Surface Reconstruction

Eliana María Vásquez Osorio

UNIVERSIDAD EAFIT
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS
MEDELLÍN
2003

Surface Reconstruction

Eliana María Vásquez Osorio

Final project presented to obtain the B.Sc. Diploma in Computer Science

Adviser
Prof. Dr. Oscar E. Ruiz

UNIVERSIDAD EAFIT
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS
MEDELLÍN
2003

Acceptation note

Jury: Dr. Georgios Sakas

Jury: Dr. Grigorios Karangelis

Adviser: Dr. Oscar Ruiz

Medellín, November 10^{th} 2003

The journey of a thousand miles begins with one step.
Lao-Tse

Acknowledgments

I would like to thank Dr. Oscar Ruiz for his guide and support during the realization of this project and he has taught me a word of things during my time at the CAD/CAM/CAE Laboratory. Besides than being my superviser, he has been a model of discipline and hard work.

I am deeply indebted to Dr. Georgios Sakas and Dr. Grigorios Karangelis. Dr. Georgios Sakas gave me the opportunity to be cooperating at Fraunhofer Institute (FhG-IGD) in the years of 2000 and 2002. Dr. Grigorios Karangelis offered me the support I needed, not only in the technical background, but in the personal field. To both of you, my deepest thanks.

The members of the CAD/CAM/CAE Laboratory deserve a recognition for their helpful contributions for this project and the good moments I shared with them. They are Dr. Carlos Cadavid, Dr. Manuel Julio García, Miguel Granados, Sebastián Peña, Carlos Toro, Juan Santiago Mejía, Miguel Henao, Mario Gómez, Leidy Suárez and Carolina Quintana.

I want to express special thanks to Mónica Henao Cálad. At the beginning of my studies she brought me the opportunity to work as a research assistant at her Laboratory. That was the best place and the best person to initiate me into research.

I owe infinite gratitude to the person who has been always next to me during all my life, and she is my mother Aracelly Osorio. Without her support, guide and love, my life could not be as happy as it has, and surely, I would never get enough strength to be here. My father, Bernardo Vásquez, started the research interests in my life by teaching me how to disarm devices when I was little. Thanks to him, now I have a north in my career. My whole family, that is part of the kernel of my life, encourage me to keep moving up. Thanks to all of them.

Last, but never least, I want to say *Thank You* to each one of my friends who lent me a hand or made me laugh whenever I needed it. Thanks to

Roberto, Jannis, Bibiana, Natalia, Angélica and Diana. I really appreciate everything that comes from you!

Glossary

The source of some of the definitions included here is [33].

CCW: Stands for counter clock-wise sense with respect to the normal of the plane where the points, polygon or contour lies.

CW: Stands for clock-wise sense with respect to the normal of the plane where the points, polygon or contour lies.

Digilab: An environment and language for manipulation of 3D digitizations developed in the CAD/CAM/CAE Laboratory in 1997 by Sebastian Schrader as final degree project. It has been improved along these years to allow more complex shapes to be reconstructed. ([34])

ISM: Abbreviation that stands for Implicit Surface Method. Refer to part III of this document.

Manifold: Topological space which is locally Euclidean. Around every point, there is a neighborhood which is topologically the same as the open unit ball in R^n. Manifolds may be open or closed. Closed manifolds are known as compact manifolds without boundary. Open manifolds may or may not contain its boundary and they are called non-compact manifolds with or without boundary.

PSM: Abbreviation that stands for Polyhedral Surface Method. Refer to part II of this document.

Surface: Two-dimensional manifold embedded in the three-dimensional Euclidean space. Also defined as the locus of a point moving with 2 degrees of freedom. See section 2.2.

Simple polygon: A polygon is said to be simple (or Jordan) if the only points of the plane belonging to two polygon edges of the polygon are the polygon vertices. Such a polygon has a well defined interior and exterior. Simple polygons are topologically equivalent to a disk.

Specific to the Polyhedral Surface Method

2DSS: Abbreviation that stands for the 2D Shape Similarity algorithm developed in the CAD/CAM/CAE Laboratory ([38], [39]). This algorithm is used to match the contours on two adjacent levels. The matched contours are used as input for the *PSM*. See section 3.

B+G method: Abbreviation that stands for the method on which the PSM is based. B+G refers to Boissonnat and Geiger initials. See section 5.

Delone Triangulation: Triangulation of the convex hull of a set P of n points in which every single triangle does not contain, not even in its circumference, any other point in P, except its vertices. See section 4.2.

DE$_{ij}$: Delone edge related to points p_i and $p_j \in P$. See section 4.2.

DT: Stands for Delone triangulation. This symbol is used as the Delone Triangulation defined over the set of points of the contours on a level. See section 4.2.

DT$_{ijk}$: Delone triangle related to points p_i, p_j and $p_k \in P$. See section 4.2.

DV$_i$: Delone vertex related to point $p_i \in P$. See section 4.2.

ER: Stands for External Region.

EVS$_i$: External Voronoi skeleton related to $level_i$. See section 5.3.1.

External Region: Union of Delone triangles such that they are outside any contour on a given level. See section 5.

External Voronoi skeleton: Voronoi skeleton such that all the Voronoi edges that belong to it, are related to an internal Delone edge. An internal Delone edge lies in the interior of any contour in the given level. See section 5.3.1.

IR: Stands for Internal Region.

Internal Region: Union of Delone triangles such that they stand inside any contour on a given level. See section 5.

Internal Voronoi skeleton: Voronoi skeleton such that all the Voronoi edges that belong to it, are related to an external Delone edge. An external Delone edge lies outside all the contours in the given level. See section 5.3.1.

IVS$_i$: Internal Voronoi skeleton related to $level_i$. See section 5.3.1.

Joint Voronoi Diagram: Planar graph resulting from the intersection of the orthogonal projections of two Voronoi diagrams (belonging to level 1 and level 2) on a common plane. There are three kinds of nodes: T_1, T_2 and T_{12}. A tetrahedron is related to every node in this graph. See section 5.4.

Joint Voronoi $\mathbf{T_1}$ node: Node of the Joint Voronoi Diagram related to a Voronoi vertex belonging to level 1. The base of its corresponding tetrahedron lies on level 1. Its neighbors are Joint Voronoi T_1 and T_{12} nodes. See section 5.4.

Joint Voronoi $\mathbf{T_{12}}$ node: Node of the Joint Voronoi Diagram related to an intersection of two Voronoi edges, one of them belonging to level 1 and the other one to level 2. Its related tetrahedron is built using the Delone edges corresponding to both Voronoi edges. It could be seen as an inclined triangular-based pyramid. See section 5.4.

Joint Voronoi $\mathbf{T_2}$ node: Node of the Joint Voronoi Diagram related to a Voronoi vertex belonging to level 2. The base of its corresponding tetrahedron lies on level 2. Its neighbors are Joint Voronoi T_2 and T_{12} nodes. See section 5.4.

Mapping group: It is a set of contours belonging to two adjacent levels such that their shapes are similar and their projections over a common plane overlap. See section 3.

Medial Axis: Set of points that are located at the same distance to two or more points of a contour. !@ !@

mg: stands for mapping group. See section 3.

MG: set of all the mapping groups created between two consecutive levels. See section 3.

Non-solid region: The region that lies between contours placed in two adjacent levels in a tree, such that outer-most contour correspond to a internal wall. See section 3.1.

Polytope: Finite region of n-dimensional space enclosed by a finite number of hyperplanes. Also may be defined as the convex hull of a finite set of points. It is composed, then, by points, lines segments, in general by i-faces, where $i = 0..n-1$.

Simplex: A simplex represents the simplest possible polytope in any given space. The boundary of a k-simplex in k-dimensional space, has $k+1$ 0-faces (vertices), $k\frac{k+1}{2}$ 1-faces (edges), and $\binom{k+1}{i+1}$ i-faces. The 1-simplex is a line segment. The 2-simplex is a triangle. The 3-simplex is a tetrahedron.

Solid regions: The region that lies between contours placed in two adjacent levels in a tree, such that the outer-most contour correspond to a external wall. See section 3.1.

T_1 node/tetrahedron: See Joint Voronoi T_1 node

T_{12} node/tetrahedron: See Joint Voronoi T_{12} node

T_2 node/tetrahedron: See Joint Voronoi T_2 node

Tetrahedron: Simplest polyhedron in 3D, or 3-simplex. It is created with four points, and it is composed by four triangular faces. In an intuitive way, it could be seen as a triangular-based pyramid.

Triangulation: Triangulation is the division of a surface or a plane polygon into a set of triangles.

Voronoi Diagram: The partitioning of a plane with n points into n convex regions such that each region contains exactly one point and every point in a given region is closer to its central point than to any other. See section 4.1.

$\mathbf{VE}_{ij}$: Voronoi edge related to points p_i and $p_j \in P$. See section 4.1.2.

$\mathbf{VV}_{ijk}$: Voronoi vertex related to points p_i, p_j and $p_k \in P$. See section 4.1.3.

$\mathbf{VR}_i$: Voronoi region related to point $p_i \in P$. See section 4.1.1.

Voronoi skeleton: Subset of Voronoi edges such that its related Delone edge does not belong to any contour. There are two kinds of skeletons, the internal Voronoi skeleton *IVS* and the external Voronoi skeleton *EVS*. See section 5.3.1.

Specific to the Implicit Surface Method

Boundary constraint: Constraint whose h-value is set to zero. See section 9.1.

Constraint: A pair composed by a location point c_i and its correspondent h-value h_i. See section 8.2.

External constraint: Constraint whose h-value is set to -1. See section 9.1.

Internal constraint: Constraint whose h-value is set to +1. See section 9.1.

LU-decomposition: A procedure to calculate a factorization of square matrices. It calculates for a given matrix A, two matrices, L and U, such that $A = LU$. L is a lower triangular matrix and U is an upper triangular matrix. If A has any zero in the diagonal, then pivoting is required.

RBF: stands for Radial Basis Functions.

Radial basis functions $\phi(x)$: Circularly symmetric functions centered at a particular point. See section 8.2.

Thin-plate interpolation: Method used to solve the scattered data interpolation problem based on variational functions. See section 8.2.

