Wavelet-based multiresolution method for surface reconstruction.

Shobha Iyer
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/4552
Wavelet Based Multiresolution Method For Surface Reconstruction

by

Shobha Iyer

A Thesis
Submitted to the Faculty of Graduate Studies and Research through the School of Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Science at the University of Windsor

Windsor, Ontario, Canada

2003
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

Canada
Shobha Iyer, 2003

© All Rights Reserved
ABSTRACT

One of the areas of research that has achieved tremendous importance over the recent past is the problem of reconstructing a surface of a solid polyhedral object. This problem has found vast applications in human anatomy, automobile design, medical imaging and therapy, etc.

One of the foremost techniques used to solve this problem included constructing the solid from a series of slices parallel to each other. This slice data was obtained by taking horizontal cross-sections passing through the interior of the solid like slicing an apple through a number of thin horizontal planes.

The piecewise linear interpolation technique was one of the traditional algorithms based on the above slicing technique. One of the problems with this algorithm was that the running time of the algorithm was large and it required fine-tuning of certain external parameters.

In the 1980s, multiresolution methods using wavelets emerged as an alternative to solve problems then solved by the windowed Fourier transforms. Wavelets were basically useful for sparse representations of various functions.

Our thesis aims at implementing the multiresolution tiling algorithm using wavelets for the purpose of surface reconstruction. This technique also uses data from slices taken through the surface being reconstructed. In this thesis, we have discussed the pros and cons of using either algorithm and our implementation provides a stable workbench for surface reconstruction using wavelets to enable further research. Experiments show that the running time of the multiresolution tiling algorithm is much lesser than that of the piecewise linear technique. Experiments also show that both algorithms show good resemblances with that of the original.
Dedicated to My Dearly Beloved Husband, Sriram
ACKNOWLEDGEMENTS

I would like to thank a number of people for their immense contributions to the completion of my work.

First and foremost, I would like to express my profound appreciation to my Supervisor, Dr. Asish Mukopadhyay for all the valuable motivation, guidance and support throughout the time of the thesis work.

I would also like to thank my Internal Reader, Dr. Ngom, my External Reader, Dr. Seth and the thesis committee Chair, Dr. Luis Rueda for their time and efforts. Thank you very much.

To this end, I want to express my ineffable gratitude to my husband, Mr. Sriram Padi for all his financial and moral support, guidance, courage and assistance. Without him, this dream could not have come through. He has always been there when I needed the help and understanding that I couldn’t find anywhere else. I give my special thanks to my father, Dr. G. Venkatachalam who has strongly motivated me to take up a professional career and contributed greatly to the actualization of this goal. My deep appreciation for my mother Mrs. Geeta for the ever-available cushion of love, her prayers, encouragements and moral supports.

Finally, I would like to thank all members of my husband’s family, especially my parents-in-law Mr. and Mrs. Balaraman for all their prayers and supports. Thanks for giving me an opportunity to do what I wanted.
2.7.1 Haar Basis
2.7.2 Notion of Haar Basis Functions
2.7.3 Mathematical Representation of Haar Basis Functions
2.7.4 Wavelet Space Representation
2.7.5 Representation Using Matrices
2.7.6 Single Knot Wavelets
2.7.7 Filter Bank Method
2.7.8 Notion of Orthogonality of Wavelets
2.7.9 Overview of Multiresolution Tiling Algorithm

3.0 PIECE-WISE LINEAR INTERPOLATION ALGORITHM
3.1 Preprocessing
3.2 Contour Matching
3.3 Stitching the Matches
3.4 Forming Clefts
3.5 Bridging the Clefts
3.6 Triangulation of 3D Clefts

4.0 MULTiresolution TILING ALGORITHM
4.1 Contour Decomposition
4.2 Base Case Tiling
4.3 Filter Bank Reconstruction

5.0 EXPERIMENTAL EVALUATION
5.1 Implementation Environments
5.2 Comparison in Terms of Complexity of Algorithms
  5.2.1 Piecewise Linear Interpolation Technique
  5.2.2 Multiresolution Technique
5.3 Comparison in Terms of Number of Vertices
5.4 Comparison in Terms of Number of Triangles 66
5.5 Comparison in Terms of Running Times 70
5.6 Comparisons with the Optimal Algorithm 71
5.7 Conclusion from All Experiments 72

6.0 FUTURE WORK AND CONCLUSIONS 74

REFERENCES 75
APPENDIX A REVIEW OF COMPUTATIONAL GEOMETRY TERMS 78
APPENDIX B SAMPLE SCREEN SHOTS OF ALGORITHMS 84
VITA AUCTORIS 91
LIST OF TABLES

Table 5.1  Comparison between Methods of Interpolation  64
Table 5.2  Results of Experiments on Interpolation  65
Table 5.3  Comparison between Methods of Triangulation  66
Table 5.4  Triangulations for Piece-wise Linear Algorithm  68
Table 5.5  Triangulations for Multi-resolution Algorithm  68
Table 5.6  Comparisons between Running Times  70
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1</td>
<td>Formation of triangular tiles</td>
<td>11</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>An alpha shape with a bad (a) and a good parameter of $\alpha$</td>
<td>14</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>r-regular shape</td>
<td>16</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>Angle formed by Delaunay disks</td>
<td>17</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>Original Image</td>
<td>22</td>
</tr>
<tr>
<td>Figure 2.6</td>
<td>256 X 128 pixels of original image</td>
<td>24</td>
</tr>
<tr>
<td>Figure 2.7</td>
<td>(a) 128 X 256 image (b) 128 X 128 image</td>
<td>24</td>
</tr>
<tr>
<td>Figure 2.8</td>
<td>(a) Box basis for $V^0$ and (b) for $V^1$</td>
<td>26</td>
</tr>
<tr>
<td>Figure 2.9</td>
<td>(a) Haar wavelets for $W^0$ and (b) for $W^1$</td>
<td>27</td>
</tr>
<tr>
<td>Figure 2.10</td>
<td>(a) Scaling functions for $V^2$ and (b) for $V^1$ and (c) Lazy wavelets for $W^1$</td>
<td>30</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>A single slice with multiple contours projected onto x-y plane</td>
<td>38</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>(a) Counter-clockwise and (b) Clockwise orientation of contour</td>
<td>39</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Original slices of the cone after pre-processing</td>
<td>40</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Contour matching</td>
<td>41</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>(a) Case when triangle PQU is formed (b) when triangle PQV is formed</td>
<td>43</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>Matched contour portions of slices stitched</td>
<td>44</td>
</tr>
<tr>
<td>Figure 3.7</td>
<td>(a) Matched and (b) unmatched portions (cleft) of contours</td>
<td>45</td>
</tr>
<tr>
<td>Figure 3.8</td>
<td>Bridge between clefts</td>
<td>47</td>
</tr>
<tr>
<td>Figure 3.9</td>
<td>A polygonal chain</td>
<td>49</td>
</tr>
<tr>
<td>Figure 3.10</td>
<td>Final reconstructed surface</td>
<td>50</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Original contours</td>
<td>55</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Decomposed Contours</td>
<td>55</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>Base Case Triangulation</td>
<td>56</td>
</tr>
</tbody>
</table>
1.0 INTRODUCTION

1.1 Problem of Surface Reconstruction

CAD (Computer-aided design) and CAM (Computer-aided manufacturing) techniques are being widely used in different industries to simulate actual physical objects from models. However the converse is quite rare. The problem of reverse engineering of a simulated digital model from an existing physical object is still obscure and evolving. A three-dimensional object has a number of properties including its shape, color, dimensions, weight, density and so on of which shape is an important one. The problem of surface reconstruction is essentially to do with the creation of the 3D shape of an existing object [BC00].

The primary aim of surface reconstruction is to build the solid topologically closely, as far as possible, using information from the surface itself or its adjacent ones [BBX95]. Consider a topographic surface, for example. To start with, the surface is not available to us per se. We have only the terrain elevation and spatial position data at several known points. Incidentally, this information is nowadays available to us in the form of scattered random sample points. The points are those with maximum information content from the surface to be reconstructed. The primary aim is to reconstruct the terrain surface from the sample points.

For example, in order to build a digital terrain model; point samples could be as diverse as spot ground elevations, groundwater levels in wells, observations in soil profiles and rain gauge readings at meteorological stations. They could also be performance indices of marketing outlets or community centers catering to certain administrative regions such as counties. The administrative regions are spatial units, which could be regular grids or irregular regions. In the latter case, point sample values are usually first interpolated to get values at regular grid-nodes. Another example of a model for the purpose would be that of the triangulated irregular network model (TINs), that is one kind of data model where a mesh of triangles whose vertices are measured at carefully located topographic points, approximates the continuous surface of the land.
Our eyes because of subconscious childhood training can interpret the surface in the model so built. This digital terrain model itself is built using some algorithms that are mostly based on computational geometry concepts.

In the case of other solids, many recent advances in the field of laser technology help to generate a cluster of sample points of known density.

1.2 Uses of Surface Reconstruction

The problem of surface reconstruction first arose due to medical diagnosis and therapy. In order to treat a patient with a damaged organ, it is necessary to take horizontal cross-sections of its surrounding organs such as bones [BS94]. Using this, it may be possible to guess the extent of damage by interpolating the shape of the damaged solid organ.

Not only in medical research, but also in CAD-CAM designs as mentioned above or in equipments, machinery design and analysis, solutions to this problem has proved useful. It could be used for morphing and animation in movies. It could be used for design of automobiles and ships. The required parts are first designed on the computer and well-developed algorithms are used to construct the three-dimensional structure from its parts and visualize the final product before actually manufacturing.

1.3 Evolution of Surface Reconstruction

Computational geometry is a field of science dealing with geometric problems. It basically aims at developing various algorithms defining and manipulating geometric objects and shapes. Usually the input to these algorithms is some geometric object such as a set of points or line segments. The output is a new geometric object like the convex hull or a geometric observation concluded such as whether the lines intersect. Most of the algorithms are developed with a polygon as the basis. A polygon in turn can be divided into a number of triangles. Triangles are found to give a more realistic depth and three-
dimensional effect. This is the basic principle used for most surface reconstruction algorithms as each triangle is on a different surface.

Researchers have attacked this problem over decades proposing their own heuristics. Some provide solutions pertaining to particular applications. Some have made assumptions in their algorithms proposed or in the available input data itself or still others have been stringently specific about the types of surfaces to be tested on in order to guarantee reliable reconstruction.

Whatever be the solution, some allowance needs to be given to the fact that the input data used may be erroneous or ambiguous. This is not unusual, since in practice, the surface may have some features of unpredictable topological types, like hidden edges or sharp features such as jagged edges with slope discontinuities and sharp corners. We say that the input data is usually "noisy". Consequently, the output so obtained may not be 100% efficient and robust. Hence, the reconstructed surface may only be approximately close to the original one.

The most important requirement is that the reconstructed surface be topologically close to the original one. How close it would be is again difficult to state in general; it depends on the density and quality of samples used. Not just the data, but also the use of suitable algorithms with minimal assumptions affects the quality of surface reconstructed. The two-dimensional version of the same problem is called curve reconstruction. Well-studied algorithms for reconstructing both open and closed curves exist.

1.4 Historical Perspective

One of the earliest methods used to solve this problem is to reconstruct solids or surfaces using horizontal cross-sections taken through the solid. These sections were called polygonal slices or contours [BS94, FKU77]. Each section was a thin slice. It can be thought of as lying in the same plane passing through the entire interior of the solid.
The outer edge for each slice was interpolated. This interpolation meant building the surface of each slice using simpler geometric objects like triangles. All the slices were joined together based on certain criteria used to determine their orientation. This generated the entire solid approximating the original one. However, this method was not sophisticated enough, since many loopholes were detected in this algorithm, mainly in the form of many possible morphological variations of the surfaces. It was indeed difficult to develop a generic algorithm spanning over most of the solids and several exceptions existed. The field of surface reconstruction had thus become extinct for a few years.

A few years later, the field of computational geometry emerged. The problem of surface reconstruction started being viewed in this new perspective. To date, most of the solutions proposed are based on computational geometry algorithms.

Most theorems related to shape of geometrical objects were those of convex hulls. (Convex hulls are described in detail in Appendix A). Convex hulls seemed to capture the shape of the underlying solids. However, the shape of the objects defined by the hull was often "crude". At around the same time, theorems on alpha-shapes started hitting the research arena as these seemed to configure the shapes of objects too [EKS83]. Later, it was found that the choice of parameter alpha really varied from solid to solid and hence, the upper bound of the algorithm was again not deterministic.

Recently, Dominique Attali [Att97] rediscovered the field of surface reconstruction that had otherwise become dormant. The course of the problem changed its direction completely. She defined the necessity to reconstruct a solid from sample data points rather than cross-sectional slices. Clearly, without any condition on the samples needed, the quality of surface so reconstructed would still be unknown. Hence, her ingenious algorithm laid down a condition between the number of samples required and the shape of the object to bring about an acceptable reconstruction.

