MESH GENERATION USING A CORRESPONDENCE DISTANCE FIELD

By

Nicholas Szapiro

Approved:

MESH GENERATION USING A CORRESPONDENCE DISTANCE FIELD

By

Nicholas Szapiro

A Thesis
Submitted to the faculty of
The University of Tennessee, Chattanooga
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computational Engineering

The University of Tennessee, Chattanooga
Chattanooga, Tennessee

August 2012

ABSTRACT

The central tool of this work is a correspondence distance field to discrete surface points embedded within a quadtree data structure. The theory, development, and implementation of the distance field tool are described, and two main applications to two-dimensional mesh generation are presented with extension to three-dimensional capabilities in mind.

First is a method for surface-oriented mesh generation from a sufficiently dense set of discrete surface points without connectivity information. Contour levels of distance from the body are specified and correspondences oriented normally to the contours are created. Regions of merging fronts inside and between objects are detected in the correspondence distance field and incorporated automatically.

Second, the boundaries in a Voronoi diagram between specified coordinates are detected adaptively and used to make a Delaunay tessellation. Tessellation of regions with holes is performed using ghost nodes.

Images of meshes for each method are given for a sample set of test cases. Possible extensions, future work, and CFD applications are also discussed.

DEDICATION

To the reader:

Standing on enough shoulders, everyone is a giant.

ACKNOWLEDGEMENTS

To pursue a topic out of curiosity is a great privilege. With deep gratitude, I thank all of those who contributed to this opportunity. Among the many, I would like to thank the following in particular. I thank Dr. Karman for both his guidance and Jedi-like faith in using the tree. I thank my family for their love and support. I thank my friends for sharing their worlds with me.

This work was funded by a grant originating from several Chattanooga-based foundations, specifically: the Lyndhurst Foundation, the Benwood Foundation, the Tucker Foundation, the UC Foundation, the Maclellan Foundation, the Community Foundation of Greater Chattanooga, as well as several anonymous donors. This support is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Overview

Meshes are used to discretize space for finite mesh-based methods in computational field simulations. The mesh describes the geometry and topology of the spatial domain explicitly through cells with connectivity. For the reader unaccustomed to these topics, a brief [background] is provided. Note that all [text] is a mouse-aware link in the PDF version of this thesis.

The shapes of the cells and the orientation of the mesh can affect the accuracy of the numerical solution. While the numerical solution is the true test of the quality of the mesh, there are guidelines for a-priori metrics. Two qualities are of specific interest to this work, shape and orientation. Regarding the shapes, high aspect ratios and skewness may lead to diffusion, slowed convergence, or instability. Regarding orientation, for viscous flow about an object, an orientation normal to the surface may lead to higher accuracy or a smaller number of elements. See [Shewchuk, 2002b] for a more precise, fuller discussion.

We discuss methods to address each of these concerns using the central tool of this work, a correspondence distance field (CDF) to discrete surface points embedded within a quadtree data structure. The theory and development of the CDF within the context of the technical literature is given in Chapter 2. Chapter 2 also provides the context for understanding the process of geometry definition to mesh generation for simulation. Our implementation of the distance field tool is described in Chapter 3, including various applications of extracting information and features of interest from the field.

Chapter 4 builds the implemented tools into the two main algorithms of this work. First is a method for surface-oriented mesh generation from discrete surface points intended to create a mesh oriented normally to the body. Contour levels of distance from the body are specified and correspondences oriented normally to the contours are created. Regions of merging fronts inside and between objects can be detected in the CDF and incorporated automatically. The implicit representation of the geometry allows the input geometry to originate from images or sufficiently dense point clouds with no connectivity information. In the second method, the boundaries in a Voronoi diagram between specified coordinates are detected in an adaptive fashion and used to make a Delaunay tessellation. Known properties of Delaunay triangulations ensure that the elements are "round." Tessellation of regions with holes is performed using ghost nodes.

Chapter 5 applies these algorithms to a set of test cases. A discussion also helps the reader understand the capabilities and limitations of the implemented approaches. Chapter 6 concludes, including a discussion of possible extensions and directions for future work.

Background

Conservation laws such as conservation of energy or conservation of momentum arise from symmetries in an underlying space [Noether, 1918]. A number of physical phenomena in fields such as fluid dynamics, solid mechanics, and electromagnetics can be modeled for engineering use by casting conservation laws as partial differential equations (PDEs). Some famous systems are the Navier-Stokes equations, Newton's Laws, Lagrange's equations, and Maxwell's equations. It is a fundamental role of an engineer to judge whether the model is sufficiently accurate for the application.

In general, analytic solutions of these governing PDEs are difficult to obtain, so we must resort to numerical solution. Note that our background is mainly in the equations

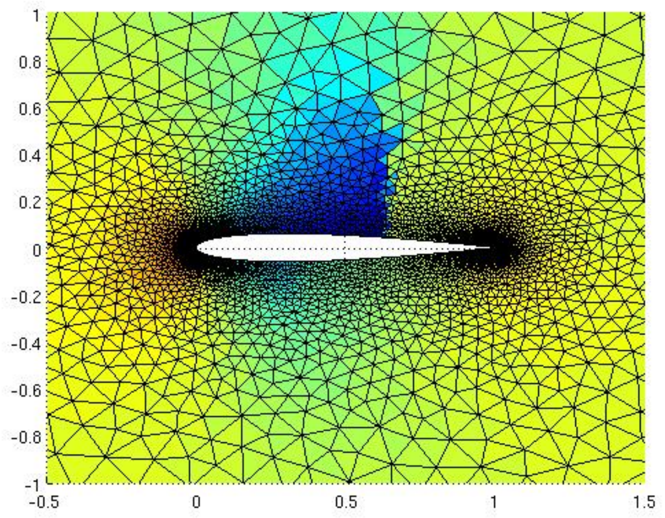


Figure 1.1 Density field for Euler flow over a triangular mesh

of computational fluid dynamics (CFD), and no distinction needs to be made between integral and differential equations for this work. The traditional tools for solving the coupled, nonlinear system of PDEs are the finite element, finite difference, and finite volume methods. In these, the space is discretized into a mesh consisting of a finite number of elements with connectivity, which cover the domain. Using these relationships and numerical methods, the governing equations can be transformed into a system of algebraic equations [Thompson et al., 1985]. The solution of these algebraic equations on the discretized domain produces an approximation of the PDEs on the entire domain, where the accuracy of the approximate solution depends on the discretization [Freitag and Ollivier-Gooch, 1997]. A universal definition for a "good" grid does not exist and is heavily dependent on the particular problem posed and the approach taken to solve the specific physics [Knupp, 2008]. Fig. 1.1 shows a density field for transonic Euler flow over a 2D NACA0012 airfoil. The mesh is composed of the triangular elements that cover the space, and the flow field can be used, for example, to compute lift and drag on the airfoil.

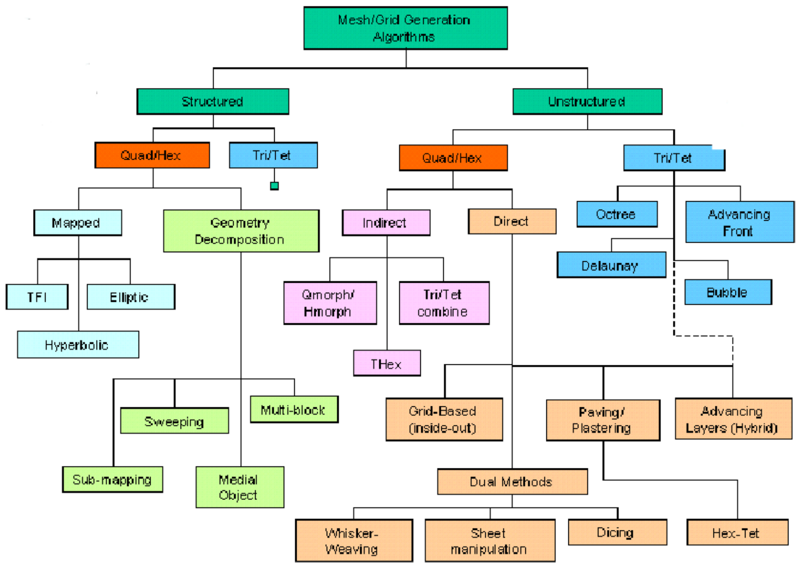


Figure 1.2 A variety of meshing algorithms [Owen, 2005]

Although not a topic of this research, it is worth noting that there are alternatives to these traditional methods. For example, particle methods establish algebraic equations for the whole problem domain without the use of a predefined mesh. They scatter a set of nodes within the problem domain and on the boundaries to represent the field so that no explicit topology is required. A local reconstruction of the field is usually obtained through a moving least-squares approach [Liu, 2003].

There exists an extensive literature for mesh generation with significant contributions from computer graphics, computational geometry, and the field simulation communities. A full review is beyond the scope of this work, but Fig. 1.2 shows a sampling of algorithms with surveys given in [Owen, 1998] and [Thompson et al., 1985]. In this categorization, the surface-oriented meshing lies in the unstructured, advancing layers region, and the Voronoi-dual meshing lies in the unstructured, Delaunay category.

We can summarize the traditional approaches to meshing based on an alternate categorization of extrusion, refinement, and smoothing. Extrusion is where points are created off of a mesh or boundary that covers a local region of the domain. Techniques such as advancing front and advancing layers are prime examples. Refinement is where we take a coarse mesh of the domain and successively divide elements. Hierarchical adaptation and Delaunay insertion algorithms are widely used techniques. Our approaches fall under this category. Smoothing is where the coordinates of or connectivities between a known number of nodes are adjusted. This would include techniques based on elliptic PDEs, transfinite interpolation, and optimization. The relevant approaches are discussed more fully in Chapter 2.

Motivation

There are a number of practical CFD applications where the geometry is not defined by parametric surfaces. For examples, Fig. 1.3 shows a possible transformation of a medical image of an artery into a mesh, and markers used to reconstruct the flight of a dragonfly. The medical applications may be used to better understand the human body and improve healthcare. The analysis of dragonfly and hummingbird flight has applications in the design of highly maneuverable small unmanned aerial vehicles. Some discussion of the research that exists on these topics is included in Chapter 2.

To our knowledge, any geometry format can be transformed into a collection of points. Our surface-oriented meshing algorithm leverages an implicit representation of the geometry to base an approach on a discrete set of surface samples. Given a method for meshing these geometries, our intent is to add tools to the fields that analyze such systems.

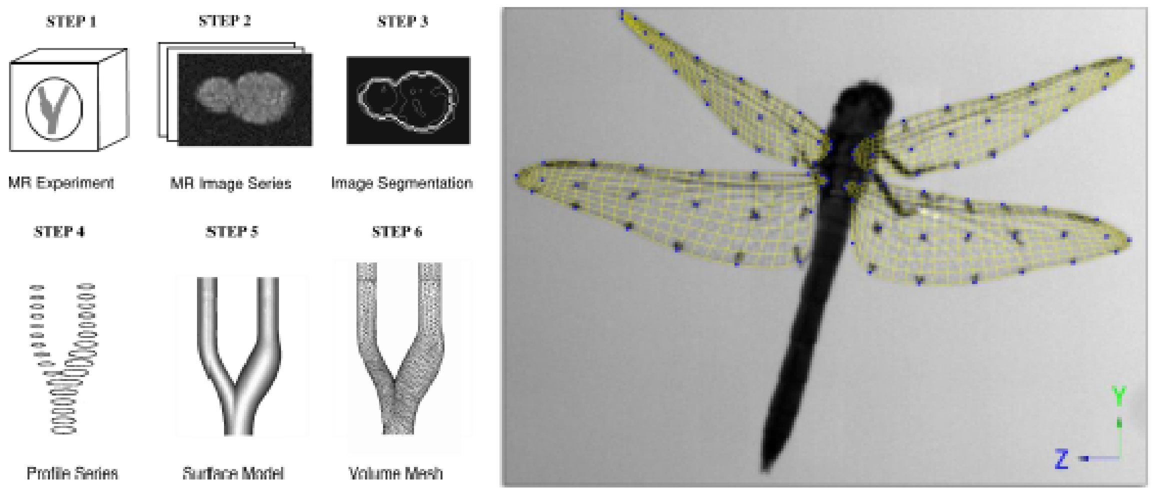


Figure 1.3 Examples of CFD applications not originating from parametric geometry [Moore et al., 1999], [Dong et al., 2010]

Development

A brief summary of the core of the historical progression of this research project may be helpful to the reader for orientation and synthesis.

For a research project looking to improve automatic target recognition software, the author gained familiarity with the tools of image segmentation. Specifically, we focused on the extraction of ships from images, including cases with atmospheric noise such as haze. During this project, the author gained additional familiarity with level sets through studying the active contour methods, and colleagues using level sets to measure geometric properties of placentas defined through slices of medical images.

After a brief period of generating structured grids "by hand" using Pointwise [Pointwise, 2012], the author was motivated to do research in the field of automated mesh generation. Noting the motivating applications, initial research was conducted along the lines of that presented in [Sethian, 1994]. Sethian's work is discussed more fully in the technical framework. For context, we present our unpublished approach here.

Given an image of an object, we segment the image using the tools in MATLAB's image processing toolbox [MATLAB, 2011]. This solid representation of the body is then used to generate the distance field over a uniform mesh with the same dimensions as the original image with MATLAB's bwdist() function. The user then specifies the desired levels for contours from the body. These contours are represented explicitly as polygons using MATLAB's contourc() function. At this point, we have explicit representations for the body-fitting curves at the specified distance levels.

We then need the edges that traverse these level curves to form a mesh. Unlike the approach taken in [Sethian, 1994], the user samples the surface by clicking in a GUI showing the original image. To generate markers, these sample coordinates are then projected onto the closest node in the polygonal representation of the surface. To create the transversals, we create correspondences between levels by connecting a point on every level to the closest point on the previous level and the next level. We identify the closest point by computing the point to point distance to all points on the relevant level. We treat convex corners as a special case by creating multiple markers to fill the resulting portion of the circle.

Figs. 1.4-1.5 show some sample results displayed as the locations of the markers overlayed on contour curves. In Fig. 1.4, note how the sampling of the field, in effect, is determined by the user's original samples. The samples on the uniformly curved surface simply propagate linearly outwards, and the markers split from the convex corners disperse. The concave corner results in markers merging to a small tolerance. We term the regions of expansion fans and regions of collapse shocks. Fig. 1.5 demonstrates the effects of choosing different contour levels.

The ad hoc approach to resolving the shocks and fans of highly curved surfaces and the lack of a robust method for running transversals between multiple objects led us to design the surface-oriented approach presented in this work. A central distinction is the use of a background mesh to sample the body-fitting curves rather than a sampling of the surface.

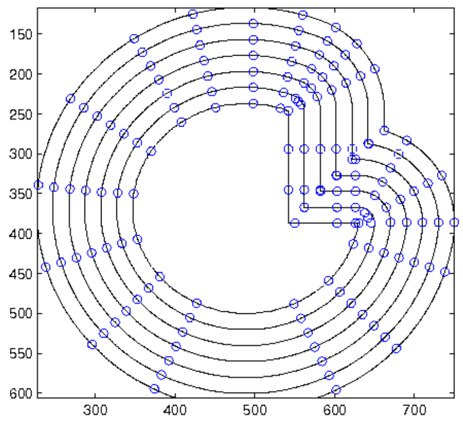
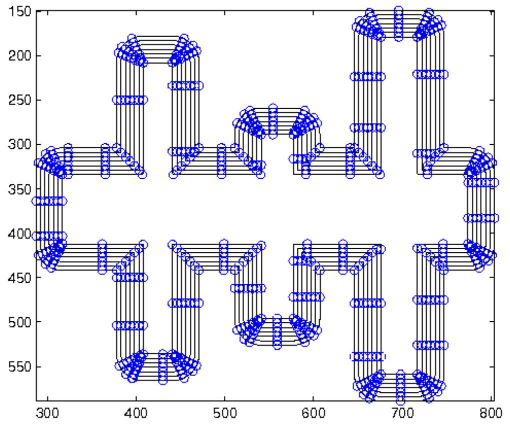


Figure 1.4 Body-fitting curves and markers about a cut circle for our preliminary surface-oriented meshing

Thus, we capture the features implicit to the shape information embedded in the field and allow the sampling to be a natural product of the features themselves.

Given this surface-oriented meshing tool, we realized that the background mesh could be adapted to a variety of features, not simply distance levels. The Voronoi-dual meshing approach is an example of this adaptation. It exploits adaptation to cells with multiple points to search for edges in a Voronoi diagram.

Given an introduction to the field and specific context on the author's experience with this work, we proceed to discuss some work of the broader technical community.



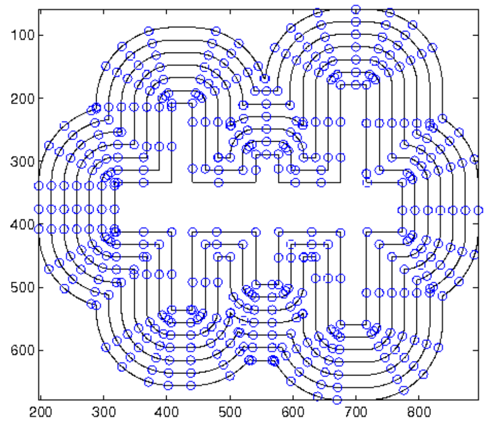


Figure 1.5 Body-fitting curves and markers about various rectangles for our preliminary surface-oriented meshing

CHAPTER 2

TECHNICAL FRAMEWORK

In this chapter, we provide the technical context for each component of our work. We begin with the intended applications and geometries for the surface-oriented meshing. Then, we discuss the distance field and its properties, including its connection to the Voronoi diagram. Using the implicit representation of the shape of the geometry in the distance field, we discuss methods to extract various features. These features include the tracking, representation, and evolution of curves in the field.

Applications

Mesh generation is a widely used tool in computational physics, geometry, and vision. Our work can be applied to a number of problems in each of these areas. We discuss one of the motivating problems for this work, biological simulation and specifically biomedical simulation.

Biological Simulation

A possible application (after significant work) of our surface-oriented meshing algorithm is to create CFD meshes for biological flows. As such, we provide a brief introduction to some relevant research. In these cases, CFD is used as an analysis tool to better understand certain flow parameters or to guide design.

A practical summary of medical imaging and guide to the use of the Insight Toolkit for manipulating real data is given in [Yoo, 2004]. Most medical images come in the form of regularly gridded images, originating from some scanning method. ”Imaging modalities currently in use are: Digital Subtraction Angiography (DSA), Magnetic Resonance Angiography (MRA), Computed Tomography (CT), and Xenon-CT (XeCT). The overall quality (sharpness) of medical images depends not only on hardware factors (resolution, wavelength, etc.), but also on the type of contrast agent used, the skill of the radiologist and patient-specific factors” [Löhner et al., 2003]. Considerable work has been done on automatically extracting the geometries of interest from ”noisy” images.

It may also take considerable work to obtain more than qualitative results. As [Prakash and Ethier, 2001] demonstrated through the use of adaptively refined meshes, extremely fine local mesh densities are required to resolve the gradient of the velocity field to within 10%. This suggests that virtually all CFD studies based on meshes created using image-based techniques are under-resolved in terms of element density [Steinman, 2002].

Voxelisation

In the simplest form, a voxel in the image that corresponds to the anatomy of interest is converted to a cubic element in the finite element mesh. Smoothing of the surface can be performed to avoid artifacts in the flow field resulting from pixelated boundaries. The work in [Keyak et al., 1993] takes these approaches to generate a mesh directly from a data set of stacked images.

Models

When knowledge of the geometry is known before meshing, we can create a map between a specific geometry and a model. If the model is analytically defined, we can then apply a full realm of mesh generation techniques. In [Wang, 2001], analytic surfaces are lofted through

Figure 2.1 Analytic model of a vessel made by lofting surfaces through contours [Wang, 2001]

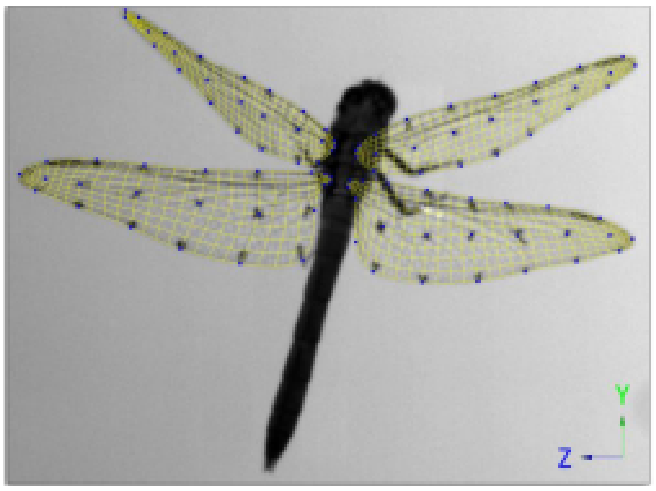


Figure 2.2 Markers used to reconstruct dragonfly motion [Dong et al., 2010]

profiles representing cross-sections of blood channels as shown in Fig. 2.1. The surface mesh is generated through the finite octree technique of [Shephard and Georges, 1991].

Another approach is given in [Pivkin et al., 2005]. Markers on a bat's body and wings are used to define the motion of a polygonal representation for simulation. Similar work is given in [Dong et al., 2010] on simulating the flight of a dragonfly. Fig. 2.2 shows the integration of the markers to reconstruct the surfaces of the wings.

Input Data

For our surface-oriented meshing algorithm, we require a description of the surface with a collection of points. As discussed, a practical source of such inputs could be medical imaging including digital images, MRI and CT scans, and ultrasounds [Steinman, 2002]. The precision of an input can range from a point cloud to an analytical surface.

For simple generality, we designed our algorithms to generate the distance field based on surface points. Each type of input data has a corresponding literature on dealing with noise. For examples, point clouds may have errors in the coordinates or gaps in the surface [Remondino and El-Hakim, 2006] and CAD models may have overlaps or holes [Barequet and Kumar, 1997]. As most of our work has been done on images, we focus our discussion to this field.

