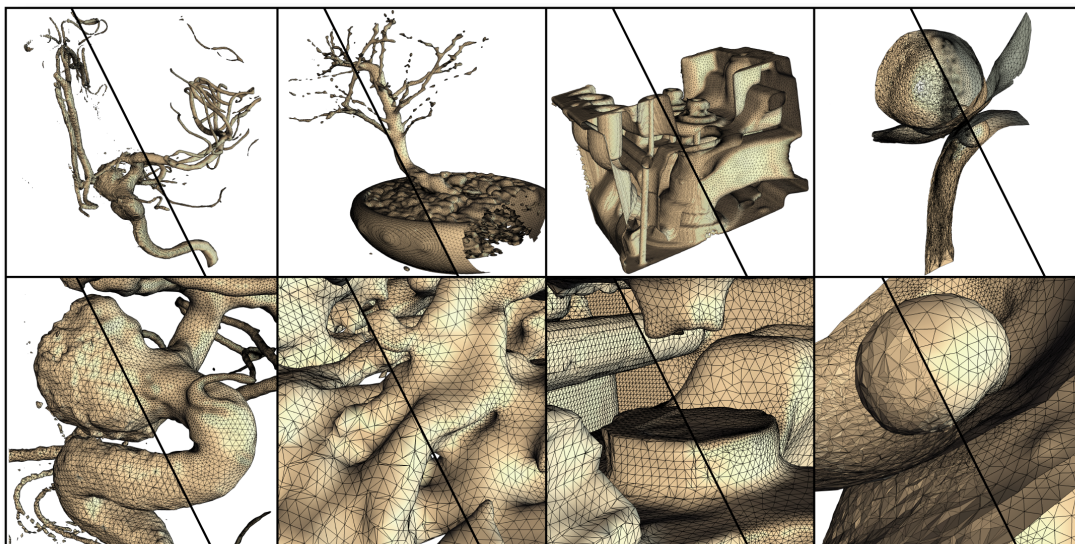
**Figure 4.26:** An example of a CSG difference operation. The output generated by Maya is shown in the upper-right corner, while the bottom row shows the results of using advancing fronts. The algorithm only changes the portion shown in blue.

### 4.7.2 Isosurface Extraction

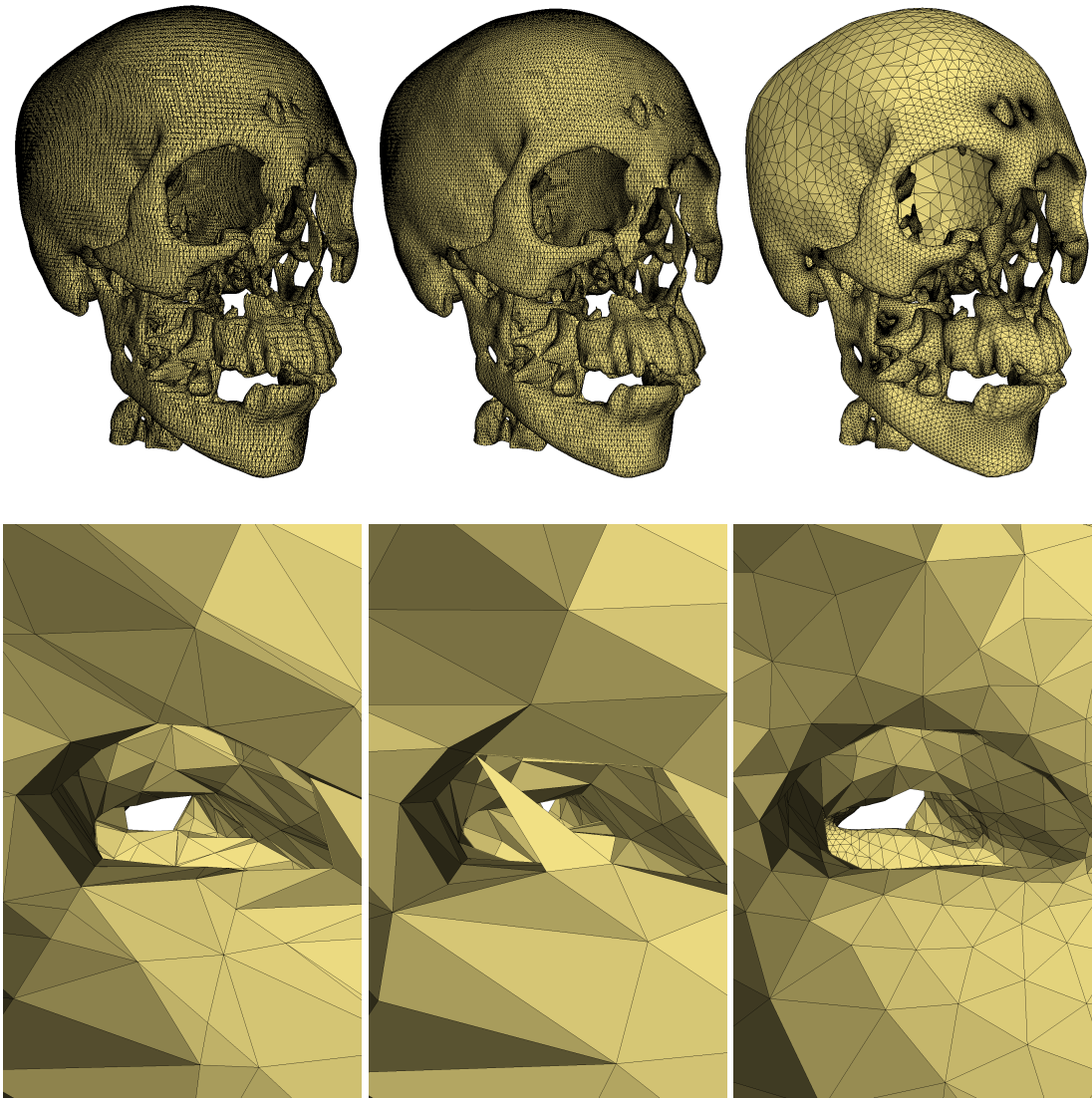
The advancing front algorithm has also been applied to a variety of isosurface extraction scenarios. Figure 4.28 compares several of the results of the advancing front algorithm to MC and MT. Figure 4.29 compares the advancing front algorithm to both MC, and Raman and Wenger's variant of MC [117]. Raman and Wenger's algorithm makes use of an extended look-up table for how to triangulate each cell of the regular grid. It includes cases for when the isosurface crosses the cell edges close to the cell corners. The mesh vertices that are created near each cell corner are merged to prevent triangles from being created with short edges. Preventing any short edges



**Figure 4.27:** The generality of the advancing front technique allows meshing of *mixed-mode* models: CSG operations between a mesh and a point-set, for example. The union of a triangle mesh with genus 48 and a torus defined by a point set is computed, with a final genus of 68. Only a local portion of the input mesh is remeshed, shown in blue. The entire input point set is triangulated, shown in green. Note that triangles meet one another in the intersection curves with the same resolution, yielding high triangle quality.



**Figure 4.28:** Some results of the advancing front algorithm, compared to MC and MT. From left to right: CT scans of an aneurism, a bonsai tree, an engine block, and isopotential surfaces of a human torso simulation. The first three datasets are regular grids, while the last one is a tetrahedral mesh.



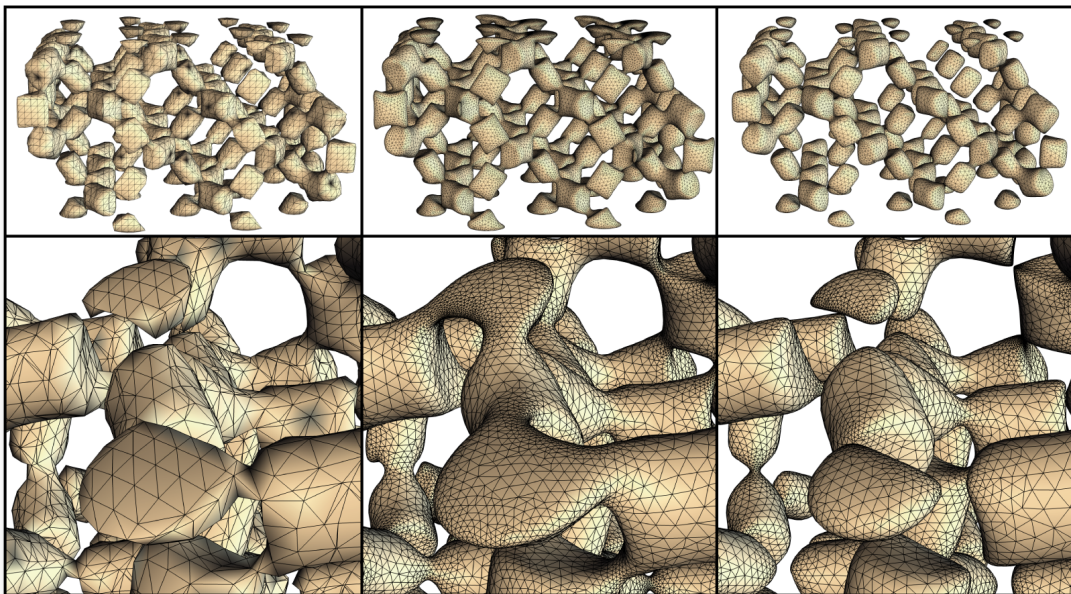
**Figure 4.29:** Triangle meshes generated from Marching Cubes have inherently biased sampling, which produces badly shaped triangles (left). A modified MC algorithm that makes use of an extended case table [117] greatly reduces the number of poorly shaped triangles, but incorrectly connects triangles across small gaps, and creates nonmanifold meshes (middle). The advancing front algorithm ensures appropriate mesh grading and curvature adaptation, and generates triangle meshes with excellent triangle shape (right).



from being created in this way greatly improves quality the triangles. However, since the MC algorithm has been altered, it is no longer guaranteed to produce manifold meshes. A closeup of a tunnel feature is shown in Figure 4.29, highlighting the adaptivity and accuracy of the advancing front algorithm. This demonstration of the advancing front algorithm differs from the remeshing example in Figure 4.25, where the algorithm is applied to the mesh generated by MC. Instead, this instance applies the algorithm directly to the isosurface.












The effect of the choice of spline used for defining the implicit function on regular grids is shown in Figure 4.30. The spline used directly affects the topology of the resulting mesh, and the topology of the MC mesh often does not match the topology of either the Catmull-Rom surface or the B-spline surface. Timing results and quality histograms are shown for regular grid isosurfaces in Table 4.3.

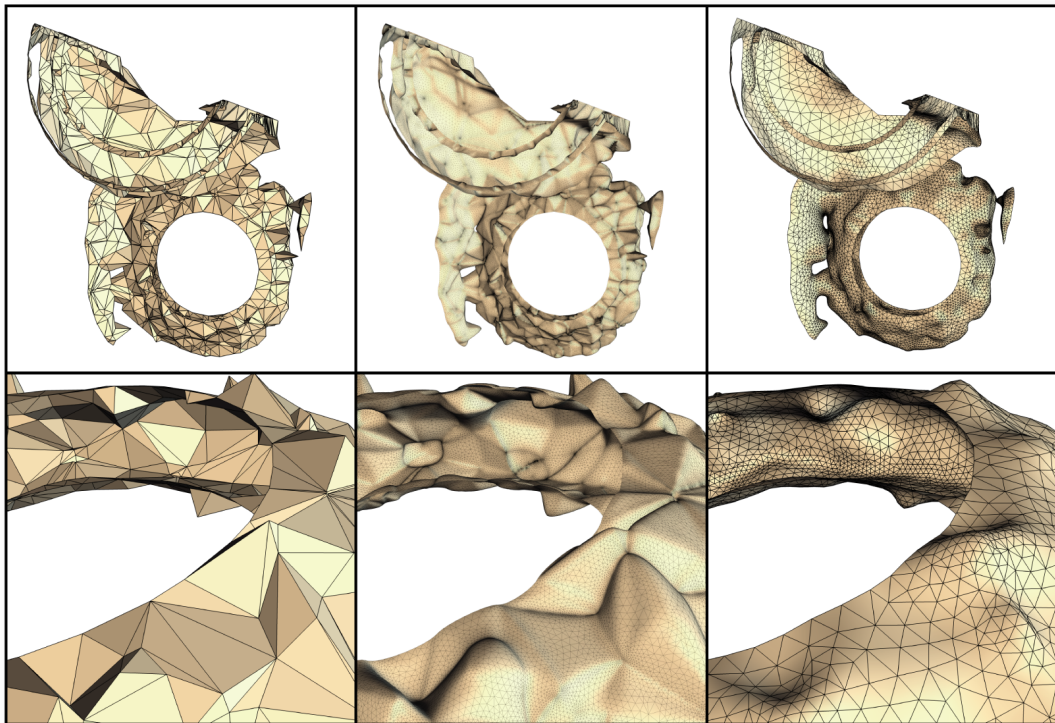
An example of isosurfacing for implicit functions defined on unstructured grids is shown in Figure 4.31. This figure highlights the effects of interpolating versus approximating implicit function definitions. When the function is reconstructed from the tetrahedral mesh vertices with Nielson's interpolating method, the surface exhibits many areas of very high curvature, and the resulting mesh contains correspondingly small triangles. For the approximating MLS implicit



**Figure 4.30:** Isosurface extraction from a structured grid of a silicium lattice simulation. From left to right: Marching Cubes output, and the advancing front method for  $\rho = 0.3$ , using, respectively, Catmull-Rom and B-splines for reconstruction.