Her research led to the finding that it is possible to build a network or mesh of triangles approximating the original solid. The mesh of triangles was formed by joining
specific sample points, with each triangle lying in a plane. A major advantage of this triangulation was that areas with less geographic variations could be depicted using fewer triangles and vice versa. The mesh is formed from triangles satisfying the Delaunay criterion and the generation of the mesh of triangles is thus called Delaunay triangulation [Aur91, For95]. Her solution was well established for 2D, but showed a snag when extended to 3D.

Curve reconstruction algorithms started developing as an offshoot of this problem. Both open and closed curve reconstruction problems are being studied. The main criterion for curve reconstruction is that only adjacent samples of the curve be joined. To ensure this, the distance between adjoining samples should not be greater than a constant multiple of the distance of the sample to its nearest point on the medial axis. (Medial axis is explained in Appendix A) This is because the medial axis of a curve approximates the Voronoi diagram. The medial axis thus divides the curve into two halves. We cannot join samples lying on opposite sides of the medial axis.

So far, a large number of algorithms have been developed using computational geometry, which are able to reconstruct the surfaces and curves fairly well. Some of the important issues that are of concern in developing algorithms for the purpose include use of minimal sample points, efficiency, how geometrically and topologically close the surfaces are with respect to the original one and how sensitive the algorithm is with respect to noisy data.

1.5 Introduction to Multiresolution Methods

Most of the time, volume of data used as input for the purpose of surface reconstruction is very high. Also, the data may be expressed as functions of two or more parameters. Before analyzing the data, it is necessary to represent data in a suitable format to enable further processing.
Representation of functions using superposition has existed since the early 1800's, when Joseph Fourier proved that functions could be expressed by superposing equivalent sines and cosine functions. This technique helped signal representations and thus analysis in a much better fashion depending on its frequency over a specific time period. Fourier transforms were extensively used in the field of mathematics and trigonometry to solve complex differential equations, signal processing, digital communication and so on [Gra95].

As with any other technique, this technique too proved to have inherent serious drawbacks with the passage of time. For example, a highly varying signal over a very short interval of time had too much Fourier series information for proper representation. For such locally varying signals, multiresolution methods using wavelets proved to have an edge over Fourier methods. Wavelets are effective in representing local aspects of a signal efficiently. Wavelets are being used extensively for the purpose of signal analysis for more than two decades. Wavelets allow a time series to be viewed in multiple resolutions.

Fourier and wavelet analyses have strong resemblances with each other. A large collection of problems is benefited by multiresolution analysis using wavelets. Wavelets represent functions at multiple levels of detail.

The primary idea behind wavelet analysis is based on representing functions at different levels from coarsest to finest. The initial finest function is discretized into a coarser function at low resolution. Obviously, some amount of information is lost in this process. This lost information is captured as detail coefficients that are necessary during the reverse process of recapturing the original function from the coarser one. Most of the wavelet coefficients are small in value and number. The few larger coefficients contain maximum information about conversion between various levels [LDW97]. Since all the levels of detail are still for the same signal or function, a function could be represented in terms of linear combinations of coefficients of all the levels.
Multiresolution analysis using wavelets was being used in a number of fields. These methods are more related to computer graphics applications the foremost being image processing and compression. Wavelets are useful for creating lower-resolution images having detail coefficients of very low value. Image compression aims at discarding a few or all of the detail coefficients depending on the level of accuracy desired. To be able to capture the lost information again, the process of reconstruction using the lowest level of coefficients is an easy task assuring a good level of accuracy. Image compression using wavelets has been studied by Wickerhauser [Wic92, Wic94]. These decomposition and reconstruction methods of multiresolution analysis are discussed in detail in Chapter 2.

1.6 Wavelets for Surface Reconstruction

Of the several sub-problems associated with reconstructing a surface, the main problem is that of tiling. The difficulty in deciding on a suitable tiling algorithm is its upper bound and that it must be optimal with respect to the goal function.

The emphasis has traditionally been on the fact that the solution to a smaller problem would give rise to a larger one with just minimal extra steps. This is exactly the technique used in dynamic programming algorithms. Dynamic Programming refers to the large class of algorithms where a problem is broken down into sub-problems such that optimal solutions to sub-problems are also known. It is always better to apply the known technique incrementally to larger size of problems. The conditions and solutions are extended incrementally without having to alter much of the already computed optimal solutions to sub-problems. Eventually, the conditions apply to all of the data and with minimal changes, and if no given input of the problem has remained untreated, the smaller solution is the solution to the larger and complete problem.

In [FKU77] the number of steps for finding an optimal tiling solution is bounded by \((\lceil \log_2 m \rceil (2mn + m) + 3mn + m^2)\) where \(m\) and \(n\) are the number of vertices of the two contours. The complexity associated by tiling adjacent slices is the same as that of the
entire solid. The time required is extremely large considering that the actual data may have thousands of point sets. It is thus necessary to turn to non-optimizing algorithms.

One of the solutions attempted to solve the problem of tiling in order to reduce the complexity is that of using multi-resolution analysis. The lowest base case resolution is small enough to reduce the complexity to a great deal. Using successive levels of resolutions, the base case is merely improved and a good deal of optimization is achieved by few extra steps and/or user intervention.

A correct and perfect tiling is undefined and it depends on the underlying nature of the solid from which the contour information is derived. Both multi-resolution tiling and optimization tiling algorithm give a great deal of accuracy and are equally adaptive to local indentations. In addition, the multi-resolution methods can incorporate data compression by leaving out coefficients that are of very small magnitudes. The overall shapes of the reconstructed solids are close to the original shape and are adequate for many purposes. The lesser storage space and time required clearly outweigh the optimization algorithms.

1.7 Thesis Motivation

"Seeing is believing" is a simple core paradigm of success. Although many algorithms are being implemented and studied, very few are available in the public domain. This report mainly aims at implementing the multiresolution tiling algorithm to be able to compare the outputs so obtained with those of the optimization algorithm and also expose the constraints and the shortcomings of the algorithm, if any. The implementation done would serve as a good experimental workbench for the surface reconstruction problem.

1.8 Thesis Idea Proposed

The problem that we propose as part of this thesis work is:

"Compare the outputs of the algorithms of the two methods of surface reconstruction – one proposed by Micha Sharir and Gill Barequet using piece-wise linear
interpolation and polygonal slices [BS94] and the other using multi-resolution analysis proposed by David Meyers [Mey92, Mey94].

Some of the important features of these methods and algorithms are: both use data from cross-sectional contours; various complex data samples, including actual medical imaging data of human organs have been used in them; data may have a series of horizontal planar slices with each slice consisting of non-crossing, possibly nested curves called contours. The heuristic proposed by [BS94] has been implemented by [GKDM96]. In practice, each contour is given as a separate circular sequence of points along it. Each point is represented in terms of its Cartesian (x,y,z) co-ordinates.

The software package developed by [GKDM96] has been written in C and has X-Windows based GUI using X toolkit, Motif and DEC-PHIGS. The package is highly user-friendly enabling viewing the object from all possible orientations. The user can control his view of the object by applying any geometric transformation such as rotation, scaling or translation on the data. Each slice of the solid can be viewed independently. User has control over parameters such as discretization parameter, voting threshold, etc. that affect the output significantly. Each phase of the algorithm can be executed separately.

The ideas we wish to include as part of our thesis work include:

- Developing an algorithm using multi-resolution analysis concept, but using the same GUI environment and C of [GKDM96], which is based on polygonal slices. However, the implementation would be made suitable for OpenGL, rather than just DEC-PHIGS.
- Incorporating in this implementation all the interactive GUI features, such as scaling, rotation and translation, mentioned in the other implementation.
- Constructing a solid model whose cross-sections along the given planes coincide with the input slices. One major difference would be to assume that every slice has just one contour.
- Implementing the algorithm to the problem of reconstructing the solid model.
• Comparing the outputs of the two algorithms – one using piecewise linear interpolation technique and the other using multi-resolution analysis by viewing the reconstructed solid models. This comparison would be in terms of
  (a) The optimality of the tilings produced - with respect to some quantitative goal function.
  (b) Alternatively, checking which of the contour shapes produced more closely resemble the original detail.
  (c) The upper bounds of the speeds of execution and the storage space.
• We shall try to overcome the drawbacks of the optimizing algorithm proposed by Fuchs et al [FKU77] such as correctly joining the corresponding local edges of organs from a pair of contours.
• Lastly, we shall try to give an estimate on the quantity of samples required to produce efficient tiling.

1.9 Thesis Outline

The remainder of this thesis report is organized as follows: Chapter 2 describes related work or early research done in the field of surface reconstruction. Chapter 3 describes the piece-wise linear interpolation algorithm in detail. We have given the details of the multi-resolution tiling algorithm in Chapter 4. In chapter 5, experimental results are discussed and conclusions are drawn in Chapter 6.
2.0 RELATED WORK

The problem of surface reconstruction is being studied for more than a decade. The notion that the convex hull of the object gives an idea of the shape of the object was no longer a stable and an acceptable form of the reconstructed solid. Researchers wanted more complex routines to handle a larger number of objects with more accuracy and precision. In this chapter, we shall review some of the algorithms developed and enhanced over the years.

2.1 Surface Reconstruction Using Cross Sectional Contours

Early work done in the 1970-1980 period, a surface was constructed from a set of cross-sectional contours [Fra87, FKL77]. Each solid was intersected by a finite number of specified parallel planes. Each plane intersected the solid in a closed curve. The curve in turn consisted of the sample points assumed to be arranged in counterclockwise direction. The curve between two points is approximately a segment and is called a contour segment ($P_i, P_{i+1}$ in Figure 2.1). All the contour segments of one parallel plane or a curve form a contour. The sequence of contours is used to construct a piecewise planar approximation to the original object surface. Tiling is done between every pair of contours. These tilings bridge the gap between two thin contours. In other words, this process is called triangulation of contours.

![Triangular Tiles](image.png)

Figure 2.1 Formation of triangular tiles

The intersections of the reconstructed surface with the parallel planes are approximately identical to the original contours. The vertices of triangular tiles are the
adjacent points taken two from one sequence and one from the other as shown in the Figure 2.1.

A span for a tile is the line segment connecting one end of the contour segment with a common vertex of the other. For example, for Figure 2.1 the spans are \( P_iQ_j \) and \( P_{i+1}Q_j \). Each tile is of the form \([P_i, P_{i+1}, Q_j]\) or of the form \([Q_j, Q_{j+1}, P_i]\). There are many sets of tiles that could be defined over the points of the two contours. A set of tiles satisfying the following two conditions is said to be an acceptable surface.

(1) A contour segment is used by only one tile.

(2) The left span of some tile should also be a right span of some other tile in the set.

Satisfying the above two conditions is also far too generic. Additional constraints need to be enforced. The solution to the problem is by reducing the representations of these tiles into a reduced graph representation. The vertices of the graph refer to the spans between the point set data of the two contours. The arcs of the graph are incident from the vertex of the left span to the vertex of the right span. An acceptable set of tiles between two contours is defined as the combination of tiles that reduce the cost of the path. The costs are assigned to every arc. A path is chosen such that the sum of the costs of arcs traversed is minimal where no vertex is repeated. Well-known methods exist to deal with finding minimum cost paths in graphs [Joh77, Chr75]. Thus the resulting overall surface consists of sets of tiles with minimal cost paths.

However, problems arose as the algorithm could not handle the case where the plane could have more than one contour. This was extremely necessary as many solids in real world are composed of several contours per plane especially when the solids had holes within them. For example, a slice through the legs of a human organ would be composed of two contours at the lower planes (one for each leg) that would then merge into one as we go up.
2.2 Piecewise Linear Approximation Using Planar Contours

In order to overcome the major drawback of handling more than one contour per plane, a new technique similar to this was proposed by [BS94]. These planes themselves were called slices.

A number of parallel planes (parallel to the x-y plane) through the solid were taken and a solid was to be reconstructed where each cross-section of the plane coincided with the input slices. The algorithm is described in detail in Chapter 3. It could successfully handle all types of data including multiple contours in each slice. Each slice could also have multiple contours nested without any bridging between them. No resemblance between any slices was required. Also acceptable were slices having different geometries or any number of contours. There was no restriction on the xy-projections of any of the contours. Some projections overlapped fully or partially and some did not.

Although the research so far gave satisfactory results for surface reconstruction, there was the necessity to construct the tiling at each cross-section [WC71]. This was not very effective as far as the running time of the algorithm was concerned.

So the problem of reconstruction was seen in a new angle as being able to reconstruct a surface given a set of random scattered data points. These points were totally unorganized. Techniques such as $\alpha$ - shapes and $r$- regular shapes came into existence. The only initial information we have about the surface is through the random scattered points.