Digital Image Processing

If the input is a digital image, we make use of the tools of the image segmentation community to extract the surface. In our framework, segmentation converts the given image to binary with black objects and a white background. The literature is extensive and numerous approaches exist including thresholding, clustering, and level set based approaches. The particular goals of the segmentation and character of the data (e.g. noise) are driving factors. Reducing the surface to a set of points loses the connectivity information we can expect to obtain from the segmentation. For certain future cases, it may be worthwhile to maintain this information as, say, boundary edges.

For concreteness, we introduce morphology for preparing images and histograms and active contours for segmentation. This discussion is taken from a previous research project.

Morphology

Mathematical morphology is rooted in understanding images as being composed of objects in space. The purpose of the morphological framework is to describe the spatial organization of these objects within the image. Morphology's goal is to produce a simpler but representative derivative of the original image by managing the loss of unnecessary information through successive transformations [Serra, 1986]. The key to using mathematical morphology effectively is a solid understanding of the fundamental operations used to transform images in order to reveal their essential shape characteristics and eliminate irrelevancies [Haralick et al., 1987]. A summary of several basic operations using the language of set theory to represent the geometric manipulations follows.

There is a hierarchy of complexity to morphological operations. The foundation to this hierarchy is the structuring element. In general, the structuring element prescribes the neighborhood in the image about which we examine each pixel. For the most effective operations, it should be in the shape of the objects sought. The size is critical because only objects smaller than the structuring element will be affected by transformations. Next in the hierarchy are the two fundamental operations: erosion and dilation. If we express the reflection of a set B about some origin (that of the structuring element for us) by $B^r = \{p : p = -b \ \forall b \in B\}$ and translation by $B_x = \{p : p = b + x \ \forall b \in B\}$, then we can define the dilation of a set A by B as:

$$A \oplus B = \{x : B_x^r \cap A \neq \emptyset\}. \tag{2.1}$$

Similarly, the erosion of A by B is defined as:

$$A \ominus B = \{x : B_x \subseteq A\}. \tag{2.2}$$

We can think of dilation as expanding an image by increasing the valleys and enlarging the maxima and erosion as a shrinking operation that reduces the peaks and enlarges the minima [Maragos and Schafer, 1990]. While erosion and dilation are not inverses, they are duals in that the erosion of the foreground of an image is equivalent to the dilation of the background.

The power of morphology lies in the successive use of the erosion and dilation operations. As examples, we will describe the opening and closing operations in detail. The opening of A by B is defined as:

$$A \circ B = (A \ominus B) \oplus B \tag{2.3}$$

The erosion then dilation operation tends to "blob" an image by rounding objects from the inside, i.e. opening smooths by cutting down peaks. Analogously, the closing of A by B is:

$$A \bullet B = (A \oplus B) \ominus B \tag{2.4}$$

Dilation then erosion tends to sharpen by smoothing from the outside, i.e. closing smooths by filling in valleys. Fig. 2.3 summarizes these operations.

Opening and closing both change the features of the image. To attenuate this impact, we can reconstruct the image from a subset by iteratively dilating the subset and intersecting the dilation with the original image until idempotence [Vincent, 1993]. Concretely, if we let $J \subset I$, then the reconstruction $\rho_I(J)$ of mask I from marker J is the union of the connected components of I which contain at least a pixel of J. Functionally, reconstruction is the limit of the following operation with a small structuring element B:

$$\rho_I(J) = ((J \oplus B) \cap I) \oplus B) \cap I ... \tag{2.5}$$

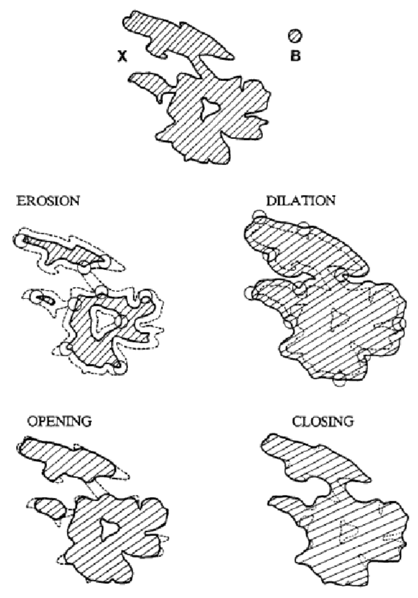


Figure 2.3 An example of several morphological operations [Maragos and Schafer, 1990]

These tools can be combined in a multitude of ways. To find a boundary of an object within the image, we can compare the original image and an eroded version:

$$\partial A = A - (A \ominus B) \tag{2.6}$$

To eliminate salt and pepper noise, we can open A by B and close the result by B. To operate on grayscale images, we simply need to alter the constructions of erosion and dilation so that they operate as local minimum and maximum operators within the neighborhood defined by the structuring element.

These operations are tools to transform an image by emphasizing the desired features. To actually segment an image, the traditional tool in morphology is the watershed transform. As described in [Haris et al., 1998], the motivation for the watershed transformation lies in conceiving of the eventual gradient of the original image as a topographical map with regions of high change corresponding to low points on the topographical surface. The local minima in this topographical surface are then pierced and the entire surface is submerged. As the surface fills from the basins surrounding the minima and the water climbs up the surface, water from different basins will begin to meet. These meeting points are the watershed boundaries dividing the catchment basins.

The high sensitivity to noise of the watersheds algorithm yields a very large number of catchment basins, leading to over-segmentation. Prior to the application of the watershed transform, the intensity image can be modified so that its regional minima are identical to a predetermined set of markers. This method achieves the suppression of the minima not related to the markers by applying geodesic reconstruction techniques and can be implemented efficiently using queues of pixels.

Because of the problem with over-segmentation, we have found morphology to be more useful to clean the image rather than use it as a segmentation method. This type of use is well documented. Morphological filtering allows for greater flexibility than median-type filters while better preserving edges [Rodríguez et al., 2002]. In [Nallaperumal et al., 2006], morphology was used to segment an image corrupted with noise. The segmentation algorithm involves three passes. In the first pass, the image is preprocessed by using an Iterative Adaptive Switching median filter which reduces the impact of impulses that cause over segmentation. In the second pass, the multiple scales of bright and dark features of different objects are extracted by the respective opening and closing of the preprocessed image. The resultant image is binarized using an optimum threshold, obtained by the fuzzy Gaussian measure. The process is repeated for multiple scales of the structuring element until all the features are extracted. In the last pass, valid segments of the bright top-hat (the difference between an image and its opening) and dark bottom-hat (the difference between an image and its closing) images are detected and the contours of these images are combined to give the final segmented image. In [Lee and Wong, 1996], morphology was used to segment images highly corrupted by noise. They separate the foreground speck noise, the object surface and the noisy background in the image by means of the grayscale skeleton transformation (morphological thinning) and the concept of maximal inscribed blocks. By removing the varying background and speck noise, the image is enhanced. To successfully recognize and separate the unwanted components, a size characterization algorithm is formulated based on the grayscale morphological opening. Finally, a global thresholding can be applied to the enhanced image to obtain the object from the background.

Histograms

In general, a histogram measures the frequency of occurrence of a certain feature. The histogram itself is a graphical display of those frequencies as a bar chart. The histogram of a grayscale image with a bar (or bin) every unit will display the number of pixels with each integer intensity value from 0 to 255. If an image contains intra-uniform and inter-distinct regions, the histogram will contain distinct and separable humps. To isolate any particular region, we threshold away any humps that do not correspond to that region. Fig. 2.4 shows an example of separating the humps to isolate the ship and sea from the sky. The ship and sea can then be separated through the horizon.

The exploitable characteristics of the histogram are the features being counted (i.e. the color space under consideration) and the thresholding method used to distinguish between humps or objects. Both are well documented.

A summary of the different color spaces can be found in [Chang et al., 2001]. For use of HSV space instead of RGB for color image segmentation, see [Sural et al., 2002]. We can also combine histograms in different spaces, fusing the pieces together for a more comprehensive result [Kurugollu et al., 2001]. An extensive summary of thresholding is found in [Sezgin and Sankur, 2004]. Typically the valleys are used as thresholding locations since peaks correspond to objects. Iterative thresholding can be applied to find the correct number of

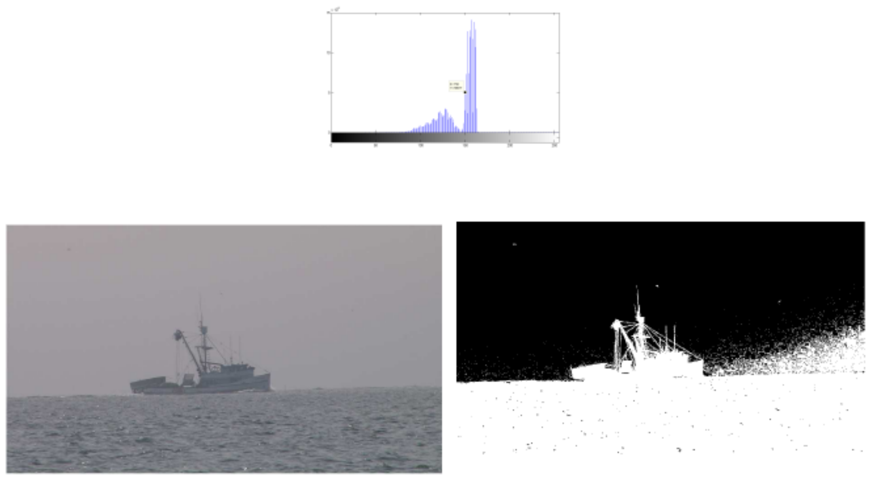


Figure 2.4 Histogram segmentation based on distinct objects

objects as in [Chang and Wang, 1997], but iterating the threshold based on the quality of the segmented result is a much more sophisticated and involved task.

Active Contours

Active contours algorithms use level sets to subject an initial contour to constraints from an image, thus evolving the contour to boundaries of objects in the image. First proposed in [Kass et al., 1988], classical active contours algorithms evolve this initial contour by minimizing a functional with terms for curve smoothness and intensity changes of adjacent pixels (numerical gradients of intensity functions). By using level sets to formulate the minimization, they allow for sudden topological changes like corners or cusps. These classical methods generally use either parametrically defined active contours or contours based on geometric minimal distance curves, as in [Caselles et al., 1997]. However, their reliance on gradients means that the classical algorithms perform poorly with blurred images which do not have sharply defined intensity changes at object boundaries. Various modifications have been proposed, including use of an external force based on gradients in the image [Xu and Prince, 1998], but this method still relies on gradient changes to segment an image.

To improve upon classical active contours models, Chan and Vese propose an active contours algorithm that does not rely on edge detection and intensity changes [Chan and Vese, 2001]. They assume that the image should be segmented into two regions (background and foreground), both of approximately uniform intensity. If C is the initial contour on an image, they show that the functional

$$F = \int_{\text{inside}(C)} (u - c_1)^2 \, dx \, dy + \int_{\text{outside}(C)} (u - c_2)^2 \, dx \, dy \tag{2.7}$$

is minimized with respect to C when C is the boundary of the object in the image. In this equation, u is the intensity of the image, c_1 the average intensity inside C, and c_2 the average intensity outside C.

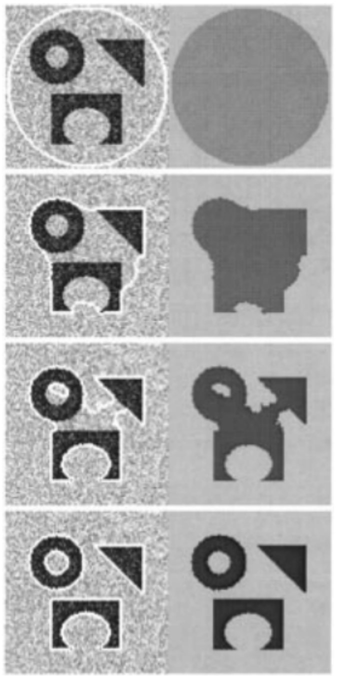


Figure 2.5 Active contour segmentation based on minimizing energy [Chan and Vese, 2001]

To control for noise, Chan and Vese add a length term $\mu \times \text{length}(C)$, arriving at the following functional

$$F = \mu \times \text{length}(C) + \lambda_1 \int_{\text{inside}(C)} (u - c_1)^2 \, dx \, dy + \lambda_2 \int_{\text{outside}(C)} (u - c_2)^2 \, dx \, dy \tag{2.8}$$

where λ_1 and λ_2 are fixed parameters. The length term smooths the contour and avoids intricate boundaries that consider noise. Fig. 2.5 shows results for various choices of parameters.

The functional F represents a special case of the *Mumford-Shah energy functional*, proposed in 1989 as a method of image segmentation [Mumford and Shah, 1989]. The length term encourages a smooth contour and the other terms minimize regional variation. Chan and Vese propose minimizing F with a numerical approximation of level sets, a method developed in [Osher and Sethian, 1988]. Since then, several other active contours methods have been proposed using versions of these equations. For example, MATLAB and C source codes have been made available for the work in [Bresson et al., 2007] by Bresson.

A particularly useful result of obtaining the object from segmentation is that the geometry may be made quickly using any standard image manipulation software (e.g. Paint). We can make use of tools based on all three approaches depending on the character of the input image.

Hierarchical

A tree is a fundamental data structure with a correspondingly large number of applications. Hierarchical meshing approaches are rooted in the tree structure and use some form of regular splitting of cells to obtain leaves that cover the domain. We can divide the field according to the type of cell and dimension (e.g. square, hexahedron, tetrahedron), subdivision conditions, and treatment of boundaries. Of particular importance to our work is the adaptation capabilities of the branching structure coupled with an organization of space.

For clarity, we discuss the application of isosurface tracking. An early work is that of [Bloomenthal, 1988], where distance values to the surface are stored in the corners of cells in an octree. In order to capture an isosurface defined implicitly by a distance level to the original surface, two methods of adapting the local resolution are presented. The first is what we term root-down and involves recursively subdividing cubes that intersect the surface. Depending on the subdivision criteria, small surface detail may not be captured by a large cell and thus not trigger refinement. The other is what we term surface-out. Here, a small (relative to surface detail) initial seed cell is identified on the surface. New cells are obtained by propagating along cell edges that cross the surface. Obtaining the initial seed may require significant work. Figs. 2.6 and 2.7 show both of these approaches.

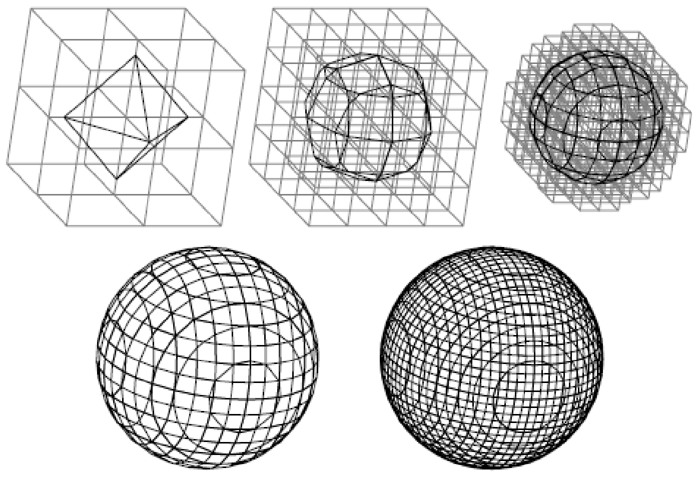


Figure 2.6 Root-down octree subdivision [Bloomenthal, 1988]

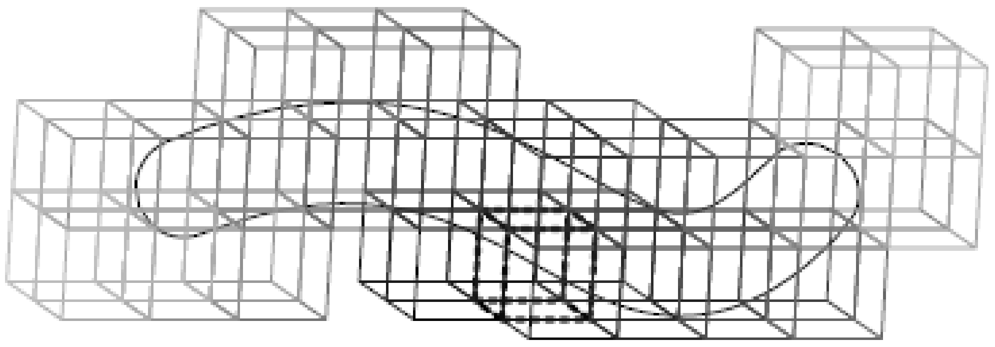


Figure 2.7 Surface-out front tracking [Bloomenthal, 1988]

Anisotropic refinement can reduce the number of cells that are required to accurately resolve complex geometries. In [Domel and Karman, 2000], an omni-tree allowed cells to be split in any combination of the three Cartesian directions with a limited aspect ratio. A control on gradation of the sizes of leaves over a region is important for some applications and can be enforced through refinement based on local neighbor properties.

Boundaries

The axis-aligned framework is an inherent barrier to resolving boundaries. While we can flag cells as inside or outside the domain defined by the surface, taking the hierarchical cells results in an integer approximation to the surface. For cases when a pixelated resolution is not sufficient, there are a number of approaches to better approximating the surface with the mesh:

- Cut cell
- Smooth surface nodes
- Buffer layers

There are alternate approaches that rely on the PDEs solver to generate the effects of boundaries by distributing forces over immersed boundaries, see [Yang et al., 2007] for example. We will only discuss methods that represent the boundary explicitly in the mesh.

Cut cell

The cut cell method is an extension of creating a pixelated representation of the boundary. If we frame this as a problem of meshing an isosurface, all of the techniques discussed in the isosurface meshing section apply. In short, the definition of the surface is used to create additional nodes in the cell through the determination of intersections. Those points are then connected inside and between cells. This introduces a polygonal surface on the boundary.

The method in [Coirier et al., 1996] uses the polygon clipping algorithm of [Sutherland and Hodgman, 1974]. This cutting can produce sliver cells that need to be optimized before attempting a numerical solution.

Smooth surface nodes

We can also better approximate the surface without changing the connectivity by adjusting the coordinates of the nodes on the pixelated boundary. The approach taken in [Dawes et al., 2007] is to project the octree front onto the surface through an optimizer based on following the local gradient of the distance field. A schematic is shown in Fig. 2.8.

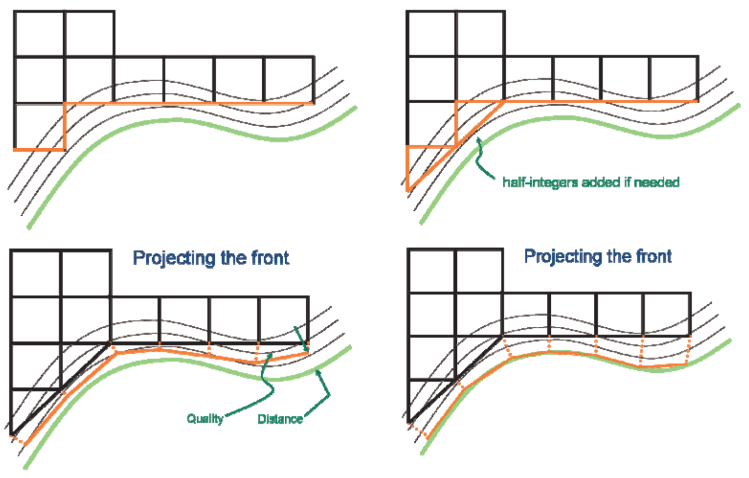


Figure 2.8 An example of projecting an integer boundary to the surface [Dawes et al., 2007]

Buffer layers

If we can accept hybrid meshes, better quality and flexibility may be obtained by utilizing a method designed for creating more resolved meshes near surfaces. A representative approach is given in [Druyor Jr, 2012] and shown in Fig. 2.9. Here, cells are cut to allow sufficient space between the surface and the integer front. A simple advancing layers algorithm based on normals at the nodes is then used to extrude the surface layers. The extrusion stops locally if a front merges sharply or separate fronts collide. The outer layer of the extrusion is then stitched to the integer front with a Delaunay triangulation.

We utilize a quadtree hierarchy as a core structure in our algorithms. The surface-oriented meshing to be presented may also be applied as a method of inserting buffer layers in a hierarchical mesh.

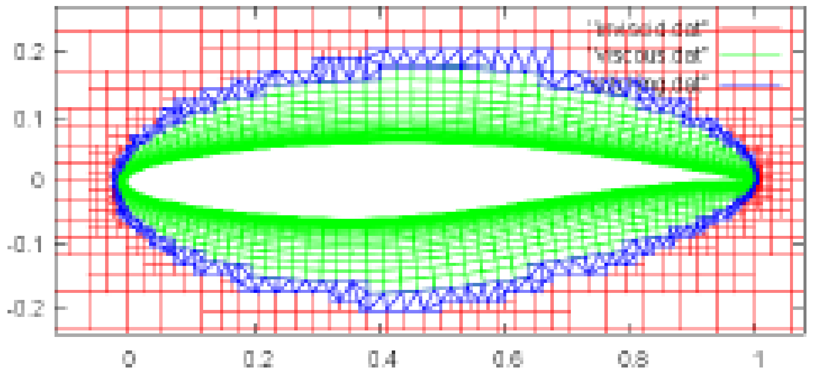


Figure 2.9 An example of buffer layers in a hierarchical mesh [Druyor Jr, 2012]

Distance field

The distance field is the most fundamental aspect of this research. The following concepts are key to both of the meshing applications presented in Chapter **??**. Given a set S in a metric space M, the value of the unsigned distance field at a point p_0 in M is

$$d(p_0) = \inf_{s \in S} ||p_0 - s||. \tag{2.9}$$

For this work, the metric is the squared Euclidean distance ($d(p_0, p_1) = \Delta x^2 + \Delta y^2$), and d represents the function in the level set definition. Fig. 2.10 shows contours in the field generated by four points. Note how the contours emanate as circular curves until the offset curves interact.