**Table 4.3:** A sample of results of isosurface extraction from regular grids, and comparison with MC and MT. The output size is measured in thousands of triangles. Dataset sizes: Aneurism,  $256^3$  bytes; Silicium,  $98 \times 34 \times 34$  bytes; Engine,  $256 \times 128 \times 128$  bytes; Skull,  $256 \times 256 \times 226$  bytes.

Model	Alg.	$\rho$	$\eta$	Time	Quality	Min Angle	Out #
Aneurism	MC	—	—	0:24		$0.0^\circ$	148.9K
	BS	0.5	1.25	7:30		$13.4^\circ$	146.1K
Silicium	MC	—	—	0:00		$0.0^\circ$	40.0K
	BS	0.5	1.25	1:07		$13.0^\circ$	99.0K
	CR	0.5	1.25	3:53		$10.9^\circ$	165.3K
Engine	MC	—	—	0:21		$0.0^\circ$	586.9K
	BS	0.5	1.25	4:59		$12.5^\circ$	143.4K
	CR	0.5	1.25	119:28		$13.7^\circ$	432.0K
Skull	MC	—	—	0:15		$0.0^\circ$	400.2K
	BS	0.5	1.25	2:52		$14.4^\circ$	187.1K
	CR	0.5	1.25	4:28		$13.9^\circ$	245.3K



**Figure 4.31:** Isosurface extraction from unstructured grids. From left to right: MT output, and the advancing front method for  $\rho = 0.5$ , using, respectively, Nielson interpolation and Moving Least Squares for reconstruction.










function reconstruction, the surface is much smoother and the algorithm produces much larger triangles. Timing results and quality histograms are shown for the unstructured grid isosurfaces in Table 4.4.

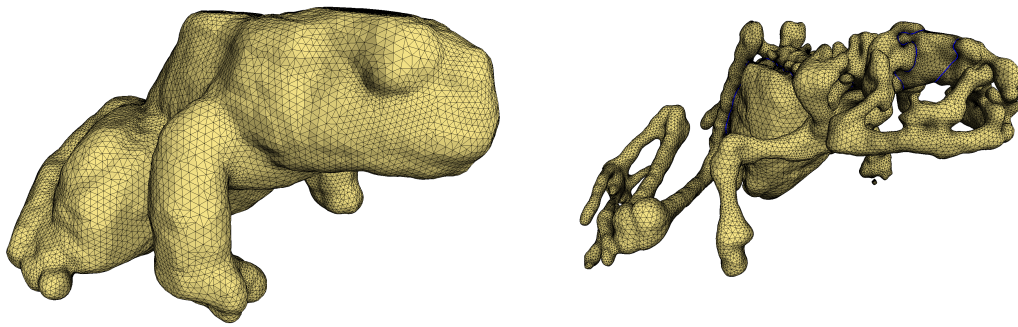
### 4.7.3 Multimaterial Volumes

Volumetric scans are sometimes segmented into a number of distinct material types. The set of interfaces between these materials may be needed for visualization, or for finite element simulations of the object being examined. The advancing front algorithm can easily be adapted to this situation. The notation of Meyer et al. [100] is adopted, and the input to the algorithm is assumed to be a set of indicator functions  $f_i : \mathbb{R}^3 \rightarrow \mathbb{R}$  associated with each material identifier  $m_i$ . An arbitrary point  $\mathbf{x}$  is labeled as material  $m_i$  if  $f_i(\mathbf{x}) > f_j(\mathbf{x})$  for all  $i \neq j$ . Note, however, that some points cannot be labeled as a single material in this way. When two indicator functions  $f_i(\mathbf{x}), f_j(\mathbf{x})$  are both equal to  $\max_k f_k(\mathbf{x})$ , there is an interface between materials  $m_i$  and  $m_j$ . Only mild conditions on the indicator functions make this set of points well defined smooth surfaces with boundary. Specifically, these points lie in subsets of the isosurface  $f_i - f_j = 0$ . The *junctions*, where multiple material interfaces meet, are found as described by Meyer et al., and used as initial fronts for each of the material interfaces. The advancing front algorithm is applied to each pairwise combination of material identifiers (some pairs may not produce any interfaces) to extract the full set of interfaces.

An example of material interfaces extracted from a multimaterial volume is shown in Figure 4.32. Here, a scan of a frog has been segmented into several materials: bone, brain, guts,

**Table 4.4:** A sample of results of isosurface extraction from unstructured grids, and comparison with MC and MT. Dataset sizes: SPX, 13k tetrahedra; Torso, 1.1 million tetrahedra.

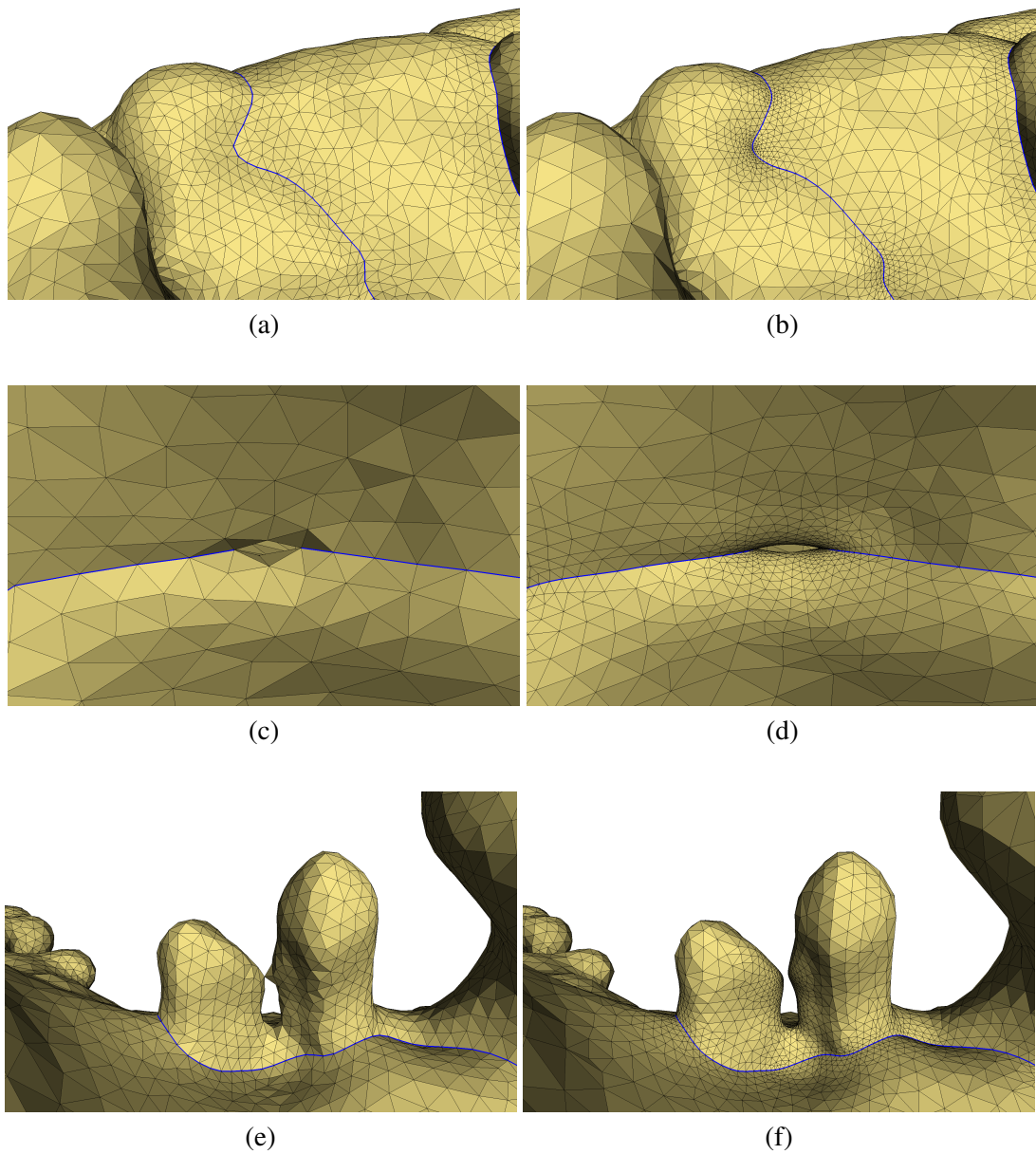
Model	Alg.	$\rho$	$\eta$	time	Quality	Min Angle	Out #
SPX	MT	—	—	0:00		0.0°	2.3K
	NI	0.5	1.25	4:17		9.1°	277.6K
	MLS	0.5	1.25	1:44		14.3°	27.1K
Torso-1	MT	—	—	0:04		0.0°	40.3K
	NI	0.5	1.25	24:50		9.1°	1419K
	MLS	0.5	1.25	2:04		14.3°	39.3K
Torso-2	MT	—	—	0:01		0.0°	3.1K
	NI	0.5	1.25	2:49		14.7°	135K
	MLS	0.5	1.25	5:30		16.3°	7.3K



**Figure 4.32:** Extraction of material interfaces from a segmented volume. The outer surface mesh is removed to show the internal structures on the right.

tissue, and the surrounding air. Since the material junctions are resampled once and shared between all the interfaces for initial fronts, the meshes meet in a watertight way. Closeups of the example are shown in Figure 4.33, comparing the results of the particle based method of Meyer et al. to the mesh generated by the advancing front algorithm. This figure highlights two main differences between the meshes generated by Meyer et al. and the advancing front algorithm, both of which stem from sizing fields used to control the edge sizes that the two algorithms create.