2.3 Alpha ($\alpha$) Shapes

Each $\alpha$ shape is a well-defined polytope or a hole, derived from the Delaunay triangulation of the point set, with a parameter $\alpha$; $\alpha$ is used for controlling the desired level of accuracy.
An alpha shape is the generalization of the convex hull of the point set. An alpha shape is a hole or a cavity surrounding the set of points [EKS83]. The parameter $\alpha$ controls the maximum curvature of any polytope. The choice of $\alpha$ may be very tricky. The alpha shapes work well with sample points that are evenly distributed throughout the interior of the object. However, for reconstructing the surfaces, alpha shapes do not serve as a good technique since the value of alpha has to be chosen with great accuracy and should be applicable to the entire surface [ABE98].

We can intuitively think of an $\alpha$ shape as the following. Imagine a huge mass of ice cream making up the surface and containing the points of surface $S$ as "hard" chocolate pieces. Using one of these sphere-shaped ice-cream spoons we carve out all parts of the ice-cream block we can reach without bumping into chocolate pieces, thereby even carving out holes in the inside (e.g. parts not reachable by simply moving the spoon from the outside). We will eventually end up with an object bounded by caps, arcs and points. The value of $\alpha$ denotes the radius of each spherical ball.

![Figure 2.2 An alpha shape with a bad (a) and good (b) choice of parameter $\alpha$](image)

Increasing the value of the parameter $\alpha$ gives a better shape. The worst shape is the convex hull itself, for the value of $\alpha$ close to infinity. As $\alpha$ decreases, the shape
shrinks and cavities are formed. The convex hull is gradually “eaten away” to expose the subtle features of the underlying solid.

Consider Figure 2.2. The alpha shape for a set of points with two different values of alpha is as shown. In Figure 2.2 (a), a sphere is drawn with a large value of $\alpha$ which covers most of the points and thus gives an idea of the “crude” shape of the object. However, the value of $\alpha$ is not large enough to form a convex hull of the object.

However, in Figure 2.2 (b), the spheres are drawn with a well-chosen value of alpha such that they totally “eat away” the convex hull of the shape and include just the original set of points. This is the finest shape that can be achieved using the set of points given. In particular, we find that two spheres are drawn using a constant value of $\alpha$, one creating a hole at the centre and the other creating a semi-hole at the bottom. Hence, the value of $\alpha$ to be chosen, to achieve this level of accuracy is tricky.

2.4 r-regular Shapes

In case of r-regular shapes as proposed by Dominique Attali [Att97], the initial data is a set of sample points with no structure belonging to a surface. Of course, it is necessary to quantify the number of samples to be taken from the surface. If ‘x’ is the number of samples taken from the original surface $O$, then $x$ is said to sample $O$ with a certain sampling density $\varepsilon$. This sampling density or path is such that if we draw a sphere with center as any point on $O$ and radius $\varepsilon$, the sphere would contain at least one sample point $x$. Smaller the sampling path, higher is the number of samples and consequently better is the topological match of the reconstructed surface with that of the original one. If the sampling path tends to zero, the samples are nearly all of the original points of the shape.

The solution to this problem is concerned with building a mesh or network of the surface known as normalized mesh. The normalized mesh is a set of cells. These cells are formed out of the Voronoi diagram of the sample points. By property of the Voronoi
diagram, each Voronoi vertex of the cell would be equidistant from 3 samples. Thus, a
normalized mesh is built up from the Delaunay diagram of the samples.

To further simplify the concept of searching for the normalized mesh, an
assumption is made that the unknown surface is an \( r \)-regular shape. An \( r \)-regular shape in
[Att97] is defined as follows –

"Let \( \text{Bo} \) be the unit ball. A shape \( X \) is said to be \( r \)-regular if it is morphologically open
and closed with respect to a disk of radius \( r > 0 \):

\[
X = (X \oplus r\text{Bo}) \oplus r\text{Bo} = (X \oplus r\text{Bo}) \oplus r\text{Bo} \, .
\]

\[\text{sample point } x\]

\[\text{B (x, 2r)}\]

Figure 2.3 \( r \)-regular shape

Figure 2.3 shows a shape that is \( r \)-regular. An \( r \)-regular shape is such that if we
slide a ball all through the boundary of the surface, the ball would always be tangential to
the surface at each point. For example, the ball \( B(x, 2r) \) defined as the ball with center at
\( x \) and radius \( 2r \) is intersected by the shape exactly through the diameter of the ball. This
larger ball can be split into 2 smaller balls each of radius \( r \). In this case, both would be
tangential to the shape at the sample point, one within and the other outside the shape. Let
us consider the 2D case first and then deal with higher dimensions.
2.4.1 2D Case

It is necessary to construct the normalized mesh by filtering some of the triangles from the Delaunay diagram of the sample points. Find the Delaunay diagram of the set of sample points of the curve as denoted by the dotted lines in Figure 3.7. The external angle formed between two adjacent Delaunay triangles is denoted by $\delta$. The triangles satisfying the condition $\delta \leq$ a certain threshold value $\delta_0$ form a set called $S_{\delta_0}$. Suppose we select the threshold value to be $\pi/2$, the set of Delaunay edges $S_{\pi/2}$ is the normalized mesh. This is achieved when the sampling path $\epsilon < \sin(\pi/8)r$. This is the key result proven in [Att97].

![Delaunay discs](image)

Figure 2.4 Angle formed by Delaunay discs

Since the solution is a subset of the Delaunay diagram satisfying a criterion, there is no need to know the values of $r$ and $\epsilon$ to compute the normalized mesh. Plus the sampling density is much lesser than the original point set data. This algorithm proves to be excellent and the first of its kind for 2D.

2.4.2 3D Case

However, when extended to 3D, their theory fails. This is because the condition misses out many Delaunay spheres. This is because the spheres may intersect the boundary without being tangential. Hence, some more workarounds may be required to complete surfaces in 3D.

However, on the minus side of this algorithm, it is necessary that the points lie close or on the boundary to keep the value of $\delta$ under a threshold. This may not always be practical. So a combination of techniques of available shapes is used to give fair reconstructions.
2.5 Crust

The Crust algorithm can be used for the reconstruction of the surfaces that have arbitrary topology [ABK98]. The input sample points are taken randomly. Given a fairly good sample from a smooth surface, we can get a topologically correct output that is close to the original surface. The greater the sampling density, the closer is the reconstruction to the original surface. The samples are taken in such a way that fewer samples are selected from the featureless areas and the samples are more in detailed areas. The Crust algorithm is based on the three-dimensional Voronoi diagram. The basic principle behind the Crust algorithm is that, given a sufficiently dense sample from a curve, the Voronoi vertices approximate the medial axis of the curve. The Crust algorithm was first defined in 2D and the same idea when extended over 3D had a snag. Hence, a workaround is required to extend the same algorithm on 3D.

This two-step algorithm in case of 2D is as follows -

1. Given a set of sample points S, first the Delaunay triangulation of S is done. Then the set of Voronoi vertices is computed. This set is called V.
2. Then, the Delaunay triangulation of $S \cup V$ is computed. This is the crust of S. The crust is said to approximate the surface.

Thus, every edge belonging to the crust has a circum-circle that contains no point in S plus no point in V. Thus, the crust is the subset of Delaunay triangulation of the input points S. All the vertices of the crust triangles are the sample points themselves. The constraint of eliminating Voronoi vertices further filters out the unwanted edges. The intuition behind this theorem is the fact that the medial axis approximates the Voronoi diagram for a sufficiently small value of r where S is an r-sample. Hence, even if two points lie very close to each other but are not adjacent, cannot be joined together in the final reconstruction if the circum-circle passing through them contains neither V nor S. This technique is called as Voronoi filtering.
For smaller values of $r < 0.06$ the reconstruction is not tightly bound. Hence, dense samples are required for which $r < 0.25$ is said to be sufficiently good. The algorithm assures that only adjacent samples on the curve are joined for $r <= 0.25$.

However, in case of 3D, it is not possible to carry out the Voronoi filtering technique very effectively. The assumption that Voronoi vertices lie close to the medial axis is not true. They often tend to be close to the sample points. Thus, instead of eliminating all the Voronoi vertices, we only eliminate a subset of vertices. Given a sample point, these vertices are two in number taken farthest from the sample one on either side of the surface. These are called the ‘poles’ of the sample point. These poles lie close to the medial axis. This constitutes set $P$. Thus, the Delaunay triangulation of $S \cup P$ is computed. Only those triangles that have their vertices as sample points constitute the crust.

### 2.6 Multiresolution Analysis

Representation of data in a suitable format is one of the foremost requirements to be able to analyze and manipulate data in order to get the desired output. The simplest way to represent data is by using time dependent waveforms. The magnitude of data in the signal varies as per time. So, every point on the waveform has a magnitude of a function at that time.

The well-known technique that has existed since years to achieve this is the Fourier transform developed by Joseph Fourier. The Fourier transform decomposes or separates a waveform or function into sines and cosines at different frequencies. The summation of all these sines and cosines leads to the original waveform. The amplitudes at different frequency sinusoids are identified. For long, the Fourier transform was considered to be a versatile tool in many fields of science as a mathematical or physical tool to alter the problem into one that can be more easily solved. In some branches of science, the Fourier transform of one function may yield another physical function.
However, the Fourier transform represents the function at one frequency completely and at that point has no information of the function at any other frequency. A large number of coefficients is needed to represent simple functions even though they may not show much fluctuations with time.

Multi-resolution methods came into being to overcome some of the serious drawbacks. The multi-resolution method consisted of representing signal at various resolutions hierarchically as a collection of coefficients. Each coefficient contains information about the signal’s amplitude and frequency.

Wavelets represent functions according to scale. The sharp perturbations that would otherwise go unnoticed in low resolutions are well defined at higher resolutions in finite time intervals. Wavelet analysis begins by using a wavelet prototype function called “mother wavelet”. All the decomposed functions are shifted versions of the prototype. The prototype represents the function at the finest possible resolution. This is called the shift-variant approach whereby the wavelets differ from each other by a mere phase shift. The resolution could be for example, in terms of, the number of control points in case of surface reconstruction or the number of pixels for image compression or editing. The next step is to represent the same function or signal at a coarser level (at a lower resolution). At every stage, the information thus lost during conversion is stored as detail coefficients that are useful during the reconstruction phase.

The original signal can be represented as linear combination of the coefficients of the wavelet functions. If we further choose to truncate some wavelet coefficients lower than a threshold, the data is sparsely represented with an acceptable level of error tolerance. Due to this sparse representation, wavelets are extremely useful in the field of data compression.
2.7 Basis Functions

Basis functions for vectors are the minimum set of vectors using which any vector can be represented. For example, for a two-dimensional vector (x, y), the vectors (0,1) and (1,0) are basis function since vectors (x, 0) and (0, y) can be generated using linear combinations of these by simply multiplying the basis by x and y respectively [Gra95].

There are a number of basis functions available of which we describe two of them that we will use throughout the thesis.

2.7.1 Haar Basis

This is the simplest wavelet basis. This can be best understood by an example. Consider an example of digital image representation and compression. We want to be able to monitor urban growth of a land using remotely sensed data. Our paper [IM02] describes this application using Neural Network techniques for the city of Mumbai, India in detail. An example of digital image data is the remotely sensed data collected by orbiting satellites, such as those of the Indian Remote Sensing Satellite series (e.g. IRS - 1A to IRS -1C). It is necessary to classify them and interpret them in order to identify the features. This is known as Image Classification. The satellite data for the analysis included 4 band IRS 1A LISS II data. This is made available in the form of imagery on paper or transparency or as digital data on tapes or floppies by the National Remote Sensing Agency at Hyderabad, India. Digital data consists of the gray levels of every pixel of the image at various bands in a .pgm (Pixel Gray Map) format.

An example of the Band 4 image (256 X 256 pixels) is as shown in the Figure 2.5.
Figure 2.5 Original image

The corresponding entries of the .pgm file would be as shown below:

```
P2
256 256
255
26 26 26 56 26 26 56 56 26 56 56 26 ............... 26 56 56 56 56 56 26 56 56
56 26 26 26 26 56 56 56 26 11 41 ............... 56 56 26 26 26 56 56 56 26 56
.
↓
256 rows
```

(The first row P2 stands for the file type. The next row gives the number of pixels column-wise and row-wise. The third row gives the maximum gray value. The other rows contain the gray levels of every pixel in the figure).

Suppose we want to accommodate a larger area, the image data would contain more pixels and consequently more storage space. However, using multi-resolution methods, it is possible to represent the same image at various resolutions using fewer
pixels by compensating a small percentage of accuracy. And consequently accommodate a larger study area if desired. Thus it can be effectively used for image compression.

The decomposition step can be applied to reduce the image column-wise or row-wise one level at a time. Let us consider column-wise decomposition. Since the image consists of 256 pixels, the decomposed image would contain 128, 64, 32 ...1 pixel at its successive resolutions depending on the level of accuracy desired.