Zero-set: Given a function f defined in a domain D, the zero set is a set composed by all the points $p \in D$, such that f evaluated on p is zero. $zero\_set(f) = \{p, p \in D | f(p) = 0\}$.

Contents

List of Figures

List of Algorithms

Part I

Introduction and Common Background

Introduction

For this project, two methods to solve the surface reconstruction problem were implemented. The target of both methods is to build a surface in 3D, starting from a set of planar samples. These samples could come from diverse sources, e.g. medical imaging, digitization of objects and GIS systems. The final result of the methods is a set of triangular faces under a B-Rep model representing the reconstructed surface.

The first method, the *Polyhedral Surface Method* or PSM is described in part II of this document. It is based on Jean-Daniel Boissonnat ([5]) and Bernhard Geiger's ([19]) research, done for the *PRISME* project in 1993, at INRIA, Sophia-Antipolis, France.

The second method, the *Implicit Surface Method* or ISM, described in part III of the document, is based on Greg Turk and James F. O'Brien's research work at the Georgia Institute of Technology in 1999, and it was originally used for shape transformation ([44]). It was developed as a tool for the physicians to segment medical images in a virtual Simulator of Radiotherapy Treatment Planning.

A reconstructed surface may represent a human organ, a tumor, a sculpture to use as part of a virtual gallery, a topographic terrain, etc. The range and variety of these representations and its applications constitute one of the reasons of the popularization of the use of computerized representation of 3D shapes in several areas, as diverse as manufacturing, virtual simulation, scientific exploration, medicine, special effects, games, virtual worlds, and so forth.

In medicine, one of the uses of the reconstructed surfaces is as support for the creation of personalized-prosthesis. The surface of the prosthesis is made based on the forms of the patient, leading to a prosthesis that fits almost perfectly to the peculiarities of the patient. Another important application in medicine, is the use of reconstructed organs and regions of patients as

complementary diagnostic aids.

Another important field where the computerized representation of surfaces is being widely used, is reverse engineering. The purpose is to create CAD models from real existing objects. These CAD models may afterward, be manufactured in series. This is useful when a part of an "old" machine needs to be redone for replacing, or when a hand-made object is going to be produced in series. In the hand-made objects case, collaborative design may be done, and several sculptors, designers, artists and craftsmen may cooperate to create a single object.

Chapter 1

Literature Review

Methods for surface reconstruction have existed since nearly 30 years. As said, the target is to associate a polyhedral shape to the given input, that is usually formed by points or contours.

The solutions have arised mainly from two fields: computational geometry and computer graphics. The *Polyhedral Surface Method* is classified in the computational geometry field and the *Implicit Surface Method* in the computer graphics field.

Included in the computational geometry field, several methods have been proposed. Much of them use geometric structures such as Voronoi Diagrams and Delone Triangulation in 2 and 3 dimensions. The α-shapes, a classical method developed by Edelsbrunner ([14]), is based on the Delone Triangulation. The surface is a simplicial subcomplex of the Delone Triangulation, where the parameter α is used to select the Delone simplices that are part of the reconstructed surface. α is manually tuned and its optimum value depends on the density of the sample. Barequet and Sharir ([3]) use a technique to match parts of the contours. The the no-matched parts, named clefts, form 3D polygons that lie partially on one level and partially on the other level. Those clefts are triangulated, and a heuristic based on the *minimum spanning tree* is used to interpolate between the no simply connected regions. Amenta and Bern ([1]) developed a method based on the Voronoi Diagram and the Delone Triangulation. Some Delone triangles are filtered by a subset of Voronoi vertices, called Poles. A pole is the farthest Voronoi vertex in the Voronoi region related to a Delone Vertex. A Delone triangle whose circumsphere contains a pole is eliminated and the set of Delone triangles remaining, form the final result that is a piecewise-linear surface.

Respect to the second field, computer graphics, Hoppe ([23]) created the first and most widely known method. It estimates a tangent plane at each sample point using the k nearest sample points. Then it creates a signed distance function that assigns to any point p in the space the distance between p and the plane of the closest sample point. After that, the *marching cubes* algorithm is used to extract the zero set of the signed distance function to, lastly, obtain a piecewise linear or polyhedral surface. Curless and Levoy ([11]) also propose the calculation of a signed distance function by calculating a weighted sum of contributions of the samples. The function is lastly discretized on voxels in order to avoid the marching cubes step.

Nowadays, a new focus has became interesting, and it is based on the calculation of a variational function that implicitly defines a surface. The first approach was proposed by Savchenko et al. ([40]), who researched in the creation of implicit surfaces from measured data such as range data or contours. This research work was later improved and applied to shape transformation by Turk and O'Brien ([44]). They proposed the replacement of the signed distance function for a variational implicit function, that allows the transformation to be performed in just one step. They also referenced the possible application of variational implicit functions to surface reconstruction. In a recent work ([45]), some algorithms to determine external, normal and internal constraints were formalized, and aspects of modeling and rendering of implicit surfaces were included as part of the research. Some authors attack critical points of this particular method and apply it to different fields. That is, Carr et al. ([9]) who use the method to create surfaces from a noisy point-cloud and automatically repair meshes. They use a representation based on the Radial Basis Functions, and show advantages for mesh simplification and remeshing applications. Also Morse et al. ([24]) worked on this method, improving the time and space requirements, by changing the Radial Basis Function by a Compactly-supported Radial Basis Function, while keeping the optimum characteristics of the created implicit surface.

Chapter 2

Common Technical Background

2.1 Input data

The input data is a set of planar samples. These samples are closed simple planar polygons, named in this document as contours. The contours are the boundaries of the regions resulting from cutting the object to be sampled by planes. The whole set of contours is organized by levels, where each level corresponds to a cutting plane. For PSM, the contours must lie on parallel levels, while ISM does not have a restriction for the global positioning of them.

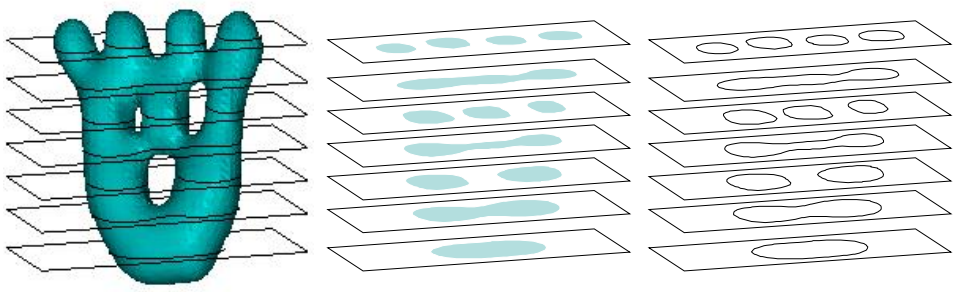


Figure 2.1: Creation of seven parallel levels from a simple object

In practical terms, these samples are obtained using different methods, commonly classified into two groups: by-contact and by-remote. When data is sampled using a by-remote method, the intensity of a determined property

present in the object is measured. It could be color, temperature, distance from a given location, radiation, etc. In these cases, a value is assigned to every spatial position sampled, and algorithms, depending on the sample method used, are executed to transform the data as required. Medical images (CT, MRI, X-Ray...), range images and level maps are examples of this kind of sampling. An articulated arm is commonly used when the sampled data is taken using a by-contact method. The sampled points are the result of solving an equation system of the positions of the joints in the device at a certain moment.

Respect to the density of the sample, there is the sampling theorem stated by H. Nyquist situated in information theory and signal processing. It states that the sampling frequency should be at least twice the highest frequency contained in the signal ([33]). This concept in surface reconstruction terms, states that the distance between the sampled points must be at least twice the smallest detail that will be reconstructed. A set of points is said well-sampled if the points accomplish the sampling theorem.

2.2 Surface Definition

A surface is mathematically defined as a 2-Manifold M embedded in R^3, where every ball B centered in any point p belonging to the surface, and with a radius r tending to zero, intersects M in a region that is isomorphic to a plane disc ([35]). In an intuitive way, it is a continuous set of points, where the closest neighborhood around every point belonging to the surface, approximates a plane. Some authors also define it as the locus of a point moving with 2 degrees of freedom.

To deal with a consistent problem, the object from where the samples are taken, must be a closed and bounded subset of R^3, whose boundary is an orientable and non-self-intersecting surface. This assumption is set by Ruiz and Cadavid ([36] and [37]) based on Morse theory ([25]).

2.3 Surface Representation

To handle the complete set of points belonging to a surface is not feasible, because this set is infinite. For this reason, an approximation of the original surface is used and handled. Geometric modeling research area creates and

improves ways to represent and handle such surfaces ([26], [30]). Some of the most used models are Boolean models, space-partitioning models, such as octrees or quadtress, and boundary models.

2.3.1 Boolean models

The boolean model is a procedural or an unevaluated model. The only available information is how to operate simple shapes, called primitives, to create the modeled object. Boolean operations, based on set theory, are used to operate the primitives. The basic Boolean operations are union ($\cup$), intersection ($\cap$) and difference ($-$). The geometric and topological information is only known when the whole model is computed.

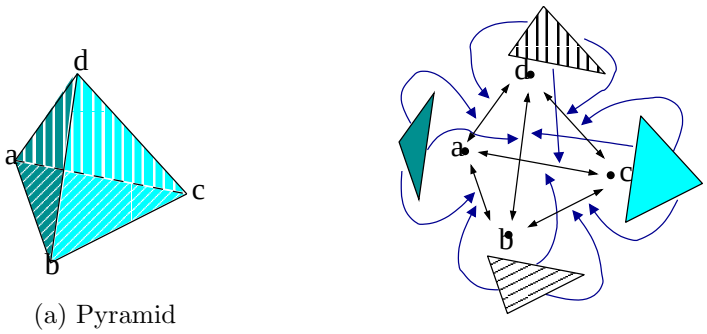
2.3.2 Space-partitioning models

These models are also known as *cell decomposition models*. They decompose the object to be modeled into separate pieces, so that each piece is easier to describe than the original. This is a special case of Boolean models, whose operations are limited to union. A wild-known space-partitioning model is the spatial-occupancy enumeration, where cells are cubes and they are located on a fixed grid. The resolution of the model depends on the cell size. The *octree encoding* is a method commonly used for the spatial-occupancy enumeration. This encoding is based on a recursive subdivision of the modeling space in octants (8 cubical regions). For each octant, if it is completely inside or outside the object, then it is marked as full or empty. If it is not completely inside nor outside the object, it is divided into octants, and they are recursively evaluated. The recursion stops when a octant is completely full or empty, or when the deepest level in the recursion is reached. Then, the cell is mark as full or empty.