The first difference is in the way that the junction curves where more than two materials join are treated. Meyer et al. perform a preprocessing step on the material indicator functions to bound the curvature of the isosurfaces that they define. They then combine all of the preprocessed indicator functions to find the material interfaces, and use the local feature size of the combined set of interfaces to define their edge sizing function. This allows features in one interface to affect the triangle sizes in another, but has the drawback that the local feature size goes to zero at the material junctions. To address this issue, and prevent arbitrarily small triangles from being created, they clamp their edge sizing function to never be smaller than a value derived from the curvature bound of the indicator function isosurfaces. While this allows large triangles near material junctions to be created, there is still the issue that the isosurfaces of the indicator functions do not coincide with the material interfaces, and the curvature of the one-dimensional junctions is not considered. The advancing front algorithm does not use the local feature size, but considers all piecewise smooth surface patches adjacent to the material junctions independently, and then merges them into a single guidance field. This avoids the issue of the edge sizing function going to zero, and does not require an explicit dependence on the isosurfaces of the indicator functions when constructing the guidance field. Also, the curvature of the one-dimensional material junctions is explicitly considered when constructing the guidance field, as described in



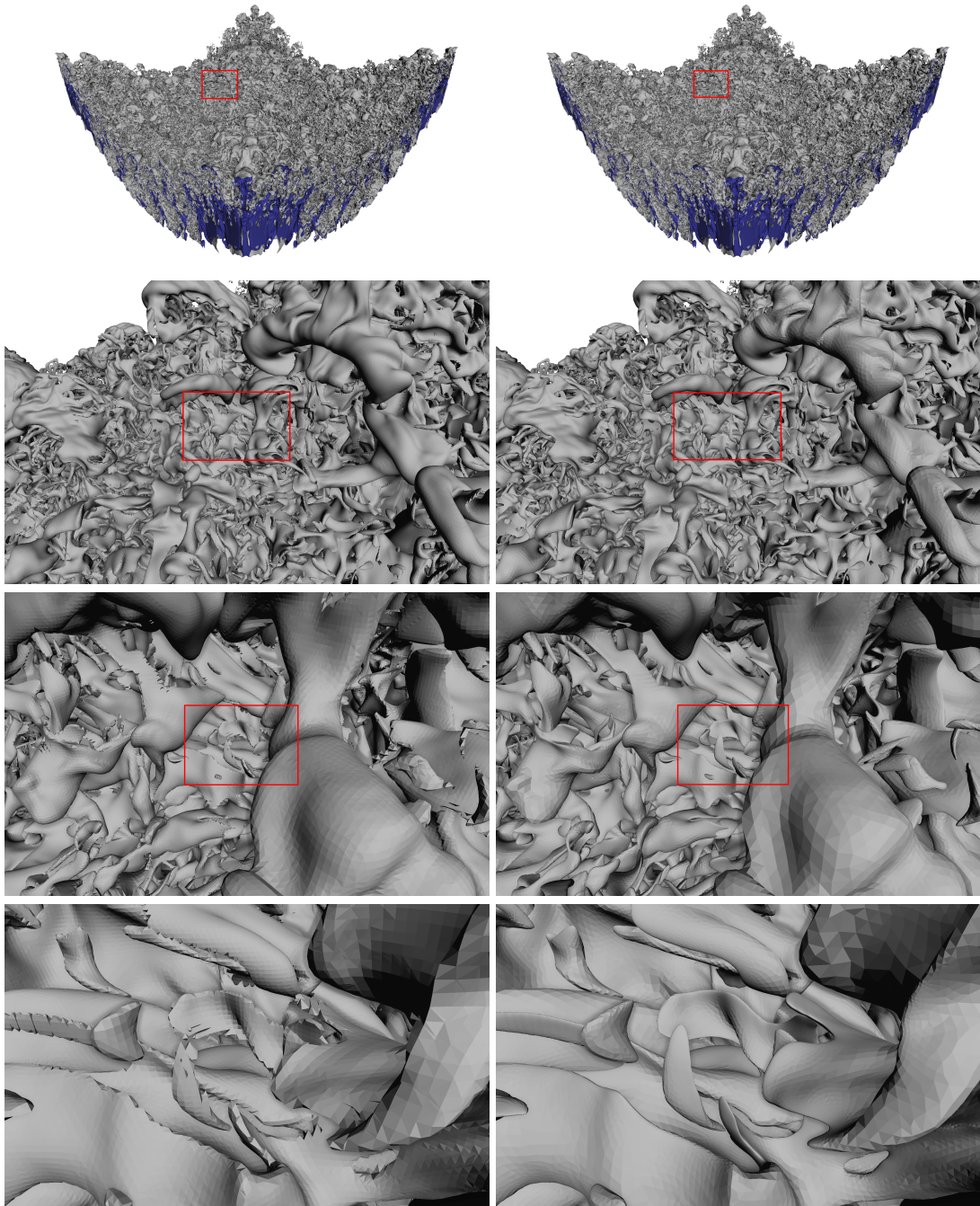
**Figure 4.33:** Comparisons of a multimaterial interface extraction between the particle based method of Meyer et al. [100] (left) and the advancing front algorithm (right). Since the curvatures of the material boundaries are directly included in the guidance field, they are more faithfully reconstructed by the advancing front algorithm (a,b,c,d). The particle based method has also created a nonmanifold point where a surface passes close to itself (e,f). The true local feature size at this point is smaller than the grid used to implement the sizing field, resulting in edge sizes larger than expected.

Section 4.3.1. This allows the triangle sizes to be constrained by all of the different material interface surfaces, as well as the curvature of the lower dimensional material junctions. Rather than clamping the edge sizes along the material junctions and providing conditions under which it will still produce a valid mesh adjacent to the junctions, the advancing front algorithm directly adapts the output mesh to the curvature of the junctions.

The second difference shown in Figure 4.33 is related to the resolution of the edge sizing function representation. Meyer et al. use a fixed resolution grid to store their sizing field, which may not be sufficient to resolve small features. This can lead directly to spaces between particles that are too large for the true local feature size, resulting in nonmanifold vertices after the particles are triangulated. While the theoretical foundation of their algorithm is sound and does not allow this to happen, it illustrates the difficulty of computing and storing the local feature size. This artifact is not an effect of either the algorithm disobeying the sizing field, or the theoretical sizing field being inappropriate. It is instead due to the specific implementation of the sizing field inaccurately representing the true local feature size. Since the preprocessing of the input data to place an upper bound on the maximum curvature also tends to increase the local feature size, this artifact only occurs when the curvature of the surface is low, but the local feature size is small as well. The situation may be addressed by specifying a finer grid for the sizing field, but the required resolution may not be known a priori. In contrast, the advancing front algorithm uses an unstructured set of points to represent the guidance field, and enforces a sampling condition to guarantee that the sizing function used by the triangulation is bounded above by the true ideal edge size.

#### 4.7.4 Out-of-Core

Out-of-core isosurface extraction is demonstrated with a dataset simulating the Richtmyer-Meshkov instability of a shockwave passing through a gas interface [101]. The volume is  $2048 \times 2048 \times 1920$ , requiring 7.7GB of uncompressed storage space. The guidance field construction was distributed to 12 computers with a total of 42 processors, and took 13 hours to produce 2.8GB of surface samples and ideal edge sizes. Extracting the surface took 124 hours on a 4 core 2.2GHz CPU with 4GB of RAM available. The output mesh has 389 million faces and 195 million vertices, requiring 6.6GB of storage in a binary streaming format. While the total data size of the input volume, guidance field, and output mesh exceeded 17GB, the maximum memory allocated was 2.2GB. The maximum number of vertices that were stored in-core at any time was 928K, or less than 0.48% of the total. Figure 4.34 compares the extracted surface to that

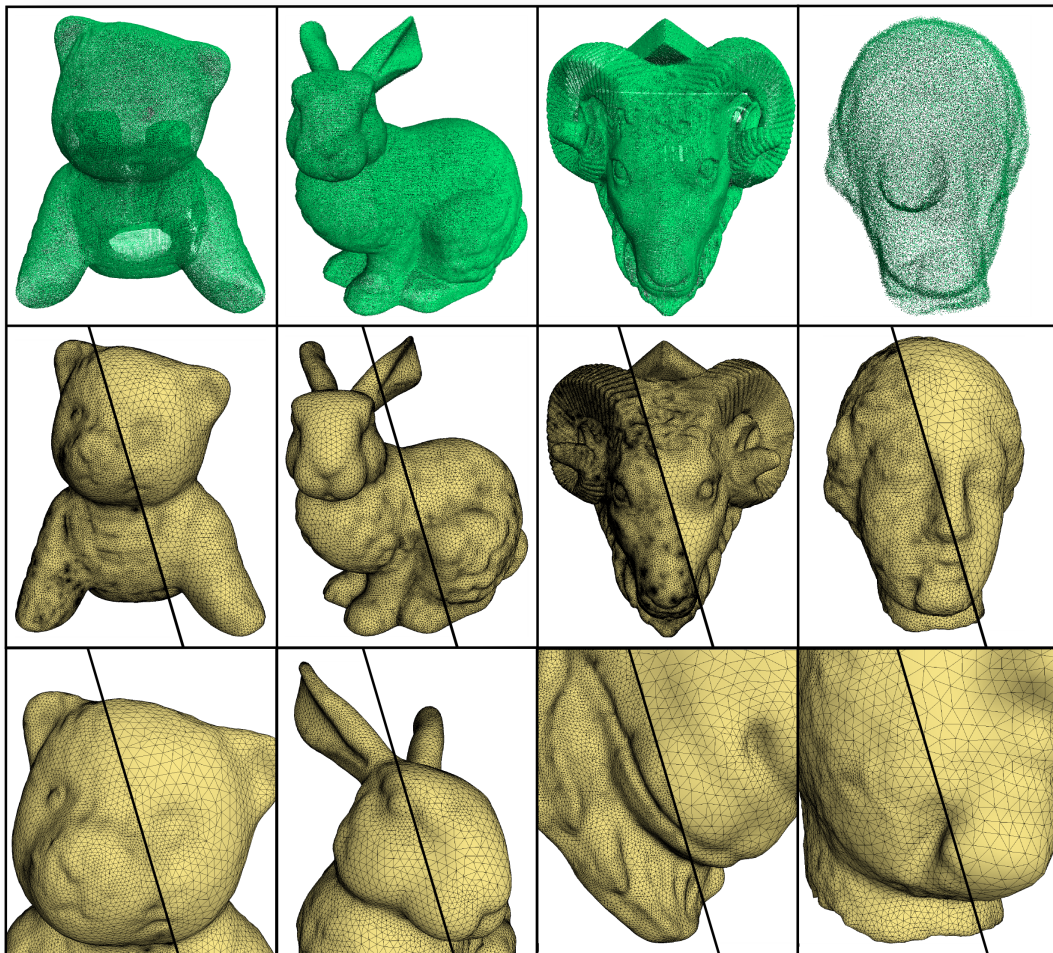


**Figure 4.34:** An isosurface of a simulation of the Richtmyer-Meshkov instability. In these images, the shockwave passed from the top to the bottom. Marching Cubes is shown on the left, and the advancing front algorithm is shown on the right. Each image is a close-up of the region in the red box above it.

of Marching Cubes. This example highlights both the flexibility of the advancing front algorithm, as well as its robustness.

#### 4.7.5 Point Set Surfaces

The advancing front triangulation algorithm can also be applied to point set surfaces. Figure 4.35 shows the meshes created from several input point sets, using both linear and nonlinear MLS surface definitions. Table 4.5 summarizes the running time and quality of the output meshes. Note that the meshes from the nonlinear surface definition tend to have smaller triangles than the meshes from the linear definition. Additionally, the nonlinear meshes exhibit artifacts where



**Figure 4.35:** Several example point set surface triangulations are shown. The top row shows the input points. In each of the split views, the left shows the surface extracted with a nonlinear MLS surface definition, and the right shows a linear surface definition.



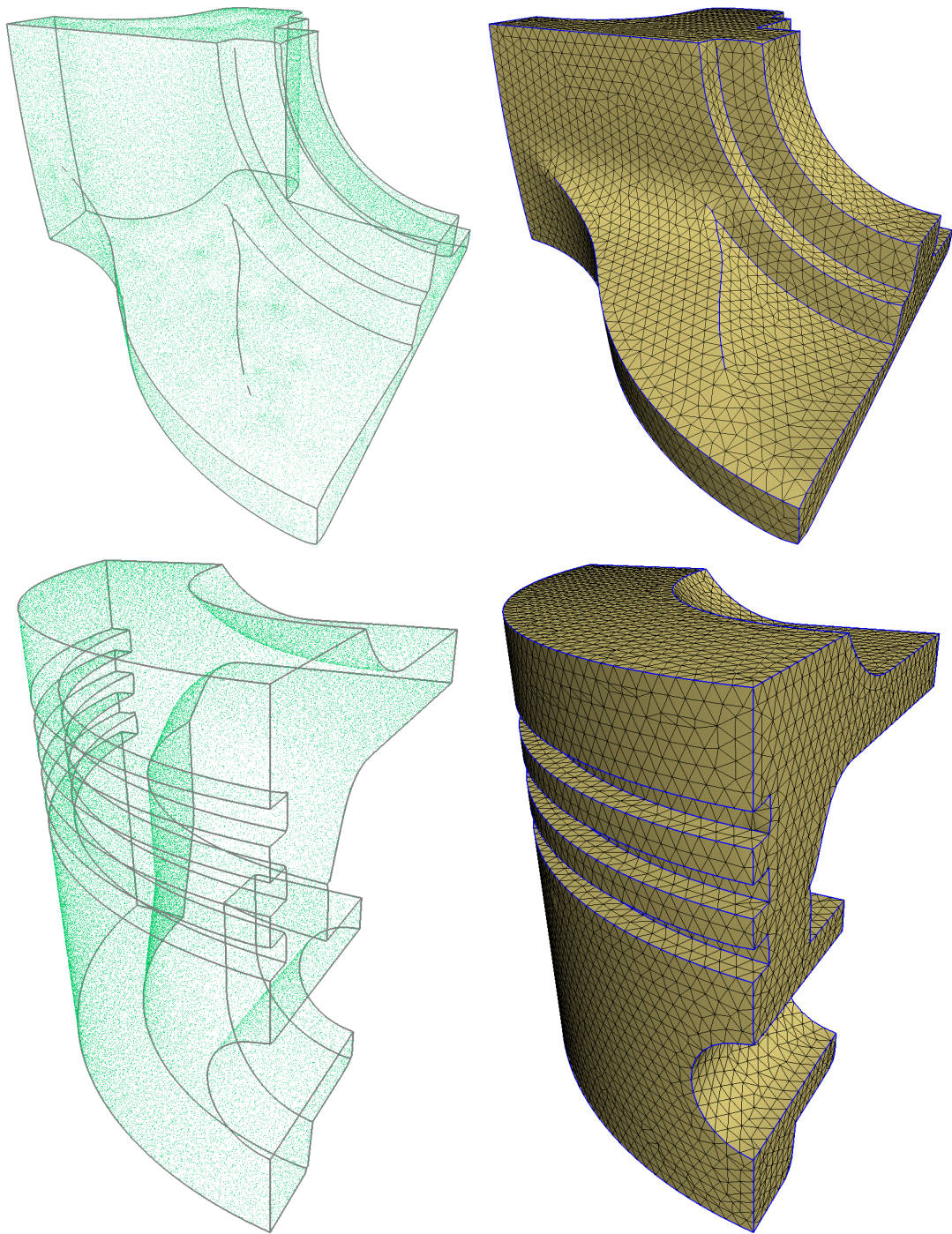
**Table 4.5:** A sample of timing and quality results of triangulating point set surfaces. The input size is the number of points, and the output size is the number of triangles. The linear surface definition tends to produce many fewer triangles, and the running times are accordingly slower. However, the applicability of the linear definition is limited to points with normals.