Suppose we want to decompose the above image using Haar basis to its just lower resolution of 128 columns, the average of a pair of pixels at each row is done and the new average coefficient replaces the original two coefficients at every row. The detail coefficients capture the lost information due to down-sampling that are of very small magnitudes. For example, for the first row,

Average = (26 + 26)/2 = 26
Detail coefficient = 0 since 26 – 26 = 0
(26 + 56)/2 = 41 and so on.
Detail coefficient = 41 – 26 = 15 and 56 – 41 = 15

So the .pgm file after this step would look like

P2
256 256
56
26 41 26 56 41 41 ..............41 56 56 41 56 0 15 0 0 15 15 ...............15 0 0 15 0

Average Coefficients (128 columns) Detail Coefficients (128 columns)

...256 columns

256 rows

We observe that the first 128 columns represent the original lower resolution coefficients and the next 128 columns are the detail coefficients stored compactly to save
more space. We also note that the detail coefficients are of much lower magnitudes as compared to the original ones.

The final image would be as shown below –

![Image](image1.png)

Figure 2.6. 256 X 128 pixels of the original image

Similarly, we can bring about row-wise decomposition in a similar way to obtain fewer rows at every level. Both column and row wise decomposition can be performed simultaneously. Some examples of images are as shown below –

![Image](image2.png)
If we continue this way to obtain the coarsest 1 X 1 image, the .pgm file would contain an overall single average coefficient representing the original image and the rest all being detail coefficients. All the resolutions have the same number of coefficients. Hence no information is lost or gained in the process. We can reconstruct the original image by the reverse procedure of reconstruction to any level by using the decomposed and detail coefficients.

2.7.2 Notion of Haar Basis Functions

Basis functions separate the smooth and detailed portions of functions. In Haar wavelet, each function is a constant step-function over a certain interval.

Suppose we express these image coefficients as functions, a one-dimensional image would be represented as a constant function over an entire half-open interval say [0,1). A half-open interval [0,1) means the function is constant over all values of x in the range 0 <= x < 1. A vector space consists of all such functions or vectors for which basic properties like vector addition and multiplication are possible. The vector space at its lowest resolution is denoted by V^0. The vector space at its next resolution would have two half – intervals which are [0,1/2) and [1/2,1) and denoted by V^1. A two-pixel image would thus have two constant pieces over these two intervals. Continuing this way, the vector space V^j would have 2^j constant pieces of functions in as many equal-sized subintervals.
We note that every function in a lower resolution is contained within a function at higher resolution. That is, a vector in \( V^j \) can be represented as a vector in \( V^{j+1} \). Hence, we say that the functions are nested. This nesting of functions is an important criterion for multi-resolution analysis.

\[
V^0 \subset V^1 \subset V^2 \subset \ldots
\]

### 2.7.3 Mathematical Representation of Haar Basis Functions

The basis functions for the Haar basis are called scaling functions and are usually denoted by the symbol \( \phi \). A simple basis for \( V^j \) is given by the set of box functions represented as

\[
\phi^j_i(x) = \phi(2^j x - i) \quad i = 0, \ldots, 2^j - 1
\]

where

\[
\phi(x) =
\begin{cases} 
1 & \text{for } 0 \leq x < 1 \\
0 & \text{otherwise}
\end{cases}
\]

For example, the basis functions for \( V^0 \) and \( V^1 \) would be as shown –

![Figure 2.8 (a) Box basis for \( V^0 \) and (b) for \( V^1 \)](image)
The support for a function is the range of the function for which the function is non-zero for example, the support for $\phi_0^0$ in $V^0$ is $[0, 1)$. If the support for a function is within a bounded interval, the support is called compact support.

2.7.4 Wavelet Space Representation

Wavelet functions represented by vectors $\psi^i_j$ span a vector space denoted by $W^j$. Wavelets in $W^j$ represent those parts of the function $V^{j+1}$ that are not present in $V^j$. Wavelet coefficients in $W^j$ are the detail coefficients obtained at level $j$. Haar wavelets are mathematically represented by the box basis functions given by,

$$
\psi^i_j(x) = \psi(2^j x - i) \quad i = 0, \ldots, 2^j - 1
$$

where $\psi(x)$

- $= 1$ for $0 \leq x < \frac{1}{2}$
- $= -1$ for $\frac{1}{2} \leq x < 1$
- $= 0$ otherwise

For example, the Haar wavelets at $W^0$ and $W^1$ are as shown.

![Haar Wavelets for W^0](image)

(a)
2.7.5 Representation Using Matrices

Representing the box basis functions in terms of matrices is often a convenient way for mathematical calculations. Suppose all the Haar basis coefficients of the scaling functions at a particular level \( j \) are put together in a single row matrix \( M^j(x) \).

2.7.5.1 Scaling Functions Representation

\[
M^j(x) = [\phi_0^j \phi_1^j \phi \ldots \text{number of scaling functions at that level}]
\]

Using the concept of nested subspaces as explained before,

\[
M^{j-1}(x) = M^j(x) \cdot P^j
\]

where \( P^j \) is a dimension of \( M^j \) X dimension of \( M^{j-1} \) matrix.

2.7.5.2 Wavelet Functions Representation

Similarly, the wavelet functions can be put into a single row matrix \( N^j(x) \) at level \( j \).

\[
N^j(x) = [\psi_0^j \psi_1^j \psi \ldots \text{number of scaling functions at that level}]
\]

Since \( W^{j-1} \) is a nested subspace of \( V^j \)

\[
N^{j-1}(x) = M^j(x) \cdot Q^j
\]

where \( Q^j \) is a dimension of \( M^j \) X dimension of \( N^{j-1} \) matrix.
2.7.5.3 Matrices Representation for Decomposition

The coefficients of the function $V^j$ at any level can be written in terms of scaling function basis. Suppose $C^j$ is the column matrix of coefficients at level $j$, we can write

$$C^j(x) = \begin{bmatrix} c_{0j} & c_{1j} & \cdots & \text{number of coefficients at that level} \end{bmatrix}^T$$

(Note that $T$ denotes transpose, hence it is a single column matrix with number of rows = number of coefficients at that level).

Thus, for coefficients at level $j-1$,

$$C^{j-1} = A_j C^j$$

where $A_j$ is a constant dimension of $C^{j-1} \times$ dimension of $C^j$ matrix.

Suppose, detail coefficients are represented by the column matrix $D^{j-1}$ at level $j-1$,

$$D^{j-1} = B_j C^j$$

where $B_j$ is a constant dimension of $D^{j-1} \times C^j$ matrix.

The process of determining the lower level coefficients from the original ones is called decomposition. $A^j$ and $B^j$ are called analysis filters.

2.7.5.4 Matrices Representation for Reconstruction

The reverse process of obtaining the higher-level coefficients (of level $j$) from $C^{j-1}$ and $D^{j-1}$ (of the lower level $j-1$) is called reconstruction. These can be obtained as

$$C^j = P^j C^{j-1} + Q^j D^{j-1}$$

The matrices $P^j$ and $Q^j$ are called synthesis filters.

For example, for the Haar basis function using

$$c^1 = A^2 c^2$$
\[
\begin{pmatrix}
101 \\
96
\end{pmatrix}
= 
\begin{pmatrix}
0.5 & 0.5 & 0 & 0 \\
0 & 0 & 0.5 & 0.5
\end{pmatrix}
\begin{pmatrix}
102 \\
100 \\
97 \\
95
\end{pmatrix}
\]

The sequence \([0.5 \ 0.5 \ \ldots]\) represents the pair-wise averaging for \(A^i\). It is present at every row at any level and is the repeating sequence differing from that of its previous level by a column shift indicating a phase shift. The differencing operation for \(B^i\) is denoted by the repeating sequence \([0.5 \ -0.5 \ \ldots]\).

### 2.7.6 Single Knot Wavelets

The other type of wavelet is the single-knot wavelet. The scaling functions for \(V^i\) are the hat functions centered over integers. The wavelet functions \(W^i\) are the hat functions \(V^j\) centered over the odd-numbered integers. These wavelets are called lazy wavelets by Wim Sweldons [Wim94] as these are computed as a subset of scaling functions.

These wavelets will be used for the purpose of surface reconstruction for hierarchically decomposing the input contours. For example, the scaling functions of \(V^2\) and \(V^1\) and the wavelets for \(W^1\) are as shown below -

![Wavelet Graphs](image-url)
The $P^j$, $Q^j$, $A^j$ and $B^j$ matrices are represented by the following matrices (say for level $j = 2$).

$$
P^2 = \frac{1}{2} \begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix} \quad Q^2 = \frac{1}{2} \begin{pmatrix} 0 & 0 \\ 2 & 0 \\ 0 & 0 \\ 0 & 2 \end{pmatrix}
$$

$$
A^2 = \frac{1}{2} \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \quad B^2 = \frac{1}{2} \begin{pmatrix} -1 & 2 & 1 & 0 \\ 0 & 0 & -1 & 2 \end{pmatrix}
$$

### 2.7.7 Filter Bank Method

The original coefficients $C^j$ is split to a lower resolution coefficients $C^{j-1}$ and detail coefficients $D^{j-1}$ by using matrices defined above. This process of subdivision can be applied $C^{j-1}$ to obtain coefficient matrix $C^{j-2}$ and so on. This recursive procedure of subdivision is called the filter bank method.
The original coefficients can be obtained one level at a time by using the lower level coefficients and detail coefficients. The number of coefficients at every level remains the same. No extra storage space is thus required.

There are three broad categories of wavelets based on how the different types of matrices especially $Q^j$ are defined. Before we go on to study the 3 broad types of wavelets, let us first try to understand the notion of orthogonality.

### 2.7.8 Notion of Orthogonality of Wavelets

As we know two vectors are orthogonal if their dot product is zero. An inner product on two vectors $f$ and $g$ in a vector space is defined as

$$<f | g> = \int_0^1 f(x) \ g(x) \ dx$$

The vectors $V^j$ and $W^j$ are said to be orthogonal if the standard inner product defined as above is zero. There are three types of wavelets based on its orthogonality.

### 2.7.8.1 Orthogonal Wavelets

A basis is said to be orthogonal if every basis function is orthogonal to every other basis function. The Haar basis functions are said to be orthogonal.

This is because the scaling functions are orthogonal to each other; the wavelets are orthogonal to one another and each wavelet is orthogonal to every coarser scaling function. So the wavelets are called orthogonal wavelets. We observe that the following is true with respect to the $A$, $B$, $P$ and $Q$ matrices for orthogonal wavelets.

- $[Q^j] = [Q^j]^T$ and $[Q^j]^T = [Q^j]^{-1}$
- $A^j = (P^j)^T$ and $B^j = (Q^j)^T$

A matrix with this property is called an orthogonal matrix. A function $<f | g>$ is said to be normalized if it is equal to 1 when $i = j$ and 0 otherwise.
\[ \langle \phi^j | \phi^j \rangle = I \text{ (identity matrix)} \]
\[ \langle \psi^j | \psi^j \rangle = I \text{ (identity matrix)} \]
\[ \langle \psi^j | \psi^j \rangle = 0 \]

2.7.8.2 Semi-orthogonal Wavelets

Since the Haar basis is the only known orthogonal basis, we need to define the wavelets in a looser way. Suppose the wavelets are orthogonal to coarser scaling functions, however, the wavelets and scaling functions are not orthogonal to each other. Such wavelets are called semi-orthogonal wavelets or pre-wavelets. Thus, orthogonal wavelets are a special case of semi-orthogonal wavelets. Examples of such types of wavelets would be the B-spline wavelets.

In mathematical terms, \[ \langle \phi^j | \psi^j \rangle = 0 \]

2.7.8.3 Biorthogonal Wavelets

It is possible to define wavelets that, although not orthogonal to scaling functions, still would have many desirable properties of those of the semi orthogonal wavelets. Multiresolution bases of this type are called biorthogonal wavelets and they were first developed by Cohen et al [CDF92]. Single knot wavelets are a type of biorthogonal wavelets. The only requirement for biorthogonal wavelets is that matrices \( A^j \) and \( B^j \) exist.

2.7.9 Overview of Multi-Resolution Tiling Algorithm

The various steps of this algorithm have been described in detail in Chapter 4. This algorithm is used for the purpose of surface reconstruction based on the multi-resolution theory of wavelets. The algorithm uses single-knot wavelets and the corresponding matrices defined in the previous section. The 3 main steps of the algorithm in brief have been described here.

1. Decomposition – The input contours are interpolated to obtain the number of vertices in each contour as a power of 2. The original wavelet coefficients are
then broken down into low-resolution contours and its detail coefficients using single-knot wavelets.

2. **Tiling** – The decomposition stops when the base case expressed in terms of the number of vertices is reached. We have taken a base case of 4 vertices for every contour. Using a larger base case may give better results but at the cost of computational expense. Using a very small base case would be erroneous as the quality of tiling may suffer. This is because the derived lowest resolution contours no longer resemble the original solid. According to Meyers [Mey92, Mey94], if the number of vertices available is very large, a base case of 64 vertices is a good choice.