There are a number of techniques and applications of the distance field to computer graphics, physics, and geometry [Jones et al., 2006]. Research is divided between applications of shape information (skeletonisation, rendering, etc.) and fundamental algorithms for calculating the field [Satherley and Jones, 2001]. Among the representations are signed distance fields distinguishing between the inside and outside of objects, vector distance fields

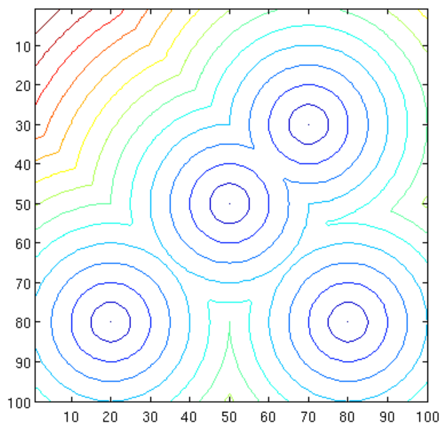


Figure 2.10 Contours of the distance field generated from four points

to the closest point in S, and correspondence distance fields that store the points in S that generate the local distance field.

Motivation

The use of an implicit representation to handle a wide variety of geometry definitions is not a novel approach. Fig. 2.11 shows the flexibility that this approach provides. In [Freytag et al., 2006], the flexibility is taken even further to use the implicit representation directly for simulating physics.

We can also extract information about the shape to guide the construction of an explicit representation such as a mesh. One particular relationship of interest is the connection between curve offsets and the local normal. A differential geometric definition of a normal to a surface is defined through derivatives and tangent planes [McCleary, 1994]. We utilize a dual notion of a normal based on offset curves.

Given a curve offset from a surface at length l, we define the normal direction(s) at each surface point to be the vectors to the closest point(s) on the offset curve. The local normal is thus obtained by creating correspondences among neighboring contour levels. For a given offset length, the normal then provides the direction of travel that yields the farthest distance from the surface.

It is through this direction of steepest descent that we can link the dual concepts. If l is infinitesimal, the offset curve and differential normals are both gradients of the distance field (except where the differential normal does not exist, for example at corners). For the offset normals, the choice of l provides a parameter for specifying the local region of influence. Our idea of offset curves is to imitate an extrusion of the surface near the body so the precise distance needs to be a function of the features of interest in a given geometry.

An intuitive understanding of offset curves is helpful for understanding the surface-oriented meshing algorithm. Fig. 2.12 is of a simple case of a right angle. If we offset the curve outward (to the top left), we see that the flat parts of the original surface are simply copied over. However, the convex corner creates the portion of the circle. If we offset

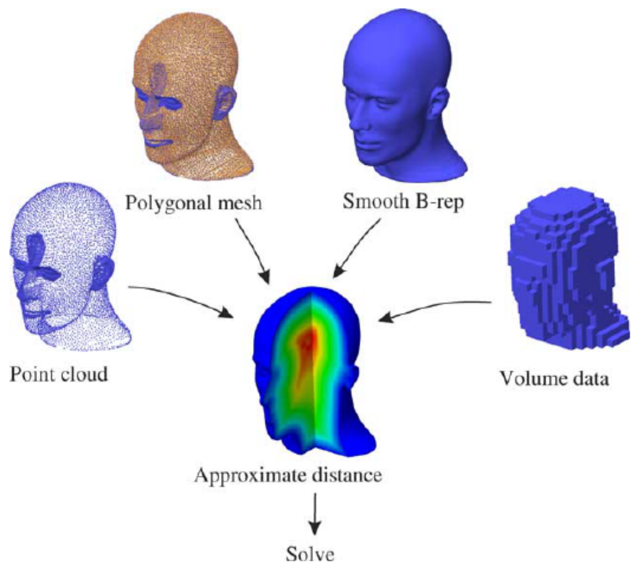


Figure 2.11 Using an implicit representation to manage various types of geometries [Freytag et al., 2006]

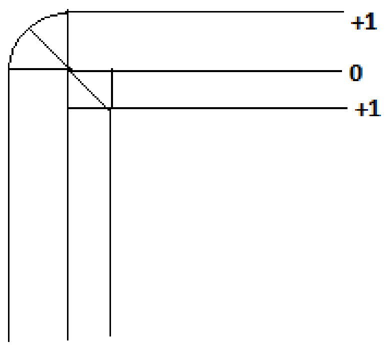


Figure 2.12 Offset curves for a right angle

inward, we see the same behavior with the flat regions, but the concave corner collapses to a single point. There is a generation and destruction of material along offset curves based on the curvature of the surface.

More generally, offset curves can merge and interact. It is because of these interactions that offsetting is not invertible. Each of these merging regions is a subset of the medial axis [Pizer et al., 1994]. The medial axis is formally defined as the locus of the centers of the maximal spheres inscribed within an object. Fig. 2.13 shows these maximal spheres and interactions inside of a leaf shape. One of the most salient properties of the distance field representation is the natural embedding of these merging regions.

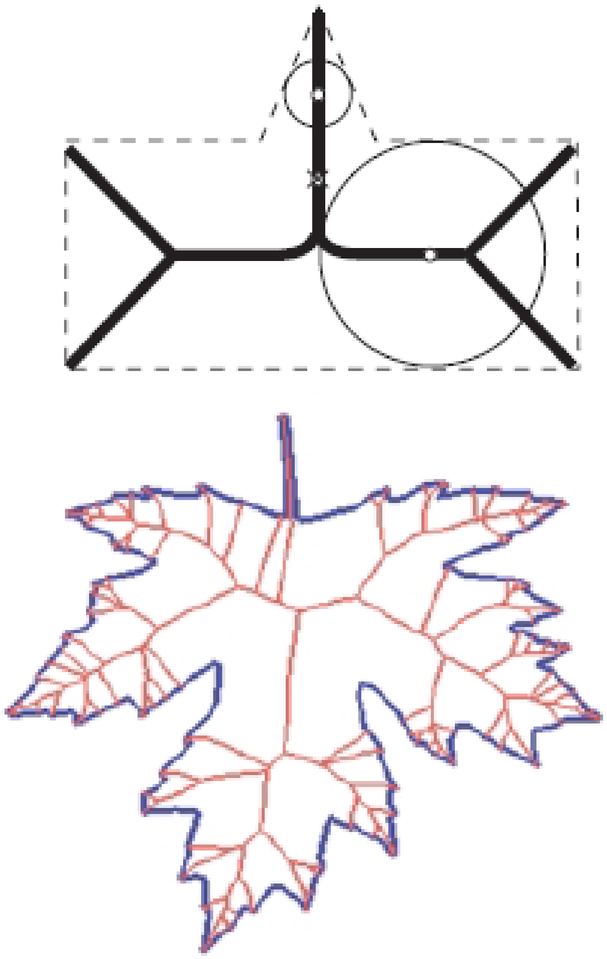


Figure 2.13 Medial axis examples [Katz and Pizer, 2003]

Computation through Spatial Partitioning

Considering the definition in Eq. 2.9, we see that the distance value at a point is a function of every point in S. It is inefficient and likely prohibitive for an algorithm to perform all of these operations for every point in the field. Attempts to reduce this cost by localizing computations through approximations are collectively termed transform methods. A full discussion of a number of transform methods is given in [Cuisenaire, 1999].

One type of approach is to limit the number of points that can influence the considered location in the field. This can be inexact or exact. We focus on exact methods that utilize a method of spatial partitioning, either marching or restricting from known values.

We can exploit the spatial organization and adaptivity of a hierarchical data structure to reduce the number of distance computations. A crucial property is that the distance field over a a subset of a cell is covered by the distance field over the original cell. Thus, we can avoid sampling in regions that are not of interest. Moreover, we can approximate the distance values of a subset from the original cell. For octrees, a common approach is to use trilinear interpolation to reconstruct the field to subcell accuracy.

As emphasized in [Frisken et al., 2000] in the development of adaptively sampled distance fields, large meshes are required for regularly sampled distance fields when any fine detail is present, even if the detail is only in a small, localized portion of the volume. The use of of a tree structure allows for arbitrarily high sampling frequencies and storage for efficient access of shape information. In Fig. 2.14, the top right R was generated by subdiving to a predetermined resolution and has 23,573 cells. The bottom right R has 1,713 cells and was generated by subdividing a boundary cell when its distance field was not well-approximated by a bilinear interpolation of its corner values. To determine whether an adaptive appraoch is more efficient than using a uniform mesh, we must also account for the structure and operations necessary to achieve the adaptation.

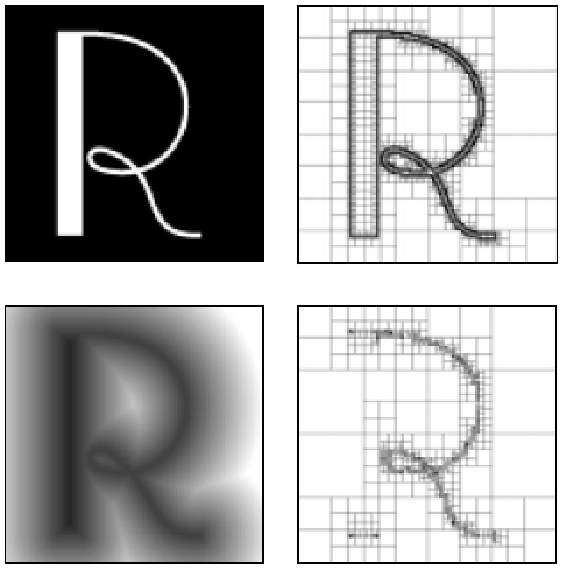


Figure 2.14 An example of adaptive hierarchical sampling [Frisken et al., 2000]

Voronoi Meshing

For a finite collection of points P and sites S, define the Voronoi map of $p_0 \in P$ to S as $V(p_0) = s$ such that $||s - p_0||$ is a minimum. This map associates every point in a set to the closest point in another set. Intuitively, this partitions a set into its nearest neighbors. A point equidistant to two sites will be mapped to both of them. For the rest of our discussion in this work, the norm is the Euclidean distance.

Fig. 2.15 shows a simple case, where the points generate the distance field, and the lines represent regions where points are equidistant. The polygons are regions of the plane that are mapped to the enclosed site.

The Voronoi diagram is a graphical representation of the mapping, and its history can be traced to the middle of the nineteenth century with a number of applications. Emphasis has been on its use in modeling natural phenomena in fields such as crystallography

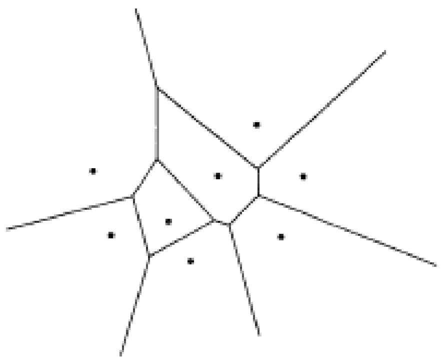


Figure 2.15 A Voronoi diagram for eight sites [Aurenhammer, 1991]

and meteorology, character as a mathematical object, and computational construction and representation [Aurenhammer, 1991].

In R^2, a dual construct is the Delaunay triangulation, which consists of a line segment between two sites in the plane if their Voronoi regions share an edge. Co-circular points are a special case as any number of points can share a vertex at the center of the circle. An important feature of a Delaunay triangulation is that it maximizes the minimum angle over all triangulations of a given set of sites [Lawson, 1986]. Delaunay refinement of a given triangulation will eliminate elements of "poor quality" [Shewchuk, 2002a]. For finite element simulations, poorly shaped elements can affect the numerical accuracy and algorithmic efficiency [Freitag and Ollivier-Gooch, 1997]. For example, as element angles become too large, the discretization error increases [Babuska and Aziz, 1976]. As angles become too small, the condition number of the element matrix increases [Fried, 1972].

There are a number of algorithms for forming Delaunay triangulations including Bowyer-Watson's, Lawson's, Fortune's, and divide and conquer. Our focus is on the approaches based on the distance field. These rely on detecting merging regions in the distance field.

In [Maignan and Gruau, 2008], boundaries of the medial axis are detected as local maxima. In [Xia and Tucker, 2009], the Laplacian or determinant of the Hessian of the distance field are used. [Vleugels and Overmars, 1998] approximates Voronoi maps in any dimension to a specified precision by locally refining hypercubes in regions of interest. More efficient identification of those regions is accomplished by exploiting the connectedness of the medial axis.

There are cases where we wish to guarantee that a particular set of edges appears in the Delaunay triangulation. Approaches that enforce certain edges are termed constrained Delaunay approaches. Note that satisfying this constraint will likely sacrifice the Delaunay properties locally. The main categories are constructive and recovery methods. Constructive methods enforce the existence of the constrained edges through the entirety of the triangulation process. Recovery methods treat the constraint by post-processing the resulting triangulation. A common approach is to flip edges of the triangulation until the constrained edge appears in the mesh.

Normal Evolution of Fronts

For certain types of physics, an intentional orientation of the mesh may increase the accuracy of a simulation. For example, to resolve high gradients in flow properties near the surface in high Reynolds number flow over an object, small spacing normal to the surface is needed in order to have an acceptable number of elements [Ito and Nakahashi, 2002].

We can classify the typical approaches to obtaining a normal orientation into two broad approaches:

- Explicit/Lagrangian/particle/advancing layers approach
- Implicit/Eulerian/field/level set approach

Explicit approaches represent the topology of the front directly by storing the nodes and their connectivities. Collisions between fronts must be resolved on the level of these nodes and connectivities. Implicit approaches embed the front and its topological information within a field. Collisions are naturally handled within the field.

Explicit

A Lagrangian approach to evolving a front is to discretize the front into markers and evolve the individual markers according to the proper equations of motion. This method is entirely natural and particularly useful if the curve does not interact with itself since the markers can move independently. However, it is difficult to resolve changes in topology because of this independence. Surgery or reconnection methods are required when the markers merge or diverge. An effective implementation of this approach is given in [Leung and Zhao, 2009], including interface movement in the normal direction.

Applied to mesh generation, a standard and industrialized approach is the advancing layers algorithm. Here, a mesh of the object's surface is marched normal to itself based on a differential geometric conception of a local normal as shown in Fig. 2.16.

Consider approximating a sphere as a soccer ball and extruding the vertices between panels outwards based on a local normal direction. Given a discretization of the surface, the necessary elements to this approach are a way to define the surface normals and a heuristic to resolve collisions and separations of neighboring faces. The idea relies on an accurate original representation of the surface because the algorithm is primarily local.

Another approach is to offset the faces instead of the nodes. As described in [Jiao, 2007], planar faces are moved under advective or wavefrontal motion. The nodes are reconstructed through an eigenvalue analysis of the incident planes and redistributed in the interests of mesh quality.

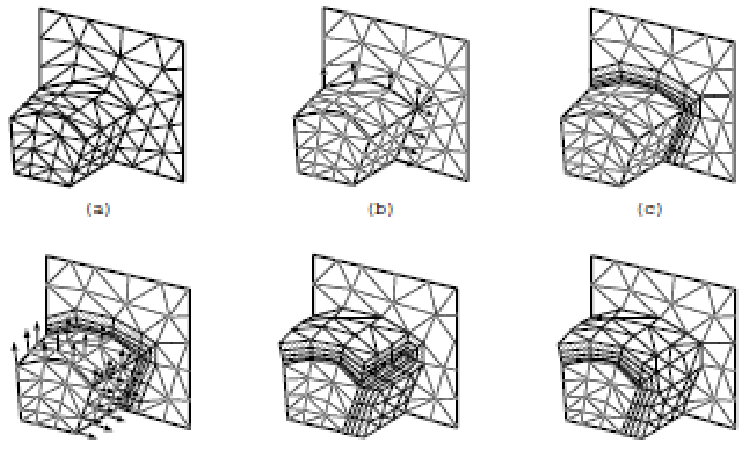


Figure 2.16 An example of advancing layers meshing [Garimella and Shephard, 1998]

Implicit

For our purposes, it is sufficient to define the level set of a function $f : R^n \to R$ as the collection of points $p \in R^n$ such that $f(p) = c$ for some scalar $c \in R$. Level sets arise as a natural way to examine the behavior of a higher dimensional function. The specific character of a level set is intimately related to the function f. A familiar example is a topographical map of terrain altitudes. Fig. 2.17 shows a collection of level sets used in modelling crack propagation in [Colombo and Massin, 2011].

An Eulerian approach to front evolution involves using scalar functions with the interface embedded within a field. The level set approach is to evolve an n-dimensional curve by traversing the level sets of an n+1-dimensional surface. Topological change is natural and curve self-independence is difficult to achieve. See [Osher and Sethian, 1988] for the foundation of the method, where the offset curves are weak solutions to the eikonal equation [Wang et al., 2005]. Considerable research has been done on the choice of stencil and numerical schemes for approximating derivatives, and narrow band methods reduce the

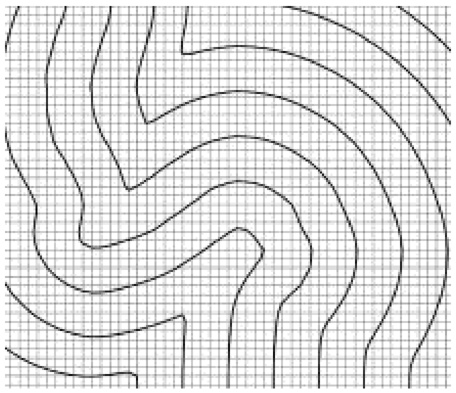


Figure 2.17 Level sets used to model crack propagation [Colombo and Massin, 2011]

computational cost. If the curve propagates "one way", a more efficient implementation is the fast marching method [Sethian, 1996].

For concreteness, consider the evolution of a shrinking square. We embed the 2D square into a 3D surface, here a four-sided pyramid like those in Giza. If we imagine the initial square as the stone blocks making up one layer, then shrinking that square is like considering higher levels of the pyramid. Eventually, we come to the tip of the pyramid as the square shrinks to a point.

In [Sethian, 1994], Sethian provides an application of level sets to grid generation. In this work, the curve is transformed into the zero-level contour of a distance field from the surface. The curve is then moved with velocity based on the local curvature and normal, both intrinsic to the level set. Using upwinding numerics where conflicting fronts of the curve merge, these body-fitting curves are generated with larger spacing in concave regions and smaller in convex. The original curve is parametrized by arc length to form markers, which are then run normal to the body-fitting curves. The paths of these markers cannot merge if curvature was incorporated into the evolution of the body-fitting curves. Spacing constraints are incorporated into layers away from the body by penalizing deviations in cell sizes tangential to the body, thus trading orthogonality far from the body for more consistent

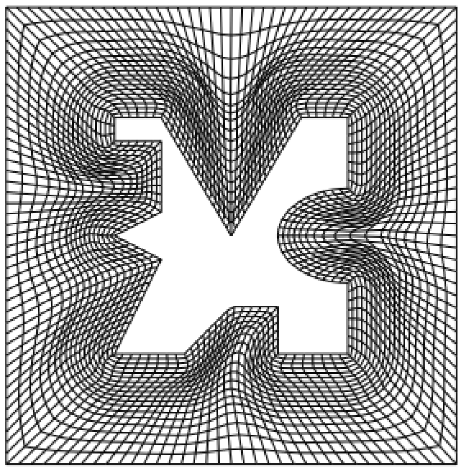


Figure 2.18 An example of Sethian's grid generation method using level sets [Sethian, 1994]

spacing in the grid. Special care is given to place markers at fan points to ensure that the grid limits the rarefaction from the diverging normal directions. The entire approach creates a body-fitted structured grid with automatically detected special points at corners as shown in Fig. 2.18.

While we categorized explicit and implicit techniques separately, these approaches can be combined to benefit from using both together. In [Enright et al., 2002], Lagrangian marker particles are used to rebuild the level set in regions which are under-resolved. The origin of the work is to simulate advecting fluid flow. For regions where the mesh is not sufficiently dense, Eulerian capturing methods can not accurately tell if characteristics merge, separate, or are parallel. The weak solution calculated with level sets can delete characteristics when they appear to be merging. Additional information is exploited from markers placed randomly along the interface. If the markers cross the interface by a certain tolerance, there is an error in the representation of the level set locally. The characteristics tracked by these crossing markers are used to regain a higher order of accuracy.

While we rely on the theory of level sets, we do not use evolution functions requiring the numerical solution of PDEs. However, in principle, our approach could also be applied to a more complex field.

Isosurface Meshing

The goal is to have a tessellated representation of a level set in a field that approximates the isosurface. Although not necessarily explicitly stated in the works that follow, the level set is typically assumed to be at least piecewise continuous.

Analytical

One approach is to first generate an approximation of the isosurface with an interpolation, with the most general form being a NURBS surface in practice. A mesh can then be generated on the analytical surface through sampling, structured mappings, or one of the other approaches below.

A practical example of the use of such an analytical surface is discussed in the [biomedical models] subsection. The analytical surface can be used to provide both a precise surface definition and a local topology for ordering.

Mapping

Structured grid generation has a long history of mapping simple domains with a known mesh to the domain of interest. As this field is tangential to our work, we will provide only a few examples of approaches.

Transfinite interpolation is an industrialized approach utilizing algebraic operations to interpolate coordinates between specified boundaries [Gordon and Hall, 1973]. The boundary points are required to map one-to-one, and smoothing functions can be used to attempt to achieve a desired clustering.

Conformal mapping is a classical approach with a strong underlying mathematical theory in complex analysis. By conformal, we mean that the angles between edges are preserved under the transformation. The Riemann mapping theorem proves a critical property that, for two topological disks, there exists a conformal map between the interiors. The meat of the problem then becomes constructing the map between the two spaces. Specific recommendations and tools can be found in [Ives, 1982].

Cell division/Templates

For a field embedded in a background mesh with some regularity, this is a common approach.

[Teran et al., 2005] generates tetrahedral meshes suitable for deformation simulations by subdividing based on local curvature and distance to the surface. The surface is then better approximated by ”compressing” the hierarchical front to the boundary with a relaxation algorithm.