Model	Alg.	$\rho$	$\eta$	In #	Time	Quality	Min Angle	Out #
Bear	Linear	0.5	1.25	651.7K	3:29	▬▬▬▬	14.5°	35.2K
	Nonlinear	0.5	1.25	651.7K	14:08	▬▬▬▬	12.1°	233K
Bunny	Linear	0.5	1.25	1120K	8:32	▬▬▬▬	15.5°	52.6K
	Nonlinear	0.5	1.25	1120K	20:14	▬▬▬▬	15.3°	80.5K
Ram	Linear	0.5	1.25	622.7K	4:00	▬▬▬▬	14.0°	81.1K
	Nonlinear	0.5	1.25	622.7K	16:14	▬▬▬▬	9.5°	339K
Venus	Linear	0.5	1.25	134.4K	0:33	▬▬▬▬	15.4°	11.5K
	Nonlinear	0.5	1.25	134.4K	2:04	▬▬▬▬	13.8°	62.3K

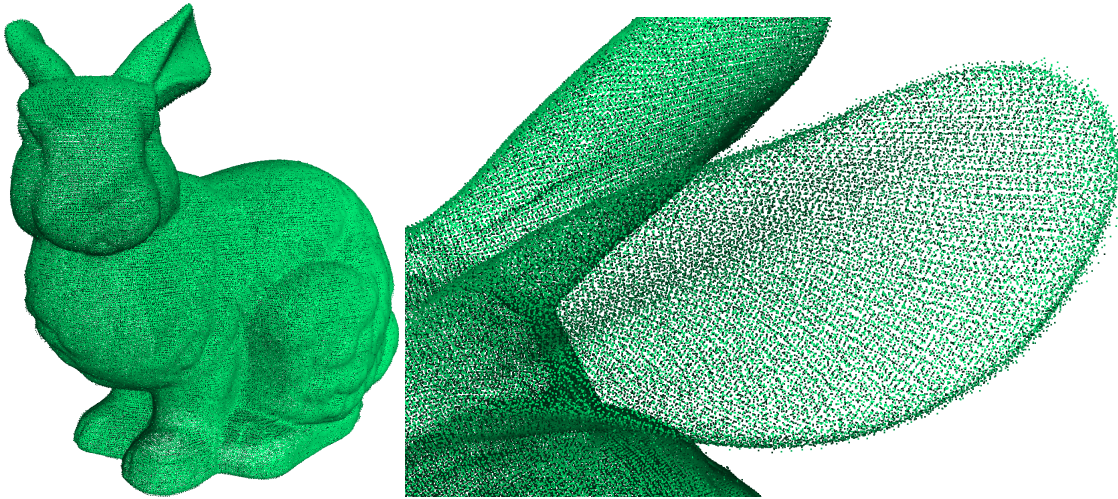
localized points are restricting the triangle size more than they should, similar to the effect when poor curvature estimates are used for triangle meshes (recall Figure 4.7). The reason for this is twofold. First, the curvature is not that of the actual surface, but is estimated from polynomials fit to the input points. Second, the projections, and thus the curvature computations, are sensitive to noise in the point set and to the initial guess used for the nonlinear optimization. These issues cause the occasional over-estimation of the maximum surface curvature, leading to the artifacts seen. The linear formulation is more stable since it additionally makes use of the normals associated with the input points, and the curvature of the surface can be computed exactly.

Point set surfaces with features identified with the method of Daniels et al. [40] are shown in Figure 4.36. The features are used as initial fronts to prevent triangles from crossing over them and rounding them off. The connectivity and topology of the feature networks do not influence the ability of the advancing front algorithm to reconstruct the features faithfully. The guidance field has been constructed to produce uniformly sized triangles.

The bunny point set shown in Figure 4.37, acquired through multiple laser range scans, has been used to compare several meshing strategies, shown in Figure 4.38. Row (a) shows a Delaunay based method [16] that uses all of the input points as vertices in the output mesh. Since there is noise in the input points, the output mesh is extremely noisy as well, and probably cannot be salvaged through simplification and smoothing operations. Row (b) shows a method where an implicit function is created from the input scans, and then an isosurface is extracted [38]. This produces a much more usable mesh, but it exhibits the issues typical of standard isosurface



**Figure 4.36:** Sharp features can be preserved by starting initial fronts along them. The features that have been identified by the method of Daniels et al. [40] are used as input and are shown to the left of the output triangulation.

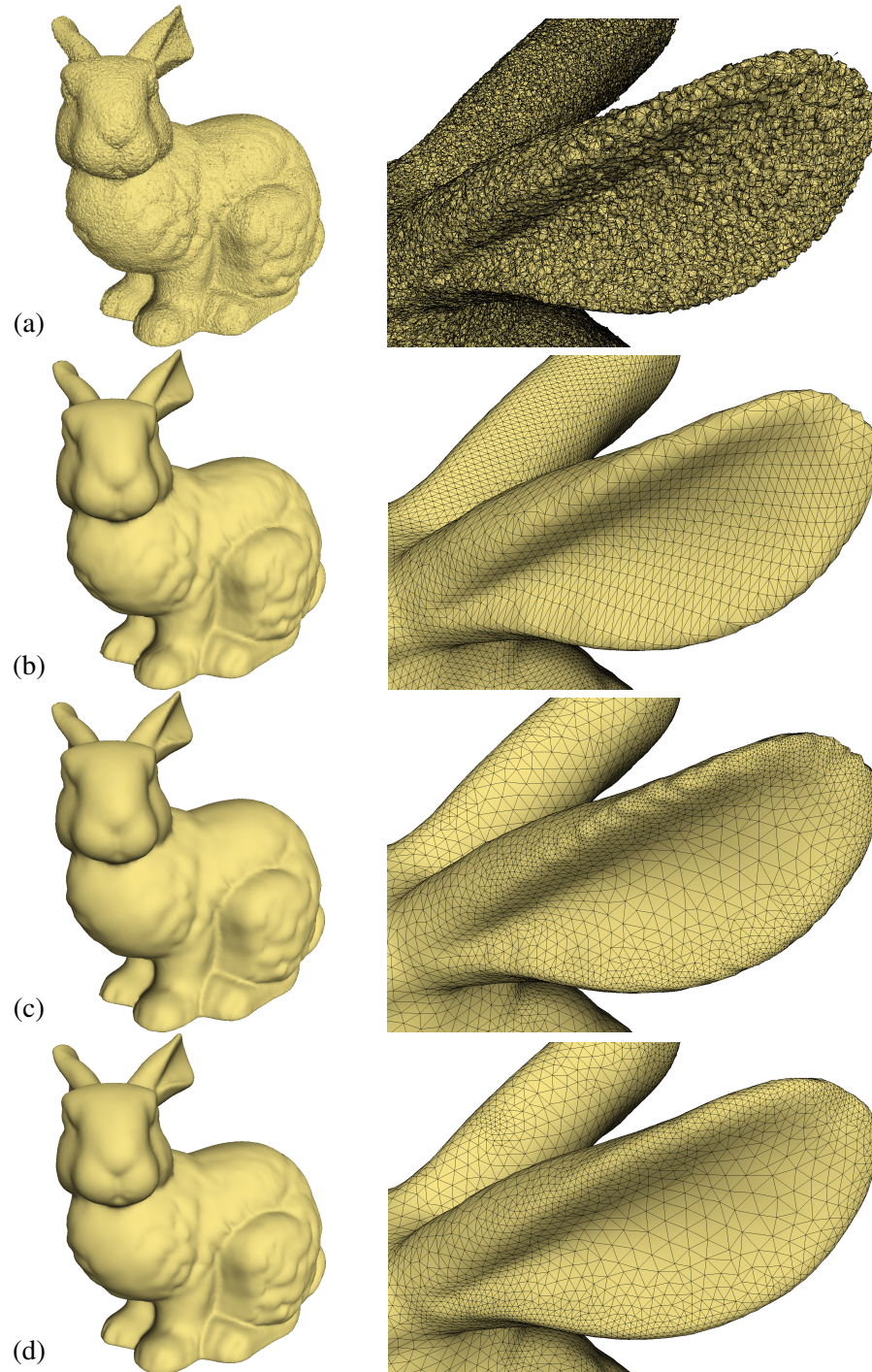


**Figure 4.37:** A point set composed of multiple laser range scans is used as the input to several meshing algorithms for comparison. There are more than 1.1 million points in the input dataset.

extraction techniques. The uniform triangle sizing creates saw-tooth effects around small sharp features, such as around the tip of the ear. This mesh can be improved by remeshing it with the advancing front algorithm, shown in row (c). This creates a mesh that adapts to the curvature, but the deficiencies of the input mesh are still apparent at the tip of the ear. The additional processing has also lost detail from the original surface, as can be seen by the decreased normal variation on the side of the bunny. Row (d) shows the result of applying the advancing front algorithm directly to the input points.

## 4.8 Discussion

The algorithm presented in this chapter addresses all of the major issues hindering advancing front algorithms, but the robustness arguments depend critically on two assumptions. First, they assume that the surface being triangulated is smooth (even though it is certainly not the case for remeshing applications). For the case of implicit surfaces, the only assumption needed is that the gradient of the implicit function is defined and nonzero at all points in the surface. This is a reasonable assumption to make. Second, it is assumed that the guidance field has been sufficiently sampled so that the first property of  $g$  will hold (i.e.,  $\tau$  is bounded below by  $g$ ). For surfaces where it is not known how to bound the curvature, such as nonlinear MLS surfaces, the input surface is simply sampled densely with no means to check for sufficiency of the guidance field. Features smaller than the sampling density can easily be missed by the triangulation. In the worst case, the assumptions of the robust front interference detection are broken, and the triangulation can fail.



**Figure 4.38:** Several methods of generating meshes from the set of points in Figure 4.37 are shown: (a) a Delaunay based method [16], (b) a volume based method [38], (c) the advancing front algorithm applied to (b), and (d) the advancing front algorithm applied directly to the original point set.

Despite these shortcomings, the algorithm is very robust when these assumptions are satisfied, as demonstrated by the out-of-core isosurface extraction.

The triangle quality generated by the advancing front algorithm is very good, as the circum-circle ratio histograms in Tables 4.2, 4.3, and 4.4, and Figure 4.14 show. For all meshes generated, 99% of the triangles have quality at least half of optimal, and the triangle with median quality is within 3% of optimal. These results hold across different input models,  $\rho$  and  $\eta$  values, and different surface definitions.

### 4.8.1 Performance

The advancing front algorithm is fast and requires a small amount of RAM, which depends largely on the size and representation of the input surface. The bottleneck is often in the construction and evaluation of the guidance field, which typically takes approximately half of the execution time, though it varies with the characteristics of the surface and its underlying definition. The running time for remeshing and point set surface reconstruction is typically measured in seconds or minutes, while isosurface extraction typically takes a small number of minutes.