3. **Reconstruction** – Filter bank method of reconstruction is used. The original contours are reconstructed one level at a time from the coarsest to the finest. At every level, vertices are added and the tiling is improved by local optimization. This step is performed until the original number of vertices is present in the contours.
3.0 PIECEWISE LINEAR INTERPOLATION ALGORITHM

This chapter describes the algorithm of reconstructing a surface using cross-sectional slices by [BS94]. This algorithm handled multiple contours and any nesting or hierarchy of contours on any slice. The input files available to us are those of medical imaging data of various medical organs. An example of a file cone2.dat is attached. We know the number of slices, the number of contours in every slice and the number of sample points in every contour. Also, given are the user-specified values of the discretization parameter, the match-neighborhood, the voting-threshold and the score-threshold. If any or all of these parameters is not given in the file, default values are assumed. The 3D Cartesian co-ordinates of every sample point are given. Let us discuss the details of all the steps of the algorithm.

We shall take an example of the file cone2.dat. The data is read from the file and a pair of slices has been selected for each of the steps. All the input co-ordinates of vertices are scaled such that they lie within a range of 0 to 1. This file has 4 slices. All have a contour each except the second that has 2. Let us assume suffix 1 for the first (lower) slice and suffix 2 for the second (upper) slice.

3.1 Preprocessing

This step primarily consists of setting up the contour data in a suitable form for all other steps.

**STEP 1:** In this step, the nesting hierarchy of the contours is determined. Each slice is taken at a time. The contours of that slice are taken in pairs to determine if one of them is nested within the other. This way, the nesting level of each of the contours of the slices is set. The nesting level is actually the number of contours enclosing the current contour starting with 0. The heuristic proposed is that the contours do not intersect in the x-y plane. Consider Figure 3.1. It shows an example of a single slice projected onto the x-y plane. The slice has many contours that are nested but do not intersect in the projection. The nesting level of the outermost contour is 1.
! Data file created by input.c version 1.0

! Comments are enclose between '!' and new-line

! The format of data is as follows :
! <Number of slices> <Number of Contours> <Number of Points>
! <slice #> <Number of contours> <slice gap>
!   <contour #> <Number of points>
!       x1  y1  x2  y2  x3  y3  ....
!   <contour #> <Number of points>
!       x1  y1  x2  y2  x3  y3  ....
!   ....

! Data begins ..... 
_slice_data

4  !Number of slices
5  !Number of contours
117 !Number of points
1 5.000000 1! <slice #> <slice height> <Number of contours>

   17  ! <Number of points>
 473 175  430 166  404 159  377 160
 348 170  320 180  306 192  297 215
 297 235  332 254  359 261  389 266
 446 264  470 243  485 216  485 193
 483 182
2 10.000000 2! <slice #> <slice height> <Number of contours>

   18  ! <Number of points>
 514 154  488 152  468 153  450 163
 434 185  438 208  444 226  466 242
 490 249  513 251  534 249  561 218
 566 205  565 191  559 178  547 168
 533 150  526 150

   15  ! <Number of points>
 285 178  255 188  235 210  232 229
 231 249  243 270  268 279  288 286
 300 285  354 263  368 222  359 192
 339 169  308 166  292 173
3 15.000000 1! <slice #> <slice height> <Number of contours>

   35  ! <Number of points>
 381 147  322 150  293 148  236 158
 206 180  201 210  199 243  214 273
 239 287  262 300  316 309  343 304
 367 288  383 266  391 247  394 220
```
399  193  406  182  416  203  425  234
427  251  444  280  481  281  510  277
545  274  574  261  594  222  595  182
572  158  548  141  511  127  482  125
455  134  433  142  405  146
4  20.000000 1! <slice #> <slice height> <Number of contours>

32  ! <Number of points>
565  121  524  112  493  108  441  112
397  112  309  115  261  126  235  133
211  148  193  156  179  174  175  203
176  234  182  268  210  299  236  311
282  324  319  326  343  326  391  305
407  285  436  298  456  310  501  303
548  289  578  281  608  259  619  228
621  201  617  170  596  145  588  128

_end
! Happy end
```
**STEP 2:** Next, the vertices of each contour are ordered such that the contour is oriented in anticlockwise direction if its nesting level is even and clockwise if it is odd. The contours represent where the surface actually exists. The portion between contours is a gap. This step is done to ensure that the existing surfaces lie to the right of the contour. The orientation is just to make the representation of contour data consistent and follow some fixed direction.

To determine the orientation of the contour, choose a point $P$ on the perpendicular bisector of any edge of the given contour. Let the edge co-ordinates be $(x_1, x_2)$ and $(y_1, y_2)$ as shown in the Figure 3.2. If $P$ is to the left of the edge and $P$ lies inside the contour $C$ or if $P$ is to the right of the edge and $P$ does not lie inside the contour $C$, the orientation is COUNTER_CLOCKWISE. (Figure 3.2 (a).) It is CLOCKWISE otherwise (Figure 3.2 (b)).

The Point $P$ is chosen using the equation:

\[
P.x = \frac{(x_1 + x_2)}{2} + d \times b / \sqrt{a^2 + b^2};
\]
\[
P.y = \frac{(y_1 + y_2)}{2} - d \times a / \sqrt{a^2 + b^2};
\]

where $a = (x_2 - x_1)$, $b = (y_2 - y_1)$

The choice of the parameter $d$ (discretization parameter) should be based on the resolution of the available data. It must be low enough to ensure that there is no edge of
the contour between the chosen point and the chosen edge. For our example, we have chosen a discretization parameter of 0.01.

![Image showing contour with points labeled (x1, y1), (x2, y2) and P]

Figure 3.2 (a) Counter-clockwise and (b) clockwise orientation of contour

**STEP 3:** To re-orient the contour, just swap the vertices half way down so that the contour is re-oriented. Now, it is necessary to generate many new points on the contour boundary so that the adjacent vertices are close and the distance is approximately equal to the constant discretization parameter. The contours are interpolated to generate new points and many smaller line segments are formed. The number of segments to be generated between two successive samples = \[\text{length of the line segment joining those samples} / \text{discretization parameter}\].

The slices after the pre-processing stage are displayed and appear as in Figure 3.3.
3.2 Contour Matching

A partial contour match is a strip of vertices from a pair of contours such that the distance between corresponding vertices in the strip is below some threshold parameter. This is done to find out a sequence of vertices between a pair of contours that show very close resemblance. The resultant strips are a pair of sub-curves (subset of the original contours), one from each contour.
Figure 3.4 Contour matching

Suppose Figure 3.4 shows the top view of a pair of slices. The lower slice has one contour while the upper slice has two contours. The distance between the strip of vertices between i and k (from C2 of S2) and j and l (from C1 of S1) is so close that the pair of sub-curves forms a contour match.

This is achieved by performing the following tasks.

**STEP 1:** Take all contours C2 of slice S2. For every contour C2, sort its vertices in increasing order of their x co-ordinates. This is done to enable range searching.

**STEP 2:** Now consider all pairs of contours between slices S1 and S2. Say the current pair under consideration is C1 (from S1) and C2 (from S2). Take every vertex from C1. Say, the vertex is ‘v’.

From the sorted list of vertices of C2, find those vertices that lie within the epsilon-neighborhood of the vertex ‘v’. Among this list, filter out those vertices whose y co-ordinates also lie within the epsilon-neighborhood of ‘v’. The range search is done by the binary search method. The result of this step is that ‘v’ is surrounded by a strip of vertices whose x and y co-ordinates are at the most at a distance of epsilon from ‘v’. The
user gives the value of epsilon. Such a neighborhood between corresponding vertices of C1 and C2 constitutes a vote. For example vertices k and l form a vote.

**STEP 3:** Calculate the shift of every vote. If the querying vertex ‘v’ is at position ‘pos’ in the strip between i and k, the shift is given by

$$\text{shift} = (k + (n2 \times (\text{pos} / n2 + 1)) - \text{pos} \mod n2)$$

where n2 is the number of vertices of C2.

**STEP 4:** We have to find the score of the vote between vertices k and l.

$$\text{Score} = (1.0 / (\text{distance}(k, l) + 0.1))$$

**STEP 5:** The resulting votes are transformed to a collection of partial contour matches. The start and end vertices of both contours of all possible matches are stored. Now, we need to filter out some of the matches that do not satisfy certain requirements.

**STEP 6:** Delete those matches not satisfying the minimum number of votes or an acceptable level of the scoring function. The matched contour portions, the votes and scores are updated to accommodate the changes due to deleted matches.

**STEP 7:** Consider all the matches stored once more. Compare the matching extents with all accepted matches so far. Truncate the extents of the match, if there is an overlap with any of the accepted matches.

**STEP 8:** The scores and shift values of the updated matches are calculated. Again the criterion for the minimum number of votes and acceptable levels of score are computed for the resultant matches. If not, those matches are further discarded.

The result of the entire curve-matching step is a set of acceptable matches between any possible pairs of contours taken from successive slices. These matches are not too short nor too long nor do they overlap.
3.3 Stitching the Matches

This step consists of triangulating or stitching together the contour matches found in the above step. The matches are taken one by one and the following tiling procedure is applied to stitch them [CS78].

**STEP 1:** Determine the length of the match between two contours C1 and C2. The length refers to the number of vertices contained in the match. Sometimes, the matches may contain all the original vertices of the contours signifying that the contours are exactly concentric.

**STEP 2:** The number of triangles would be twice the match length. The actual tiling procedure is illustrated with the help of the diagram in Figure 3.5. Let p and u be the successive vertices in contour C1. Let q and v be the successive vertices in C2.

If \((\text{distance (pu)} + \text{distance (qu)}) < (\text{distance (pv)} + \text{distance (qv)})\)

Generate triangle PQU. Advance the vertices p and u by one for the next iteration.

else

Generate triangle PQV. Advance the vertices q and v by one for the next iteration.

![Diagram](image)

Figure 3.5. (a) Case when triangle (PQU) is formed (b) Case when triangle (PQV) is formed.

Two combinations of triangles (shown by dotted and solid lines) are possible. Depending on which combination satisfies the condition, the solid triangles are formed instead of the dotted ones.
**STEP 3:** This step is repeated until we have reached the end vertex of one of the contour matches. If C1 matched vertices are exhausted, form triangles using q and v of C2 and the end vertex of C1.

Similarly, if C2 is first exhausted, form triangles using the remaining successive vertices of C1 and the end-vertex of C2. This is similar to merging two sorted lists of numbers. The process terminates when we reach the ends of both contour matches.

The result of the entire stitching process is the generation of mesh of triangles between the matched contour portions as shown in Figure 3.6.

Figure 3.6 Matched Contour Portions of Slices Stitched
3.4 Forming Clefts

After tiling the matched contour portions, the unmatched portions remain. These along with the extreme edges of the triangulations of the matched portions form closed 3D polygons known as clefts. This step basically aims at finding such clefts.

Take every slice in turn. Take all the matches between any pairs of contours in this slice.

**STEP 1:** First, preprocess and store the matched vertices in a suitable way so as to ensure easy jumps between vertices of different contours to generate closed polygons. A jump is always from starting match vertex of a contour C1 to the corresponding start match vertex of another contour C2. Similarly, a jump is always made from end vertex of contour C2 to corresponding end vertex of contour C1. Consider Figure 3.7 (a). Let vertices i and j be start and end vertices of matched portion of one of the contours C1 (of S1). Let k and l be the start and end vertices of the matched portion of C2 (of S2).

![Figure 3.7 (a) Matched and (b) unmatched portions (cleft) of contours](image)

We first store the information on which vertex to jump to when we arrive at a particular vertex. For example, we have to jump to k if we reach i, and to j if we reach l. Also, the unmatched vertices in C2 are arranged in a clock-wise direction while the unmatched vertices of C1 are arranged in counter-clockwise direction. We need to build clefts of the form shown in Figure 3.7 (b) by leaving out the matched sub-curves. The complete closed polygon consisting of all 4 vertices is known as a cleft.
**STEP 2**: Sometimes, a contour may have no match and is itself a cleft.

**STEP 3**: After pre-processing, the task of finding clefts for even matched contours is easy. We begin with one of the vertices of contour C1, say i. We jump to the vertex k of the contour C2. This information of where to jump is already available from the previous step. We then walk through all the unmatched vertices in clock-wise direction on C2 till we reach the end-vertex l of the match. We then make a jump to vertex j of C1. Again walk through the unmatched vertices of C1 in counter-clockwise direction till we reach i.

The above steps are repeated on all matched contour pairs between successive slices. The result is that all the unmatched (non-triangulated) closed polygonal portions or cycles called clefts between any pair of contours in successive slices are given.

### 3.5 Bridging the Clefts

We already have the unmatched portions of all the contours in a pair of slices. Now, since the contours may be nested up to any level, the clefts also may be nested and contained within an outer contour.