2.3.3 Boundary Representation

The target of this models, also known as *B-Rep models*, is to represent the complete object as an organized collection of surfaces. This model is supported on a graph, which emphasizes on the topological information, using an structure made of data pointers linking the faces, edges and vertices of the modeled object. The object is represented as a list of its faces and their respective surface equation. The edges of these faces are represented

by a curve equation, and they have pointers to their vertices and neighbor faces. A simplified B-Rep model is composed by a set of triangular faces and straight edges connecting two vertices (figure 2.3.3), where through-holes are not present. The described structure is used in the CAD/CAM/CAE Lab. and it is also used in the implementation of both methods. A format for a file was defined as part of this project. It is shown in appendix 12.3.



(a) Pyramid

(b) B-Rep model

Figure 2.2: B-Rep model of a pyramid

Part II

Polyhedral Surface Method

Introduction for *PSM*

The *Polyhedral Surface Method* or *PSM*, reconstructs a surface that is a flat-faced polyhedron, from a set of contours that lie on parallel planes or levels. The process is based on geometric closeness, supported over two geometric structures, the 2D Delone Triangulation and the 2D Voronoi Diagram. There is no limitation on the distance between the levels. Without loss of generality the levels are considered as parallel to the XY plane.

The B+G method was developed by Jean-Daniel Boissonnat ([5]) at INRIA and later improved by Bernhard Geiger ([19]). The basic idea of the *PSM* is similar to the voxel-heuristic, where the interior of each contour on each level is divided in small parts, and then, every part belonging to one level is connected to the nearest part on the consecutive level. Notice that in the interior of the contours a division is done when two adjacent parts are matched to two different contours on the consecutive level, creating a "natural" surface between the levels. The B+G uses the 2D Delone Triangulation of the contour points on each level, and then, the corresponding Delone triangles in the interior of each contour on each level are linked by building tetrahedrons.

In the following chapters, a review of the 2D shape similarity algorithm is given. After that, a technical background on the used structures and a review of the B+G method is provide for the sake of completeness. The *PSM* is then explained in detail including the changes done, and finally some results of this method applied to some set of contours are shown.

Chapter 3

Background. The 2DSS algorithm

The surface reconstruction problem starting from parallel contours has been attacked in the CAD/CAM/CAE laboratory since 1996. In 1996 and 1997, HormaCAD and Digilab were implemented; these applications were the first attempt to solve the surface reconstruction problem for simple cases. The 2D shape similarity algorithm, named as $2DSS$ in this document, is the result of years of experience and it was developed in the CAD/CAM/CAE Laboratory ([38], [39]). An algorithm to create surfaces is required as last step in the surface reconstruction process. A version of the method on which the *PSM* is based, named as B+G in this document and implemented at INRIA, was used for this purpose as a library in Digilab. This version gave positive results. However, some defects were identified. The solution of those defects and the implementation of a domestic surfacing algorithm constituted the seed of the *PSM*.

The 2DSS, 2-D shape similarity, is an algorithm that seeks for a partition of the original set of contours, such that subsets of contours belonging to two different levels, with similar 2D shapes and whose projections over a common plane overlap, are set together. These subsets of contours are known as mapping groups mg, and they declare which subsets of contours optimally match between them.

The input to this algorithm is a set of contours that lie on parallel planes. Every contour is a simple, closed and planar polygon defined by at least three different points. The contours are grouped in levels, where each level corresponds to a plane, and the distance between the levels is usually constant

along the whole model. The contours that lie on a level may be nested, but never intersect among them. If contours are nested, it means that the object that is being reconstructed may contain fold or internal cavities. Under this context, a contour may represent an internal or external wall. When two or more contours are nested, they describe solid or non-solid regions. (section 3.1).

The complete surface reconstruction process is composed by five steps:

1. Contour orientation and inclusion calculation.
2. Calculation of 2D-similar composed shapes (mapping groups).
3. Post-processing of mapping groups.
4. Skin construction.

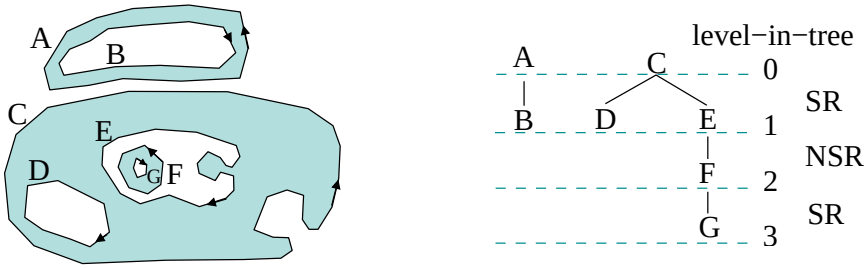
A brief description of the process steps is given here. The material in this section is from a previous work done in the CAD/CAM/CAE Laboratory ([38] and [39]).

3.1 Contour orientation and inclusion calculation

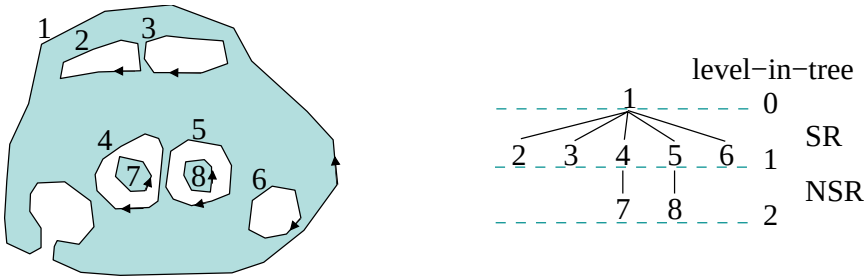
Each level is preprocessed to ensure that the sense of its contours is correct. The contours must be oriented, such that the interior of the model being reconstructed lies to the right of the edges. It implies that contours representing external walls are oriented CCW, and contours representing internal walls are CW. Based on this orientation, the area of every contour is signed according to its sense, CCW implies a positive area and CW a negative one.

In this step a forest is built for each level, where a tree in a forest represents the hierarchical relation between the nested contours (figure 3.1). Notice that contours set in an even level-in-a-tree have positive area and its sense is CCW, because they represent external walls. The contours which are set in an odd levels-in-a-tree have negative area, its sense is CW and correspond to internal walls. The term level-in-a-tree refers to the the depth of the contour in a tree.

Two kinds of regions are then identified between the contours on a level:



(a) Orientation of the contours and forest of level i



(b) Orientation of the contours and forest of level j

Figure 3.1: Contours on a level: orientation of contours and forest.

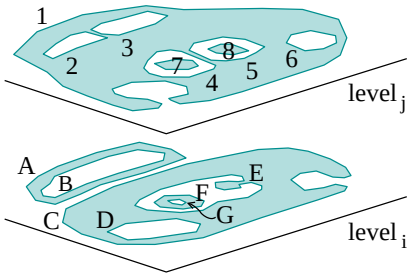
Solid regions SR**:** 2D region that lies between contours, set in two adjacent levels-in-a-tree, such that the outer-most contour correspond to a external wall. It represents a polygon with holes. The regions (A, B), (C, D, E), (F, G), $(1, 2, 3, 4, 5, 6)$, (7) and (8) that are presented in figure 3.1 are solid.

Non-solid regions NSR**:** 2D region that lies between contours, set in two adjacent levels-in-a-tree, such that outer-most contour correspond to a internal wall. The regions (E, F), (B), (D), (G), $(4, 7)$, $(5, 8)$, (2), (3) and (6) that are shown in figure 3.1 are non-solid.

3.2 Calculation of 2D-similar solid and non-solid regions

The 2DSS generates a list of non-repeated mapping-groups. In this step, sets of regions (solid and non-solid) of a level are matched against similar ones on the adjacent level, forming mapping groups mg. The set of all the mapping groups, denoted as MG contains all the mapping groups created for two consecutive levels. MG must be filtered before the creation of the skin is done (section 3.3).

The criterion to determine if two regions create a matching group mg is based on the percentage of overlap of the two regions. The hypothesis is that the intersection of the projections of two regions belonging to adjacent levels, which have a real relation and therefore must be linked by a surface, represents a significant portion of the area of each region (figure 3.2). A *threshold* is used to decide when the regions match or not. The tests done over several data sets showed a great deal of stability of matches with respect to the threshold used and a robust performance.



$$\begin{aligned}
\{(A\ B), (C\ D\ E)\} &\text{ vs. } \{(1\ 2\ 3\ 4\ 5\ 6), (8)\} \\
\{(F\ G)\} &\text{ vs. } \{(7)\} \\
\{(B)\} &\text{ vs. } \{(2), (3)\} \\
\{(D)\} &\text{ vs. } \{(\Phi)\} \\
\{(E\ F)\} &\text{ vs. } \{(4\ 7), (5\ 8)\} \\
\{(G)\} &\text{ vs. } \{(\Phi)\} \\
\{(\Phi)\} &\text{ vs. } \{(6)\}
\end{aligned}$$

Figure 3.2: Mapping groups of the contours on level i and j

Every contour in the levels being matched, level i and j in figure 3.2, must appear in at least one mapping group. For this reason an empty region Φ is defined. It is used when a region R is not related to any other in the adjacent level, and the mapping group $mg = \{R\}$ vs. $\{\Phi\}$ or $mg = \{\Phi\}$ vs. $\{R\}$ is created. For example, in figure 3.2, the mapping groups $\{(D)\}$ vs. $\{(\Phi)\}$, $\{(G)\}$ vs. $\{(\Phi)\}$ and $\{(\Phi)\}$ vs. $\{(6)\}$ were created using Φ.

The combinatorial algorithm of testing all possible regions on both levels against each other to determine the mapping groups is too expensive. A linear approximation to the solution was implemented in the CAD/CAM/CAE

Laboratory. See [38] and [39] for details.