The output mesh is streamed to disk as it is created, so the memory footprint is only dependent on the size of the input surface and the number of edges in the advancing fronts. To increase the locality of the triangulation and reduce the lengths of the fronts, *working areas* are used. All triangles, free and connection, are created within a working area before the triangulation progresses. If the fronts are allowed to grow over the entire surface before creating any connection triangles, the memory usage will be proportional to the size of the output mesh. The use of working areas reduces the memory use to be a small fraction of the total output size. Restricting the triangulation to working areas allows free triangles to be grown from edges of connection triangles, but this does not significantly affect the overall quality of the output mesh.

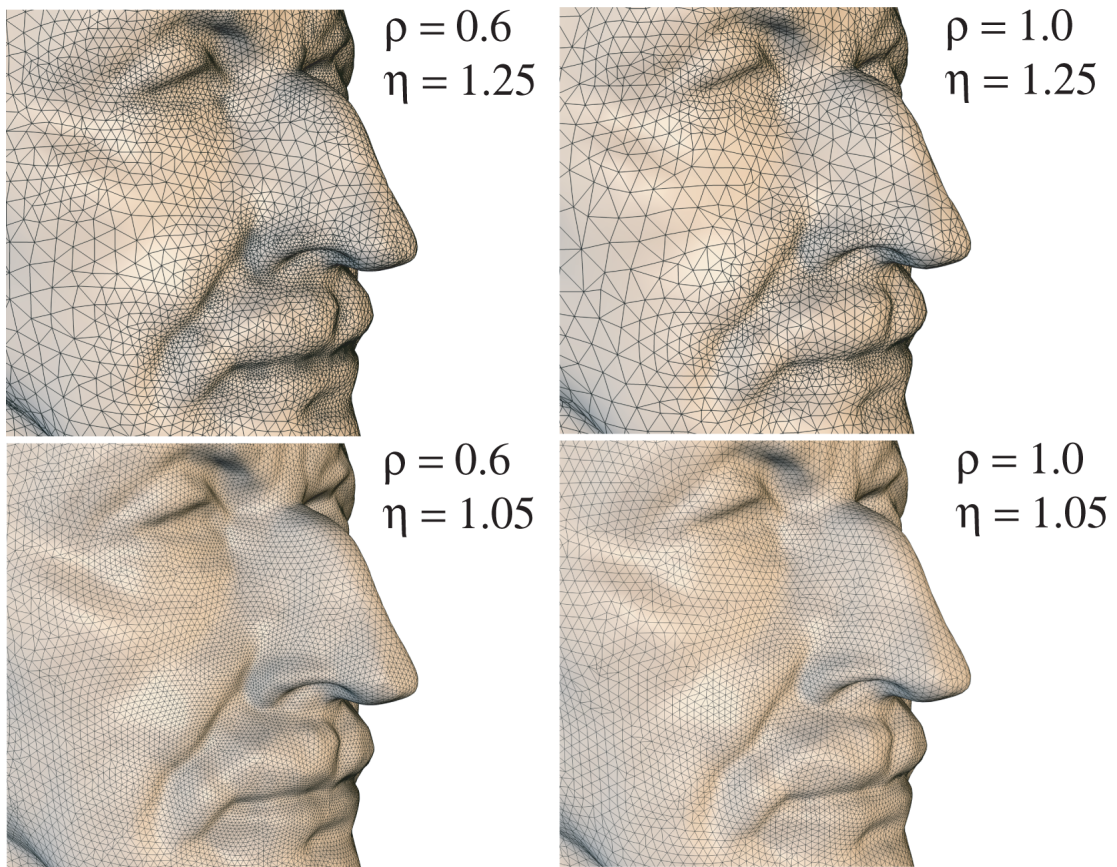
The running times for isosurface extractions with the advancing front algorithm are not as fast as MC or MT implementations. However, it should be noted that the triangulations created by this technique will tend to require negligible or no downstream processing, unlike the results of MC or MT. The advancing front triangle counts are often less than MC triangle counts, and while the Nielson method of defining the implicit function results generates much denser triangle meshes than seems necessary, overall they tend to be well within the applicability range of downstream processing methods.

The system used to generate these results is a dual AMD Opteron 275 (2.2GHz, 4 cores total) with 4GB of RAM. This multicore system is used to exploit trivially parallelizable aspects of the

algorithm, such as finding the surface samples  $s$  for the guidance field construction and computing the ideal edge lengths. The amount of RAM required for the remeshing experiments ranged from 17MB for the bunny with 69 thousand triangles to 470MB for the Pensatore mesh which has almost two million triangles. Memory consumption is mostly dependent on the data structures used for keeping the guidance field and do not grow significantly with the density of the output mesh as the fronts are the only other data structure that remains in memory.

#### 4.8.2 User Control

The coarseness and quality of the output mesh are entirely controlled by two intuitive user parameters:  $\rho \in (0, 2\pi/3]$  and  $\eta \in (1, 2)$ .  $\rho$  directly controls the coarseness of the triangles by scaling the ideal edge length at each point.  $\eta$  controls the compromise between adaptability of the triangle size, and quality of the triangle shape. A small  $\eta$  will not allow the triangle size to change quickly, resulting in nearly equilateral triangles, but sizing them much smaller than required for the local curvature over much of the surface. Using a larger  $\eta$  allows more adaptive triangles, but results in more triangles with poor aspect ratios. Setting  $\eta = 1.2$  provides a good balance, and was used for the results described unless stated otherwise. Depending on the desired coarseness of the resulting mesh,  $0.2 \leq \rho \leq 1.5$  is typically used. Figure 4.39 demonstrates how different combinations of the two user parameters  $\rho$  and  $\eta$  effect the resulting mesh.



**Figure 4.39:** The effects of  $\rho$  and  $\eta$  on the output mesh.  $\rho$  controls the approximation accuracy: a bigger  $\rho$  will result in a coarser triangulation.  $\eta$  controls triangle grading: a larger  $\eta$  will result in more adaptive triangulations.

## CHAPTER 5

### CONCLUSIONS AND FUTURE WORK

Two distinct methods for generating high quality meshes have been presented in this dissertation. They are both very general, but in quite different ways. The intersurface mapping algorithm presented in Chapter 3 creates a map between two meshes, allowing a single generalized procedure to create planar, spherical, torroidal, and simplicial parameterizations, and thus the creation of highly regular remeshes for a variety of input mesh topologies. The advancing front algorithm presented in Chapter 4 uses a generalized surface reconstruction based approach to create high quality adaptive meshes for arbitrary input surface definitions. Taken together, these methods can address many of the meshing and remeshing problems present in computer graphics, visualization, and scientific computing.

There are several avenues for future work related to the intersurface mapping algorithm. To improve speed, it may be possible to use fine-to-coarse propagation of information [120] to obtain better configurations at low resolutions. Though the map is already optimized in a fine-grained way, which largely accounts for the quality of the resulting correspondence, the quality may be able to be further improved by even finer optimization. Individual optimizations of the edge-edge intersections of the map would give even more control over the map. This may also significantly simplify the implementation of the algorithm, since the temporary 2D parameterization used for the vertex optimization could be avoided entirely. Huge meshes could be handled using a hybrid strategy; after running the intersurface mapping algorithm to create a good midresolution map, the finer map could be defined using simplicial map composition, since the simplicial pieces may be small and flat enough to avoid numerical stability issues and geometric detail mismatch.

An interesting open question is how to extend the intersurface mapping method to handle multiple input meshes. Simultaneously optimizing an all-to-all map would not scale, and additional constraints on the maps would need to be introduced to maintain consistency (e.g., to enforce that  $\phi_{M^2 \rightarrow M^3} \circ \phi_{M^1 \rightarrow M^2} = \phi_{M^1 \rightarrow M^3}$ ). Using one model as domain would lose some benefits of directly optimizing intersurface maps, and the use of an intermediate domain is explicitly avoided in this work. Such an atlas of intersurface maps enables compatible remeshing of all of the input meshes.



Computing maps between surfaces of different genus is a difficult problem that is now receiving attention (e.g., [20]). Singularities are required in the map, associated with the topological features in one mesh that do not have corresponding features in the other. This introduces unbounded distortion in the parameterization, which makes optimization problematic.

The advancing front method based on the Lipschitz guidance field is a novel algorithm for meshing a large class of surfaces. The technique can be used for meshing a complete surface, or selectively triangulating a portion of it. The algorithm generates meshes that combine triangle quality, adaptivity and fidelity, which directly affect downstream processing. The algorithm is fast, robust, simple to implement, and produces very high quality meshes.

The advancing front algorithm can be applied to surfaces with many different underlying definitions, which is exploited to perform high quality CSG operations and mixed-mode CSG. This flexibility also allows it to be adapted for processing gigantic data with minimal changes to the core algorithm. The results generated by the advancing front algorithm are easily generalizable to different surface formulations. If a surface allows computation (or at least estimation) of the curvature, and points can be projected onto it, the advancing front technique can be used to create a high-quality, high-fidelity triangulation of it.

The meshes generated by the advancing front algorithm are dependent on the intrinsic geometry of the input surface, and not on aspects of the underlying surface definition, such as the grid on which implicit functions are sampled. This flexibility is especially important for extracting isosurfaces from high-order meshes, where the implicit function in each cell is dictated by a high-order (typically around degree 10 [104]) polynomial. In this case, pieces of the isosurface might have a much higher frequency than the domain vertex spacing. MC or MT would fail to capture these features, while the advancing front algorithm would reconstruct them faithfully.

The output mesh has both bounded distance and bounded normal error relative to the input surface. If that proves to be too restrictive, a downstream tool might choose to relax any of these constraints and simplify the mesh accordingly. Generating a good mesh at the beginning of the processing pipeline allows this as an option, without hindering alternative operations that may require those properties.

There are several intriguing avenues of future work related to the advancing front algorithm. While the user parameters  $\rho$  and  $\eta$  intuitively control the size and shape of the triangles, the user often requires direct control over the total number of triangles in the output. For example, if the mesh is to be used for a finite element simulation, there may be an upper limit on the size of the mesh to make the simulation tractable, but if fewer triangles are used, the results will suffer. The

number of triangles in the output depends not just on the user parameters, but also on aspects of the surface such as the area and distribution of curvature. It may be possible to relate the guidance field to the size of the output, but the user parameters affect  $g$  in nonlinear ways, making this a difficult problem.

Stronger properties of the algorithm may be able to be shown by using different ideal edge size functions  $\iota$ , such as the popular local feature size [118]. Perhaps it would be possible to show that the output mesh would be either homeomorphic or isotopic to the input surface, when using such a function. Similarly, it may also be possible to use the algorithm to produce an  $\varepsilon$ -sample of the surface.

Finally, providing triangle shape guarantees for all of the triangles generated by the algorithm is another interesting problem. Since connection triangles use an already existing vertex, they do not satisfy the aspect ratio constraint specified by  $\eta$ . The fact that poorly shaped connection triangles can almost always be improved through local edge flips implies that there may be a way to bound their qualities when connected in an optimal way. Dynamic programming provides a way of triangulating a polygon in an optimal and efficient way. Since each front is simply a polygon in  $\mathbb{R}^3$ , this may provide a tool to replace the heuristics used when closing the fronts after all the free triangles have been created. The dynamic programming solution, however, requires that only a single polygon be considered at a time. If each connected component of the input surface has only a single initial seed front on it, this will be the case. In practice this is not always true, especially when there are boundaries and features. Considering multiple fronts at the same time makes the optimal connection problem exponential in the total number of fronts, so it quickly becomes impractical.