Marching cubes and its descendants [Lorensen and Cline, 1987],[Chernyaev, 1995] are popular algorithms. The methods classify vertices as positive or negative, according to their comparison with a given isovalue. Then, a lookup table is used to tile the surface inside the cube. The original lookup table may lead to cracks or incorrect topology and is shown in Fig. 2.19. Interpolation of the field can be performed on the relevant edges to obtain the coordinates.

While the templates are designed to cover the possible configurations of the corner values, the actual isosurface within the cell may have a complicated shape and topology which cannot

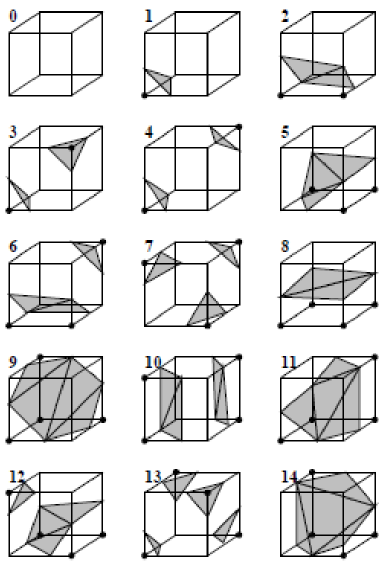


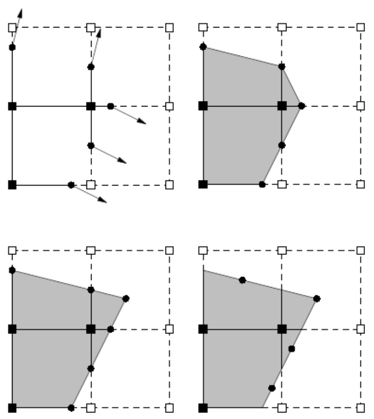
Figure 2.19 Lookup table for original Marching Cubes algorithm [Lorensen and Cline, 1987]

be reconstructed correctly using Marching Cubes. [Natarajan, 1994] use saddle information of a linear interpolation in the interior and faces to determine the proper polygonization.

Dual cube methods generate a node for each cell that crosses the isosurface. Since the vertices are not restricted to lie on an edge, the resulting polygons typically have better aspect ratios. Extended marching cubes is a hybrid method that uses normal information to identify cells with features. It uses Marching Cubes in featureless cubes, and a vertex placed to minimize a quadratic functional in cubes with features. Dual contouring [Ju et al., 2002] uses a quadratic error functional in all cubes to position vertices. For every edge that crosses the isosurface, a quad connects the four cells that contain the edge.

These approaches to sampling are summarized in Fig. 2.20.

We can also perform a Delaunay-like refinement algorithm with some concept of error to the surface. [Boissonnat and Oudot, 2005] inserts points in the centers of Delaunay balls to obtain successively better approximations.



A signed grid with edges tagged by Hermite data (upper left), its Marching Cubes contour (upper right), its Extended Marching Cubes contour (lower left), and its dual contour (lower right).

Figure 2.20 An example of sampling for MC, EMC, and dual contouring [Ju et al., 2002]

Advancing Front

Given an initial mesh, the goal is to cover the domain by continuing to add cells to the front. A number of approaches exist for determining where to add elements. For examples, [Frey and Borouchaki, 2003] uses notions of curvature, and [Schreiner et al., 2006] focuses on generating a guidance field for edge and triangle sizes. [Schöberl, 1997] provides an abstraction for the rules used to generate elements.

Paving and plastering are methods designed to create all square or hexahedral meshes. In [Staten et al., 2005], rows are advanced from the boundaries and the voids between the expanding fronts must be resolved with squares. Fig. 2.21 shows an example from this work.

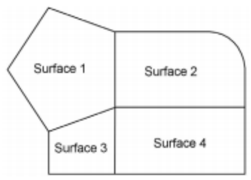


Fig. 13 Multiple adjacent surfaces requiring a conformal mesh

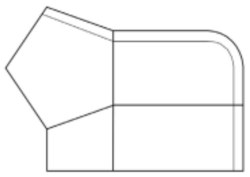


Fig. 14 An unconstrained row has been advanced extending through multiple surfaces

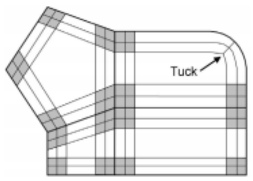


Fig. 15 Additional rows are advanced including a tuck

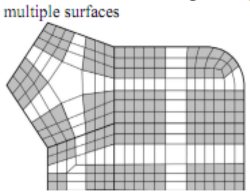


Fig. 16 Only small voids and connecting tubes remain

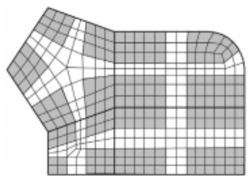


Fig. 17 Unmeshed voids and connecting tubes are meshed

Figure 2.21 Unconstrained paving of multiple regions [Staten et al., 2005]

CHAPTER 3
TOOLS

In this chapter, we describe our implementation of the CDF and some of its basic applications. These tools are our implementations of the concepts presented in Chapter 2. The reader may find it worthwhile to refer back to this chapter when reading Chapter 4.

Except where noted, all software was implemented in C++ and makes extensive use of classes and dynamic memory allocation.

CDF Computation

We use a quadtree to represent the spatial domain and compute bounds on the distance over the quadtree squares to the surface points. The correspondence information of the surface points that generate those distance bounds is also stored.

Spatial Quadtree

We make use of the simple, isotropic, and hierarchical structure of the quadtree to organize space. The quadtree also has a clear extension to an octree in 3D. Examples of the quadtree structure are given in Figs. 3.1 and 3.2 for tree and spatial representations, respectively. Each parent is divided into four cells by halving along each axis.

Each cell contains pointers to its parent and child cells. An integer tag is used mainly as an index into the OwnPt array. A cell-centered approach is taken, which allows us to avoid creating unique nodes that are shared between cells.

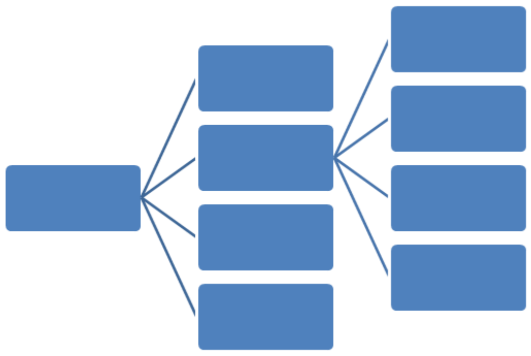


Figure 3.1 Tree representation of three levels in a quadtree

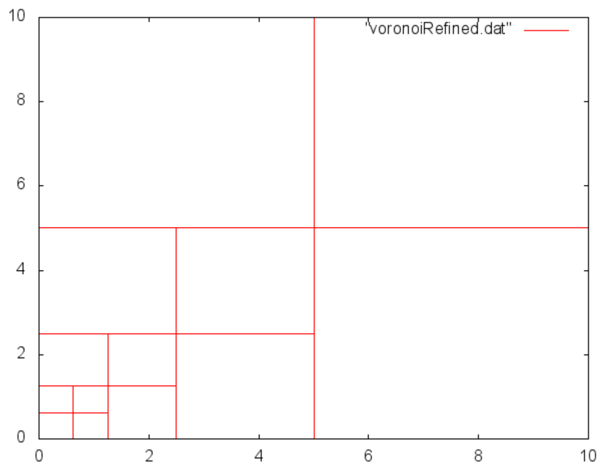


Figure 3.2 Voxel representation of five levels in a quadtree

Data Structure

We term each cell in the tree a voxel. Each voxel object stores the following information:

Type	Description
double[2]	x, y coordinate of lower left corner
double	Side length of square
Voxel*	Parent voxel
(Voxel*)[4]	Child voxels ordered lower left to top left
int	Used for utility and indexing

The root of the tree can be identified as the Voxel with no parent. The leaves are the Voxels with no children. The relevant pointer is set to NULL in each case. The memory for the children is only allocated when the Voxel is subdivided.

Note that we can allocate memory for each voxel's four children together in a contiguous chunk from the heap, so we actually only need one pointer that can be incremented to access all four children. Also, after development of the final versions of our algorithms, the link to the parent voxel is never utilized, so it can be removed from the data structure. This savings should be included in any future versions.

There is a trade-off between storing information and re-calculating. As a general rule, we store the information that does not change and is reused. Thus, for example, the coordinates of a voxel are stored, but its leaf neighbors are not as the tree continues to branch under adaptation.

Neighbor Finding

In the surface-oriented meshing algorithm, we need to find the leaves surrounding a given leaf for contouring features and performing a flood-fill operation. We obtain these neighbors

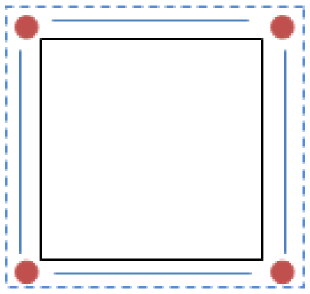


Figure 3.3 Objects for capturing corner, face, and all neighbors of a voxel

from the quadtree structure. There are several approaches to finding the neighbor to a given voxel, including point and object offsets (top-down) and tree traversal (bottom-up) [Samet, 1982].

As we have an unbalanced quadtree without gradation enforcement with leaves generally at different levels, we have found the top-down methods to be the cleanest approach. These are implemented as recursive calls down through a cell's children testing whether the point or object in Fig. 3.3 intersects the voxel.

All Neighbors An expanded box is passed down the root. The original voxel is in the list of neighbors. We use the dashed line shown in Fig. 3.3. These are the 8-connectivity neighbors.

Face Neighbors A line is created that is outside of that face and shrunken to not capture the corners. We use the solid lines shown in Fig. 3.3. These are the 4-connectivity neighbors.

Corner Neighbors A point offset is calculated to be just outside each face of the corner. We use the corner points shown in Fig. 3.3. These are the difference between the 8-connectivity and 4-connectivity neighbors.

For slightly better efficiency, the test for whether the object intersects the voxel depends on the type of object. For a point, its x-coordinate must fall in the horizontal range of the box, and y-coordinate must fall in the vertical range of the box. For two overlapping lines, an endpoint of one falls in the range of the other. To test whether an axis-aligned line crosses the box, the line must overlap with the appropriate horizontal or vertical bounds of the box. For two overlapping boxes, we test whether the two axis-aligned lines that represent one box both overlap with the other box. All of these tests are robust and significantly cheaper than testing for intersection between two arbitrary polygons.

In order to create these neighbor objects, the user inputs a distance that is smaller than the smallest voxel edge length. The default is 1e-12, which is likely orders of magnitude smaller than the smallest voxel edge.

Distance Bounds and Correspondence

We compute the exact minimum and approximate maximum distance over a quadtree square to all surface points. We also keep track of the surface points whose distance to the square falls within those bounds.

Computation over a Box

Consider the distance of a single point to the space covered by a box. By inspection, we note the following:

- If the point falls within the box, the minimum distance is zero
- If the point falls outside of the box, the closest point in the box is the point's projection onto the closest face
- A corner of the box is the farthest distance from the point

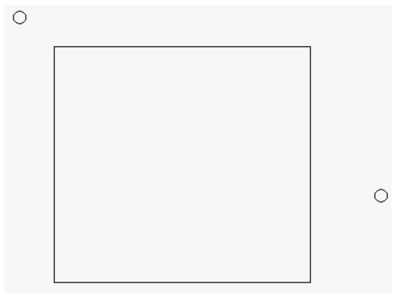


Figure 3.4 Example used to demonstrate surface points generating distance bounds over a box

- The distance bounds and point correspondences of a subset of the box are covered by any enclosing box

The first three points are sufficient to give exact bounds to the distance field over a box from a single point.

Now consider the distance field over the box to multiple points. The minimum of the minimum distances to each point is exact. For maximum distance, the fronts expanding from multiple points can interact by merging. So, the minimum of the distances of the points to each one's farthest corner is a loose upper bound on the actual maximum distance over the box. A tighter upper bound can be obtained by sub-sampling the entire space of the original box and taking the maximum of the subsamples.

Fig. 3.4 is a simple case we use for demonstration. For the top left point, the closest point in the box is the top left corner. The farthest is the bottom right corner. For the right point, the closest point is its projection onto the right face. The farthest point is the top left corner. For both points together, the closest point is the top left corner. The farthest is the bottom left corner. Given distance bounds, the corresponding points are those whose distance over the box falls within those bounds. Both points are stored in this case.

Data Structure

The object that stores the distance information is termed an OwnPt. Each OwnPt stores the following information:

Type	Description
Voxel*	Geometry for the box
std::vector$< int >$	Points corresponding to the distance bounds
double[2]	Minimum and maximum distance over the box

All OwnPts are stored in a linear array. To allow for more efficient indexing, the integer field in the Voxel is set to be the index of the associated OwnPt in the array. Thus, we have doubly-linked access between the Voxels and OwnPts.

Computation over the Tree

For our purposes, we require the CDF over all leaves in the tree. There are two main approaches. Root-down exploits the branching structure of the tree, and surface-out exploits the spatial organization through local neighbor operations.

Root-down Since the root covers the domain, distance field, and surface points, we can refine it to obtain the regions of interest.

We exploit the property that the root covers the distance field of the child during subdivision. To compute the field over a child to all surface points, we compute the field over the child only to those (multiple) points corresponding to the parent.

More efficient usage of memory can be performed if we note that the four children cover the correspondence distance information of the parent. In fact, the information from the children is likely more accurate than that of the parent. Once refined, the memory used to store the distance information of the parent can be freed if not allocated as part of a contiguous chunk if memory capacity is a constraint.

Surface-out Imagine waves propagating outwards from the generating surface points, as in Fig. 2.10, within a quadtree with some degree of branching. Obtaining the surface voxels is a matter of searching the tree for leaves containing the surface coordinates. Given the surface voxels, we can march outwards from the surface through local neighbor operations using the tree. An iterative procedure uses the following idea for a propagation algorithm. We add all voxels that contain a surface point to an active queue. While there is a voxel in the active queue, we take a voxel from the queue and try to update the CDF information of its 8-connectivity neighbors. If any of the 8-connectivity neighbors updates its distance information, that neighbor is added to the active queue so that it can update its neighbors as well.

These approaches can also be combined. We can first perform a surface-out computation of the CDF over a tree with a given branching. Then, we can use a root-down approach to refine to regions of interest within a given leaf by using that leaf as the root.

Feature Adaptation

We can use a root-down approach to resolve a given feature in the quadtree by subdividing the children that contain a feature of interest. Note that for some complicated geometries, local feature sizes can vary by orders of magnitude [Branets and Carey, 2005]. The quadtree structure is suited to resolve different levels of resolution through branching.

We discuss resolving two features in particular: a prescribed distance level or offset from the surface and regions between points.

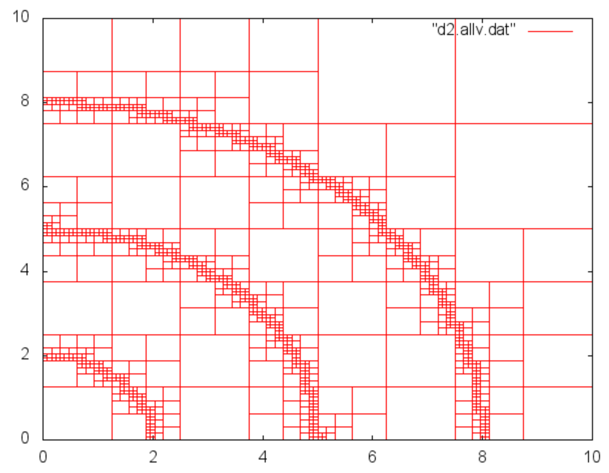


Figure 3.5 Adaptation to $2^2, 5^2, 8^2$ d levels about $(0, 0)$

Adaptation to Distance Level

We can subdivide any leaf that contains a distance level within its bounds. Fig. 3.5 shows the root-down adaptation of a root cell to a fixed resolution to three distance levels. Note that there is some gradation in the sizes of the cells outside of the specified levels as we must divide the root regularly to resolve the level of interest. This example also shows how we will use the CDF and quadtree to resolve the body-fitting curves in the surface-oriented meshing algorithm.

Adaptation to Multiple Points

We can subdivide any child that contains multiple corresponding points. Fig. 3.6 shows the subdivision of any cell that has more than one corresponding point in a field generated by eight points to a maximum leaf size of .01. This example also shows how we can use the CDF and quadtree to construct a Voronoi diagram.

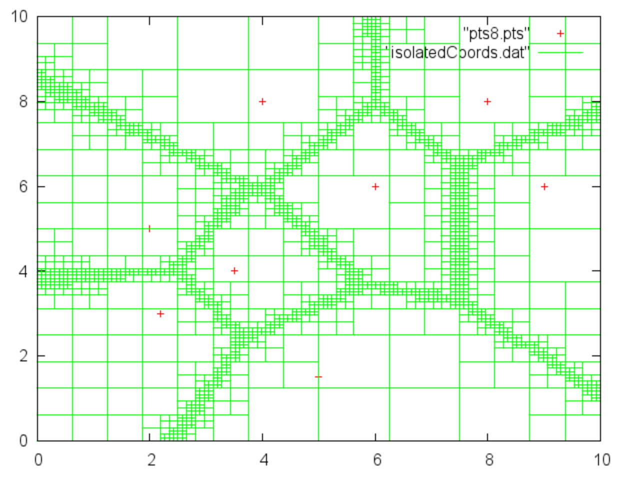


Figure 3.6 Adaptation to multiple corresponding points

If we have an association between surface points and objects, we can subdivide any child that contains corresponding points from different objects instead.

Contouring of a Feature

Our goal is to obtain an explicit representation of a feature represented implicitly in a field. We represent the feauture explicitly with nodes and connectivities. The problem then becomes obtaining the coordinates of the nodes and the connections between them.

A traditional approach in Cartesian meshing is to use the local size of the leaves to dictate the sampling resolution of the geometry and obtain the surface mesh. Here we apply this idea to sample the features.

Coordinates

We make one coordinate per feature per leaf. The location of the coordinate depends on the type of feature that is being contoured. For our purposes, we need coordinates for the features of offset curves and merged regions.

Distance Level

We subsample each box that contains the contour level and average the midpoints of the subsamples that also cross the contour. Note that, if the distance field is computed just over the coarser voxel, a subsample is not guaranteed to cross the contour level, since the maximum distance we calculate is a loose upper bound on the exact maximum distance for multiple points.

Figure 3.7 show the results for this approach for several types of contours.

Note that this approach is guaranteed to miss certain topological information at the leaf level. Higher order approximations such as splines, templating, and cell subdivisions used in the field of isosurface meshing can be utilized if topology preservation is critical and further refinement is not an option.

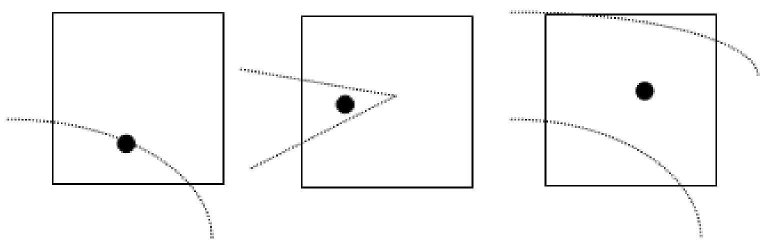


Figure 3.7 Examples of coordinates for various contours

Merged Region

We subsample each box that contains the contour level and take the midpoint of a subsample with the greatest minimum distance bound. It would also be reasonable to create a uniform mesh within the voxel and calculate the distance of each node to the corresponding surface points. Again, we would take the node with the maximum distance, as representing the location of a local maxima is a goal.

Contour Connectivity

Since both of these features originate from expanding waves out of the surface points, they should be piecewise continuous. Our goal is to connect the feature's path through the voxels. We do this through local neighbor operations.

For simplicity, we make use of the terminology of image processing to discuss connectivity. The collection of neighbors of all the faces of a cell are termed 4-connectivity, and 8-connectivity is the 4-connectivity and corner neighbors.

Offset Contour Connectivity

The offset contours are piecewise continuous. For two neighboring voxels to be connected along a continous contour, it is necessary for both of them to contain the contour level within

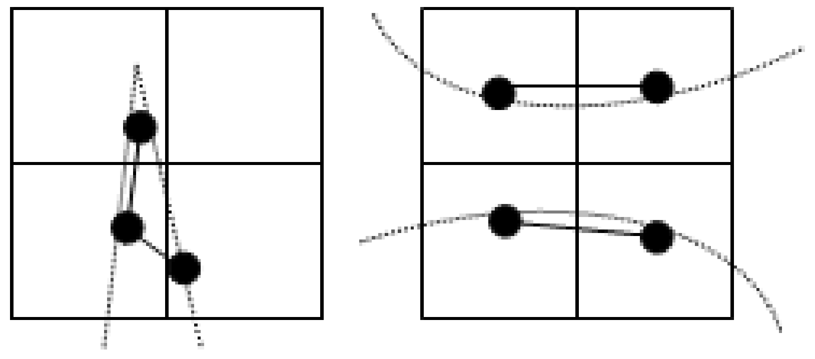


Figure 3.8 Examples of connectivities along contours

their distance bounds. However, this is not a sufficient test to generate the correct topology for the curve. As can be seen in Fig. 3.8, neighboring curves need not connect.

Our approach is to first consider the 4-connected neighbors. We then test whether the contour passes through the face between the voxels. A subsampled voxel of one on the level must be a neighbor of a subsample of the other voxel that is also on the level. The number of refinements taken to obtain subsamples dictates the sharpness of the contour feature that will yield the correct connectivity.

Fig. 3.8 shows the resulting connectivities for sample cases.

The corner example has been chosen to show the effect of having an under-resolved geometry. We are essentially left with a hanging segment for the corner.

Surface and Merging Region Connectivity

Since the surface is input as a discrete number of points, the zero contour is not necessarily 4-connected. A clear case is a set of points sampling a line running diagonally through the corners of the boxes. Since a sample need not lie on the corner, we need to include the corner neighbors not already connected to a face neighbor. The connectivities for merging regions are generated in the same fashion.