3.3 Post-processing of mapping groups

A contour may appear in more than one mapping groups in MG. This repetition must be filtered to avoid the creation of the same skin more than once. In addition, mapping groups representing impossible topologies must be also eliminated from MG.

3.3.1 Filter of mapping groups involving internal walls

A mapping group mg involving internal walls requires one of these actions:

1. If a contour B representing an internal wall is mapped to any contour k, then k must be an internal wall, and their parents in the trees (of each level) must be also mapped. If these conditions are not satisfied by B, k or their parents, then this mapping group is considered invalid, and it is discarded.

2. If a contour B representing an internal wall is mapped to Φ and

$$\frac{Area(B \cap Q)}{Area(B)} \geq (1 - threshold),$$

   for every contour Q that represents an external wall on the level opposite to B, then the contour is tiled. The contour G in the mapping group $\{(G)\}$ vs. $\{(\Phi)\}$, shown in figure 3.2, is tiled.

3. If a contour B representing an internal wall is mapped to Φ, and

$$\frac{Area(B \cap Q)}{Area(B)} \leq threshold,$$

   for every contour Q that represents an external wall on the level opposite to B, then this mapping group is ignored, B is not discarded nor tiled. The mapping groups $\{(D)\}$ vs. $\{(\Phi)\}$ and $\{(\Phi)\}$ vs. $\{(6)\}$ shown in figure 3.2 require this action.

3.3.2 Filter of redundant mapping groups

The intersection of all the mapping groups in MG is not empty. The reason is that a contour that represents an internal wall participates in a group mapping solid regions and in another group mapping non-solid regions. This redundancy must be eliminated. Some definitions are important before the filter of redundant mapping groups:

Level of a mapping group $level(mg)$**:** It is the lowest level-in-the-tree of the contours belonging to such mapping group.

Ordering of mapping groups $\prec$**:** An ordering $\prec$ may be defined using the level of a mapping group. A mapping group mg_i is said lower than other mg_j if $level(mg_i) < level(mg_j)$.

The mapping groups are processed in ascending order. First, the lowest mapping group $mg_{lowest} \in MG$ is removed from that set. Then, the intersection between mg_{lowest} and every mapping group mg remaining in MG is removed from mg. After that, the new lowest mapping group $mg_{lowest} \in MG$ is identified and removed, and the procedure is repeated until MG becomes empty. The set of processed mapping groups are not redundant.

3.4 Skin construction

A surfacing-algorithm is used to create the skin that joins the mapping groups created in the previous step. There are several algorithms that have been developed to create a polyhedral surface that joins different set of contours. The B+G algorithm ([19] and [5]) was chosen and used as complement of this approach, and it constitutes the seed of this project.

Chapter 4

Theoretical Basis for *PSM*

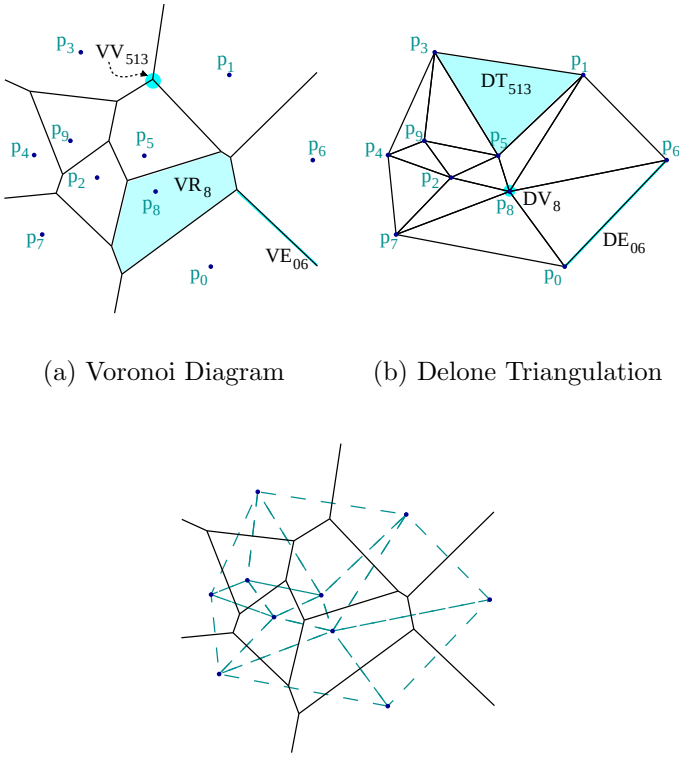
The *PSM* is based on the 2D Delone Triangulation and the 2D Voronoi Diagram. These structures are *dual*. This means that both diagrams represent the same information in different ways. Both structures give a closeness criterion, and establish a "neighborhood" relation between the given set of points.

In the following sections a definition of both structures is given and some algorithms to calculate them are referenced, emphasizing in the *Random Incremental Delone Triangulation algorithm* based on the *Delone Tree*. For more information check a survey in Voronoi Diagrams and Delone Triangulations ([16], [2]).

4.1 Voronoi Diagram

Let P be a set of n points p_0, p_1, $\ldots p_n$ in R^2, with $n \geq 3$ and with no more than three co-circular points. The Voronoi Diagram $Vor(P)$, also known as the Dirichlet Tessellation, is the partition of the plane in n convex regions, such that each region contains one point $p_i \in P$ and all the points in R^2 that are closer to p_i than to any other point $p_j \in P$. (Figure 4.1(a)).

The Voronoi Diagram has three kinds of elements: Voronoi regions VR_i, Voronoi edges VE_{ij} and Voronoi vertices VV_{ijk}, as shown in figure 4.1(a). Each of these elements are related to one, two or three points in P respectively.



(a) Voronoi Diagram (b) Delone Triangulation

(c) Delone Triangulation
and Voronoi Diagram

Figure 4.1: The *2D Voronoi Diagram* and the *2D Delone Triangulation* of 10 points in R^2

4.1.1 Voronoi region

A voronoi region, VR_i, is a convex area containing $p_i \in P$, limited by two or more Voronoi edges and composed by all the points in R^2 that are closer to p_i than to any other $p_j \in P$ with $p_i \neq p_j$. There are closed and open Voronoi regions; the open Voronoi regions are related to points that belong to the boundary of the convex hull of P, and in a similar way, the closed Voronoi regions are related to points that are in the interior of the convex hull. A Voronoi region is defined in equation 4.1.

$$VR_i = \left\{q, q \in R^2 | d(q, p_i) < d(q, p_j); p_i \neq p_j; p_i, p_j \in P\right\} \tag{4.1}$$

where $d(q, p_j)$ represents the Euclidean distance. Notice that points that are at the same distance to more than one point $p_i \in P$ do not belongs to a Voronoi region; implying that all the Voronoi regions are disjoint. The union of all Voronoi regions for the set of points P is denoted by $V(P)$ (Equation 4.2).

$$V(P) = \{q | q \in VR_i\} \tag{4.2}$$

where VR_i is the Voronoi region related to the point $p_i \in P$. The points that do not belong to $V(P)$ are Voronoi vertices or belong to a Voronoi edge.

4.1.2 Voronoi edge

A Voronoi edge VE_{ij} is placed between two adjacent Voronoi regions VR_i and VR_j and is related to both regions. All the points belonging to VE_{ij} are at the same distance from both points p_i and $p_j \in P$. It is possible to say that two points p_i and p_j that are related to two adjacent Voronoi regions VR_i and VR_j, and therefore are related to the Voronoi edge VE_{ij}, are neighbors. A Voronoi edge is defined in equation 4.3. Note that there are infinite Voronoi edges, related to open Voronoi regions.

$$\begin{aligned} VE_{ij} = \{q, q \in R^2 | d(q, p_i) = d(q, p_j); p_i, p_j \in P; \\ \forall p_k \in P, p_k \neq p_i \neq p_j, d(q, p_i) < d(q, p_k)\} \end{aligned} \tag{4.3}$$

4.1.3 Voronoi vertex

The point that is equidistant to three points p_i, p_j and $p_k \in P$ is known as a Voronoi vertex VV_{ijk}, see equation 4.4. VV_{ijk} is related to p_i, p_j and p_k, and, therefore, to its corresponding Voronoi regions VR_i, VR_j and VR_k. A Voronoi vertex, also may be seen as the place where three Voronoi edges converge, defining a relation between the Voronoi vertex VV_{ijk} and the three converging edges VE_{ij}, VE_{jk} and VE_{ki}. Notice that a point in R^2 may be equidistant to, at most, three points because of the assumption made of no more than three co-circular points. This implies that only three Voronoi edges and three Voronoi regions are related to a Voronoi vertex.

$$VV_{ijk} = \left\{q, q \in R^2 | d(q,p_i) = d(q,p_j) = d(q,p_k); p_i \neq p_j \neq p_k; p_i, p_j, p_k \in P\right\} \tag{4.4}$$

4.2 Delone Triangulation

Triangulation is the division of a surface or plane polygon into a set of triangles. The Delone Triangulation of a finite set of point P, known as DT, is a triangulation of the convex hull of the n points p_0, p_1, ..., $p_n \in P$ in which every single triangle satisfies the "empty circle" condition (figure 4.1(b)). The "empty circle" condition states that the circumcircle of every *Delaunay triangle* does not contain, not even in its circumference, any other point in P, except its vertices. For a given set of points P, the Delone Triangulation is unique if there are no more than three co-circular points. The reason is that for more than three co-circular points the Delone Triangulation becomes ambiguous, because there is more than one set of triangles that satisfies the "empty circle" condition (figure 4.2).

The Delone Triangulation has three kinds of elements: Delone vertices DV_i, Delone edges DE_{ij} and Delone triangles DT_{ijk}, as shown in fig 4.1(b). Each of these elements uses one, two or three points in P respectively.

Each Delone Triangulation has associated a Voronoi Diagram and vice versa, see figure 4.1(c). Each element from one of the graphs corresponds to another one in the *dual* graph, as it is shown in table 4.1.