# APPENDIX

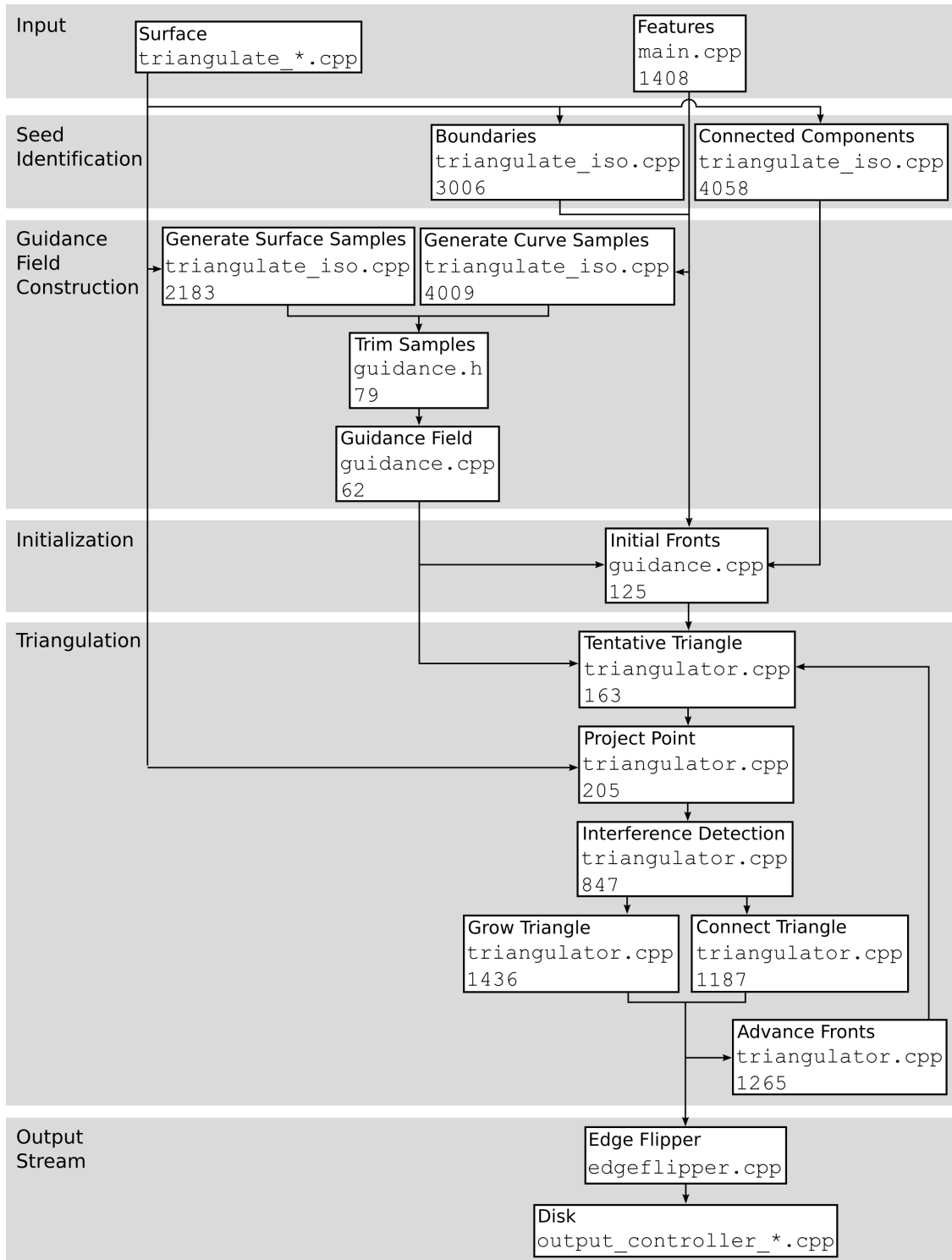
## ADVANCING FRONT SOFTWARE

### ARCHITECTURE

The prototype implementation of the work described in Chapter 4, called *Afront*, has been made available under the GNU General Public License (GPL). The source code, as well as binary executables are available for Windows, Linux, and Mac OS X, from <http://afront.sourceforge.net/>.

Afront is written in C++, and takes advantage of two significant language features: templates and abstract classes. Templates are used to allow all calculations to be performed in either single or double precision. This allows a trade-off between memory use and stability of some of the projection procedures. Little difference in efficiency has been seen, so double precision was used for all the results in this dissertation. Templates have also been used, for example, to implement implicit function surface definitions, where the data type of the source may be any numerical type (e.g., **unsigned char**, **short**, **float**, etc). Abstract classes are used throughout the algorithm to hide the underlying surface definition from the core triangulation module, and provide flexibility in the handling of the output mesh.

The diagram from Figure 4.1 has been annotated with source code files and lines in Figure A.1. This provides a direct mapping from the high-level components of the algorithm to the actual implementations. Some of the components that depend on the underlying surface definition are implemented in several places — once for each surface type. In these cases, the regular grid isosurface extraction is used, as it is often the most involved implementation.



**Figure A.1:** The advancing front diagram is annotated with source code files and lines for implementations of the individual components.

## A.1 Guidance Field

The guidance field is implemented for each surface type from an abstract base class:

```

class GuidanceField {
  virtual void* OrderedPointTraverseStart(const Point &p) = 0;
  virtual bool OrderedPointTraverseNext(void *ctx ,
                                         real &dist ,
                                         Point &point ,
                                         real &ideal) = 0;
  virtual void OrderedPointTraverseEnd(void *ctx) = 0;

  real MaxStepLength(const Point &p);
};

```

The guidance field value is evaluated by the nonvirtual `MaxStepLength` function. This calls the abstract `OrderedPointTraverse` functions to traverse the surface samples  $s$  and their associated ideal edge lengths, ordered by their distance from the evaluation point  $p$ . `OrderedPointTraverseStart` returns a traversal context, which allows multiple guidance field evaluations to be performed in parallel, without any details of how the traversal is implemented. `MaxStepLength` uses the traversal to evaluate the guidance field value as illustrated in Figure 4.18.

An abstract class is used to implement the guidance field since the surface samples  $s$  and computing the surface curvature for the ideal edge length depend on the underlying surface definition. Allowing the implementation details to be tailored to the underlying surface definition can significantly reduce the memory use and allows a great amount of flexibility. Input meshes, for example, use the vertices of the mesh as the guidance field samples. A *kd*-tree is built directly over these points, rather than duplicating them in a separate guidance field class. A `GuidanceField` class has also been implemented for CSG operations, which simply contains two `GuidanceField` instances of its own. The point traversal is performed by traversing both sub-instances in parallel, and always returning the closer of the two results. This idea is taken farther with the out-of-core guidance fields for implicit functions on regular grids. In this case, a separate guidance field for each block in the volume is used, and they are swapped in and out of memory as needed. Since the `MaxStepLength` function is only given a sequence of  $(point, ideal)$  pairs, it would also be straightforward plug in a different ideal edge size function, such as the local feature size.

## A.2 Surface Definitions

Many of the surface properties are encapsulated by abstract class:

```
class Surface {
    virtual int ProjectPoint(const FrontElement &base1 ,
                            const FrontElement &base2 ,
                            const Point &from_point ,
                            Point &to_point ,
                            Vector &to_normal) const = 0;
    virtual bool GetNextBlock(vector<vector<Point>> &points ,
                              vector<vector<Vector>> &normals ,
                              vector<bool> &loops) const = 0;
};
```

Implementations of this class are responsible for projecting points onto the surface, and well as finding the initial fronts (boundaries and connected component seed edges) for each block of the surface (only out-of-core surfaces have more than one block). The `FrontElement` structures contain information about vertices in the advancing fronts, including their position, normal, and guidance field value. When projecting a point, the tentative triangle being created is represented by the two bases, and the `from_point`. The normals in the `FrontElement` structures can be used to provide consistently oriented normals for surface definitions that do not have explicit orientations (e.g., nonlinear MLS surfaces). The `FrontElement` bases can also be used to ensure that the edge lengths of the resulting triangle match the guidance field, but may be ignored if the surface definition cannot easily make use of them. In `Afront`, they are used for projections on all surface definitions except for point set surfaces.

The `GetNextBlock` function is used to update the current block and create initial fronts for the triangulation. These initial fronts are given as simple ordered sets of points and normals. Each set also has an associated boolean `loops` value, specifying whether the front is a complete loop or just a segment. This allows incremental construction of the boundaries, as encountered during out-of-core meshing.

### A.3 Output Streams

The output of the advancing front algorithm is handled by a set of `OutputController` classes:

```
class OutputController {
    virtual void AddVertex(int index,
                          const Point &p,
                          const Vector &n) = 0;
    virtual void AddTriangle(int v1,
                            int v2,
                            int v3) = 0;
    virtual void FinalizeVertex(int index) = 0;
};
```

As vertices are generated, they are assigned an index and passed to the `OutputController`. Triangles refer to these indices when they are created. When a vertex is no longer on any of the advancing fronts, the output stream is notified with a `FinalizeVertex` call. This base class can be implemented to serve a number of different purposes.

The most fundamental `OutputController` classes are those that write the output mesh to disk. Many mesh file formats require all of the vertices to be written before any of the triangles, such as `.ply` and `.off`. Straightforward implementations would require storing the entire mesh in memory before writing it to disk. Instead, file formats that allow interleaving vertices and triangles are used. In particular, `OutputController` classes have been implemented for the `.sma` and `.smb` formats of Isenburg and Lindstrom [72].

An `OutputController` class has also been written that performs edge flips within a band of the advancing fronts, as described in Section 4.5. When vertices are passed to this class, they are cached for its own use, as well as passed on to another `OutputController`. Triangles are not passed on, but cache until its vertices are finalized with `FinalizeVertex`. When this happens, edge flips are performed around the vertex. Any triangles that are sufficiently far from all nonfinalized vertices are then passed down to the next `OutputController`, and any vertices left with no triangles are finalized.

An `OutputController` class has also been implemented to visualize the advancing fronts, as well as provide a tool for debugging. As it receives vertices and triangles, they are stored in memory and drawn in a GUI with OpenGL. This allows interactive inspection of the output mesh, while it is being generated. Since this stores the entire mesh in memory, and continually updates the display, it can be a significant bottleneck in the algorithm. To address this, `Afront` provides a batch-mode which simply does not attach an instance of this class to the output stream.

## A.4 Triangulator Module

The core advancing front triangulation algorithm is implemented in a `Triangulator` class:

```
class Triangulator {
    Triangulator(GuidanceField &guidance ,
                Surface &surface ,
                OutputController &output);

    Heap<FrontElement*> heap_local;
    Heap<FrontElement*> heap_global;

    KDTree<FrontElement*> kdtree_local;
    KDTree<FrontElement*> kdtree_global;
};
```

When this class is instantiated, it is passed instances of the `GuidanceField`, `Surface`, and `OutputController` classes. This completely hides the details of the underlying surface representation from the core triangulator. This class is responsible for the high-level triangulation procedure, including choosing which edges to grow triangles from, performing the front interference detection, and maintaining the advancing fronts as they new triangles are created and they split and merge.

The triangulation is started by retrieving the initial fronts from the `Surface` class. The fronts are stored as doubly linked lists of `FrontElements`, which allows the split and merge operations to be performed in constant time. When a vertex is inserted into a front, the `GuidanceField` class is used to determine the allowed edge size at that point. A heap is used to efficiently choose which edge would create the highest quality triangle when using those edge lengths. The best edge is used to create a vertex for a tentative triangle that is then projected with the `Surface` class. A *kd-tree* is then used to check this new triangle for interference with any other fronts. Either a connection triangle, or a free triangle and a new vertex are then sent to the `OutputController` class.

The triangulator restricts the advancing fronts to choose only edges within a small working area by modifying the priorities of the edges in the heap. Since edges are continually added and removed from the *kd-tree*, it may become highly unbalanced. To address this, the tree is rebuilt every time the working block is updated. This, however, can be extremely slow when there are a large number of edges in the fronts. To accommodate out-of-core surfaces, two levels of heaps and *kd-trees* are used. All edges within the active block are inserted into the *local* heap and *kd-tree*. Only this local *kd-tree* is rebuilt when the working area is changed. When the entire active block has been triangulated, it is changed and the new seeds and boundaries are retrieved from the `Surface` class with `GetNextBlock`. At this time, all of the edges in the local heap and



*kd-tree* are migrated to a *global* heap and *kd-tree*, and old edges that are now in the active block are move to the local structures. Partitioning the edges into two sets in this way greatly improves efficiency by reducing the depths of the *kd-trees* and heaps, since usually only the local ones need to be queried.