**STEP 1**: First, the nesting level or hierarchy of the 3D clefts present must be determined. A pair of clefts is taken one by one. Check if one of the clefts is contained within the other. For this, take every point in one and check if it lies within the boundary of other. For each cleft i lying within another cleft j, nesting level of cleft i is increased by 1 and the child count of j is increased by 1. The heuristic proposed is that the 2D projections of the clefts do not intersect each other similar to the heuristic for contours as shown in Figure 3.1.

**STEP 2**: We form an undirected weighted complete graph G with the vertices of G corresponding to the nested clefts determined. One of the clefts with minimum possible nesting level and having some children is chosen. The nodes of G correspond to either this parent cleft or its children.
STEP 3: The weights of the graph edges are to be computed. The weights correspond to the distances between the corresponding nodes. Since each node is a cleft, the 2D distance between two nearest points of the clefts is the weight. The distance between same nodes is set to a very high value of infinity.

STEP 4: We now compute a minimum spanning tree (MST) of G.

STEP 5: Then store the information about the various bridge points for every node. The two nearest points v1 and v2 of clefts C1 and C2 of Figure 3.8 have already been determined earlier. v3 and v4 are the successive vertices to v1 and v2 on the clefts c1 and c2 respectively.

STEP 6: Traversing the cleft cycles forms a bridge. Each bridge between clefts consists of a pair of edges that are oppositely oriented as shown in Figure 3.8. One of these clefts is nested within the other. One edge leads to the cleft C1 and the other leads to the cleft C2. This is necessary because all the bridged clefts can be represented as one complete closed polygonal cycle.

![Figure 3.8 Bridge between clefts](image)

Start from a branch point v1 of the highest node in the MST. Jump over to v2, collect all the vertices till you reach v4. Jump over to v3 and then traverse all the vertices
till you reach v1 back. This way, we have one closed polygon over all the contours up to any level connected by bridges. Since an edge of MST is the edge formed by minimum distance between the two clefts, the bridge is also of the smallest possible length between the contours.

Thus, all the nested contours are replaced by one complete polygon. At the end of this step, we are left with just closed 3D polygons whose 2D projections do not intersect.

3.6 Triangulation of 3D Clefts

Now, we need to triangulate the 3D polygons generated. Triangulation is based on the principle of reducing the total cost of the triangulation. This means that the set of triangles ultimately formed should have the lowest cost as compared to any other combination of triangles formed out of any other set of vertices. The polygons of a slice are taken one by one and the following steps are performed.

**STEP 1:** Discard the middle vertices that produce straight edges. If edge (i, j) and edge (j, j+1) of a polygon lie on the same straight line, the vertex j is deleted.

**STEP 2:** Costs between various vertices are initialized. The cost is assigned as per the following rules where v1, v2 and v3 are the successive vertices in that order.

2.1 Cost (v1, v2) = Distance (v1, v2)
2.2 Cost (v1, v3) = Function (v1, v2, v3)
   The function itself is assigned as follows –
   Function (v1, v2, v3)
   = (0.85 * area of triangle formed by v1, v2, v3)
   + (0.45 * perimeter of the triangle)
   + 0.6 (ratio between the smallest and largest of the edges).
2.3 Cost of all the polygonal chains of lengths >= 3 are also computed. The chain would contain 3 or more vertices in order but not successive (In order means the
vertices are traversed in one direction, either clockwise or anti-clockwise without retracing). Cost of the chain of vertices with a start vertex v1 and end vertex v2 is given as –

\[
\text{Cost (v1, v2) = Cost (v1, m) + Cost (m, v2) + Function (v1, m, v2)}
\]

where m is any vertex after v1 and before v2. The vertex m is moved from v1+1 to v2-1 in order and at every step and the cost is calculated. The final cost is the minimum of all costs and the corresponding vertex m at which this is achieved is also stored.

Thus, this step basically aims at finding out costs or weights between all possible combinations of chains (in order) with each chain having any number of vertices starting from 2 to the maximum length of chain possible.

**STEP 3:** The actual triangulation using the costs obtained above is done. Consider Figure 3.9.

![Figure 3.9 A polygonal chain](image)

The chain formed by vertices starting from i and ending with k is of maximum possible chain length. A triangle (i, j, k) is generated. Next, the untriangulated polygonal chains between (i,j) and (j,k) are considered. Again the maximum length polygonal chains for these 2 chains are determined and triangulated. Again leftover untriangulated sub-chains are considered. This recursive procedure is done until the entire polygon is triangulated.
**STEP 4:** These triangulations along with the stitched triangulations of the matched contour portions gives the entire surface so reconstructed between the pair of slices.

These steps are repeated for every pair of successive slices recursively starting from 0. The final reconstructed surface of the slices in cone2.dat is as shown in Figure 3.10.

![Figure 3.10 Final Reconstructed Surface](image)

Some more sample screen shots of outputs using this algorithm for other data are attached in Appendix B.
4.0 MULTI-RESOLUTION TILING ALGORITHM

In this chapter, we shall provide the details of the multi-resolution algorithm. This algorithm is based on the single-knot wavelets to hierarchically decompose polygonal contours. The approach called multi-resolution tiling algorithm was first described by Meyers [Mey92, Mey94]. The algorithm is defined for the purpose of surface reconstruction using a number of slices parallel to each other taken through the solid. However, the algorithm described by Meyers assumes only one contour per slice. The data used is the same as that of the previous algorithm described in Chapter 3. However, we have taken only those examples where there is just one contour per slice. The structure of the input file is thus the same. We are given a number of slices for each surface. Each slice has one contour. Each contour has a number of sample points on it. The points are given in terms of their (x, y, z) co-ordinates. Now, we shall discuss all the steps of the algorithm in detail.

Again, we shall take an example of the file cone2.dat which has been modified and is as attached. The data is read from the file and a pair of slices has been selected to perform the steps outlined below. All the input co-ordinates of vertices are scaled such that they lie within a range of 0 to 1. This file has 4 slices with one contour each. Let us assume suffix 1 for the first (lower) slice and suffix 2 for the second (upper) slice.

4.1 Contour Decomposition

The first and foremost step for carrying out multi-resolution analysis is to determine if the number of coefficients to be decomposed is a power of 2. For this application, the coefficients are the control points on the contours. Hence, the number of points on all the contours must be a power of 2. The following procedure is followed to accomplish the decomposition procedure.
! Data file created by input.c version 1.0

! Comments are enclose between '!' and new-line

! The format of data is as follows:
! <Number of slices> <Number of Contours> <Number of Points>
! <slice #> <Number of contours> <slice gap>
!   <contour #> <Number of points>
!   x1 y1 x2 y2 x3 y3 ....
!   <contour #> <Number of points>
!   x1 y1 x2 y2 x3 y3 ....
!   ....
!   ....
!   ....
!   ....
!   ....

! Data begins ..... 

_slice_data

4  !Number of slices
4  !Number of contours
99 !Number of points
1 5.000000 1! <slice #> <slice height> <Number of contours>

17  ! <Number of points>
473 175  430 166  404 159  377 160
348 170  320 180  306 192  297 215
297 235  332 254  359 261  389 266
446 264  470 243  485 216  485 193
483 182
2 10.000000 1! <slice #> <slice height> <Number of contours>

18  ! <Number of points>
514 154  488 152  468 153  450 163
434 185  438 208  444 226  466 242
490 249  513 251  534 249  561 218
566 205  565 191  559 178  547 168
533 150  526 150

3 15.000000 1! <slice #> <slice height> <Number of contours>

32  ! <Number of points>
381 147  322 150  293 148  236 158
206 180  201 210  199 243  214 273
239 287  262 300  316 309  343 304
367 288  383 266  391 247  394 220
399 193  406 182  416 203  425 234
427 251  444 280  481 281  510 277
545 274  574 261  594 222  595 182
572 158  548 141  511 127  482 125

52
4 20.000000 1! <slice #> <slice height> <Number of contours>

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>565</td>
<td>121</td>
<td>524</td>
<td>112</td>
<td>493</td>
<td>108</td>
<td>441</td>
</tr>
<tr>
<td>397</td>
<td>112</td>
<td>309</td>
<td>115</td>
<td>261</td>
<td>126</td>
<td>235</td>
</tr>
<tr>
<td>211</td>
<td>148</td>
<td>193</td>
<td>156</td>
<td>179</td>
<td>174</td>
<td>175</td>
</tr>
<tr>
<td>176</td>
<td>234</td>
<td>182</td>
<td>268</td>
<td>210</td>
<td>299</td>
<td>236</td>
</tr>
<tr>
<td>282</td>
<td>324</td>
<td>319</td>
<td>326</td>
<td>343</td>
<td>326</td>
<td>391</td>
</tr>
<tr>
<td>407</td>
<td>285</td>
<td>436</td>
<td>298</td>
<td>456</td>
<td>310</td>
<td>501</td>
</tr>
<tr>
<td>548</td>
<td>289</td>
<td>578</td>
<td>281</td>
<td>608</td>
<td>259</td>
<td>619</td>
</tr>
<tr>
<td>621</td>
<td>201</td>
<td>617</td>
<td>170</td>
<td>596</td>
<td>145</td>
<td>588</td>
</tr>
</tbody>
</table>

_end

! Happy end
**STEP 1:** If the number of vertices in the 2 contours is not a power of 2, we try to add some vertices such that the number becomes equal to the nearest power of 2. For example, if there are 14 vertices, the nearest power of 2 is 16. Hence, we have to add 2 extra vertices to make the total 16. To add these extra vertices, the procedure used is –

1.1 Form edges out of the n (initial number of vertices). There would be n edges.
1.2 Put these n edges into a priority queue the priority being the length of an edge.
   The queue would have the longest edge at its root.
1.3 Do while (number of vertices is not a power of 2)
   
   { 
   The longest edge (always the root) is removed.
   This edge is split into two to give a middle vertex.
   Up heap the priority queue structure.
   Increase number of vertices by 1.
   }
1.4 Report the total number of vertices.

**STEP 2:** Once the number of vertices is a power of two, the contours are decomposed to lower resolutions in steps. If the initial number of vertices, for example, is 32, make it 16, then 8 and lastly 4. This is done with single-knot wavelets by using appropriate A and B matrices. The general formula for decomposing is

\[ C^j = A^j C^{j-1} \]
\[ D^j = B^j C^{j-1} \]

where \( C^j \) and \( C^{j-1} \) are the column matrices containing vertices in levels \( j \) and \( j-1 \) respectively. \( D^j \) is the column matrix containing the detail coefficients in level \( j \). \( A^j \) and \( B^j \) are the analysis filters for single-knot wavelets.

**STEP 3:** \( j = 2 \) or vertices in the lowest resolution = 4 is called the base case where all the contour decomposition ultimately leads to. Using a large base case gives better tiling but requires more time for execution. Since our data mostly consisted of contours of up to a
maximum of 32 or 64 vertices, we chose a base case of 4 to have 3-4 layers of low-resolution contours.

The original and the decomposed contours from the input data in file cone2.dat are as shown in Figure 4.1 and Figure 4.2 respectively.

Figure 4.1 Original Contours

Figure 4.2. Decomposed Contours
4.2 **Base Case Tiling**

The tiling of the base case is done similar to the stitching of the matched portions of the contours in the piece-wise linear surface reconstruction algorithm as described in Section 3.3.

Consider Figure 4.3 where vertices 1, 2, 3, 4 are from one contour and vertices A, B, C, D are from the next. This is how a decomposed pair of contours with 4 vertices would look like.

**STEP 1:** Consider first quadrilateral AB12 formed by first two vertices of both contours each.

If the sum of the edge lengths \((1, 2) + (A, 2)\) < sum of edge lengths \((A, B) + (B, 1)\)

Generate triangle (A12) and triangle (AB2).

Else

Generate triangle (B12) and triangle (AB1).

![Diagram of base case triangulation](image)

**Figure 4.3. Base Case Triangulation**

The idea is to have the shorter of the two diagonals, either A2 or B1 (One of the dotted lines of the figure).

**STEP 2:** Do the same for the remaining 3 quadrilaterals (BC32, CD43 and AD41). There would be 8 triangles formed at the end of this step.
**STEP 3:** The triangle edges that connect vertices from one contour to the other are called suspect edges. In the figure, edges A1, B2 and the diagonal (A2 or B1) are suspect edges. These suspect edges are stored as they are useful for the purpose of local optimization during reconstruction step.

The base case triangulation of contours is as shown in Figure 4.4.

![Figure 4.4 Lowest Resolution Contour Triangulation](image)

### 4.3 Filter Bank Reconstruction

For filter-bank reconstruction, one level of wavelet coefficients is returned to the contours one filter-bank level at a time from the base case to the level where the original vertices are all present.

This is the last step of the algorithm and comprises of the triangulation of the entire surface. The output generated is the final surface reconstructed. The following steps are carried out to bring about the reconstruction.
**STEP 1:** Introduce one level of reconstruction for every contour. Hence, the total number of vertices would now be 8 in every contour. The 4 new vertices in each contour would now lie between each original pair of vertices respectively.