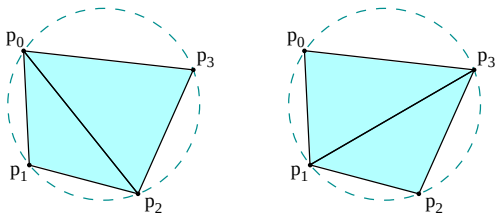


Figure 4.2: Two possible Delone Triangulation for four co-circular points.

Delone Element	Voronoi Element	Related by
Delone triangle DT_{ijk}	Voronoi vertex VV_{ijk}	VV_{ijk} lies in the center of the circumcircle of DT_{ijk}.
Delone edge DE_{ij}	Voronoi edge VE_{ij}	VE_{ij} is perpendicular to DE_{ij}, and they are related even when they do not intersect.
Delone vertex DV_i	Voronoi region VR_i	DV_I lies on p_i that is the point that defines VR_i.

Table 4.1: Relation between the Voronoi Diagram and the Delone Triangulation

4.3 Algorithms for Voronoi Diagram and Delone Triangulation construction

Several algorithms to build the Delone Triangulation and the Voronoi Diagram exist. When one of these structures is created, the *dual* structure can be built using the correspondence of elements between the Delone Triangulation and the Voronoi Diagram in $O(n)$. A lower bound of $O(n \log n)$ for the time for computing a planar triangulation was analytically established by Shamos ([42]). A brief survey of some algorithms for computing the Voronoi Diagram is given in the following paragraphs and finally the algorithm chose and implemented in this project is described.

4.3.1 The Accumulative algorithm

This algorithm is the most intuitive and also most time- and space-consuming algorithm. It just takes every possible triad of points to check the empty circle condition on them. If the three elected points satisfy the condition, then, a Delone triangle is created and kept. This is $O(n^4)$ for the triad election and checking of the empty circle condition.

4.3.2 The Divide and Conquer algorithm

It was the first worst-case optimal algorithm for Voronoi Diagrams in two dimensions, developed by Shamos and Hoey approx. in 1975 ([41]). It divides the set of points with a vertical line, into two subsets of approximately the same size. Then, the Voronoi Diagram of each half is recursively computed, and lastly, the Voronoi Diagrams are merged into one. The merge takes linear time in the worst case, so the algorithm runs in $O(n \log n)$ for the worst-case. A detailed description for this algorithm in terms of Delone Triangulation is given by Guibas and Stolfi ([21]).

4.3.3 The Flipping algorithm

It is also simple. It starts with any triangulation of all the points in P, and begins to check every two opposite triangles *abc* and *bcd*, if they are *locally Delone*. Two opposite triangles are *locally Delone* when the non-common vertex of one of the triangles is not inside the circumcircle of the

second triangle or vice versa. If the checked triangles are not *locally Delone*, then the diagonal *bd* is flipped, implying that *abc* and *bcd* are replaced by *abd* and *bcd*. All the triangles are checked, flipping some of them, and the triangulation is checked again, until the complete set of triangles are Delone. This algorithm runs in time $O(n \log n + f)$, where f is the number of flips and n the edges in the triangulation. An upper bound for f of $\binom{n}{2}$ is given by Fortune ([16]). If the points are added in random order, and flips are performed to keep all the triangles satisfying the "empty circle" condition, then Guibas, Knuth and Sharir ([22]) show that the expected number of flips is linear. This algorithm was extended to general dimension by Rajan ([32]), Edelsbrunner and Shah ([15]).

4.3.4 The Incremental algorithm

It starts with an initial triangulation T of one or more triangles, and the rest of points are added one by one. For the sake of simplicity, the initial triangulation covers the whole space where the added points will lie. Given a new point p, the set of triangles T_o whose circumcircles contain p is found and replaced with a triangulation T_p, built with the edges of the triangles in the border of T_o and p. In the worst case, all the triangles of T could be affected by the insertion of p, so adding a single point could take linear time, and adding all the points, under this condition, takes $O(n^2)$. In general, this algorithm runs in d-dimensional space in $O(n^{\frac{d+1}{2}})$ and uses $O(n^{\frac{d}{2}})$ in space. Edelsbrunner ([13]) gives more details of this algorithm, and also some adaptations to minimize the time consumed by finding the affected triangles may be found in ([8]) and ([29]).

4.3.5 The Random Incremental algorithm

It is a special case of *the incremental algorithm*, initially proposed by Clarkson and Shor ([10]). In this algorithm, the points are inserted in random order and takes, in the worst-case, $O(n \log n)$ in $2D$ and $O(n^{\frac{d}{2}})$ in a d-dimensional space. This is better than the worst-case complexity of the *Incremental Algorithm*. This algorithm requires a data structure to find the affected triangles when a point is inserted. Several data structures have been proposed ([6], [28] and [27]). This algorithm, based on the *Delone Tree* developed by Boissonnat and Teillaud ([7]) has been implemented for this project.

4.3.6 The Random Incremental algorithm based on the Delone Tree

The Delone tree data structure was developed by Boissonnat and Teillaud in 1986 ([6], [7]). The algorithm and the structure are defined for any dimension, where a Delone Triangulation is composed by simplices, and they share facets among them. The data structure is a *rooted directed acyclic graph* that stores the history of the construction of the Delone Triangulation and saves relations between the eliminated simplices with the ones that replaced them. These relations are used to locate the simplices that are affected when a new point is inserted. Much of the material in this section is from [7], but presented just for two dimensions, where the finite simplices are triangles and the facets are edges.

The algorithm starts with a triangulation of three points p_0, p_1 and p_2. Using these points, four simplices are created and set as *sons of the root*. One of the created simplices, t_{012}, is finite and the other three, $t_{02\infty}$, $t_{10\infty}$ and $t_{21\infty}$, are infinite. t_{012} represents the Delone triangle DT_{012}, and is built with p_0, p_1 and p_2 in ccw-sense. See figure 4.3.

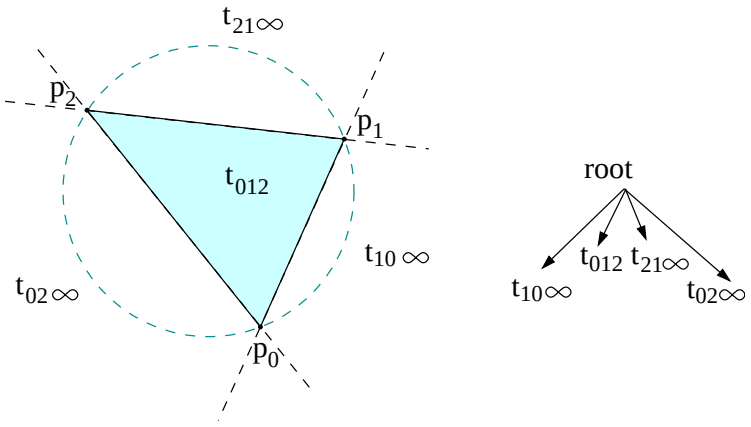


Figure 4.3: Initialization of the random incremental algorithm based on Delone Tree

Any infinite simplex $t_{ij\infty}$ is built using an edge e_{ij} shared with a finite simplex and a point in ∞. The infinite version of e_{ij} divides the plane into two half-planes. The third point used to create an infinite simplex lies in the halfplane to the right of e_{ij}, or in the direction $z \otimes e_{ij}$, where z is the normal vector of the plane where the set of points lie. The sense of e_{ij} is opposite

to the edge e_{ji} in the finite simplex, and it is called the visible edge of $t_{ij\infty}$. The center of the circumcircle of $t_{ij\infty}$ also lies in ∞, and the circumcircle is deformed to a straight line that corresponds to the infinite version of e_{ij}.

When a new point q is inserted into the triangulation, the algorithm follows two main steps:

1. *Localization*: It localizes the affected simplices T_o whose circumcircle contains q.

2. *Creation New Simplices*: It replaces the affected simplices T_o for a set of new simplices T_q built using q.

A simplex is affected by q when q falls inside its circumcircle. For an infinite simplex $t_{ij\infty}$ the "inside-the-circumcircle" test is simplified to a "side-of-the-line" test, where the sense of e_{ij} will tell where the interior and exterior of the circumcircle is placed.

The union of the affected simplices T_o is referred to as the affected region $AR(q)$. The boundary of $AR(q)$ is a star-shaped polygon. This implies that for any point p in $AR(q)$, the segment qp is contained in $AR(q)$.

After the insertion of q, the affected simplices T_o become dead and q is declared as their killer. Let $t_{ijq} \in T_q$ be a new simplex created using q and an edge e_{ij} on the border of $AR(q)$. The simplex $t_{ijk} \in T_o$ that originally contained e_{ij} is declared father of t_{ijq} through edge e_{ij}, and the simplex t_{jil} that contains e_{ji} is set as the stepfather of t_{ijq} through edge e_{ji}. It is important to notice that the circumcircle of t_{ijq} stands inside the union of the circumcircles of t_{ijk} and t_{jil}. This implies that when t_{ijq} is affected, t_{ijk} or t_{jil} are also affected. See figure 4.4, where the new created simplex is t_{014}. t_{012} is father of t_{014} through e_{01} and t_{103} is stepfather of t_{014} through e_{10}.

In formal terms, the *Localize* and *Create New Simplices* steps are shown in algorithms 1 and 2 respectively. Algorithm 3 shows the whole process used to add a new point to the Delone Triangulation.

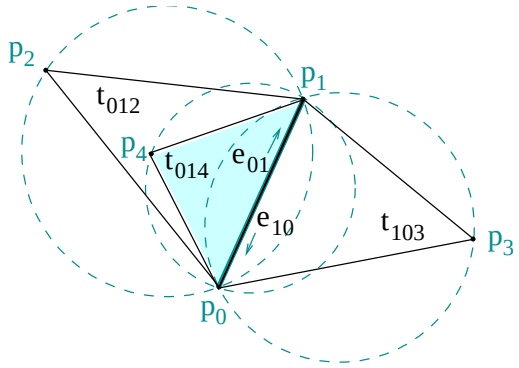


Figure 4.4: Father and stepfather relations and circumcircles

Algorithm 1 Localizing the affected simplices

T_o = **localize**(q, s)

Input: q: new added point
t: simplex to check
Output: T_o: List of simplices affected by q
Precondition: t is a valid simplex. It could be infinite or finite
Postcondition: the simplices in T_o would be affected by the insertion of q

1: **if** t is not marked as visited and q lies inside the circumcircle of t **then**
2: mark t as visited
3: **for** every simplex s, that is stepson of t **do**
4: T_{os} = **localize** (q, s)
5: add T_{os} to T_o
6: **end for**
7: **if** t is dead **then**
8: **for** every simplex s, that is son of t **do**
9: T_{os} = **localize**(q, s)
10: add T_{os} to T_o
11: **end for**
12: **else**
13: mark t as killed by q
14: add t to T_o
15: **end if**
16: **end if**

Algorithm 2 Creating new simplices containing q that replace T_o

createNewSimplices(q, T_o)

Input:	q: new added point T_o: List of simplices affected by q
Output:	
Precondition:	T_o contains at least one affected simplex
Postcondition:	The Delone tree contains at least three new simplices containing q
Comment:	The Delone tree is implicitly updated by the creation of the relations between the new added simplices and the old ones.