Merging Regions

The two cases we concern ourselves with are regions inside and between objects. Here, the underlying implicit representation allows us more flexibility than the typical explicit advancing front schemes, which usually halt the marching when fronts collide and then stitch the fronts together.

While we can also incorporate the stitching approach, we present another method based more on the geometry of the domain itself. Note that we use the minimum distance over the box for the identification of the regions below as it is exact, even to multiple surface points.

Interior

Imagine the interior contour levels of a circular pipe. The contours inside the surface have successively smaller lengths until they reach a point at the center and then vanish. While no hole will be present if that center point is specified as a contour level, that is hardly a robust or desirable solution.

We exploit the distance values in the field to identify these regions. A region where the front is collapsing inside of an object will have a local maximum. Specifically, we know that the distance will be greater than any of its neighbors.

Note that sharply concave regions can also be treated with a similar approach. In these regions, the test is whether points on one contour level have a corresponding point on the next level as described in Section 3. If not, the corner was sharp enough to collapse the front away.

Between Objects

Imagine the distance field around two points in the plane. As the circular contours expand from each point, there is a line dividing the two points where the fronts collide. Using the correspondence information we store, it is simply a matter of identifying the voxels that contain surface points from different objects.

If surface points are input without any connectivity information, we must first determine how many objects exist and associate each surface point to its object. In the surface-oriented meshing approach, edges are created along the zero isosurface. The voxels crossing the surface are connected along the zero contour and disconnected from the other surface curves as an input requirement. So, we determine the number of objects and associate the voxels to those objects by following the paths of those edges.

Normal Correspondences

An apparently unique aspect of our approach is obtaining the normal direction from correspondences in the distance field rather than derivatives, thus avoiding differentiability issues in the weak solution that the contour levels represent. It is important to note that normals coming from the distance field are guaranteed not to cross.

For a more formal description of the normal, let $C_1, ..., C_n$ be the body-fitting curves for the surface C_0. For points $p \in C_m, q \in C_{m+1}$, we have three possibilities for these correspondences, with 2-1 as shorthand for multiple values taking on a single value:

- 1-1: Normal
- 2-1: Shock
- 1-2: Fan

While these terms are intended to depict geometric behavior, they also represent physical processes. For example, the original work on level sets refers to the behavior of burning flames, crystal growth, and viscous phenomena. The different cases can also be imagined as the amount of material in the offset curves changing under evolution.

1-1 is the simplest case. It occurs when q has a unique correspondence with a single p. For typical objects and offset distances, the majority of points will fall in this category.

2-1 occurs in concave regions. Multiple p's will correspond to the same q. This occurs when the offset distance is greater than the local radius of curvature. The highly curved section collapses into a single point under normal motion.

1-2 occurs in convex regions. A single p will correspond to multiple q's. Essentially, this is the dual process of the creation of a shock; here, regions of the offset curve collapse into a single point under normal motion in the opposite direction. For example, any corner is transformed into a portion of a circle after offsetting.

Finding the closest coordinate

A brute force approach to determining the closest coordinate for correspondence would be to calculate the distance from the given point to all points on the corresponding level. If there are many coordinates per level, we can be more efficient by exploiting the point correspondence and neighbor information in the voxels of the distance field.

Consider the case given in Fig. 3.9. First, we find the point corresponding to the top right corner of the box in the offset arc. Note that the voxel containing the corner coordinate has the surface points that generated its distance bounds stored. We first add this original, containing voxel to a list of candidate neighbors. We then iteratively add to the list the 8-connectivity neighbors of the voxel's in the list that (1) have a corresponding point in common with the original voxel and (2) fall between the specified distance levels. Out of

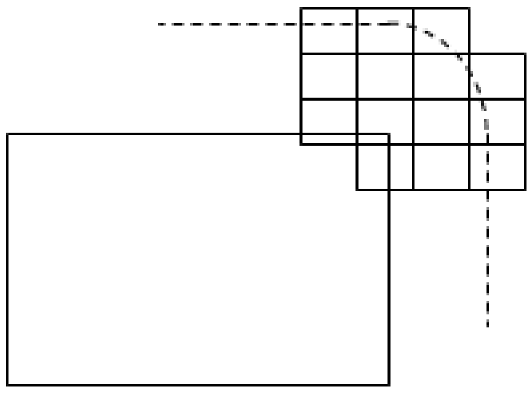


Figure 3.9 Finding the closest corresponding point through a flood-fill approach

this candidate list, we can isolate the voxels that lie on the next distance level. We then take as the corresponding point any coordinate that is the minimum distance to the corner point. Calculating the minimum of all of the distances in this smaller set requires (far) fewer distance computations. Note that this flood-fill operation will not find a voxel on the next level for concave regions where the front collapses and the correspondences vanish. In fact, this is our method for detecting vanishing regions in merging fronts.

We can also create correspondences from a point on a given distance level to the closest point on the previous level. The procedure is exactly the same as just described. Importantly, the flood-fill operation will always find a neighbor on the previous level since the surface points generate the distance field. For correspondence, the corner of the box goes to some point in the arc, and all of the points in the arc correspond to the corner.

Edges to Cells

Given edges between nodes, we need to obtain the polygons in the mesh. For a valid mesh, these polygons are disjoint and cover the domain. In 2D, this is straightforward to do

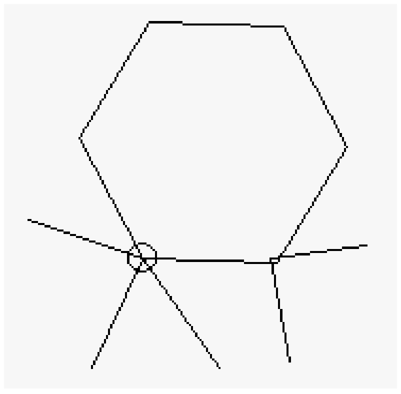


Figure 3.10 Sorting edges by angle into a polygon

by sequentially sorting the edges around a given edge by clockwise angle. We then assemble a polygon by tracing the counterclockwise path of an edge back to itself. Each internal edge is a part of two polygons. The edges that form the boundary to the exterior of the domain are only members of one valid polygon.

Fig. 3.10 shows a simple example of sequentially choosing the most counterclockwise next node to construct each polygon.

CHAPTER 4
COMPOSITE ALGORITHMS

In this chapter, we describe the algorithms for surface-oriented and Voronoi-dual meshing. The tools described in Chapter 3 were implemented specifically for these algorithms.

Surface-oriented Meshing

We construct a mesh by first making edges along and between offset curves generated from the distance field. Additional edges may be added to regions between merging fronts. We then transform these edges into polygons.

Pre-processing

We input the geometry as a collection of points that represent the surface, i.e. a point cloud. For simplicity, we convert any other format to a point cloud. Any connectivity information is lost in this conversion, and it may be worthwhile to incorporate that information separately.

Parametric Boundaries

Given polynomial segments that represent the boundaries, we take a fixed number of samples for each segment to create a dense set of surface nodes.

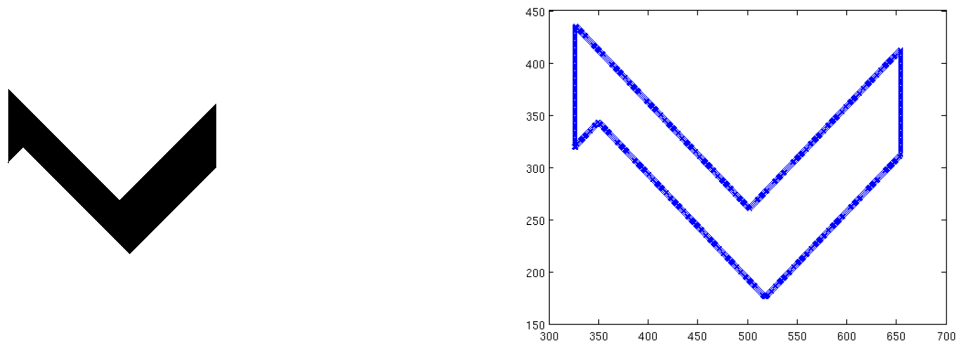


Figure 4.1 An example of an image and its boundary points

Digital Images

The images given in the results were created in black and white in a typical Paint-type program. Segmentation was done using thresholding, and surface coordinates were created at the midpoints of the pixel faces between object and background. Fig. 4.1 shows a result of this approach.

Density Requirements

For this algorithm to behave as intended, the surface points must be sufficiently dense. For our purposes, this requires the following conditions depicted in Fig. 4.2:

- Surface voxel size > minimum distance between neighboring surface points → surface mesh is connected
- First offset distance > maximum distance between neighboring surface points → offset curve "looks like" surface
- Points from disconnected surfaces do not lie in neighboring voxels → disjoint objects
- Separate curves on same isosurface must fall in separate voxels → coordinate represents one curve

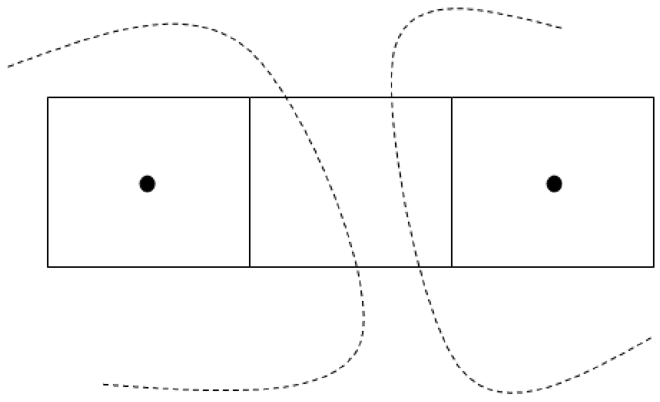


Figure 4.2 A depiction of surface density requirements. In short, close objects must be neighbors, and disjoint objects must not be close.

The first condition ensures that the surface mesh is connected. The second ensures that the offset curves "look like" the surface. A separate condition on voxel sizing requires that the surfaces of disjoint objects must not lie in the same object.

Algorithm

The steps of the algorithm are as follows:

1. Input surface points and distance levels for offsets as data files
2. Make covering root voxel and set parameters for minimum voxel size and number of refinements for subsamples
3. Refine about the distance contour levels
4. If desired, identify voxels in merged regions and contour the merged features
5. Contour the offset curves
6. Create correspondences between levels (each coordinate comes from the closest coordinate on the previous level)

7. Create correspondences to merged voxels (each coordinate comes from the closest coordinate on the previous level)
8. If desired, collapse edges whose points are within a specified tolerance
9. Output coordinates and node to node connectivity
10. Generate polygons from node to node information

Input

We take in the following inputs from files:

- Point cloud
- Desired distance levels for body-fitting curves
- Dimensions of the root voxel
- Minimum leaf size and number of refinements for subsamples

Example

We proceed with a simple case to demonstrate the steps of the algorithm. Fig. 4.3 shows the input point cloud of 50 points sampling a circle of radius 1 centered at (4,4), the adaptation to a resolution of .2 to distance levels of $0, .7^2, 2^2, 5^2$ in the quadtree, the contouring of the offset curves, and the normal correspondences of each level to the closest coordinate in the previous level. This example is constructed to be instructive, not an example of a "good" mesh.

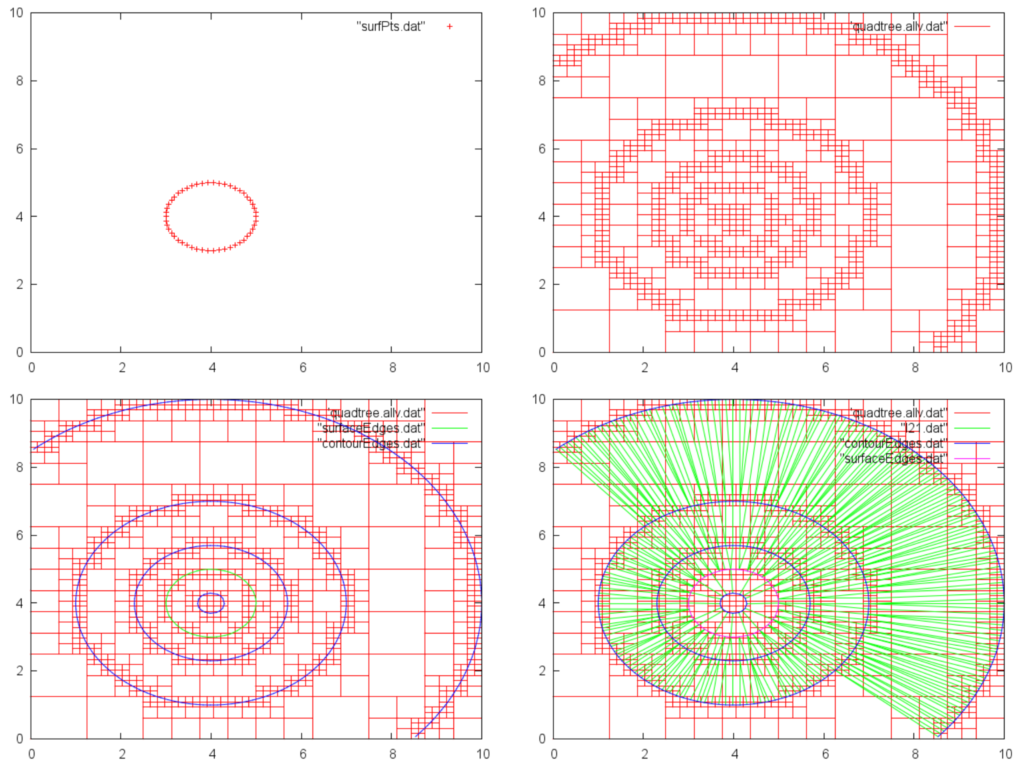


Figure 4.3 Step by step example of surface-oriented meshing algorithm without merging regions from surface points, to distance level adaptation, to contours of offsets, to normal correspondences

Voronoi-dual Meshing

Given the correspondence information in the CDF, it is straightforward to adapt the quadtree to the Voronoi boundaries by refining the leaves with multiple corresponding points. We can thus resolve the boundaries of the Voronoi diagram to any desired resolution.

We can go even farther to create a divide and conquer approach to the search for Delaunay edges. Remember that an edge in the Delaunay tessellation occurs when exactly two points are equidistant. For example, five co-circular points create a pentagon. An adaptive approach grants us added efficiency since we only need to detect each edge once, and we do not need to store the full quadtree down to the finest level. The first quality allows us to target our adaptation in the quadtree, and the second allows better memory management. In short, we search each leaf individually for undiscovered edges.

Holes

There are a number of cases in which holes exist in the domain. For example, impermeable objects such as most airfoils are holes in the space through which a fluid can flow. A Delaunay mesh over the domain will generally cover these holes.

We take one solution where we exploit the CDF to "block" nodes from "seeing" across the hole by the creation of ghost points. Here, we must remember that the Voronoi diagram is a representation of natural neighbors. Consider two sites in the plane. If we insert a dense line of additional sites between the original two, the two original sites will no longer be neighbors. By intelligently adding additional sites between two sites, we can achieve the same effect as the line.

Fig. 4.4 shows an example of the locations of the ghost points, and the edges they can block.

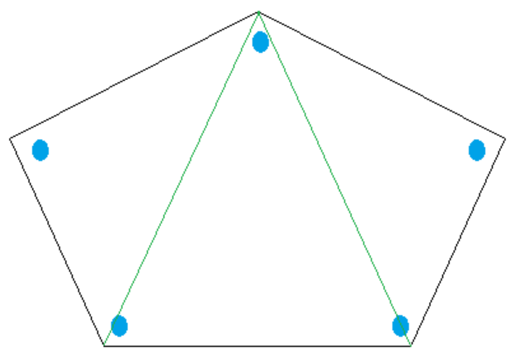


Figure 4.4 Location of ghost points for a hole region

The construction of the ghost points depends on the definition of the hole. If the segments are unknown, we need the ghost points to block across the hole but not along the surface. In practice, this does not seem possible for general boundaries.

Oriented Segments

Oriented segments are one way to represent a domain with holes through directed edges. As we walk along each edge in the specified direction, we define the interior of the domain to be to the left. Thus, the exterior domain or hole region is to the right. We block such a hole by creating a ghost node for each boundary node. The coordinates of these ghost nodes are offset to lie just in the hole. The steps are summarized below.

1. Tag each node as not a boundary node
2. Loop over boundary segments and tag nodes as on a boundary
3. Create ghost node coordinates equal to the corresponding node's coordinates
4. For each boundary segment
    (a) Calculate a normalized normal facing outside the domain
    (b) Add as a vector a scaling of the normal to the ghost node's coordinates

Fig. 4.4 shows an example of the locations of the ghost points, and the segments (in green) that they are designed to block.

Algorithm

The steps for creating the Voronoi duals are as follows.

1. Input coordinates and definition of holes
2. Refine the quadtree to isolate each coordinate in distinct voxels
3. Generate ghost points for holes
4. Calculate the correspondence distance field over each leaf
5. Make any known boundary-boundary and boundary-ghost edges
6. For each leaf with multiple corresponding points not already connected,

   (a) Calculate the minimum leaf size as a fraction of the minimum distance between the corresponding points
   (b) Refine the leaf to the undiscovered pairs of points to the minimum distance
   (c) If the the points correspond in a small voxel, make an edge between them
7. Output node-to-node information for non-ghost edges
8. Generate polygons from node to node connectivity

The adaptive nature of this algorithm means a storyboard example of each step is difficult to present. After isolating each coordinate in the quadtree for an initial adaptation, each leaf is examined much as in Fig. 3.6, but with the refinement terminating whenever an edge is known. An optimization of the algorithm does exist in the order that we investigate leaves for the edges. We loop through the cells containing surface points, then the cells that are smaller than a given multiple of the refinement tolerance, and then the remaining cells. The effect of this order is to reduce the total number of refinements since we only search for new edges.

Input

If the domains have no holes, the only input is the coordinates that will be vertices in the dual tessellation. The vertices are the sites in the plane as shown in Fig. 2.15.

Abstraction of Approaches

We provide a layer of abstraction in order to unite the two specific approaches. A generalization of the core of the approaches is as follows:

- Detect: Test a region for a given feature
  - Surface-oriented: Distance levels from the surface

  - Voronoi-dual: Multiple points

- Resolve: Adapt the quadtree locally to resolve the detected feature
- Represent: Explicitly describe a resolved feature
  - Surface-oriented: Contour offset curves
  - Voronoi-dual: Edge for cell with two corresponding points
- Gather additional information
  - Surface-oriented: Add edges for normal correspondences and subsets of the medial axis
- Transform the explicit representations: Edges to polygons

CHAPTER 5
RESULTS AND DISCUSSION

In this chapter, we present and discuss the results of our two composite algorithms to better understand their effectiveness. All cases were executed using a single core of an Intel(R) Core(TM) i5 CPU 750 at 1.2GHz with over 3GB of RAM.

Surface-oriented Meshing

Parameter Choices

Here we discuss the influences of the choices of the number of subsamples, distance levels, and leaf size on the resulting mesh. For concreteness and simplicity, our base case is represented in Fig. 5.1. We can see the sine curve in green, body-fitting curves in blue, correspondences between levels in purple, and the background quadtree in red. The distance levels are $0, 1^2, 2^2, 3^2$, the number of subsamples is 5, and the maximum leaf size is .4.

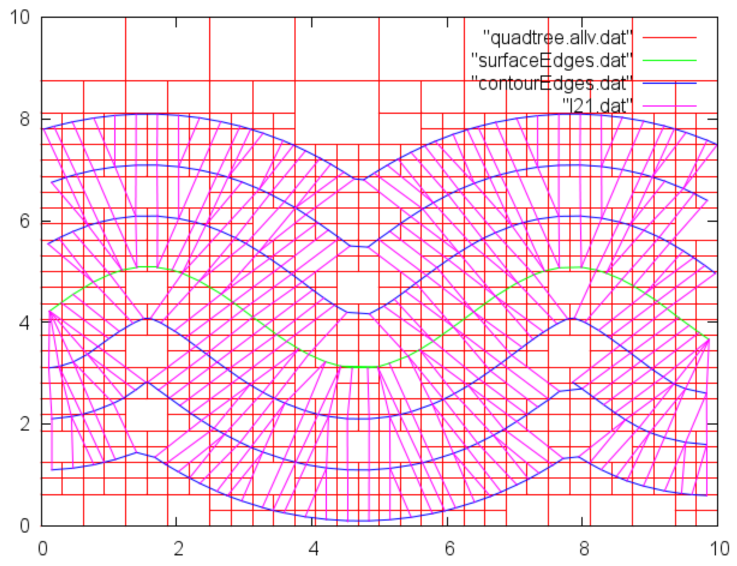


Figure 5.1 Base case of sine curve for effects of parameters on surface-oriented meshing

Leaf Size and Subsamples

These choices affect the sampling resolution of the given features.

A resolution that is too coarse does not approximate the feature well. For example, not subsampling a leaf will return the midpoint of the leaf as the coordinate for all contained features. All contour levels and merged regions would collapse to that point.

Although not as extreme, Fig. 5.2 shows that a coarse subsampling of one level will also alias the curves as there are not enough samples to obtain a representative average coordinate. Note that the subsampling is also used to update the bound on the maximum distance on each leaf.

A resolution that is too fine is more expensive and may lead to an undesirable element count in the case of leaf size. Fig. 5.3 shows a case with a smaller leaf size of .05.