## REFERENCES

- [1] A. SHEFFER, E. PRAUN, K. R. Mesh parameterization methods and their applications. *Foundation and Trends in Computer Graphics and Vision* 2, 2 (2006), 105–171.
- [2] ADAMSON, A., AND ALEXA, M. Approximating and intersecting surfaces from points. In *Symposium on Geometry Processing* (2003), 230–239.
- [3] AKKOUCHE, S., AND GALIN, E. Adaptive implicit surface polygonization using marching triangles. *Computer Graphics Forum* 20, 2 (2001), 67–80.
- [4] AKSOYLU, B., KHODAKOVSKY, A., AND SCHRÖDER, P. Multilevel solvers for unstructured surface meshes. *SIAM J. Sci. Comput* 26 (2003).
- [5] ALEXA, M. Merging polyhedral shapes with scattered features. In *International Conference on Shape Modeling and Applications* (1999), 202.
- [6] ALEXA, M., AND ADAMSON, A. On normals and projection operators for surfaces defined by point sets. In *Symposium on Point-Based Graphics* (2004), 149–155.
- [7] ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. T. Point set surfaces. In *IEEE Visualization 2001* (2001), 21–28.
- [8] ALLGOWER, E. L., AND SCHMIDT, P. H. An algorithm for piecewise-linear approximation of an implicitly defined manifold. *SIAM Journal on Numerical Analysis* 22, 2 (1985), 322–346.
- [9] ALLIEZ, P., COHEN-STEINER, D., YVINEC, M., AND DESBRUN, M. Variational tetrahedral meshing. *ACM Trans. Graph.* 24, 3 (2005), 617–625.
- [10] ALLIEZ, P., DE VERDIÈRE, É. C., DEVILLERS, O., AND ISENBURG, M. Isotropic surface remeshing. In *Shape Modeling International* (2003), 49–58.
- [11] ALLIEZ, P., MEYER, M., AND DESBRUN, M. Interactive geometry remeshing. *ACM Transactions on Graphics* 21, 3 (2002), 347–354.
- [12] ALLIEZ, P., UCELLI, G., GOTSMAN, C., AND ATTENE, M. Recent advances in remeshing of surfaces. In *State-of-the-art report of the AIM@SHAPE EU network*. Springer, 2005.
- [13] AMENTA, N., AND BERN, M. Surface reconstruction by voronoi filtering. In *Symposium on Computational Geometry* (1998), 39–48.
- [14] AMENTA, N., BERN, M., AND KAMVYSSSELIS, M. A new voronoi-based surface reconstruction algorithm. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1998), 415–422.

- [15] AMENTA, N., CHOI, S., DEY, T. K., AND LEEKHA, N. A simple algorithm for homeomorphic surface reconstruction. In *Symposium on Computational Geometry* (2000), 213–222.
- [16] AMENTA, N., CHOI, S., AND KOLLURI, R. K. The power crust. In *Symposium on Solid Modeling and Applications* (2001), 249–266.
- [17] AMENTA, N., AND KIL, Y. J. Defining point-set surfaces. *ACM Transactions on Graphics* 23, 3 (2004), 264–270.
- [18] BAJAJ, C. L., CHEN, J., AND XU, G. Modeling with cubic a-patches. *ACM Transactions on Graphics* 14, 2 (1995), 103–133.
- [19] BAJAJ, C. L., PASCUCCI, V., AND SCHIKORE, D. R. Fast isocontouring for improved interactivity. In *1996 Volume Visualization Symposium* (1996), 39–46.
- [20] BENNETT, J., PASCUCCI, V., AND JOY, K. Genus oblivious cross parameterization: Robust topological management of inter-surface maps. In *Pacific Conference on Computer Graphics and Applications* (2007), 238–247.
- [21] BERNARDINI, F., MITTLEMAN, J., RUSHMEIER, H., SILVA, C., AND TAUBIN, G. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics* 5, 4 (1999), 349–359.
- [22] BLOOMENTHAL, J. An implicit surface polygonizer. In *Graphics Gems IV*, P. Heckbert, Ed. Academic Press, Boston, 1994, 324–349.
- [23] BLOOMENTHAL, J., Ed. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.
- [24] BOISSONNAT, J.-D., COHEN-STEINER, D., AND VEGTER, G. Isotopic implicit surface meshing. In *Symposium on Theory of Computing* (2004), 301–309.
- [25] BOISSONNAT, J. D., AND OUDOT, S. Provably good surface sampling and approximation. In *Symposium on Geometry Processing* (2003), 9–18.
- [26] BOLITHO, M., KAZHDAN, M., BURNS, R., AND HOPPE, H. Multilevel streaming for out-of-core surface reconstruction. In *Symposium on Geometry Processing* (2007), 69–78.
- [27] BOROUCAKI, H., HECHT, F., AND FREY, P. J. Mesh gradation control. In *6th International Meshing Roundtable* (1997), 131–141.
- [28] CARR, J. C., BEATSON, R. K., CHERRIE, J. B., MITCHELL, T. J., FRIGHT, W. R., MCCALLUM, B. C., AND EVANS, T. R. Reconstruction and representation of 3d objects with radial basis functions. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (2001), 67–76.
- [29] CATMULL, E., AND ROM, R. A class of local interpolating splines. In *Computer Aided Geometric Design* (1974), 317–326.
- [30] CEBRAL, J. R., AND LÖHNER, R. From medical images to CFD meshes. In *8th International Meshing Roundtable* (1999), 321–331.

- [31] CERMÁK, M., AND SKALA, V. Adaptive edge spinning algorithm for polygonization of implicit surfaces. In *Computer Graphics International* (2004), 36–43.
- [32] CHIANG, Y.-J., AND SILVA, C. T. I/O optimal isosurface extraction. In *IEEE Visualization* (1997), 293–300.
- [33] CIGNONI, P., GANOVEII, F., MONTANI, C., AND SCOPIGNO, R. Reconstruction of topologically correct and adaptive trilinear isosurfaces. *Computers & Graphics* 24, 3 (2000), 399–418.
- [34] CIGNONI, P., MARINO, P., MONTANI, C., PUPPO, E., AND SCOPIGNO, R. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics* 3, 2 (1997), 158–170.
- [35] CIGNONI, P., ROCCHINI, C., AND SCOPIGNO, R. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum* 17, 2 (1998), 167–174.
- [36] COURANT, R., AND JOHN, F. *Introduction to Calculus and Analysis*, vol. 2. John Wiley and sons, 1974.
- [37] CROSSNO, P., AND ANGEL, E. Isosurface extraction using particle systems. In *IEEE Visualization 1997* (1997), R. Yagel and H. Hagen, Eds., 495–498.
- [38] CURLESS, B., AND LEVOY, M. A volumetric method for building complex models from range images. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1996), 303–312.
- [39] DAHMEN, W., MICCHELLI, C. A., AND SEIDEL, H.-P. Blossoming begets bsplines built better by b-patches. *Mathematics of Computation* 59, 199 (1992), 97–115.
- [40] DANIELS, J. I., HA, L. K., OCHOTTA, T., AND SILVA, C. T. Robust smooth feature extraction from point clouds. In *IEEE International Conference on Shape Modeling and Applications* (2007), 123–136.
- [41] DE FIGUEIREDO, L. H., GOMES, J., TERZOPOLOUS, D., AND VELHO, L. Physically-based methods for polygonization of implicit surfaces. In *Graphics Interface* (1992), 250–257.
- [42] DESBRUN, M., MEYER, M., AND ALLIEZ, P. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum* 21 (2002), 209–218.
- [43] DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. Implicit fairing of irregular meshes using diffusion and curvature flow. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1999), 317–324.
- [44] DEY, T. K., AND GIESEN, J. Detecting undersampling in surface reconstruction. In *Symposium on Computational Geometry* (2001), 257–263.
- [45] DIETRICH, C. A., SCHEIDEGGER, C. E., SCHREINER, J., JO A. L. D. C., NEDEL, L. P., AND SILVA, C. T. Edge transformations for improving mesh quality of marching cubes. *IEEE Transactions on Visualization and Computer Graphics* 15, 1 (2009), 150–159.

- [46] DOBKIN, D. P., WILKS, A. R., LEVY, S. V. F., AND THURSTON, W. P. Contour tracing by piecewise linear approximations. *ACM Transactions on Graphics* 9, 4 (1990), 389–423.
- [47] DU, Q., FABER, V., AND GUNZBURGER, M. Centroidal voronoi tessellations: Applications and algorithms. *Siam Review* 41, 4 (1999), 637–676.
- [48] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution analysis of arbitrary meshes. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1995), 173–182.
- [49] FAN, Z., JIN, X., FENG, J., AND SUN, H. Mesh morphing using polycube-based cross-parameterization: Animating geometrical models. *Computer Animation and Virtual Worlds* 16, 3-4 (2005), 499–508.
- [50] FIORIN, V., CIGNONI, P., AND SCOPIGNO, R. Out-of-core mls reconstruction. In *IASTED International Conference on Computer Graphics and Imaging* (2007), 27–34.
- [51] FLEISHMAN, S., COHEN-OR, D., AND SILVA, C. Robust moving least squares fitting with sharp features. *ACM Transactions on Graphics* 24, 3 (2005), 544–552.
- [52] FLOATER, M. S. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design* 14, 3 (1997), 231–250.
- [53] FLOATER, M. S. Mean value coordinates. *Computer Aided Geometric Design* 20, 1 (2003), 19–27.
- [54] FLOATER, M. S., AND HORMANN, K. Surface parameterization: a tutorial and survey. In *Advances in Multiresolution for Geometric Modelling*. 2005, 157–186.
- [55] GARIMELLA, R. V., AND SWARTZ, B. K. Curvature estimation for unstructured triangulations of surfaces. Tech. Rep. LA-UR-03-8240, Los Alamos National Laboratory, Nov. 2003.
- [56] GARLAND, M., AND HECKBERT, P. S. Surface simplification using quadric error metrics. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1997), 209–216.
- [57] GOTSMAN, C., GU, X., AND SHEFFER, A. Fundamentals of spherical parameterization for 3d meshes. *ACM Transactions on Graphics* 22, 3 (2003), 358–363.
- [58] GU, X., GORTLER, S. J., AND HOPPE, H. Geometry images. *ACM Transactions on Graphics* (2002), 355–361.
- [59] GU, X., AND YAU, S.-T. Global conformal surface parameterization. In *Symposium on Geometry Processing* (2003), 127–137.
- [60] GUSKOV, I., SWELDENS, W., AND SCHRÖDER, P. Multiresolution signal processing for meshes. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1999), 325–334.
- [61] GUSKOV, I., VIDIMČE, K., SWELDENS, W., AND SCHRÖDER, P. Normal meshes. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (2000), 95–102.

- [62] HAKER, S., ANGENENT, S., TANNENBAUM, A., KIKINIS, R., SAPIRO, G., AND HALLE, M. Conformal surface parameterization for texture mapping. *IEEE Transactions on Visualization and Computer Graphics* 6, 2 (2000), 181–189.
- [63] HARTMANN, E. A marching method for the triangulation of surfaces. *The Visual Computer* 14, 3 (1998), 95–108.
- [64] HILDEBRANDT, K., POLTHIER, K., AND WARDETZKY, M. Smooth feature lines on surface meshes. In *Eurographics Symposium on Geometry Processing* (2005), 85–90.
- [65] HILTON, A., STODDART, A., ILLINGWORTH, J., AND WINDEATT, T. Marching triangles: Range image fusion for complex object modelling. *International Conference on Image Processing B* (1996), 381–384.
- [66] HOPPE, H. Progressive meshes. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1996), 99–108.
- [67] HOPPE, H., DE ROSE, T., DUCHAMP, T., HALSTEAD, M., JIN, H., McDONALD, J., SCHWEITZER, J., AND STUETZLE, W. Piecewise smooth surface reconstruction. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1994), 295–302.
- [68] HOPPE, H., DE ROSE, T., DUCHAMP, T., McDONALD, J., AND STUETZLE, W. Surface reconstruction from unorganized points. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1992), 71–78.
- [69] HOPPE, H., DE ROSE, T., DUCHAMP, T., McDONALD, J., AND STUETZLE, W. Mesh optimization. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1993), 19–26.
- [70] HORMANN, K., AND GREINER, G. MIPS: An efficient global parametrization method. In *Curve and Surface Design* (1999), 153–162.
- [71] HORMANN, K., GREINER, G., AND CAMPAGNA, S. Hierarchical parametrization of triangulated surfaces. In *Vision, Modeling, and Visualization* (1999), 219–226.
- [72] ISENBURG, M., AND LINDSTROM, P. Streaming meshes. In *IEEE Visualization* (October 2005), 231–238.
- [73] JIN, M., LUO, F., AND GU, X. D. Computing general geometric structures on surfaces using ricci flow. *Computer Aided Design* 39, 8 (2007), 663–675.
- [74] JOSHI, S. C., PIZER, S. M., FLETCHER, P. T., THALL, A., AND TRACTON, G. Multi-scale 3-D deformable model segmentation based on medial description. In *International Conference on Information Processing in Medical Imaging* (2001), 64–77.
- [75] JU, T., LOSASSO, F., SCHAEFER, S., AND WARREN, J. Dual contouring of hermite data. *ACM Transactions on Graphics* 21, 3 (2002), 339–346.
- [76] KARKANIS, T., AND STEWART, A. Curvature-dependent triangulation of implicit surfaces. *IEEE Computer Graphics and Applications* 21, 2 (2001), 60–69.