1: **for** every simplex $t_o \in T_o$ **do**
2: **for** every simplex t_n, neighbor of t_o through its edge e_{ij} **do**
3: **if** t_n is not affected by q **then**
4: create simplex t_q with e_{ij} and q
5: set t_o as the father of t_q
6: set t_q as the son of t_o through e_{ij}
7: set t_n as the stepfather of t_q
8: set t_q as a stepson of t_n through e_{ji}
9: set t_q as neighbor of t_n trough e_{ij}
10: set t_n as neighbor of t_q trough e_{ji}
11: add t_q in T_q
12: **end if**
13: **end for**
14: **end for**
15: update adjacency relation between the created simplices.

Algorithm 3 Add a new point q to a Delone Triangulation

dtree = **addNewPoint**(*dtree*, q)

Input:	*dtree*: Delone tree containing a Delone Triangulation
	q: new added point
Output:	*dtree*: Updated Delone tree containing q as a Delone vertex
Precondition:	*dtree* has at least four simplices (after initialization)
Postcondition:	*dtree* contains at least three new simplices
	dtree contains q as a Delone vertex

1: set T_o as an empty list of simplices
2: **for** every simplex t, son of the root of *dtree* **do**
3: T_{os} = **localize**(q, t) (algorithm 1)
4: add T_{os} to T_o
5: **end for**
6: **createNewSimplices**(q, T_o) (algorithm 2)

Chapter 5

Review of the B+G method

A review of the method developed by Boissonnat ([5]) and improved by Geiger ([19]) is given in this chapter. This method is refered as B+G in this document. The basis of the material presented in this chapter is from [19], enriched with additional elements, concepts, explanations and changes.

The B+G method processes each pair of adjacent levels, $level_i$ and $level_j$ and creates a flat-faced polyhedral surface that joins the contours of both levels. Notice that on $level_i$ or on $level_j$, the given contours may be nested, indicating that the objects to reconstruct may have internal cavities, possibly holding inside solid regions.

The Delone Triangulation and the Voronoi Diagram are built using the points of the contours on each level. A graph, named the Joint Voronoi Diagram is then constructed by the intersection of the Voronoi Diagrams related to $level_i$ and $level_j$ (section 5.4). After that, the Joint Voronoi Diagram is translated to tetrahedrons (section 5.4), which are finally filtered (section 5.5) to isolate the faces that belong to the surface.

Before the construction of the *Joint Voronoi Diagram*, some conditions must be fulfilled for the Delone Triangulation on each level:

Condition 0: Completeness of DT with respect to the contours The triangulation should include all the edges which form the contours C_i. In formal terms, condition 0 is defined in equation 5.1.

$$\forall \text{ edge } e \in C_i : \qquad \exists DE_{ij} \text{ s.t. } e = DE_{ij} \tag{5.1}$$

Condition 1: Partition of DT by contours The classification of every triangle in the Delone Triangulation as internal or external with respect

to the contours must be possible. Let the union of all the external triangles be called External Region ER, and the union of all the internal triangles be the Internal Region IR, then, condition 1 is formalized in equation 5.2.

$$\forall DT_{ijk} \in DT: \qquad (DT_{ijk} \subset IR) \vee (DT_{ijk} \subset ER) \tag{5.2}$$

Condition 2: Confinement of circumcenters The circumcenter of every Delone triangle DT_{ijk} must lie inside the region to which DT_{ijk} belongs. In formal terms, it is defined in equation 5.3.

$$\forall DT_{ijk} \in DT: \qquad \begin{cases} \text{if } DT_{ijk} \subset IR \Rightarrow circumcenter(DT_{ijk}) \in IR \\ \text{if } DT_{ijk} \subset ER \Rightarrow circumcenter(DT_{ijk}) \in ER \end{cases} \tag{5.3}$$

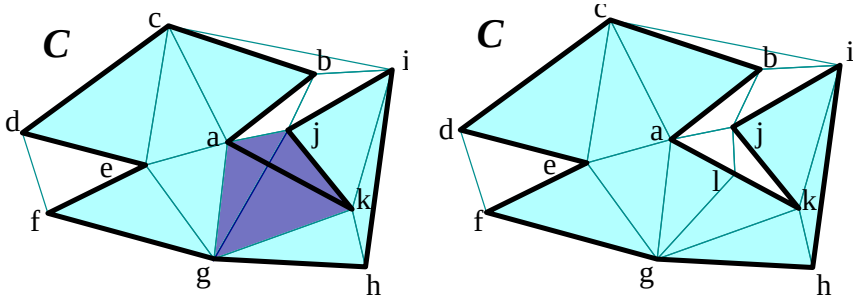
Notice that the satisfaction of condition 0 leads to the satisfaction of condition 1 and vice versa. When all the contour edges are included in the triangulation (condition 0 is satisfied), the triangles may be classified as internal or external triangles with respect to the contours on the level (condition 1 is satisfied). Both conditions are equivalent, and they are both kept for the sake of understandability (Condition 0 is not presented in the original method). The term "condition 0/1" indifferent refers to condition 0 or condition 1. Also, it is important to note that condition 2 needs the satisfaction of condition 1 to be guaranteed.

To accomplish these conditions, some operations must be performed over the contours on each level $level_i$ and $level_j$. In section 5.1 the addition of points to the contour edges is done to guarantee condition 0, and therefore condition 1. Section 5.2 explains the operations followed to accomplish condition 2. After both conditions are satisfied, points in the interior of some contours are added to the Delone Triangulation of each level (section 5.3). Up to this point, the Delone Triangulation, and its related Voronoi Diagram on each level, are considered adequate to create the Joint Voronoi Diagram.

5.1 Addition of points to contour edges to guarantee condition 0/1

The Delone Triangulation of the vertices of the contours on a level may result in a triangulation where some edges of the contours do not appear.

For example, in figure 5.1(a) there is a contour C defined by the ordered set of points $P = a, b, ...k$. The edge defined by the vertices a and k does not appear in the Delone Triangulation of P. Notice that DT_{agj} and DT_{kjg} may not be classified as internal nor external triangles with respect to C. So, the Delone Triangulation shown in that figure does not satisfies condition 0/1.



(a) DT_{agj} and DT_{kjg} are not inside nor outside C

(b) White triangles are external and colored triangles are internal with respect to C

Figure 5.1: Added points to ensure that all the contour edges appear in the Delone Triangulation

A simple procedure, described in algorithm 4, must be followed to create a Delone Triangulation that satisfies condition 0/1. Each contour edge e that does not appear in the triangulation is divided into two edges by adding its midpoint to the contour (line 4). Then, the new vertices of the contour are added to the Delone Triangulation (line 13) and the updated Delone Triangulation is checked again. This operation is done untill all the edges are included in the Delone Triangulation on the level (cycle repeat-until in lines 1, 15). In this point, a different algorithm from the one presented by Geiger ([19]) is given. In the original code, a recursive call is made. Even, when the results do not differ from the ones obtained from the algorithm given by Geiger, this presentation is more understandable and it follows the style of the algorithms presented in this document.

Algorithm 4 converges in a Delone Triangulation containing all the contours edges. This fact was proven by Boissonnat ([5]). Notice that the contour shape does not change because the added vertices are included into

Algorithm 4 Add points to the triangulation to satisfy condition 1

dtree = **addPointsToSatisfy01**(*dtree*, *C*)

Input: *dtree*: Delone tree containing the Delone Triangulation of the level
C: set of contours of the level

Output: *dtree*: Delone tree containing the updated Delone Triangulation

Precondition: The Delone Triangulation hold in *dtree* contains all the contour vertices

Postcondition: The Delone Triangulation hold in *dtree* contains all the contour edges

```
1: repeat
2:   points = [ ]
3:   for every contour c_i in C do
4:     for every edge e_j belonging to c_i do
5:       if e_j does not belong to any triangle in dtree then
6:         p_m = midpoint between the vertices of e_j
7:         add p_m in points
8:         add p_m in contour c_i in between the vertices of e_j
9:       end if
10:    end for
11:  end for
12:  for every point p in points do
13:    dtree = addNewPoint(dtree, p) (algorithm 3)
14:  end for
15: until points is empty
```

the original contour edges. See figure 5.1(b) where all the contour edges are part of the Delone Triangulation. In this case, the addition of l was enough for the Delone Triangulation to accomplish condition 0/1 and the contour C is then defined by the ordered set of points $P = a, b, c, d, e, f, g, h, i, j, k, l$.

5.2 Addition of points to contour edges to guarantee condition 2

Notice that adjacent internal and external triangles share a contour edge.