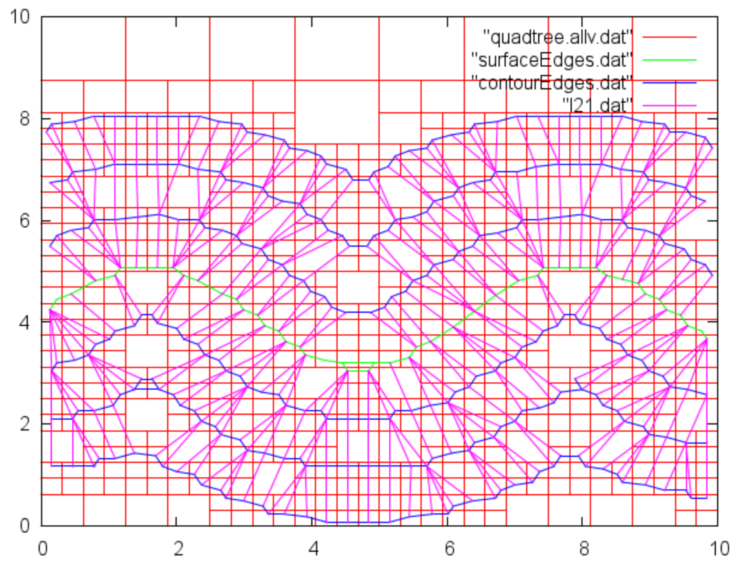


Figure 5.2 Aliasing of coordinates due to coarse subsampling

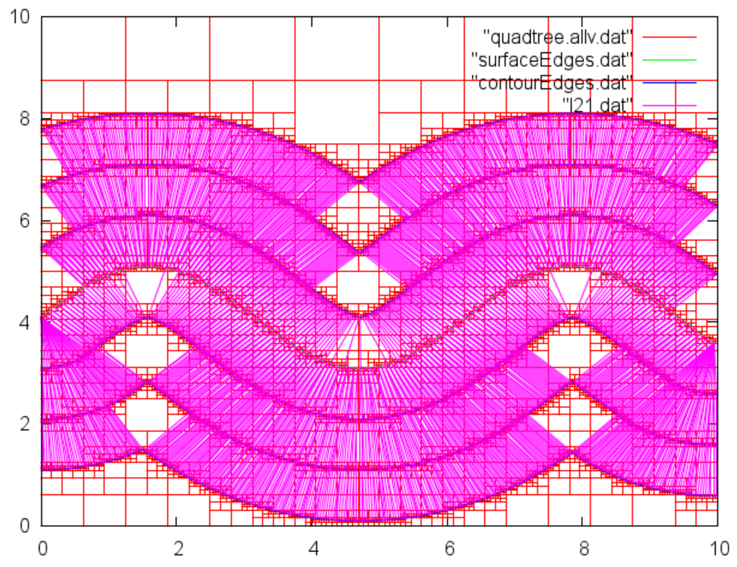


Figure 5.3 Greater contour sampling and element count due to smaller leaf size

Distance Levels

The distance levels determine the locations of the contours and the refinement of the quadtree. The levels should be chosen to cover the domain of interest. The choice of specific values depends on the sizes of the features of interest. For a large offset, distinct surface features will tend to interact and merge. A minimum bound depends on the density of the surface sample.

Fig. 5.4 shows a case with the distance levels set at $0, .5^2, 1^2, 4^2$. The closer spacing near the surface preserves the curvature of the original surface. The farther spacing to the last level creates large concave regions without correspondences due to only using the rule that every coordinate comes from the closest on the previous level. We can also see that there are larger leaves in the quadtree, and the boundary of the root chops the bottom contour.

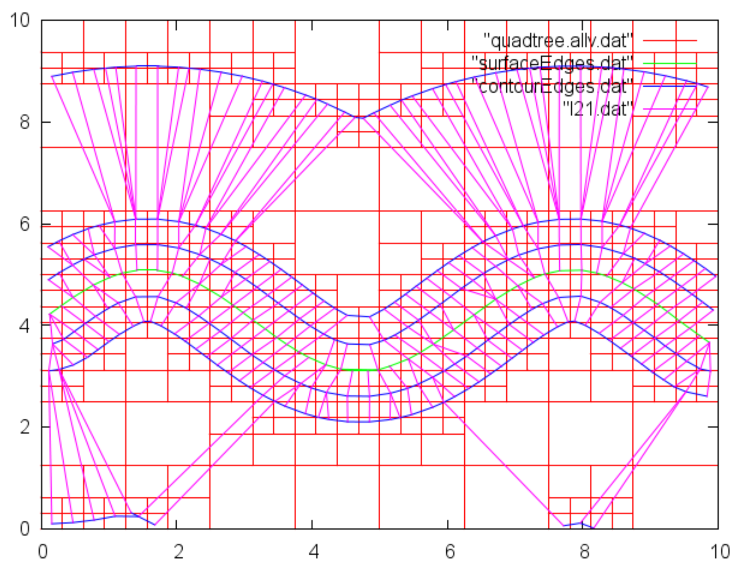


Figure 5.4 Feature preservation and smoothing based on choice of distance levels

Image Geometry

For all image cases given here, 100 distance contours were specified in increments of 5, where 1 unit represents the edge length of a pixel in the original image. The sizes of the images are on the order of 500 pixels by 400 pixels. The adaptation was set to stop refining once the width of the leaf capturing the feature was less than 5 units.

The far view in Fig. 5.6 shows the captured surfaces, the local maxima, and the regions between the objects from the original image show in Fig. 5.5. The surface point cloud has 1236 points. The case required 16s to execute, and the mesh has 14,755 nodes with the maximum number of edges in a polygon as 28.

Fig. 5.7 shows the interior of the rectangle. Note the isotropic appearance of the cells and the local maxima at the center. In Fig. 5.8, we see the interior of the diamond. Note the isotropic appearance of the cells and the local maxima at the center. We can also see an incorrect connectivity for the surface at the top, left, and bottom corners. Fig. 5.9 shows

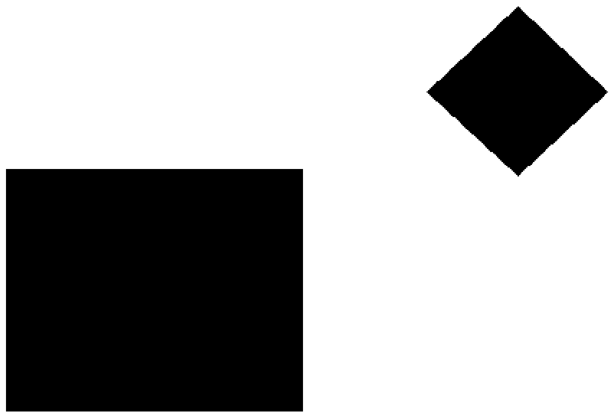


Figure 5.5 Rectangle and diamond geometry

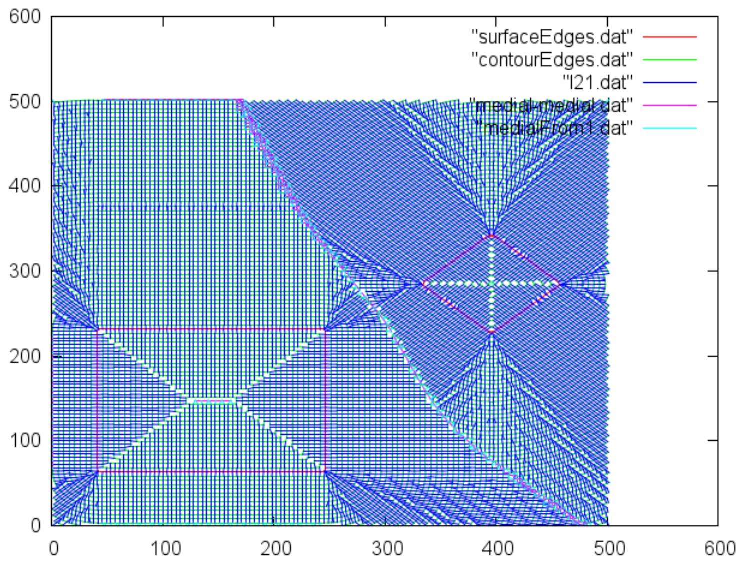


Figure 5.6 Far view of rectangle and diamond geometry

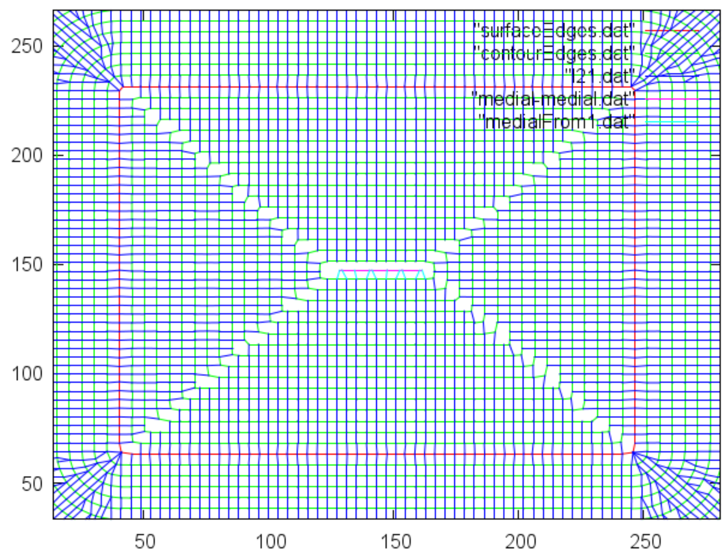


Figure 5.7 Interior of rectangle in rectangle and diamond geometry

the region between the rectangle and diamond. Note how the edges for the merging region act to stitch together the meshes between the separate objects.

The far view in Fig. 5.11 shows the surface represented as an inner and outer ellipse extracted from the original image show in Fig. 5.10. We also see an elliptical curve where those surfaces merge. The surface point cloud has 1896 points. This case required 16s to execute, and the mesh has 15,821 nodes with the maximum number of edges in a polygon as 26. Fig. 5.12 shows the region between the inner and outer ellipses. Note how the correspondence edges become arbitrarily small as the distance level approaches the same space as the merged curve. In Fig. 5.13, we can see the region inside the inner ellipse. Note how the shrinking offsets develop corners under the sharply concave evolution.

The far view in Fig. 5.15 shows the four curves that represent the surface and the merged region between all of those curves from the original image show in Fig. 5.14. The surface point cloud has 1839 points. The case required 12s to execute, and the mesh has 14,717 nodes with the maximum number of edges in a polygon as 20. Fig. 5.16 shows the merging regions

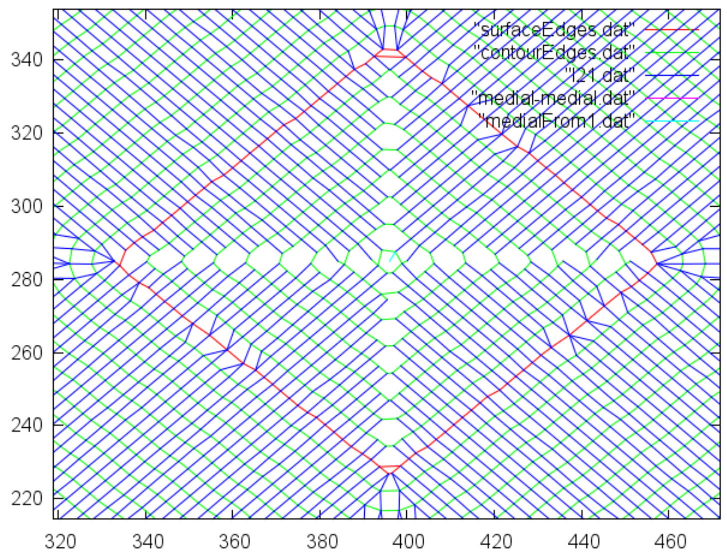


Figure 5.8 Interior of diamond in rectangle and diamond geometry

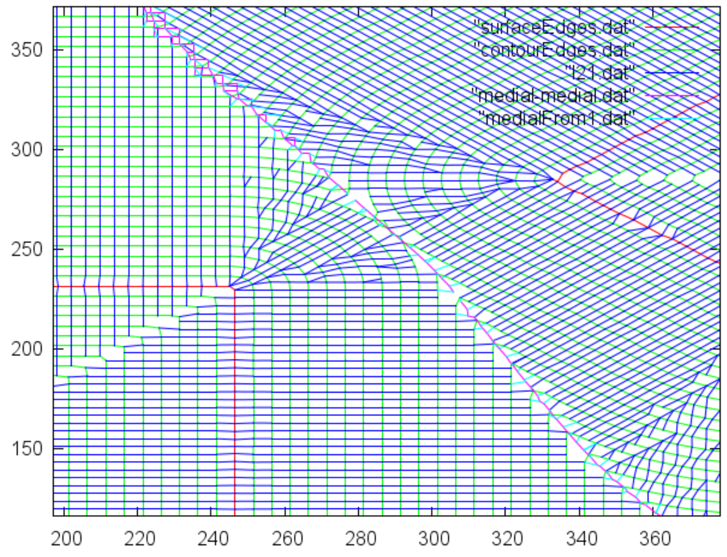


Figure 5.9 Merging regions in rectangle and diamond geometry

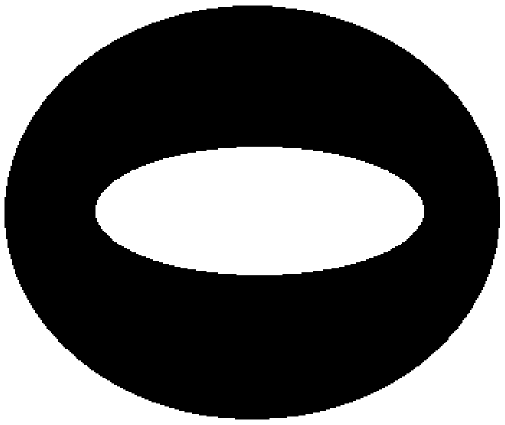


Figure 5.10 Donut geometry

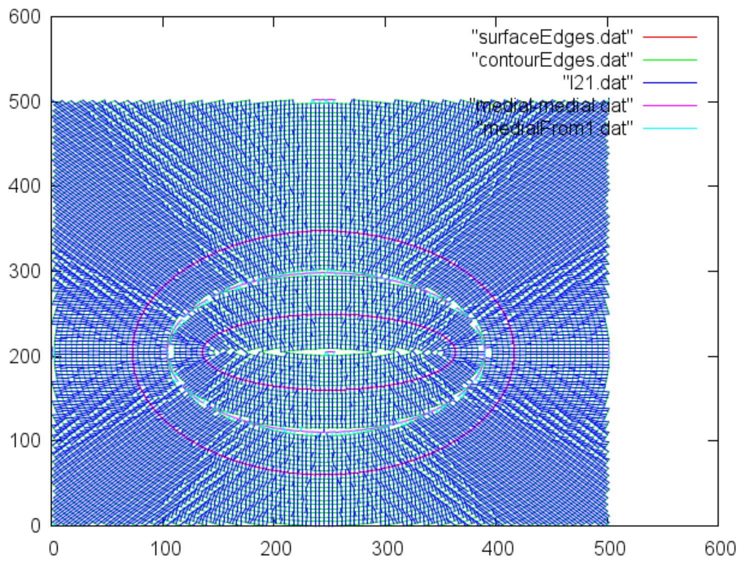


Figure 5.11 Far view of donut geometry

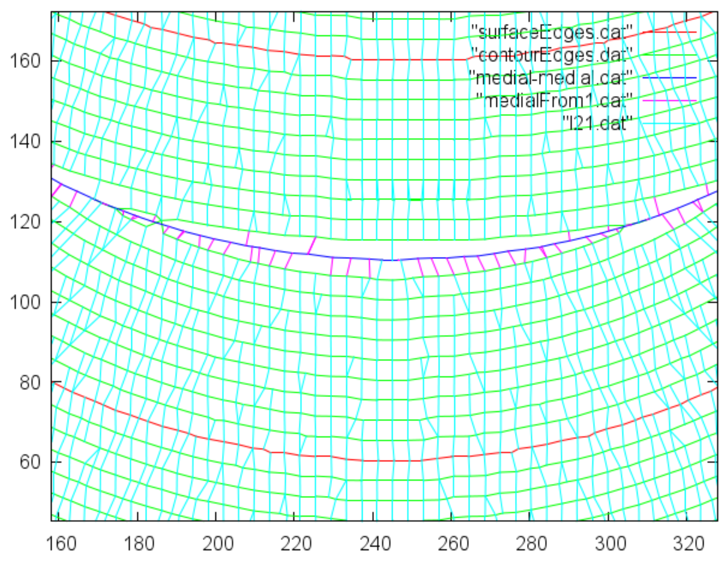


Figure 5.12 Merging regions between ellipses of donut geometry

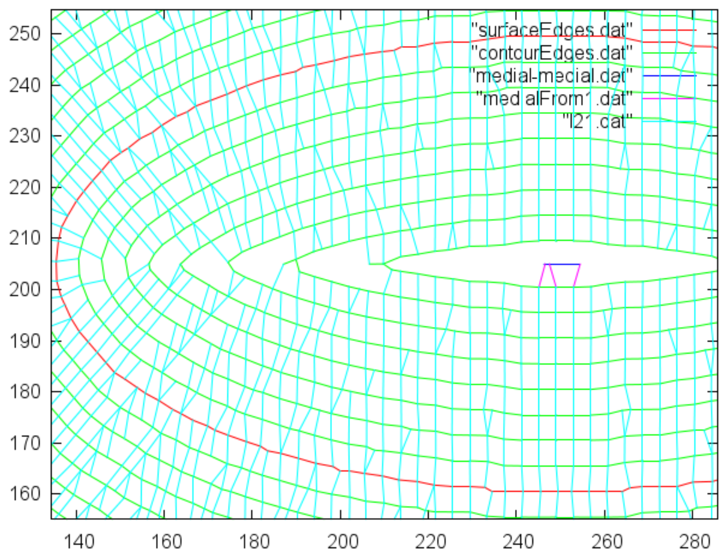


Figure 5.13 Interior merging regions of donut geometry

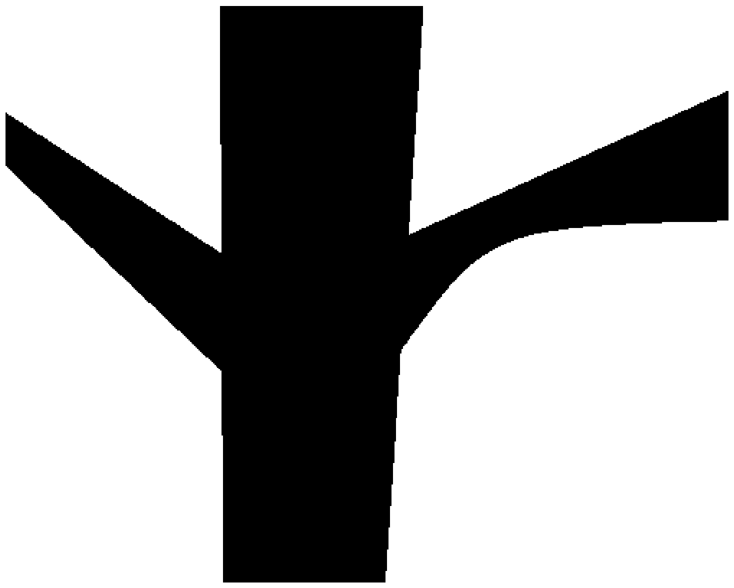


Figure 5.14 Branch geometry

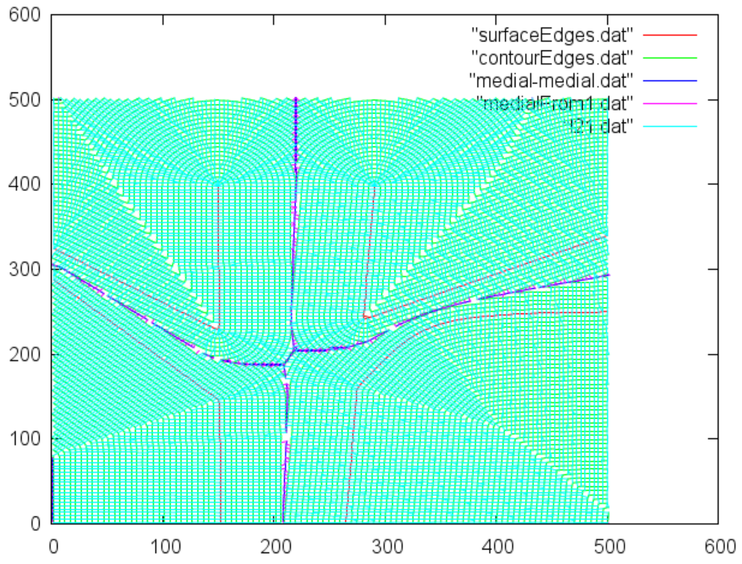


Figure 5.15 Far view of branch geometry

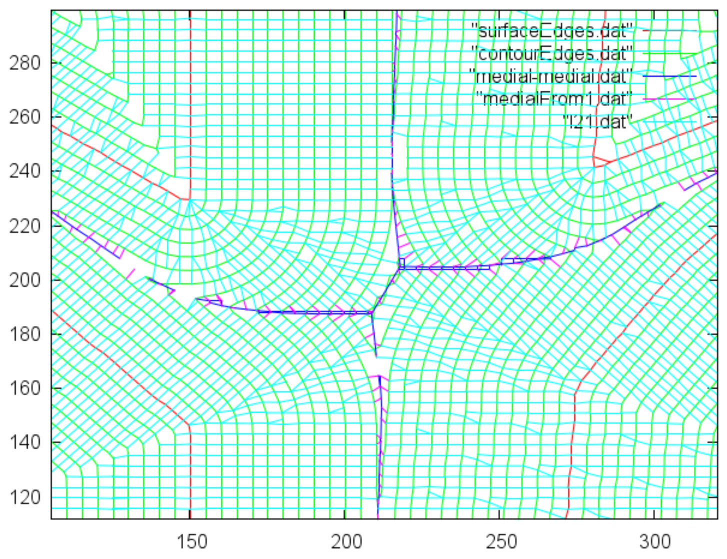


Figure 5.16 Merging regions of branch geometry

between the four surface curves. Note that the merged curve is not connected. For example, we see gaps near (150,200). While we spent considerable effort to determine whether this is due to a coding error or a property of using discrete surface points, we were not able to establish a conclusive justification.