![Diagram showing triangulation with one level of filter bank.](image)

**Figure 4.5. Triangulation with one level of filter bank**

**STEP 2:** Consider Figure 4.5. Again vertices A, B, etc are from one contour and 1,2,etc. are from the other. Consider the front-most polygon 152DEC. Vertices C12 and CD2 that were joined as triangles in the base case are now quadrilaterals. Split each of these quadrilaterals into two triangles each. These triangles are simply formed by joining the newly introduced vertex with its opposite vertex in the same quadrilateral. For example, the dashed edge C5 splits quadrilateral 152C into two triangles C15 and C52. Similarly, edge E2 forms two triangles out of quadrilateral CED2.

This is done for all other 3 polygons introduced due to the addition of the vertices. We would thus get 16 triangles at the end of this step. Update the list of suspect edges (16 in number).
STEP 3: After the triangles are formed with one level of filter-bank vertices added, local optimization is done. This can be explained with the help of Figure 4.6 again with respect to the front-most polygon. The local optimization is similar for all other polygons too.

![Figure 4.6. Local Optimization](image)

Consider each suspect edge starting from C1. Two possibilities exist.

**Case 1:**
Let us take an example of edge C5. Take the other two edges of two triangles formed by this suspect edge (not the suspect edge itself). The triangles are C15 and C52 and the other edges are C1 and C2. If these two triangles share a common end-point (C in this case) ignore and simply go on to the next suspect edge.

**Case 2:**
Let us take for example edge C2. In this case, the two triangles formed are C52 and C2E. The edges C5 and E2 do not share a common end-point. So within quadrilateral C52E, consider the sum of edge lengths as before.

If \(d(CE) + d(C2) < d(52) + d(5E)\)

{} 
Let the edge C2 be as is

{} 
Else
Swap edge 5E for C2. Update the list of suspect edges.

Go on to the next suspect edge.

The local optimization step terminates when all suspect edges in the current tiling have been considered. This step basically ensures that the goal function that is the sum of edge lengths is minimal for the formation of triangles.

**STEP 4:** Introduce levels of reconstruction (16, 32, \ldots vertices for every contour) in succession and at every level, split the new quadrilaterals into 2 triangles and perform local optimization on all suspect edges. This is done till all the initial vertices from all contours are present.

Here, we must mention a subtle issue pertaining to the reconstruction process. We consider contours in pairs starting from 0-1, then 1-2 and so on. For each pair, we add vertices using successive filter bank levels on both contours and tile them. Say contours 0 and 1 contain 4 vertices in the lowest resolution and 16 vertices each in the highest (original) resolution. We reconstruct starting from the base case and generate 8 and 16 vertices in 2 successive levels.

Next, we again consider the base case vertices of contours 1 and 2 and add successive filter-bank levels to these to have 8 and 16 vertices. Now, if contour 2 had more number of vertices initially (say 32). One more level of contour 2 reconstructions from 16 to 32 vertices is possible; however there would be no corresponding level for contour 1 (Since contour 1 had only 16 vertices originally). This would result in non-uniform iterations of the reconstruction procedure for input contours having different number of vertices.

Hence, this algorithm requires the number of vertices in all contours of a particular surface to be equal to enter the reconstruction procedure. Recall that we
interpolate the contours if the number of vertices is not a power of two. So, their initial number may be different. But, after interpolation is done, the number of vertices should be the same for all contours and also a power of 2.

The above 3 steps is repeated using the next pair of slices and the final surface is the summation of reconstructions between all such pairs. The final reconstructed surface for the example of cone2.dat is as shown in Figure 4.7.

![Figure 4.7. Final Reconstructed Surface](image)

Some more sample screen shots of outputs using this algorithm for other data are attached in Appendix B.
5.0 EXPERIMENTAL EVALUATION

In this chapter, we present some results of the experiments conducted to analyze the performance of multi-resolution tiling algorithm. We shall compare this algorithm with the piecewise linear classification technique to know what limitations exist in each.

5.1 Implementation Environments

We tried our experiments on Sun Solaris OS Version 5.8 with an X-Windows environment. The software package has been written in C and has X-Windows based GUI using Xtoolkit, Motif and OpenGL for rendering.

We used real medical imaging data obtained from Gill Barequet and Micha Sharir [BS94] of Tel Aviv University. However, since the multiresolution tiling algorithm can handle data with only one contour per slice unlike the piecewise linear algorithm, we modified out input files to have single contours per slice for the former. Both single (modified input files) and multiple contours (original input files) were used to perform experiments to test the latter. However, the rest of this chapter for comparing the two algorithms is based solely on the modified input files. Some outputs of the piece-wise linear surface reconstruction algorithm on original data of multiple contours are as attached in Appendix B.

5.2 Comparison in Terms of Complexity of Algorithms

5.2.1 Piecewise Linear Interpolation Technique

The first step, which is the pre-processing step of the piecewise linear interpolation technique requires \(O(k)\) time on a contour with \(k\) initial vertices. Let the number of vertices in the contours after the arc discretization step be \(n\). Usually, \(k\) is much smaller than \(n\) (\(k << n\)).

The curve-matching step requires considering all the \(n\) vertices each of a pair of contours. This requires \(O(n)\) time. The stitching of \(n\) vertices can be done in \(O(n)\) time as the vertices are traversed in the same order without back tracking. Again, formation of
clefts requires $O(n)$ time since the $n$ vertices of contours at the most need to be traversed to form clefts. The number of clefts may be in the worst-case equal to $O(k^2)$. However, it has been proven in [BS94] that the number of clefts is lesser or at least comparable to the number of input contours. Let the number of clefts be $c$. Bridging and finding the MST would take $O((k+c) \log (k+c))$ time.

Triangulation is the most expensive part of this algorithm. Triangulation takes $O(ch^3)$ time where $h$ is the average number of vertices in the clefts.

Hence the overall complexity of the algorithm is $O((k+c) \log (k+c) + n + ch^3)$ in the worst case. If a contour has no match, $h = n$. That is, the contour forms a cleft by itself. Hence, the total complexity of the algorithm is mainly dependent on triangulation and it could be $O(n^3)$.

### 5.2.2 Multiresolution Technique

The first step in the multiresolution tiling algorithm adds vertices to ensure that the number of vertices in each contour is a power of 2. If we have $n$ vertices, the number of vertices in the worst case is at the most doubled. Priority queue implementation to determine $n$ extra vertices on contours of $n$ original vertices takes $O(n \log n)$ vertices.

Decomposition using filter-bank takes $O(n)$ time. Base case triangulation is just on 4 vertices of 2 contours and hence it takes constant time. Reconstruction can be done in $O(n)$ time disregarding local optimization. Local optimization may take $O(n)$ time in the worst case if all $2n$ suspect edges need to be swapped. However, Meyers has proven that local optimization that occurs in each iteration is nearly constant for contours ranging in size from 128 to 1024 vertices [Mey94]. Hence, time complexity is not increased by the local optimization step.

So the overall complexity of the multiresolution algorithm is $O(n \log n)$.

### 5.3 Comparison in Terms of Number of Vertices

The input data used for both is a number of sample points on each contour. For both the algorithms, the contours are interpolated in order to obtain intermediate vertices. The following table distinguishes the interpolation procedures for both algorithms.
<table>
<thead>
<tr>
<th></th>
<th>Piecewise Linear Algorithm</th>
<th>Multiresolution Tiling Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need for interpolation</td>
<td>To have the arc length between a pair of vertices small and equal to the discretization parameter.</td>
<td>To have the number of vertices in each contour as a power of 2.</td>
</tr>
<tr>
<td>Final number of vertices</td>
<td>Any number (not necessarily a power of 2).</td>
<td>Always a power of 2.</td>
</tr>
<tr>
<td>Final edge lengths</td>
<td>Constant = discretization parameter</td>
<td>Variable edge lengths</td>
</tr>
<tr>
<td>Edges split</td>
<td>Each and every edge is discretized into smaller segments.</td>
<td>The first $n$ longer edges are split where $n =$ the number of extra vertices needed to make the total near a power of 2.</td>
</tr>
<tr>
<td>Final number of vertices in all contours</td>
<td>Need not be equal</td>
<td>Have to be equal.</td>
</tr>
<tr>
<td>Number of segments for each split edge</td>
<td>Equal to length of edge/discretization parameter</td>
<td>Equal to length of edge/2. (2 in number).</td>
</tr>
</tbody>
</table>

Table 5.1 Comparison between methods of Interpolation

We have conducted some experiments by using different number of input vertices and have noted down the number of output vertices generated due to interpolation. Table 5.2 generalizes these experiments and a graph has been plotted for the same to compare the behaviors of the interpolation processes of the 2 algorithms. The first column is the average number of input vertices from all slices. This is obtained by dividing the total number of input vertices by the number of slices. The second and third columns give the average number of vertices generated from all slices after interpolation using the two
techniques respectively. We have chosen a uniform discretization parameter of 0.01 for the piece-wise linear technique.

<table>
<thead>
<tr>
<th>Average number of Input Vertices</th>
<th>Average Number of Output Vertices in Piecewise Linear Algorithm</th>
<th>Average Number of Vertices in Multiresolution Tiling Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.75</td>
<td>73</td>
<td>32</td>
</tr>
<tr>
<td>40.66</td>
<td>80.73</td>
<td>64</td>
</tr>
<tr>
<td>45.8</td>
<td>85.67</td>
<td>64</td>
</tr>
<tr>
<td>48.06</td>
<td>123.25</td>
<td>64</td>
</tr>
<tr>
<td>60.01</td>
<td>128.39</td>
<td>64</td>
</tr>
<tr>
<td>70.55</td>
<td>131.23</td>
<td>128</td>
</tr>
<tr>
<td>105.9</td>
<td>133.2</td>
<td>128</td>
</tr>
<tr>
<td>120.38</td>
<td>141.77</td>
<td>128</td>
</tr>
</tbody>
</table>

Table 5.2 Results of Experiments on Interpolation

A graph for the above table has been drawn and it appears as in Figure 5.1
Figure 5.1 Graph of Average Input vs Average Output Vertices

From the graph we conclude that,

- The multi-resolution tiling algorithm has fixed number of output vertices that are powers of 2.
- The output increases with the increase in the number of input vertices beyond a range (The range is between two successive powers of 2).
- The piece-wise linear algorithm increases proportionately with the increase in number of input vertices for our test data.

5.4 **Comparison in Terms of Number of Triangles**

We have compared the triangulation procedures of both the algorithms in Table 5.3.

<table>
<thead>
<tr>
<th>Number of vertices in Initial Triangulation</th>
<th>Piecewise Linear Algorithm</th>
<th>Multiresolution Tiling Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>All the final vertices of clefts of contours.</td>
<td>The lowest resolution contour of 4 vertices.</td>
<td></td>
</tr>
<tr>
<td>Number of vertices in Final Triangulation</td>
<td>Equals the number of vertices in clefts of contours. Lower than the number of input vertices.</td>
<td>Equals the number of input vertices of contours.</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>Number of triangles</td>
<td>Varies from contour to contour on the same surface.</td>
<td>Is the same for all contours of the same surface.</td>
</tr>
<tr>
<td>Triangulation Method Used</td>
<td>By choosing lowest of all costs assigned to edges, triplet of vertices and all sub-curves (of 3 to total number of vertices) of contours.</td>
<td>Triangulating a set of 4 vertices and improving tiling in stages by local optimization</td>
</tr>
<tr>
<td>Goal Function Reduced</td>
<td>Function on area, perimeter and surface area of triangles – area being the prominent one</td>
<td>Function on the sum of edge lengths of triangles</td>
</tr>
</tbody>
</table>

Table 5.3. Comparison between methods of Triangulation

Some results of experiments carried out on the various sets of sample points on various contours to check triangulations are as shown in Table 5.4 and Table 5.5 for piecewise linear and multi-resolution algorithms respectively. The first column for each algorithm indicates the number of vertices to be triangulated. For piece-wise linear reconstruction algorithm, this is the number of vertices on clefts. For the multi-resolution tiling algorithm, this is the number of vertices in the highest resolution that is a power of 2. The second column for both algorithms is the number of triangles using as many vertices.
<table>
<thead>
<tr>
<th>Number of Vertices of Clefts to be Triangulated</th>
<th>Final Number of Triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>27</td>
</tr>
<tr>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>24</td>
<td>21</td>
</tr>
<tr>
<td>36</td>
<td>52</td>
</tr>
<tr>
<td>53</td>
<td>107</td>
</tr>
<tr>
<td>116</td>
<td>244</td>
</tr>
</tbody>
</table>

Table 5.4 Triangulations for Piece-wise Linear Algorithm

<table>
<thead>
<tr>
<th>Number of Vertices in Highest Resolution Contour</th>
<th>Final Number of Triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>64</td>
<td>128</td>
</tr>
<tr>
<td>128</td>
<td>256</td>
</tr>
</tbody>
</table>

Table 5.5 Triangulations for Multi-resolution Algorithm
The graphs for the above tables are as in figures 5.2 and 5.3.