Algorithm 5 Add points to the triangulation to satisfy condition 2

dtree = **addPointsToSatisfy2**(*dtree*, *C*)

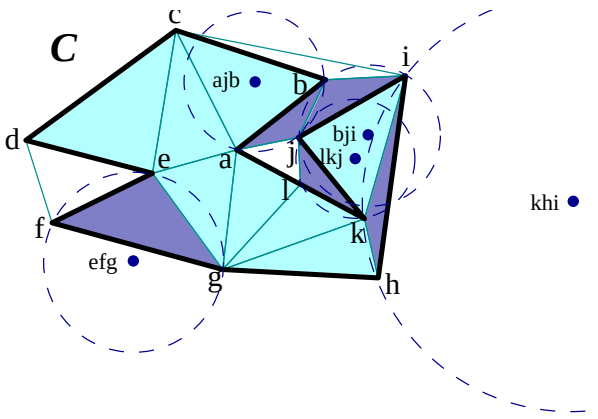
Input: *dtree*: Delone tree containing the Delone Triangulation of the level
C: set of contours of the level

Output: *dtree*: Delone tree containing the updated Delone Triangulation

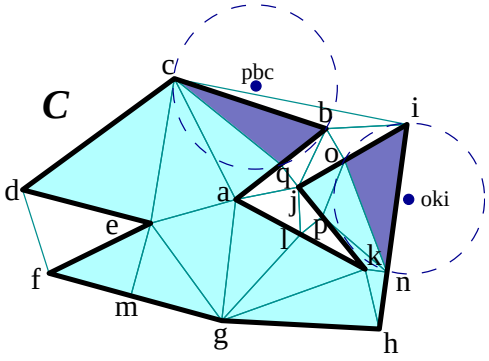
Precondition: The Delone Triangulation hold in *dtree* contains all the contour edges

Postcondition: The angles in front of the contour edges are not obtuse

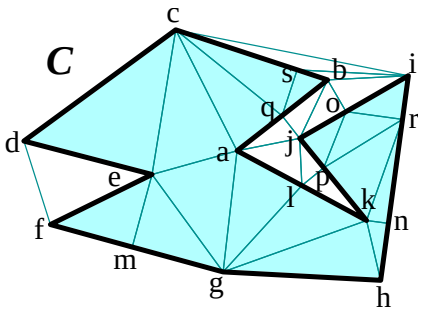
```
1: repeat
2:   dtree = addPointsToSatisfy01( dtree, C ) (algorithm 4)
3:   points = [ ]
4:   for every contour c_i in C do
5:     for every edge e_j belonging to c_i do
6:       t_0 = triangle to the left of e_j if any.
7:       t_1 = triangle to the right of e_j if any.
8:       add_point = FALSE
9:       if angle in front of e_j in t_0 is OBTUSE then
10:        p_m = project the vertex opposite to e_j in t_0 on e_j
11:        add_point = TRUE
12:      else if angle in front of e_j in t_1 is OBTUSE then
13:        p_m = project the vertex opposite to e_j in t_1 on e_j
14:        add_point = TRUE
15:      end if
16:      if add_point is TRUE then
17:        add p_m in points
18:        add p_m in contour c_i in between the vertices of e_j
19:      end if
20:    end for
21:  end for
22:  for every point p in points do
23:    dtree = addNewPoint(dtree, p) (algorithm 3)
24:  end for
25: until points is empty
```



(a) Triangulation that satisfies Condition 0/1 but not Condition 2



(b) First insertion of points



(c) Refined Triangulation

Figure 5.2: Added points to ensure that the angles in front of contour edges are not obtuse

Any internal or external triangle whose circumcenter lies outside the region to which the triangle belongs (IR or ER), (i) have an obtuse angle and (ii) contain a contour edge. The circumcenter and also the contour edge lie in front of the obtuse angle of the triangle. In figure 5.2(a) the Delone triangles DT_{efg}, DT_{khi}, DT_{lkj}, DT_{ajb} and DT_{bji} contain an obtuse angle, and its circumcenters (denoted as ijk) lie out of its corresponding triangles.

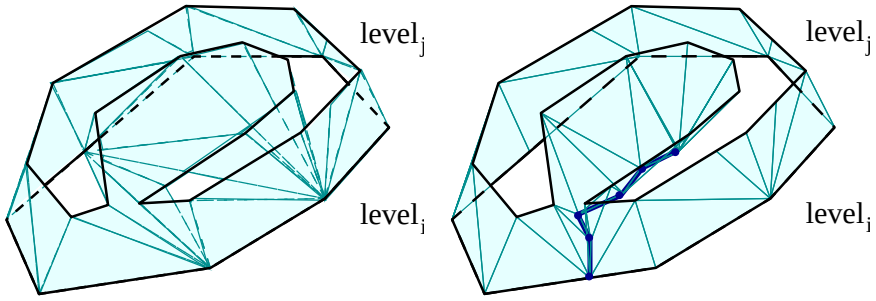
Each triangle whose circumcenter lies outside the region to which it belongs, must be divided to accomplish condition 2. The process is implemented in algorithm 5. Notice that each contour edge e_j is contained in, at most, two triangles (t_0 and t_1); so, two different triangles must be tested. The angles opposite to the contour edge e_j in both triangles are checked (lines 9 and 12). If any of these angles is obtuse, then a new point at the normal projection of the opposite vertex on the contour edge (lines 10 and 13) is added to divide the contour edge (line 18). The Delone Triangulation is updated (line 23), condition 0/1 is guaranteed (line 2) and condition 2 is checked again (cycle repeat-until in lines 1, 25). See figure 5.2 where seven points were added to the contour in two iterations and C is then defined by the ordered set of points $P = a, q, b, s, c, d, e, f, m, g, h, n, r, i, o, j, p, k, l$.

Algorithm 5 differs from the one given by Geiger ([19]). In this algorithm condition 0/1 is checked before the addition of points to satisfy condition 2 is done (line 2). It is important to notice that condition 0/1 (Completeness of DT with respect to the contours/Partition of DT by contours) must be fulfilled before condition 2 (Confinement of circumcenters) is checked. This dependence between the conditions is not taken into account in the original method. Refer to section 6.1 to obtain more details.

5.3 Insertion of internal points

Up to this point the Delone Triangulation and its dual Voronoi diagram are considered apt to calculate the Joint Voronoi Diagram. Nevertheless, using contours to reconstruct the surface as they are up to this point, a surface like the one shown in figure 5.3(a) is created. Geiger ([19]) improved the method developed by Boissonnat ([5]) by adding points in the interior of some contours. The basic idea is to use the medial axis, internal and external, of the contours on one level, to divide internal regions on the adjacent level, and finally create a surface that links similar regions, even if those regions are not completely bordered by contours (figure 5.3(b)).

Calculate the medial axes of a set of contours is not easy. Geiger proposed to use a subset of edges of the Voronoi Diagram instead of the medial axes. This subset of edges is named Voronoi Skeleton. When the contours on a level have a big number of vertices, the Voronoi Skeletons tend toward the medial axes of the contours.

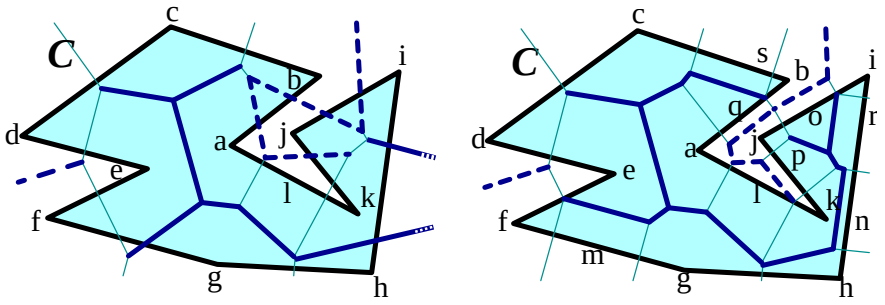


(a) Reconstructed surface with no internal points added on level i

(b) Reconstructed surface with internal points added on level i

Figure 5.3: Surface before and after adding internal points

5.3.1 Voronoi Skeleton



(a) Voronoi Skeletons of the contour before Condition 2 is satisfied

(b) Voronoi Skeletons of the contour after the satisfaction of Condition 2

Figure 5.4: Voronoi skeletons of the same polygon before and after the satisfaction of Condition 2

A Voronoi Skeleton is a subset of edges of the Voronoi Diagram, such that its related Delone edge does not belong to any contour. There are two kinds of skeletons, the Internal Voronoi Skeleton *IVS* and the External Voronoi

Skeleton EVS. The Internal Voronoi Skeleton IVS is the subset of Voronoi edges whose dual Delone edge is shared by two internal triangles. In the same way, the External Voronoi Skeleton, EVS, is defined as the subset of Voronoi edges whose dual Delone edge is shared by two external triangles. The vertices belonging to the EVS are known as *External Voronoi Vertices* and conversely the vertices belonging to the IVS are called *Internal Voronoi Vertices*. See figure 5.4, where the Voronoi Skeletons of the Contour C are shown, the Internal Voronoi Skeletons is represented by a solid line and the External Voronoi Skeletons is represented by a dashed line.

If condition 2 is not satisfied by the contours on the level, as it is shown in figure 5.4(a), the *External Voronoi Skeleton* may invade the internal region IR. Likewise, any *Internal Voronoi Skeleton* may cross contour edges and invade the external region ER.

Algorithm 6 Add internal points to contours on level i

$dtree_i$ = **addInternalPoints** (C_i, $dtree_i$, EVS_j)

Input:	C_i: set of contours on level i $dtree_i$: Delone tree containing the Delone Triangulation of level i EVS_j: *External Voronoi skeleton* on level j
Output:	$dtree_i$: Delone tree containing the updated Delone Triangulation
Precondition:	The Delone Triangulation hold in $dtree$ contains all the contour edges
Postcondition:	zero or more points are added in the interior of the contours in the triangulation.

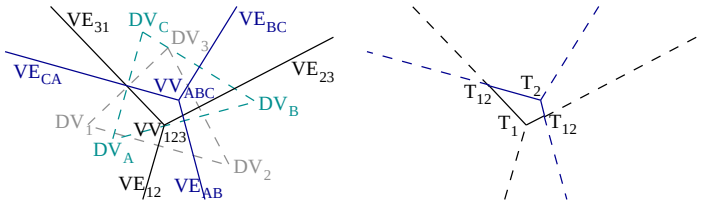
```
1: points = [ ]
2: for every Extern Voronoi vertex VV in EVS_j do
3:    if VV falls inside any contour in C_i then
4:       add VV to points
5:    end if
6: end for
7: for every point p in points do
8:    dtree_i = addNewPoint(dtree_i, p) (algorithm 3)
9: end for
10: dtree_i = addPointsToSatisfy2( dtree, C_i )
```

The External Voronoi Vertices on level i whose projections fall inside any

contour on level j are added to the triangulation of level j and vice versa. The addition of these points enables the separation of contours along an approximated external medial axis, as it is shown in figure 5.3(b). Algorithm 6 implements the process of adding internal points to a level. This algorithm must be called once per level.