Airfoil Geometry

We now provide results from an airfoil defined by boundary segments. To sample, we take one point at the beginning of each line segment. For the distance contours, we create a geometric progression of 20 levels starting at an offset of .005 with a growth factor of 1.1. The leaf size is refined to .005 units.

Fig. 5.17 shows the extracted airfoil and the spacing between the body-fitting lines increasing as we progress farther from the surface. The surface point cloud has 533 points. The case required 22s to execute, and the mesh has 5457 nodes with the maximum number of edges in a polygon as 13.

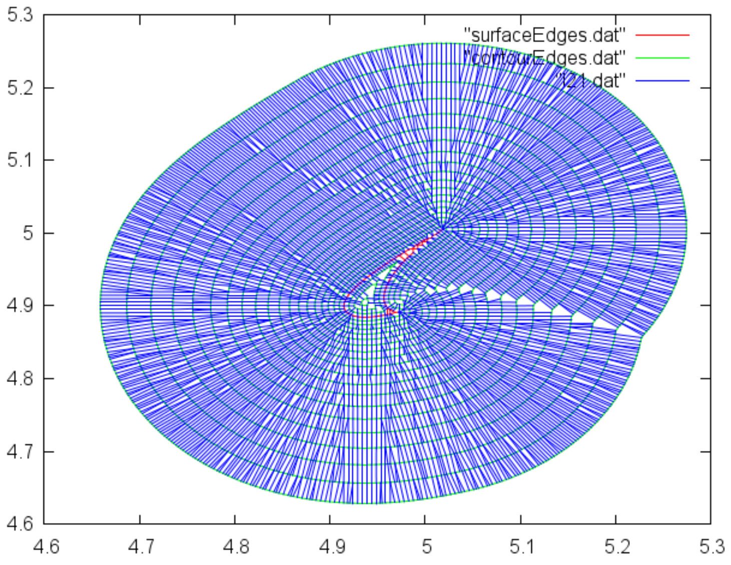


Figure 5.17 Far view of airfoil mesh

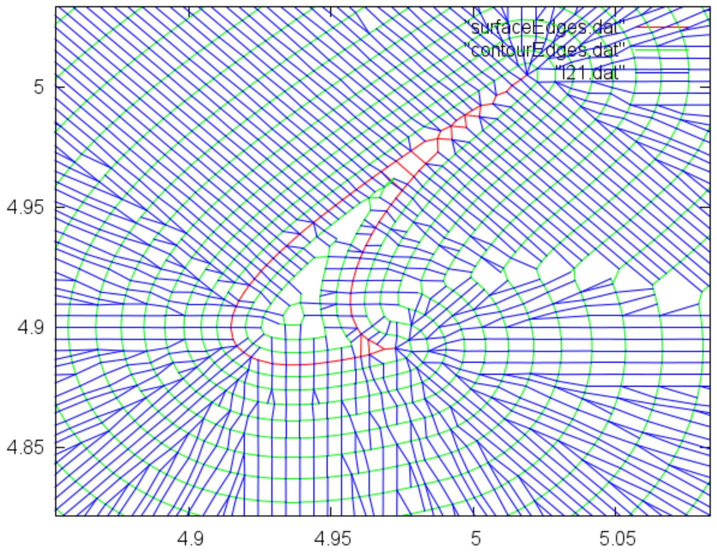


Figure 5.18 Close view of airfoil mesh

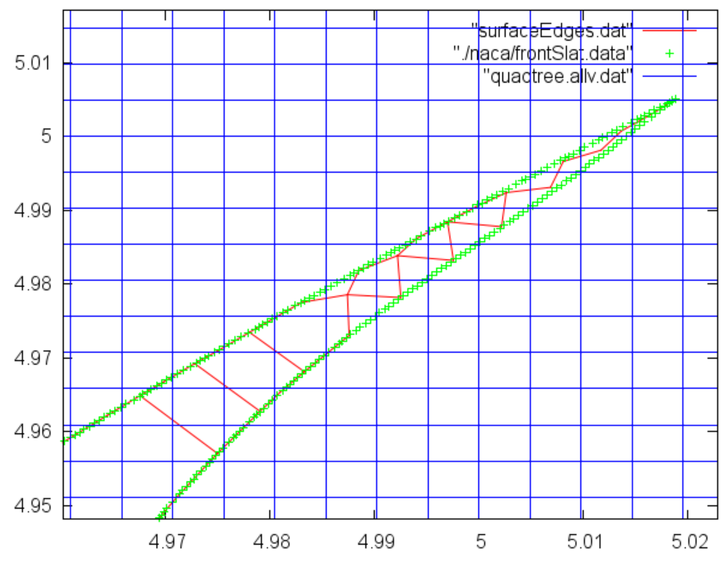


Figure 5.19 Zero-contour approximation of surface

Fig. 5.18 shows a closer view of the airfoil. Note how the cells are isotropic near the surface due to similar spacing along and between levels. As the spacing between levels increases, the cells become more stretched. In this case, no edges for merged regions are shown. Note the natural handling of the merging inside the airfoil due to the implicit evolution. We can also see the curvature on the surface generating the three types of branchings for the correspondences.

Fig. 5.19 shows the surface points, surface edges, and background quadtree used for sampling. We see that extraneous edges result from the violation of the density requirement where disconnected voxels on the same isosurface must be non-neighbors. These ”extra” edges cut across the corner in the real geometry.

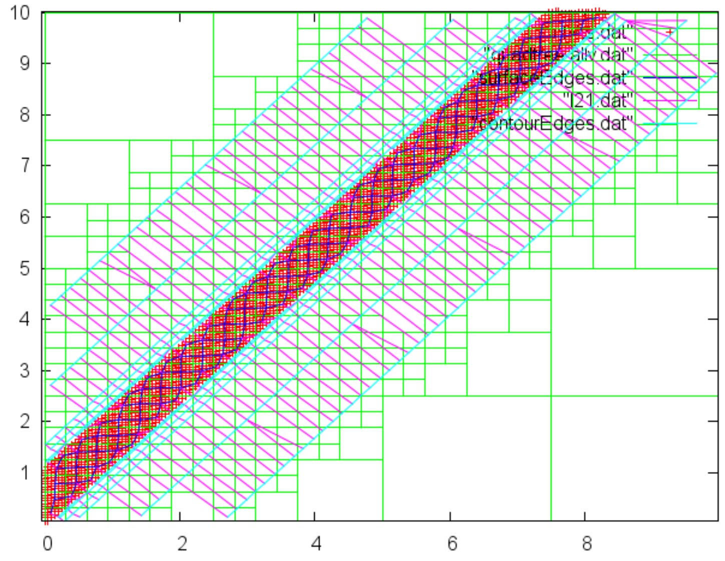
Viscous Spacing

By viscous spacing, we mean that the spacing along the contours is greater than the length of the correspondences between contours. These types of elements are used in flow simulations that possess high gradients normal to the surface to capture the physics.

However, our approach to isosurface sampling creates only one coordinate per isosurface per cell. Since we have an unsigned distance field, there is no distinction between offsets inside and outside of objects. Thus, separate offset curves for the same level near the surface need to fall in separate leaves. We achieve this by requiring the surface to be sufficiently dense or thick relative to the voxel size to separate the distinct curves. If the inside and outside regions are known, it is also possible to create a signed CDF.

For a specific geometry input, we can also generate this thick representation to flood out the surface in the distance field. For a parametric surface, the orientation of the interior and exterior of the domain is defined by the ordering of the faces. We can offset ghost points opposite the direction of interest to thicken the surface. For a segmented image, we have a representation for the solid object. Thus, we simply sample the faces between the objects and background as well as the bodies of the objects themselves. For a point cloud with local normal information such as from a laser scanner, we have an orientation similar to the parametric case. If no orientation information is available, an alternate approach is needed, perhaps converting the point cloud into another surface format.

Figs. 5.20 and 5.21 show fine correspondence edges relative to the spacing along contours, and the background sampling used to generate the field. The axes on the plots indicate the locations of the zoomed figures. The red points show thick sets of surface points, which are necessary to separate the viscous offsets. For the line geometry, we can also see an aliasing of the correspondences due to the regular sampling.



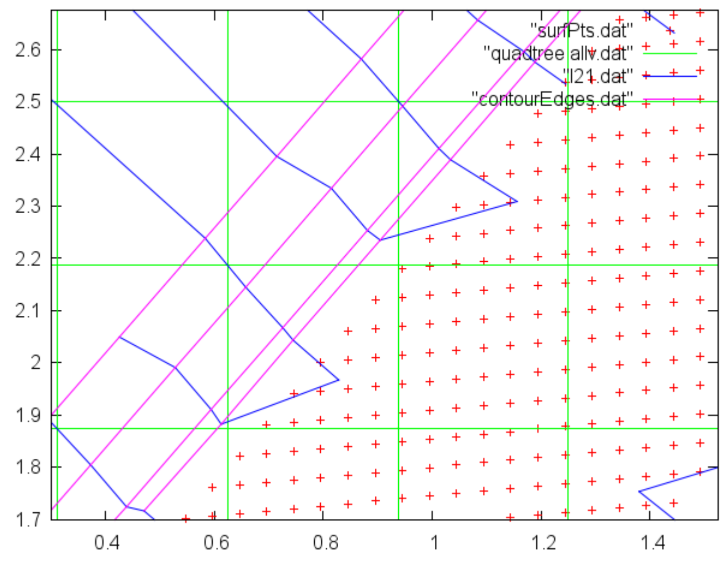


Figure 5.20 Viscous spacing for a line geometry

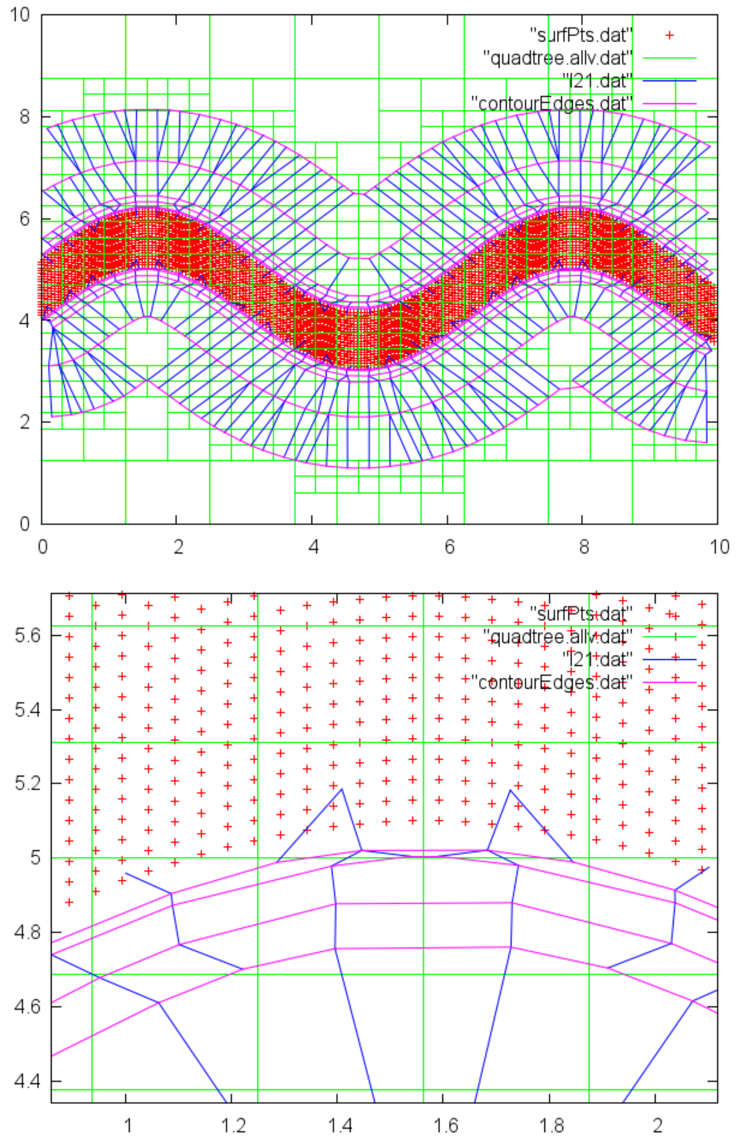


Figure 5.21 Viscous spacing for a sine curve geometry

Discussion

We begin with a recapitulation of the specific results given and then extend our comments.

The isotropic appearance of the meshes originating from images is achieved by having the spacing along the contour be similar to the spacing between the contours. Anisotropy can be obtained by changing the ratio of the leaf size, which controls the spacing along the contour, to the offset distance.

The surface may not be approximated well by the zero-contour of the distance field. While averaging of the surface nodes within a voxel provides a reasonable approximation, it will not generally preserve sharp features exactly. Optimization of the node locations can also be done. A more serious problem is the incorrect connectivity, as in the sharp edge of the airfoil shown in Fig. 5.19 for example. This is an expected deficiency of our approach. Reconstructing the correct topology is a persistent difficulty of contouring algorithms in general. Note that the correct topology is likely only critical for the surface contour for a CFD simulation. Until the construction of polygons in the final step of the algorithm, the connections between surface nodes do not affect any other region of the mesh. The mapping of surface points to surface curves only matters in the detection of voxels between objects.

A separate approach is needed to recover the correct surface. While many of the isosurface methods that output topologically correct polygonizations may seem appropriate, those that require refinement to ensure correct topology will generate many points near corners. There is an inherent tradeoff between accuracy and the number of elements. It is worth noting that there may be other solutions specifically for certain applications. If the surface originates from segments or an image, recovery is straightforward as the boundary segments or connected segmentation define the surface. If not, a level set or active contour algorithm designed for point clouds may also be applicable. Our results can couple with such an algorithm as we can generally construct the proper mapping of surface points to objects.

We can also obtain normal directions on the surface from the distance field. Essentially, while we do not use a surface mesh to construct a mesh in the space around the object, a surface mesh is required for simulation and is not a product of our approach. We list below a number of ideas to generate the surface or test candidate edges for future reference.

- Active contour or shrink wrapping around points
- Adaptive refinement of quadtree to edge
- Voronoi-dual meshing with ghost nodes blocking the interior
- Midpoint of edge should be near isovalue
- Edge should be tangent to surface
- Use correspondences in a convex offset curve resolved with small voxels to order neighbors
- Generate the polygon that encloses all other edges
- Once polygonized, every surface edge should be between the interior and exterior
- Generate a polygon with the maximum length using any edge at most once

Unfortunately, none of the approaches listed above will resolve very well the geometries shown in Fig. 5.22 given variable surface sampling. The selection of boundary edges, however, is not a complicated task for the user, especially for the 2D cases presented here. The only operation needed to delete an edge is the removal of the relevant nodes from each other's node-to-node information.

Although the airfoil geometry case had the smallest surface and resulting mesh, it had the largest run time. This is likely a product of distribution of points per level. For the image cases, the surface level had the largest number of points. For the airfoil case, the farthest three offsets each had about the same number of points as the surface levels of the image cases. Since normal correspondences are not made for each surface point, there is a

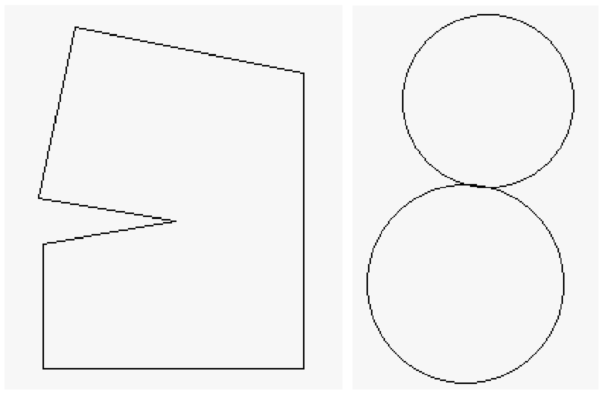


Figure 5.22 Counterexample geometries for surface recovery methods

difference in the number of operations. Moreover, the number of points per level decreases from the surface in the image cases due to the merging fronts and size of the root voxel.

Small edges may occur in the mesh. If the contoured feature oscillates near corners of the voxels, the neighboring coordinates will be close. The consistency of edge lengths due to the quadtree sampling also tends to degrade in regions where fronts are merging. Including local maxima and boundaries between objects are natural attempts to achieve better results. However, the specification of contour levels before computation of the distance field in effect eliminates the possibility of avoiding small edges in the mesh. Considering the merging regions in the branch case, refinement of non-convex polygons will yield a mesh more suitable for numerical simulation. Thus, we do not need to include any subsets of the medial axis, and the algorithm is significantly simpler. Regardless, a quality control step where edges below a certain length are collapsed is simple and could be quite fruitful.

If the contours are close relative to the voxel size and fill the root cell, the quadtree is full at the finest level. In this case, it would be more efficient to use a uniform mesh. However, it should be noted that we do not know the regions of interest a-priori.

Quadtree sampling may lead to an aliasing effect. For a straight line, we expect all of the correspondences between extrusions to be 1-1. This is not the case as shown in Fig. 5.20

for several corresponding edges. If the surface line were axis-aligned, the correspondences would be 1-1.

Assuming that the set of surface points is sufficiently dense to resolve the surface, a key parameter is the choice of offset contour levels. The distances from the object used to define the contours should be relative to the local scale of interesting feature sizes. Distances smaller than the resolution of the object simply create circles around the individual points and are too local to be meaningful. Distances too far from the body essentially create circles around the point mass of the object. If levels very near to the surface are required, they can be interpolated from the existing contours. This refining interpolation can be done as post-processing due to the inherent ordering of grid points based on distance from the surface.

A guiding philosophy for making decisions in the construction of this algorithm was to extract as much use from our own tools as possible. Where alternate approaches may be beneficial, they were considered (with most introduced in this report) but not necessarily utilized. The rationale of our approach is that documentation already exists describing the characteristics of an existing tool.

Voronoi Meshing

Simple Case

A simple, representative test case is shown in Figs. 5.23 and 5.24 for a domain with a four-sided hole in the middle.

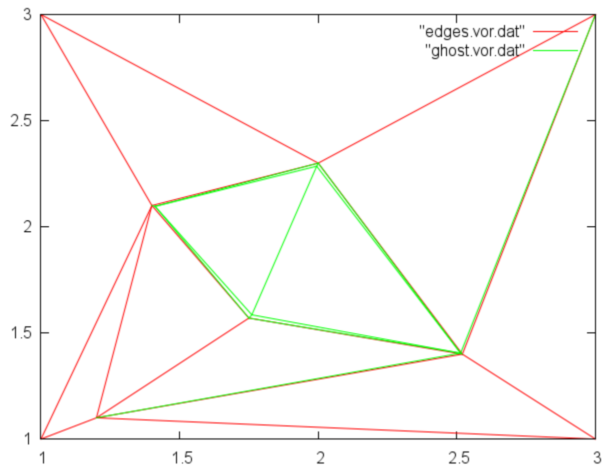


Figure 5.23 Delaunay tessellation of coordinates and ghost nodes

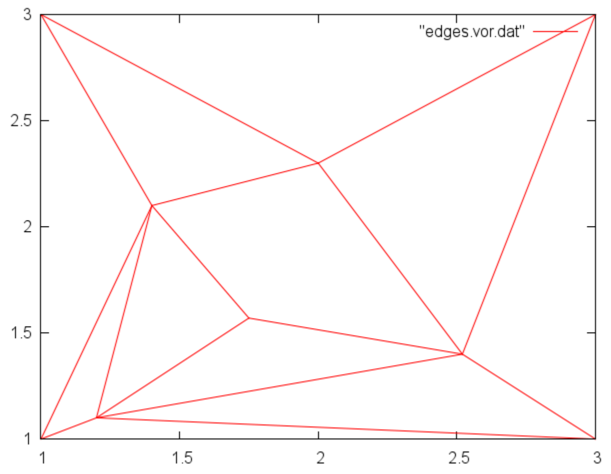
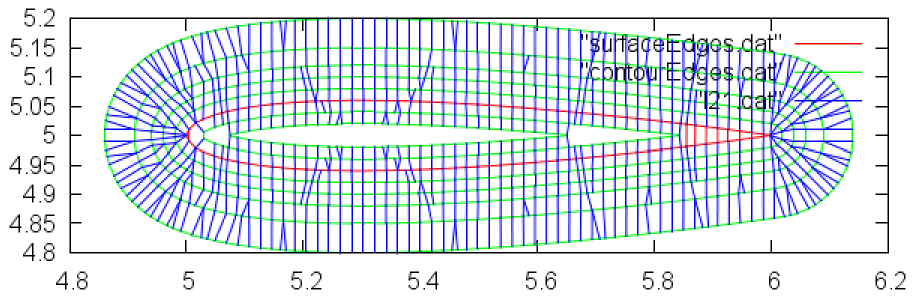


Figure 5.24 Delaunay tessellation of coordinates with center hole enforced

Combined Case

Our Voronoi-dual method is termed a batch approach in the Delaunay literature as all of the vertices are known a-priori. As one approach to generating the locations of these vertices, we can use some of the components of the surface-oriented meshing algorithm. Specifically, we can output all of the coordinates of the contoured features and use those coordinates as input to the Voronoi-dual algorithm.

Fig. 5.25 shows the meshes we obtain from the surface-oriented approach and the Voronoi-dual of those coordinates. The geometry is a NACA0012 airfoil defined by line segments. We also attempted to recover the boundary using the Voronoi-dual approach. We can identify the surface points in the surface-oriented approach as the points on the zero-contour. Since the geometry is convex, we tested the creation of a ghost point for every surface point offset towards the middle of the airfoil. However, the hole was not recovered at the trailing edge, and edges along the airfoil were lost.



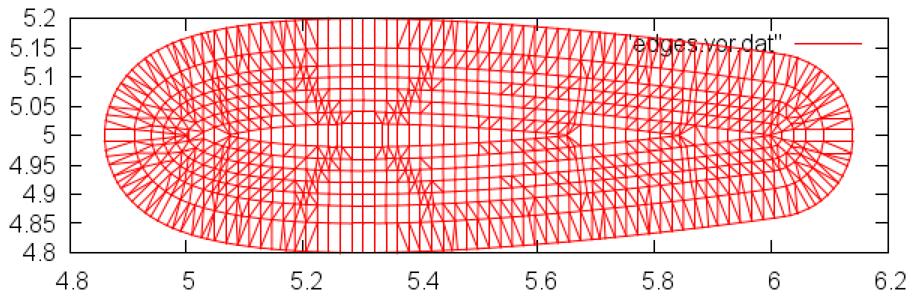


Figure 5.25 Surface-oriented meshing about a NACA0012 and a Voronoi-dual mesh of the coordinates

Discussion

The algorithm connects the neighboring nodes as expected.