![Triangulation Graph](image)

**Figure 5.2 Graph of Number of Triangles vs Number of Cleft Vertices**

![Triangulation Graph](image)

**Figure 5.3 Graph of Number of Triangles vs Number of Points**

From the above graphs and tables, we observe that,

- For the piece-wise linear algorithm, there is no steady increase or decrease in the number of triangles as the number of cleft vertices go up.
• For the multi-resolution algorithm, the number of triangles is exactly twice the number of vertices in the highest resolution contour.

5.5 Comparison in Terms of Running Times

The following table shows the total algorithm average running times (in seconds) of both the algorithms. The time is measured as time taken from the first step through the last step for both the algorithms.

<table>
<thead>
<tr>
<th>Total Number of Sample Points</th>
<th>Piecewise Algorithm Average Running Time (in seconds)</th>
<th>Linear Average Time (in seconds)</th>
<th>Multiresolution Algorithm Average Running Time (in seconds)</th>
<th>Tiling Average Time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>17</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>187</td>
<td>20</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>408</td>
<td>47</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>769</td>
<td>76</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>913</td>
<td>86</td>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>1059</td>
<td>90</td>
<td></td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.6 Comparisons between Running Times

A graph for comparing the running times has been plotted below.
The running times using the same test data may vary during the day depending on number of users, processes running, etc. So we have taken the average running time over all experiments performed using each test data set. The average running times are as indicated above. From the experiments we can conclude that

- In general, the multi-resolution tiling algorithm runs much faster than the piece-wise linear technique.
- The running times increase with increase in the number of input vertices for both algorithms.

5.6 Comparisons with the Optimal Algorithm

Both the optimal algorithm of [FKU77] and the multiresolution tiling algorithm handle single contours per slice. However, the former has a worst-case complexity of $O(n^3 \log n)$ as opposed to $O(n \log n)$ of the latter for a contour with $n$ vertices.

Both algorithms do not guarantee joining the corresponding local edges of the organs from a pair of contours. Consider Figure B.12 where the human head has been reconstructed using the multiresolution tiling algorithm. The nose region of the head has
not been reconstructed well to depict the true shape of the nose. The nose along with the rest of the head appears as one complete connected surface instead of the nose being a sharp feature standing out separately. This is because there are edges joining samples on the portions of the contours of the nose region and samples lying in contour portions outside the nose region. This is especially a problem if contours show sharp branches or indentations with the optimal as well as multiresolution tiling algorithms.

This is not a problem in case of the piecewise linear algorithm as the nose region is one of the matched contour portions which is tiled separately and independent of the rest of the surface. This would help joining samples lying in portions of contours within the nose region only. This is evident from Figure B.8.

5.7 Conclusions from All Experiments

- The multi-resolution algorithm runs much faster than the piece-wise linear algorithm using the same number of input points on the test cases we conducted. This may be because interpolation gives fewer vertices for multi-resolution method.

- The multi-resolution tiling does not require any user intervention in terms of external parameters such as match – neighborhood, score threshold, discretization parameter, etc. The piece-wise linear algorithm is very sensitive to these parameters and the algorithm may even fail in its basic heuristic proposed that the x-y projections of contours and clefts do not intersect for certain values of parameters. Hence, the choice of such parameters is very tricky.

  For example, the clefts are the unmatched contour portions. The contour matches again depend on the scoring threshold and the distance between corresponding vertices (match-neighborhood). If these parameters are too high, the contours may be reported as fully unmatched. These unmatched contours form clefts that may intersect in 2D rendering the algorithm useless.

- From our experiments, both the algorithm outputs showed good resemblances to the original surfaces. However, the piece-wise linear algorithm showed more resemblance than the multi-resolution tiling algorithm. The ability of the piece-wise
linear technique to handle multiple contours per slice is a clear advantage over the multiresolution technique. This was especially experienced when treating surfaces that had holes and cavities.

- In general, if the sampling density increases, the output is better and shows more resemblance with the original surface for either algorithm. Based on the discretization parameter, we believe that the distance between adjacent samples around 0.01 – 0.06 is a good choice of sampling density.

- The multiresolution tiling does not perform well in case of surfaces having sharp features and/or branchings.
6.0 FUTURE WORK AND CONCLUSION

In this thesis, we have first discussed the traditional algorithms for the surface reconstruction problem. In early years, the reconstruction was carried out efficiently using horizontal cross-sectional contours taken one disc at a time.

As computational geometry concepts received further attention, researchers tried to correlate the problem of surface reconstruction using Delaunay triangulation. Extension of this property is discussed in some algorithms.

We have then touched on theoretical aspects of multiresolution analysis and have shown how theory can be used in developing a more complete mathematical framework for multiresolution analysis. Wavelets are finite windows through which the signal can be viewed. The wavelets can be translated about time in addition to being compressed and widened. Meyers' multiresolution polygon tiling algorithm is unique and it gives an optimal solution to the problem of surface reconstruction.

We have discussed the various pros and cons of using either the multi-resolution tiling algorithm or the piece-wise linear surface reconstruction algorithm. We have thus provided a workbench for surface reconstruction using wavelets to enable further research for a better solution.

An important research area worth exploring is to enhance the multiresolution tiling algorithm to be able to handle multiple contours per slice. Another open problem would be to solve the problem using both algorithms by taking vertical cross-sections of the original solid rather than horizontal ones. To apply Delaunay triangulation between two successive slices and check the experimental results sounds interesting.
REFERENCES


APPENDIX A

REVIEW OF COMPUTATIONAL GEOMETRY TERMS

Surface reconstruction and multiresolution analysis rely heavily on fundamental ideas from computational geometry. In this appendix, we shall review some of the important concepts used throughout the thesis.

A.1 Computational Geometry

Computational geometry is a branch of computer science that deals with algorithms to geometric problems. This field has found vast applications in mathematics, graphics, terrain modeling, geographic information systems and statistics.

CGAL (Computational Geometry Algorithms Library) is a central database of useful computational geometry algorithms developed as a collaboration of many sites in Europe and Israel. CGAL provides robust, efficient and easy to use implementations of geometric algorithms and data structures. The primary aim of CGAL is to make available globally a vast number of algorithmic solutions developed by proprietary vendors and researchers [BY98, BDTY00].

Some of the commonly used terminologies in the field of computational geometry for the problem of surface reconstruction are as explained below:

A.1.1 Voronoi Polygons

Voronoi polygons are one of the fundamental structures satisfying the criterion for the nearest-neighborhood problem. These polygons divided the set of sample points into a number of polygons such that each polygon has exactly one sample point within.

Assume that S contains a finite set of points called P₁, ..., Pₙ in Euclidean space as shown in Figure A.1. We partition the region into a number of polygons such that every polygon is the perpendicular bisector of the sides joining the input points.
Each such polygon is called as the Voronoi polygon or region denoted by $V(P_i)$. The edges of the polygons are called Voronoi edges with its vertices called Voronoi vertices. All the original $n$ points thus belong to a specific region. Each polygon consists of the loci of points that are closest to the point $P_i$ (that lies within) than to any other point. Voronoi vertices are essentially the circum-centers of the triangles formed by joining the $n$ points.

The Voronoi diagram of the points in $S$ is as shown in Figure A.1. Let us take for example the polygon $ahcb$. This polygon consists of one of the $n$ input points $P_8$. Hence, this polygonal region is represented as $V(P_8)$. All the points within $V(P_8)$ are closest to $P_8$ than to any other point. Consequently, each vertex of the triangle $P_1P_2P_8$ is closest to point ‘a’ than to any other vertex. This point ‘a’ becomes the Voronoi vertex or circum-center for that triangle [Aur91, For95].

![Voronoi Diagram](image)

Figure A.1 Voronoi diagram

a,b,...h – Voronoi vertices
Every Voronoi vertex is thus the common intersection point of exactly three edges of the diagram. The polygons V (Pi) are unbounded, if Pi ∈ S lies on the convex hull boundary of S.

The nearest neighbor of a point P_i ∈ S defines a Voronoi edge of the diagram. For example, the nearest neighbor of P_1 is P_8 and a Voronoi edge P_1P_8 exists.

### A.1.2 Delaunay Triangulation

By definition, 3 points form a Delaunay triangle if and only if the circum-circle defined by them contains no other point. Figure A.2 can illustrate this definition. Consider the points P_1, P_2, P_3 and P_4 to be given. Let us generate 2 triangles out of the 4 given points and draw circum-circles around them. Two possibilities exist for generating triangles around the 4 points as shown in figures A.2 (a) and (b). In case of Figure A.2 (a) the circum-circle around triangle P_1P_3P_4 contains point P_2 and thus the triangle is said to be a non-Delaunay triangle. Similarly, the circum-circle around triangle P_1P_2P_4 also contains point P_3. However, as in Figure A.2 (b), the circumsircle around both triangles do not contain any other vertex and each triangle is called a Delaunay triangle.

![Diagram of Delaunay Triangles](image)

Figure A.2 Two triangles that violate (a) and honor (b) the Delaunay criterion
The boundary edges on the Delaunay network form the convex hull, which is the smallest area convex polygon to contain all of the vertices (described next).

### A.1.3 Convex Hull

The convex hull of a set of points is the smallest convex set containing all the sample points. That is, a smallest polygonal area that is of the smallest length and so that any pair of points within the area has the line segment between them contained entirely inside the area. A number of algorithms have been developed in order to find the convex hull of the points or of the polygon, the most common amongst them being the Graham’s Scan algorithm [Gra72], Jarvis March algorithm [Jar73] and the Incremental algorithm.

Consider the points shown in black circles given as input as shown in Figure A.3. The convex hull for the set of points is as shown. This is the smallest polygonal area to include all the points. A line segment between any two points would lie entirely within the hull.

![Convex Hull](image)

**Figure A.3 Convex hull for the set of points**

Convex hulls can be used to solve many interesting problems such as the farthest-pair problem. Given a set of $n$ points in the plane, and it is desired to find the two points
whose distance from each other is maximum. These two points must be the vertices of the convex hull.

A.1.4 Medial Axis

The medial axis is the set of line segments joining the centers of circles that touch the shape tangentially at two or more points. The radii of the circles may be different and would be such that the circles are tangential to the shape in its interior [Lee82]. These discs do not lie outside the boundary of the shape. For example, for a circle the medial axis is defined as its center. For an ellipse, it is its major axis. The medial axis provides the local axis of symmetry of the shape everywhere.

![Diagram of a curve and its medial axis]

**Figure A.4** Medial axis of a curve

Let us take for example Figure A.4. The curve $\Gamma$ consists of sample points 1, 2,...,11. We need to construct the medial axis for this curve. Circles are drawn with centers at $m_1$, $m_2$, $m_3$ and $m_4$ such that these circles touch the curve $\Gamma$ at two or more points. The positions of these centers and radii of circles are arbitrary, so long as they touch the boundaries of the curve. Thus, the medial axis consists of points $m_1$, $m_2$, $m_3$, $m_4$ and point 6 (since point 6 is the extreme point that completes the shape of the curve and
provides continuity between the 2 portions of the curve). Conversely, given just the
points on medial axis, it is possible to reconstruct the curve as the union of discs with
centers at these points and their corresponding radii. The medial axis of a shape is said to
approximate its Voronoi diagram. The medial axis is a subset of edges from the boundary
of the Voronoi diagram.
APPENDIX B
SAMPLE SCREEN SHOTS OF ALGORITHMS

This appendix contains screen shots of the outputs of the piecewise linear surface reconstruction and the multiresolution tiling algorithms. Screen shots for the initial slices/contours and the final reconstructed surface have been taken for the piecewise-linear algorithm.

For the multi-resolution tiling algorithm, the original slices/contours, the lowest resolution contours, the triangulation of the base case contours and the final complete reconstructed surfaces are attached.
Figure B.1 Original Contours of Hip Bone

Figure B.2 Fully Reconstructed Hip Bone using Piecewise Linear Algorithm
Figure B.3 Original Contours of Hip Bone

Figure B.4 Lowest Resolution Contours
Figure B.5 Triangulation of Lowest Resolution Contours

Figure B.6 Fully Reconstructed Hip Bone using Multiresolution Tiling Algorithm
Figure B.7 Original Contours of Human Head

Figure B.8 Fully Reconstructed Head using Piecewise Linear Algorithm
Figure B.9 Original Contours of Human Head

Figure B.10 Lowest Resolution Contours
Figure B.11 Triangulation of Lowest Resolution Contours

Figure B.12 Fully Reconstructed Head using Multiresolution Tiling Algorithm
VITA AUCTORIS

Shobha Iyer was born in Coimbatore, India in 1975. She received her Bachelors degree in Computer Science from Bombay University, India in 1997. She was working till mid 2000 as a Senior Systems Engineer at Tata Infotech Limited, Mumbai, India. She is currently a candidate for the Master’s degree program in Computer Science at the University of Windsor, Ontario, Canada. She is expected to pursue a career in Information Technology building on her academics and professional experience.