- [77] KAZHDAN, M., BOLITHO, M., AND HOPPE, H. Poisson surface reconstruction. In *Symposium on Geometry Processing* (2006), 61–70.
- [78] KAZHDAN, M. M. Reconstruction of solid models from oriented point sets. In *Symposium on Geometry Processing* (2005), 73–82.
- [79] KHODAKOVSKY, A., LITKE, N., AND SCHRÖDER, P. Globally smooth parameterizations with low distortion. *ACM Transactions on Graphics* 22, 3 (2003), 350–357.
- [80] KINDLMANN, G., WHITAKER, R., TASDIZEN, T., AND MLLER, T. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *IEEE Visualization* (October 2003), 513–520.
- [81] KRAEVOY, V., AND SHEFFER, A. Cross-parameterization and compatible remeshing of 3d models. *ACM Transactions on Graphics* (2004), 861–869.
- [82] KRAEVOY, V., SHEFFER, A., AND GOTSMAN, C. Matchmaker: Constructing constrained texture maps. *ACM Transactions on Graphics* 22, 3 (2003), 326–333.
- [83] LANCASTER, P., AND SALKAUSKAS, K. Surfaces generated by moving least squares methods. *Mathematics of Computation* 87 (1981), 141–158.
- [84] LAZARUS, F., POCCHIOLA, M., VEGTER, G., AND VERROUST, A. Computing a canonical polygonal schema of an orientable triangulated surface. In *ACM Symposium on Computational Geometry* (2001), 80–89.
- [85] LEDERGERBER, C., GUENNEBAUD, G., MEYER, M., BÄCHER, M., AND PFISTER, H. Volume mls ray casting. *IEEE Transaction on Visualization and Computer Graphics* 14, 6 (2008), 1372–1379.
- [86] LEE, A. W. F., DOBKIN, D., SWELDENS, W., AND SCHRÖDER, P. Multiresolution mesh morphing. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1999), 343–350.
- [87] LEE, A. W. F., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. Maps: Multiresolution adaptive parameterization of surfaces. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1998), 95–104.
- [88] LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics* 21, 3 (2002), 362–371.
- [89] LINDSTROM, P. Out-of-core simplification of large polygonal models. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (2000), 259–262.
- [90] LIVNAT, Y., SHEN, H., AND JOHNSON, C. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics* 2, 1 (1996), 73–84.
- [91] LLOYD, S. Least squares quantization in pcm. *IEEE Transactions on Information Theory* 28 (1982), 129–137.

- [92] LORENSEN, W., AND CLINE, H. Marching cubes: A high resolution 3D surface reconstruction algorithm. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1987), 163–169.
- [93] MAILLOT, J., YAHIA, H., AND VERROUST, A. Interactive texture mapping. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1993), 27–34.
- [94] MASCARENHAS, A., ISENBURG, M., PASCUCCI, V., AND SNOEYINK, J. Encoding volumetric grids for streaming isosurface extraction. In *Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium* (2004), IEEE Computer Society, 665–672.
- [95] MATSUMOTO, Y. *Introduction to Morse Theory*. American Mathematical Society, 1997.
- [96] MEDEROS, B., AMENTA, N., VELHO, L., AND DE FIGUEIREDO, L. H. Surface reconstruction from noisy point clouds. In *Symposium on Geometry Processing* (2005), 53.
- [97] MEYER, M., DESBRUN, M., SCHRÖDER, AND BARR, A. H. Discrete differential-geometry operators for triangulated 2-manifolds. In *VisMath* (2002), 35–57.
- [98] MEYER, M., GEORGEL, P., AND WHITAKER, R. Robust particle systems for curvature dependent sampling of implicit surfaces. In *International Conference on Shape Modeling and Applications* (2005), 123–133.
- [99] MEYER, M., KIRBY, R. M., AND WHITAKER, R. Topology, accuracy, and quality of isosurface meshes using dynamic particles. *IEEE Transactions on Visualization and Computer Graphics* 12, 5 (September-October 2007).
- [100] MEYER, M., WHITAKER, R., KIRBY, R. M., LEDERGERBER, C., AND PFISTER, H. Particle-based sampling and meshing of surfaces in multimaterial volumes. *IEEE Transactions on Visualization and Computer Graphics* 14, 6 (November-December 2008), 1539–1546.
- [101] MIRIN, A. A., COHEN, R. H., CURTIS, B. C., DANNEVIK, W. P., DIMITS, A. M., DUCHAINEAU, M. A., ELIASON, D. E., SCHIKORE, D. R., ANDERSON, S. E., PORTER, D. H., WOODWARD, P. R., SHIEH, L. J., AND WHITE, S. W. Very high resolution simulation of compressible turbulence on the ibm-sp system. In *Supercomputing '99: Proceedings of the 1999 ACM/IEEE Conference on Supercomputing (CDROM)* (1999), ACM, 70.
- [102] MOORE, R. E. *Interval Analysis*. Prentice-Hall, 1966.
- [103] NATARAJAN, B. K. On generating topologically consistent isosurfaces from uniform samples. *The Visual Computer* 11, 1 (1994), 52–62.
- [104] NELSON, B., AND KIRBY, R. M. Ray-tracing polymorphic multi-domain spectral/hp elements for isosurface rendering. *IEEE Transactions in Visualization and Computer Graphics* 12, 1 (2006), 114–125.
- [105] NIELSON, G. On marching cubes. *IEEE Transactions on Visualization and Computer Graphics* 9, 3 (2003), 283–297.



- [106] NIELSON, G., HAGEN, H., AND MÜLLER, H. *Scientific Visualization*. IEEE Computer Society, 1997, ch. 20.
- [107] NIELSON, G. M. Dual marching cubes. In *IEEE Visualization* (2004), 489–496.
- [108] NIELSON, G. M., AND HAMANN, B. The asymptotic decider: Resolving the ambiguity in marching cubes. In *IEEE Visualization* (1991), 83–91.
- [109] NING, P., AND BLOOMENTHAL, J. An evaluation of implicit surface tilers. *IEEE Computer Graphics and Applications* 13, 6 (1993), 33–41.
- [110] OWEN, S. A survey of unstructured mesh generation technology. <http://www.andrew.cmu.edu/user/sowen/survey/index.html>.
- [111] PERSSON, P.-O. Pde-based gradient limiting for mesh size functions. In *Proceedings of 13th International Meshing Roundtable* (2004), 377–387.
- [112] PIEGL, L., AND TILLER, W. *The NURBS Book*. Springer, 1997.
- [113] PINKALL, U., JUNI, S. D., AND POLTHIER, K. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics* 2 (1993), 15–36.
- [114] PLANTINGA, S., AND VEGTER, G. Isotopic approximation of implicit curves and surfaces. In *Symposium on Geometry Processing* (2004), 245–254.
- [115] PRAUN, E., AND HOPPE, H. Spherical parametrization and remeshing. *ACM Transactions on Graphics* 22, 3 (2003), 340–349.
- [116] PRAUN, E., SWELDENS, W., AND SCHRÖDER, P. Consistent mesh parameterizations. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (2001), 179–184.
- [117] RAMAN, S., AND WENGER, R. Quality isosurface mesh generation using an extended marching cubes lookup table. *Computer Graphics Forum* 27, 3 (2008), 791–798.
- [118] RUPPERT, J. A new and simple algorithm for quality 2-dimensional mesh generation. In *Symposium on Discrete Algorithms* (1993), 83–92.
- [119] SABA, S., YAVNEH, I., GOTSMAN, C., AND SHEFFER, A. Practical spherical embedding of manifold triangle meshes. In *International Conference on Shape Modeling and Applications* (2005), 258–267.
- [120] SANDER, P. V., GORTLER, S. J., SNYDER, J., AND HOPPE, H. Signal-specialized parametrization. In *Eurographics Workshop on Rendering* (2002), 87–98.
- [121] SANDER, P. V., SNYDER, J., GORTLER, S. J., AND HOPPE, H. Texture mapping progressive meshes. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (2001), 409–416.
- [122] SCHEIDEGGER, C. E., FLEISHMAN, S., AND SILVA, C. T. Triangulating point set surfaces with bounded error. In *Symposium on Geometry Processing* (2005), 63–72.

- [123] SCHREINER, J., ASIRVATHAM, A., PRAUN, E., AND HOPPE, H. Inter-surface mapping. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (2004), 870–877.
- [124] SCHREINER, J., SCHEIDEGGER, C., FLEISHMAN, S., AND SILVA, C. T. Direct (re)meshing for efficient surface processing. *Computer Graphics Forum (Proceedings of Eurographics 2006)* 25, 3 (2006), 527–536.
- [125] SCHREINER, J., SCHEIDEGGER, C., AND SILVA, C. T. High-quality extraction of isosurfaces from regular and irregular grids. *IEEE Transactions on Visualization and Computer Graphics* 12, 5 (2006), 1205–1212.
- [126] SHEFFER, A., AND HART, J. C. Seamster: Inconspicuous low-distortion texture seam layout. In *IEEE Visualization* (2002), 291–298.
- [127] SILVA, C. T., AND MITCHELL, J. S. B. Greedy cuts: An advancing front terrain triangulation algorithm. In *ACM International Symposium on Advances in Geographic Information Systems* (1998), 137–144.
- [128] SORKINE, O. Differential representations for mesh processing. *Computer Graphics Forum* 25, 4 (2006).
- [129] SORKINE, O., COHEN-OR, D., GOLDENTHAL, R., AND LISCHINSKI, D. Bounded-distortion piecewise mesh parameterization. In *IEEE Visualization* (2002), 355–362.
- [130] SORKINE, O., LIPMAN, Y., COHEN-OR, D., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. Laplacian surface editing. In *Symposium on Geometry Processing* (2004), 179–188.
- [131] SUMNER, R. W., AND POPOVIĆ, J. Deformation transfer for triangle meshes. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (2004), 399–405.
- [132] SUMNER, R. W., ZWICKER, M., GOTSMAN, C., AND POPOVIĆ, J. Mesh-based inverse kinematics. *ACM Transactions on Graphics* 24, 3 (2005), 488–495.
- [133] SURAZHSKY, V., ALLIEZ, P., AND GOTSMAN, C. Isotropic remeshing of surfaces: A local parametrization approach. In *International Meshing Roundtable* (2003).
- [134] SURAZHSKY, V., AND GOTSMAN, C. Explicit surface remeshing. In *Symposium on Geometry Processing* (2003), 20–30.
- [135] TREECE, G., PRAGER, R., AND GEE, A. Regularised marching tetrahedra: Improved iso-surface extraction. *Computers & Graphics* 23, 4 (1999), 583–598.
- [136] TURK, G. Re-tiling polygonal surfaces. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1992), 55–64.
- [137] TURK, G., AND O’BRIEN, J. F. Shape transformation using variational implicit functions. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (Aug. 1999), 335–342.
- [138] TUTTE, W. T. How to draw a graph. *London Mathematical Society* 13, 3 (1963), 743–768.

- [139] WILEY, D. F., AMENTA, N., ALCANTARA, D. A., GHOSH, D., KIL, Y. J., DELSON, E., HARCOURT-SMITH, W., JOHN, K. S., ROHLF, F. J., AND HAMANN, B. Evolutionary morphing. In *IEEE Visualization* (October 2005), 431–438.
- [140] WILHELMS, J., AND GELDER, A. V. Octrees for faster isosurface generation. *ACM Transactions on Graphics* 11, 3 (1992), 201–227.
- [141] WITKIN, A. P., AND HECKBERT, P. S. Using particles to sample and control implicit surfaces. In *International Conference on Computer Graphics and Interactive Techniques (Proc. SIGGRAPH)* (1994), 269–277.
- [142] WOOD, Z. J., SCHRÖDER, P., BREEN, D., AND DESBRUN, M. Semi-regular mesh extraction from volumes. In *IEEE Visualization* (2000), 275–282.
- [143] YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. Mesh editing with poisson-based gradient field manipulation. *ACM Transactions on Graphics* 23, 3 (2004), 644–651.