As given here, there are no edges between the four coordinates at the far corners of the simple test case. This is because the root voxel was not large enough for the boundary in the Voronoi diagram to be detected between these coordinates. We intentionally include this result to emphasize that the size of the root voxel is a critical parameter.

A number of the previously cited works compute the distance field over a fine mesh. The adaptive nature of our search for Voronoi boundaries makes our approach considerably more efficient for cases when the spacing between neighbors varies. This is likely the case for, say, a CFD mesh for an airfoil as more nodes are typically clustered near the surface to capture the physics more effectively.

CHAPTER 6
CONCLUSIONS

We have presented the theory, development, and implementation of a correspondence distance field embedded in a quadtree data structure for use in creating meshes for field simulation. A number of tools based on this field have been implemented and described.

These tools were combined into two composite algorithms. The first generates a surface-oriented mesh based on contouring the distance field. Results have been presented for surfaces originating from images and parametric boundaries. As the surface is not generally approximated very well, we should consider the algorithm an approach to the generation of a "volume" mesh.

The second algorithm searches adaptively for boundaries in a Voronoi diagram to create a Delaunay tessellation. Results have been presented for meshing of regions with holes by incorporating ghost nodes that "block" connectivity across the holes. We also demonstrated the combination of the two approaches.

We now discuss possible future directions to take this research.

Extensions

There are a number of promising applications that were not pursued due to time constraints. We believe that implementation of the following topics would be straightforward.

3D CDF

The extension of the quadtree to an octree and the other updates resulting from the increase in dimension are the only changes needed to generate a 3D CDF with the same capabilities to adapt to features of interest. The framing of the offset contours as level sets provides a strong mathematical foundation to this extension.

Approximate Medial Axis

Since the focus of this work was on mesh generation, alternate applications were not fully discussed. Given the correspondence distance field, we can determine arbitrarily small bounds on the regions where the offset curves collide. There may also be a worthwhile connection between vanishing correspondences between levels and the discrete $\lambda - medial$ axis described in [Chaussard et al., 2009].

Point Generation

There are a number of mesh generation techniques that utilize a known batch of specified points. For example, some Delaunay insertion algorithms operate in this fashion. As our surface-oriented meshing algorithm generates a collection of points structured along distance levels from the body, it provides a sampling of the space with a known structure. In 3D, an application could be point generation for tetrahedral mesh generation.

Future Work

3D Voronoi Meshing

There is little difference between the 2D and 3D algorithms except for the generation of cells from edges. Sorting edges counterclockwise does not work outside of the plane. As the cells are convex, a shortest path algorithm should suffice.

The remainder of this section will focus on the surface-oriented mesh generation.

Couple with Flow Solver

As emphasized in [Dawes et al., 2007], the key quality of a mesh for our purposes is that it be solvable. Making a grid generation algorithm without considering how it will be used for the flow or a particular flow solver limits the algorithm's usefulness. Element shape quality tests can be an a-priori indicator of mesh quality [Beall et al., 2003], although only generally effective by identifying defective elements that would introduce significant interpolation errors. For anisotropic or varying flow, element quality metrics become much more reliant on engineering heuristics with better mesh density achieved by adapting the mesh to the flow solution. To measure the quality of our meshes, physics simulations should be executed using different meshes and results compared.

Our process generates more information than is contained in the final polygon output as well. There are additional steps between polygon definitions and a mesh for a flow solver. One step is the association of boundary elements to the boundary condition. If the subsets of the medial axis are not included, every node is associated to a level, and the surface and outermost offset curves are known. Some turbulence models require the distance to the nearest wall as discussed in [Tucker et al., 2005]. This can be output easily from our mesh generation process, and properties such as surface curvature can be calculated to a higher accuracy than that contained in the polygonal mesh.

CFD Applications

The underlying implicit representation of the geometry is intentionally robust to the type of input that can be addressed. In fact, any explicit format used for representing geometry can be converted into a set of surface points.

If the segmentation step can resolve an object sufficiently well and an adequate flow solver is coupled, CFD may be applied to a broader and more realistic range of problems. Given medical imaging of blood flow through different channels in the body, CFD could yield information on how the body transports material, quantify the harm of different types of blockages, and simulate the effects of treatments [Dyedov et al., 2009], [Wang, 2001]. Given sonar imaging of water distribution piping, CFD could determine whether a blockage is significant enough to require removal. For clandestine intelligence, performance estimates of foreign aircraft and missiles could be simulated from representations of surfaces found by laser range-finder depth profiles.

Sequential Sizing

The uniform maximum leaf size was a simple choice. We can also adapt the size of the voxels to local feature sizes (like curvature). This would likely require creating some form of a sequence of meshes in order to measure the relevant geometric properties. However, tight control of voxel size is not possible if subdivision is performed by the regular halving of quadtrees or octrees.

Smarter Correspondences

Some applications may not accept the hanging nodes that are a product of each node on a contour corresponding to a node on a previous contour. While remeshing of these polygons is a practical solution, we can also resolve this within our framework.

A solution is to make the correspondences based on the local curvature relative to the direction of propagation. For convex regions, each node comes from a node on the previous level. For concave regions, each node goes to a node on the next level. This approach does require the inclusion of edges in the merged regions to ensure that no edges overlap in the mesh.

Parallelization

While several works comment that storing the full correspondences is prohibitive, we have not found that to be the case. If it is later found to be so, we can employ parallelization to ease memory requirements. There are a number of opportunities to obtain speedups or lessen memory requirements.

For a shared memory system, contouring features and level-to-level correspondences can be implemented as parallel multithreaded algorithms. For a distributed memory architecture, we can use a modified tree where each processor stores its region of the hierarchy and possibly the leaves outside of its boundary. Every processor can then perform operations within its stored domain.

Geometry Definition

The goal of defining the geometry as a point cloud for our surface-oriented meshing algorithm was to have a single representation of all formats. We then represent this point cloud implicitly. It is the implicit representation that we depend on, and this transformation to a point cloud may lose information such as connectivity or interior and exterior orientations of the domain. An even more flexible approach would be to calculate the CDF from the given geometry definition. For example, we could calculate the closest distance to all line segments defining a parametric surface and store a reference to the line segment as the correspondence information. This added capability may resolve some of the previously discussed issues that involve dense surface requirements and orientation in the domain.

REFERENCES

F. Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991. xii, 35

I. Babuska and A.K. Aziz. On the angle condition in the finite element method. *SIAM Journal on Numerical Analysis*, pages 214–226, 1976. 35

G. Barequet and S. Kumar. Repairing cad models. In *Visualization'97., Proceedings*, pages 363–370. IEEE, 1997. 13

M.W. Beall, J. Walsh, and M.S. Shephard. Accessing CAD geometry for mesh generation. In *12th International Meshing Roundtable*, pages 31–42. Citeseer, 2003. 101

J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5 (4):341–355, 1988. xi, 22, 23

J.D. Boissonnat and S. Oudot. Provably good sampling and meshing of surfaces. *Graphical Models*, 67(5):405–451, 2005. 43

L. Branets and GF Carey. A local cell quality metric and variational grid smoothing algorithm. *Engineering with Computers*, 21(1):19–28, 2005. 53

X. Bresson, S. Esedoglu, P. Vandergheynst, J.P. Thiran, and S. Osher. Fast global minimization of the active contour/snake model. *Journal of Mathematical Imaging and Vision*, 28(2):151–167, 2007. 22

V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *International journal of computer vision*, 22(1):61–79, 1997. 20

T.F. Chan and L.A. Vese. Active contours without edges. *IEEE Transactions on image processing*, 10(2):266–277, 2001. xi, 20, 21

C.C. Chang and L.L. Wang. A Fast Multilevel Thresholding Method Based on Lowpass and Highpass Filtering. *Pattern Recognition Letters*, 18(14):1469–1478, 1997. 19

H.D. Chang, X.H. Jiang, Y. Sun, and J. Wang. Color Image Segmentation: Advances and Prospects. *Pattern Recognition*, 34(12):2259–2281, 2001. 18

J. Chaussard, M. Couprie, and H. Talbot. A discrete λ-medial axis. In *Proceedings of the 15th IAPR international conference on Discrete geometry for computer imagery*, pages 421–433. Springer-Verlag, 2009. 100

E.V. Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces. *Institute for High Energy Physics, Moscow, Russia, Report CN/95-17*, 1995. 42

W.J. Coirier, K. Powell, et al. Solution-adaptive cartesian cell approach for viscous and inviscid flows. In *AIAA J.* Citeseer, 1996. 25

D. Colombo and P. Massin. Fast and robust level set update for 3d non-planar x-fem crack propagation modelling. *Computer Methods in Applied Mechanics and Engineering*, 2011. xii, 38, 39

O. Cuisenaire. Distance transformations: fast algorithms and applications to medical image processing. *Laboratoire de Telocommunications et Teledetection*, 1999. 33

WN Dawes, SA Harvey, S. Fellows, CF Favaretto, and A. Vellivelli. Viscous layer meshes from level sets on cartesian meshes. In *45th AIAA Aerospace Sciences Meeting & Exhibit*, pages 8–11, 2007. xi, 25, 26, 101

ND Domel and SL Karman. Splitflow: Progress in 3d cfd with cartesian omni-tree grids for complex geometries. *AIAA Paper*, 1006, 2000. 24

H. Dong, C. Koehler, ZX Liang, H. Wan, and Z. Gaston. An integrated analysis of a dragonfly in free flight. In *40th AIAA Fluid Dynamics Conference and Exhibit, AIAA2010-4390*, 2010. xi, 6, 12

C.T. Druyor Jr. *An adaptive hybrid mesh generation method for complex geometries*. PhD thesis, The University of Tennessee at Chattanooga, 2012. xi, 26, 27

V. Dyedov, D.R. Einstein, X. Jiao, A.P. Kuprat, J.P. Carson, and F. del Pin. Variational generation of prismatic boundary-layer meshes for biomedical computing. *International journal for numerical methods in engineering*, 79(8):907–945, 2009. 102

D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics*, 183(1):83–116, 2002. 40

L. Freitag and C. Ollivier-Gooch. The effect of mesh quality on solution efficiency. In *6th International Meshing Roundtable*. Sandia National Laboratories, 1997. 3, 35

PJ Frey and H. Borouchaki. Surface meshing using a geometric error estimate. *International journal for numerical methods in engineering*, 58(2):227–245, 2003. 44

M. Freytag, V. Shapiro, and I. Tsukanov. Field modeling with sampled distances. *Computer-Aided Design*, 38(2):87–100, 2006. xii, 29, 30

I. Fried. Condition of finite element matrices generated from nonuniform meshes. *Aiaa Journal*, 10:219–221, 1972. 35

S.F. Frisken, R.N. Perry, A.P. Rockwood, and T.R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 249–254. ACM Press/Addison-Wesley Publishing Co., 2000. xii, 33, 34

R.V. Garimella and M.S. Shephard. Boundary layer meshing for viscous flows in complex domains. In *7th Int. Meshing Roundtable*, pages 107–118. Citeseer, 1998. xii, 38

W.J. Gordon and C.A. Hall. Transfinite element methods: blending-function interpolation over arbitrary curved element domains. *Numerische Mathematik*, 21(2):109–129, 1973. 41

R.M. Haralick, S.R. Sternberg, and X. Zhuang. Image Analysis Using Mathematical Morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4): 532–550, 1987. 14

K. Haris, S.N. Efstratiadis, N. Maglaveras, and A.K. Katsaggelos. Hybrid Image Segmentation Using Watersheds and Fast Region Merging. *IEEE Transactions on Image Processing*, 7(12):1684–1699, 1998. 16

Y. Ito and K. Nakahashi. Unstructured mesh generation for viscous flow computations. In *Proceedings of the 11th International Meshing Roundtable*, pages 367–377. Citeseer, 2002. 36

D.C. Ives. Conformal grid generation. *Applied Mathematics and Computation*, 10:107–135, 1982. 42

X. Jiao. Face offsetting: A unified approach for explicit moving interfaces. *Journal of Computational Physics*, 220(2):612–625, 2007. 37

M.W. Jones et al. 3D distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics*, pages 581–599, 2006. 27

T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. *ACM Transactions on Graphics (TOG)*, 21(3):339–346, 2002. xii, 43, 44

M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988. 20

R.A. Katz and S.M. Pizer. Untangling the blum medial axis transform. *International Journal of Computer Vision*, 55(2):139–153, 2003. xii, 32

JH Keyak, MG Fourkas, JM Meagher, and HB Skinner. Validation of an automated method of three-dimensional finite element modelling of bone. *Journal of Biomedical Engineering*, 15(6):505–509, 1993. 11

P.M. Knupp. Remarks on mesh quality. In *AIAA 46th Aerospace Sciences Meeting and Exhibition*, 2008. 3

F. Kurugollu, B. Sankur, and A.E. Harmanci. Color Image Segmentation Using Histogram Multithresholding and Fusion. *Image and Vision Computing*, 19(13):915–928, 2001. 18

C.L. Lawson. Properties of n-dimensional triangulations. *Computer Aided Geometric Design*, 3(4):231–246, 1986. 35

C.K. Lee and S.P. Wong. A Mathematical Morphological Approach for Segmenting Heavily Noise-Corrupted Images. *Pattern Recognition*, 29(8):1347–1358, 1996. 17

S. Leung and H. Zhao. A grid based particle method for moving interface problems. *Journal of Computational Physics*, 228(8):2993–3024, 2009. 37

G.R. Liu. *Mesh free methods: moving beyond the finite element method*. CRC, 2003. 4

R. Löhner, J. Cebral, O. Soto, P. Yim, and J.E. Burgess. Applications of patient-specific cfd in medicine and life sciences. *International journal for numerical methods in fluids*, 43 (6-7):637–650, 2003. 11

W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM Siggraph Computer Graphics*, volume 21, pages 163–169. ACM, 1987. xii, 42, 43

L. Maignan and F. Gruau. Integer gradient for cellular automata: Principle and examples. In *Self-Adaptive and Self-Organizing Systems Workshops, 2008. SASOW 2008. Second IEEE International Conference on*, pages 321–325. IEEE, 2008. 35

P. Maragos and R.W. Schafer. Morphological systems for multidimensional signal processing. *Proceedings of the IEEE*, 78(4):690–710, 1990. xi, 15, 16

MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2011. 7

J. McCleary. *Geometry from a differentiable viewpoint*. Cambridge Univ Pr, 1994. 29

JA Moore, DA Steinman, DW Holdsworth, and CR Ethier. Accuracy of computational hemodynamics in complex arterial geometries reconstructed from magnetic resonance imaging. *Annals of biomedical engineering*, 27(1):32–41, 1999. xi, 6

D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Comm. Pure Appl. Math*, 42(5):577–685, 1989. 21

K. Nallaperumal, K. Krishnaveni, J. Varghese, S. Saudia, RK Selvakumar, R. Subban, and JJ Ranjani. Fuzzy Optimal Thresholded Multiscale Morphological Segmentation of Digital Images. In *Wireless and Optical Communications Networks, 2006 IFIP International Conference on*, page 5, 2006. 17

B.K. Natarajan. On generating topologically consistent isosurfaces from uniform samples. *The Visual Computer*, 11(1):52–62, 1994. 43

E. Noether. Invariante Variationsprobleme. *Nachr. v. d. Ges. d. Wiss. zu Gttingen, Math-phys. Klasse*, pages 235–257, 1918. 2

S. Osher and J.A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988. 21, 38

S.J. Owen. A survey of unstructured mesh generation technology. In *7th International Meshing Roundtable*, volume 3. Citeseer, 1998. 4

S.J. Owen. An introduction to mesh generation algorithms. International Meshing Roundtable, 2005. xi, 4

I. Pivkin, E. Hueso, R. Weinstein, D. Laidlaw, S. Swartz, and G. Karniadakis. Simulation and visualization of air flow around bat wings during flight. *Computational Science–ICCS 2005*, pages 15–36, 2005. 12

S.M. Pizer, C.A. Burbeck, J.M. Coggins, D.S. Fritsch, and B.S. Morse. Object shape before boundary shape: scale-space medial axes. *Journal of Mathematical Imaging and Vision*, 4(3):303–313, 1994. 31

Pointwise. *V17.0 Release 2 Candidate 2*. Pointwise, Inc., Fort Worth, Texas, 2012. 6

S. Prakash and C.R. Ethier. Requirements for mesh resolution in 3d computational hemodynamics. *Journal of biomechanical engineering*, 123:134, 2001. 11

F. Remondino and S. El-Hakim. Image-based 3d modelling: A review. *The Photogrammetric Record*, 21(115):269–291, 2006. 13

R.M. Rodríguez, T.E.M. Alarcón, and I.B. Castellanos. A Strategy for Reduction of Noise in Segmented Images. Its Use in the Study of Angiogenesis. *Journal of Intelligent and Robotic Systems*, 33(1):99–112, 2002. 17

H. Samet. Neighbor finding techniques for images represented by quadtrees. *Computer Graphics and Image Processing*, 18(1):37–57, 1982. 49

R. Satherley and M.W. Jones. Hybrid distance field computation for volumetric objects. In *Proceedings International Workshop on Volume Graphics 2001*, pages 121–133, 2001. 27

J. Schöberl. Netgen an advancing front 2d/3d-mesh generator based on abstract rules. *Computing and visualization in science*, 1(1):41–52, 1997. 44

J. Schreiner, CE Scheiclegger, and C.T. Silva. High-quality extraction of isosurfaces from regular and irregular grids. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):1205–1212, 2006. 44

J. Serra. Introduction to Mathematical Morphology. *Computer Vision, Graphics, and Image Processing*, 35(3):283–305, 1986. 14

JA Sethian. Curvature flow and entropy conditions applied to grid generation. *Journal of Computational Physics*, 115(2):440–454, 1994. xii, 6, 7, 39, 40

J.A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences of the United States of America*, 93 (4):1591, 1996. 39

M. Sezgin and B. Sankur. Survey Over Image Thresholding Techniques and Quantitative Performance Evaluation. *Journal of Electronic Imaging*, 13:146, 2004. 18

M.S. Shephard and M.K. Georges. Automatic three-dimensional mesh generation by the finite octree technique. *International Journal for Numerical methods in engineering*, 32 (4):709–749, 1991. 12

J.R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational geometry*, 22(1):21–74, 2002a. 35

J.R. Shewchuk. What is a good linear finite element? interpolation, conditioning, anisotropy, and quality measures. *University of California at Berkeley*, 2002b. 1

M. Staten, S. Owen, and T. Blacker. Unconstrained paving & plastering: A new idea for all hexahedral mesh generation. In *Proceedings of the 14th International Meshing Roundtable*, pages 399–416. Springer, 2005. xii, 44, 45

D.A. Steinman. Image-based computational fluid dynamics modeling in realistic arterial geometries. *Annals of Biomedical Engineering*, 30(4):483–497, 2002. 11, 13

S. Sural, G. Qian, and S. Pramanik. Segmentation and Histogram Generation Using the HSV Color Space for Image Retrieval. In *International Conference on Image Processing (ICIP)*, pages 589–592, 2002. 18

I.E. Sutherland and G.W. Hodgman. Reentrant polygon clipping. *Communications of the ACM*, 17(1):32–42, 1974. 25

J. Teran, N. Molino, R. Fedkiw, and R. Bridson. Adaptive physics based tetrahedral mesh generation using level sets. *Engineering with Computers*, 21(1):2–18, 2005. 42

J.F. Thompson, ZUA Warsi, and C.W. Mastin. *Numerical grid generation: foundations and applications*. North-holland Amsterdam, 1985. 3, 4

P.G. Tucker, C.L. Rumsey, P.R. Spalart, R.E. Bartels, and R.T. Biedron. Computations of wall distances based on differential equations. *AIAA journal*, 43(3):539–549, 2005. 101

L. Vincent. Morphological Grayscale Reconstruction in Image Analysis: Applications and Efficient Algorithms. *IEEE Transactions on Image Processing*, 2(2):176–201, 1993. 15

J. Vleugels and M. Overmars. Approximating voronoi diagrams of convex sites in any dimension. *International Journal of Computational Geometry and Applications*, 8(2):201–222, 1998. 36

K.C.Y. Wang. *Level set methods for computational prototyping with application to hemodynamic modeling*. PhD thesis, stanford university, 2001. xi, 11, 12, 102

Y. Wang, F. Guibault, R. Camarero, and K.F. Tchon. A parameterization transporting surface offset construction method based on the Eikonal equation. In *Seventeenth AIAA CFD Conference*. Citeseer, 2005. 38

H. Xia and P.G. Tucker. Distance solutions for medial axis transform. *Proceedings of the 18th International Meshing Roundtable*, pages 247–265, 2009. 35

C. Xu and J.L. Prince. Snakes, shapes, and gradient vector flow. *IEEE transactions on image processing*, 7(3):359–369, 1998. 20

J. Yang, N. Sakamoto, Z. Wang, P. Carrica, and F. Stern. Two phase level-set/immersed-boundary cartesian grid method for ship hydrodynamics. In *Proc. Nineth Int. Conf. Numer. Ship Hydrodynamics, Ann Arbor, MI*, 2007. 24

T.S. Yoo. *Insight into images: principles and practice for segmentation, registration, and image analysis*. AK Peters, Ltd., 2004. 10

VITA

Nicholas Szapiro was born in Colorado in 1987 to parents from Argentina. He obtained Bachelor's degrees in engineering and math from Swarthmore College. He continued his studies at the SimCenter to learn the computational tools used to solve cool problems. He will continue to apply these studies at the University of Oklahoma in meteorology.