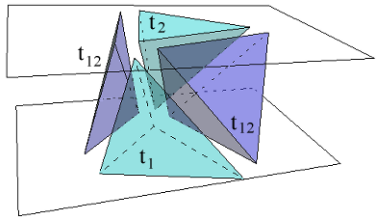
5.4 The Joint Voronoi Diagram

Using the Voronoi Diagrams of both levels, the *Joint Voronoi Diagram* is built. This graph is a planar graph that results from intersecting the orthogonal projections of two Voronoi Diagrams on a common plane. There are three kinds of nodes in the graph: T_1, T_2 and T_{12}. The T_i nodes correspond to the Voronoi vertices belonging to the Voronoi Diagram on level i, with $i = 1, 2$. The T_{12} nodes correspond to the intersection of two Voronoi edges.



(a) The Voronoi Diagrams of two simple contours

(b) The Joint Voronoi Diagram for two simple contours



(c) The corresponding tetrahedrons

Figure 5.5: The Joint Voronoi Diagram of two contours and its tetrahedrons

Every node in the graph corresponds to a tetrahedron, and the union of all these tetrahedrons form the 3D Delone Triangulation of the contour points P of both levels i and j. Because the tetrahedrons that are translated from the graph are Delone tetrahedrons, they satisfy the "empty-sphere" condition, that is, the sphere that circumscribes the tetrahedron does not contain any other point in P except its vertices. The tetrahedron corresponding to a T_i node, named T_i tetrahedron, is built using a Delone triangle on level i as base and a Delone vertex on level j as apex. The base is the Delone triangle DT_{klm}, related to the Voronoi vertex VV_{klm} on which the node lies. The apex is the Delone vertex belonging to level j that is the closest vertex to the circumcenter of DT_{klm}. The tetrahedron corresponding to a T_{12} node, named T_{12} tetrahedron, is built using the Delone edges related to both Voronoi edges that intersect. See figure 5.5(c) where a T_1, T_2 and two T_{12} tetrahedrons are translated from the Joint Voronoi Diagram in figure 5.5(b).

An edge on the *Joint Voronoi Diagram* connecting two nodes, may only connect a T_i to a T_i or a T_{12} node. A T_1 node is never directly connected to a T_2 node, nor vice versa. A T_{12} node must be in between those T_1 and T_2 nodes in order to be indirectly connected. Every T_i node is connected to, at most, three nodes, and a T_{12} to four nodes.

Notice that every tetrahedron is created with four Delone vertices. When more than four Delone vertices satisfy the "empty sphere" condition, the creation of tetrahedrons and the definition of the Joint Voronoi Diagram becomes ambiguous, issue that is not addressed by Geiger and Boissonnat ([19] and [5]). These cases are handle in an analytical way in section 6.3. In the implementation proved in the CAD/CAM/CAE Laboratory, a perturbation is applied when more than four points that shares the same empty sphere are found.

5.5 Elimination of tetrahedrons

Some nodes, and their related tetrahedrons, must be eliminated from the Joint Voronoi Diagram and only the tetrahedrons corresponding to the interior of the reconstructed object are to be kept. The triangular faces that are not shared by two tetrahedrons, compose the reconstructed surface that is the target of all this process.

Two steps are performed on the set of nodes and tetrahedrons:

1. External tetrahedrons are eliminated.

2. Tetrahedrons contributing to non-solid connections are eliminated.

A tetrahedron is called external if any of its edges lies outside all the contours. For a T_i node, if the Voronoi vertex on which the T_i is defined is an External Voronoi vertex, then the T_i tetrahedron has at least one edge outside of the contours and must be eliminated. If any of the Voronoi edges that intersect and create a T_{12} node belongs to an External Voronoi Skeleton EVS, then, its related Delone edge lies outside the contours, and the T_{12} must be erased.

The non-solid connections are a set of tetrahedrons that only have an edge or one single point on one of the levels. If the T_i nodes related to the Voronoi vertices of the Voronoi edges that intersect and create a T_{12} node are eliminated, then the T_{12} tetrahedron is considered non-solid and must be also eliminated. If a set of T_i nodes connected among them does not have a connection to a T_{12}, then the whole set of T_i must be eliminated. This specific configuration of nodes is reflected in a set of T_i tetrahedrons that shares the same apex and usually the apex belongs to a contour.

Chapter 6

The Polyhedral Surface Method

The *PSM* is based on the B+G method. The B+G method ([19]) reconstructs incomplete surfaces and presents no-manifold situations. In the version implemented as part of this project, the incomplete surface problem was solved (section 6.4) and the no-manifold situations were minimized by taking into account special cases when creating the jointVoronoi Diagram (section 6.3). Also minor changes, in implementation and conceptualization, were done to reach better results.

6.1 Satisfaction of conditions

There are two conditions, (extended to three conditions in this document) that must be fulfilled before the Joint Voronoi Diagram may be created (section 5.1 and 5.2). These conditions are dependent between them, condition 0/1 must be fulfilled before condition 2 is checked. This detail is not considered on the original document, and some presented algorithms fails because of this omission. Notice that the satisfaction of both conditions is not straightforward, because the addition of points to satisfy condition 2 may cause some contour edges to disappear from the current Delone Triangulation. Also, the addition of points to satisfy condition 1 may create obtuse triangles, making the set of points do not satisfy condition 2. A cycle of "checking and correcting" converges to a Delone Triangulation which accomplish both conditions, since the Delone Triangulation maximizes the minimal angle in all the Delone triangles (satisfying condition 2) and the contours edges would eventually appear in the Delone Triangulation (satisfying condition 1 - [5]).

The satisfaction of both conditions must be tested every time a point is inserted into the Delone Triangulation. It implies that algorithm 5.1 must be called in algorithm 5 (as it is shown in line 2), and algorithm 5 must be called after the addition of internal points, as it is shown in algorithm 6, line 10.

6.2 Lifting of internal points

As an improvement to the B+G method, the points that are inserted in section 5.3 are orthogonally projected on a level between the processed levels before the surface is finished. This projection allows getting the original contours if the resulting surface is re-sampled using the same original planes.

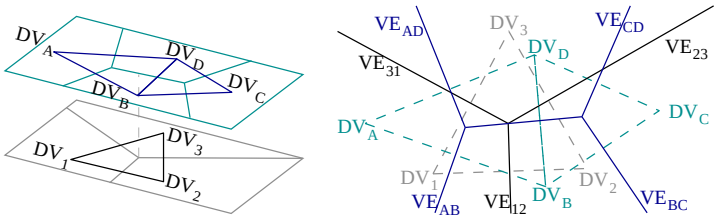
6.3 Special cases in the creation of the Joint Voronoi Diagram

The special cases are generated when more than four Delone vertices stand on the surface of an empty sphere. In these situations, the construction of the graph becomes ambiguous because there is more than one configuration of nodes (and therefore, tetrahedrons) that could be generated, as it happens in the construction of the Delone Triangulation when more than three co-circular points exist (section 4.2). If nodes of different configurations are kept together at the same time, the graph becomes inconsistent, because the faces of the related tetrahedrons intersect among them and the number of connections between the nodes exceed the limit. The problem is solved when a valid configuration of nodes, defined as the set of nodes whose related tetrahedrons properly share faces, is found and it is inserted into the graph. These special cases are not considered in the original version of this method ([19]).

As an upper bound, at most six Delone vertices may share the same empty sphere, because on each level the limit of co-circular vertices is three and the graph generation involves just two levels.

6.3.1 Case 1: Voronoi Vertex vs. Voronoi Edge

This case is generated when five Delone vertices are co-spherical, leading to a Voronoi vertex belonging to level i be projected on a Voronoi edge belonging to level j, or vice versa. An example is shown in figure 6.3.1



(a) Vertices DV_1, DV_2, DV_3, DV_B and DV_D are co-spherical

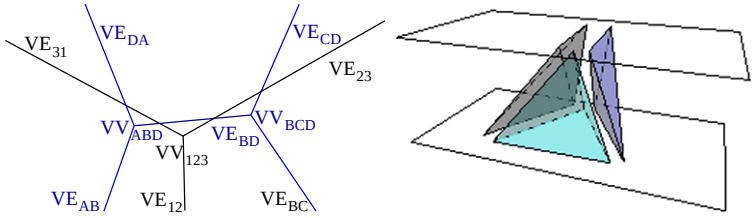
(b) Projected case

Figure 6.1: Special cases in the creation of the Joint Voronoi Diagram: example of Voronoi vertex vs. Voronoi edge case

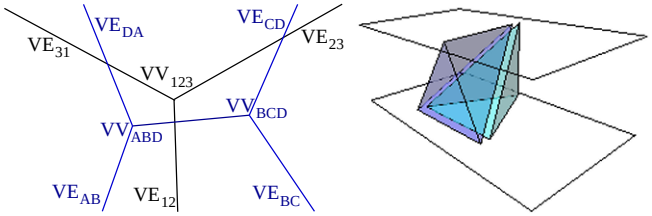
Each pair of Voronoi edges that intersect each other, generate a T_{12} tetrahedron, as it is seen in section 5.4. In this case three intersections are found, VE_{12} vs. VE_{BD}, VE_{23} vs. VE_{BD} and VE_{31} vs. VE_{BD}. The creation of all these T_{12} tetrahedrons is illegal, because their faces intersect and one face is shared by more than two tetrahedrons.

The ambiguity is essentially shown when creating the T_i tetrahedron related to VV_{123}. In this case, two different Delone vertices are found at the same distance to VV_{123} and the distance is the minimum among all the points, so, any of them could be used as the apex. These two found Delone vertices are the ones related to the Voronoi edge on which VV_{123} is projected, in figure 6.3.1 they are DV_B and DV_D. Because there are two alternatives to choose from, this situation allows two configurations as solutions.

The election of the apex for the T_i tetrahedron between these Delone vertices, states that the distance from the elected vertex to VV_{123} is virtually smaller than the distance from the non-elected vertex to VV_{123}. It is equivalent to move the level j in direction $\overrightarrow{VV_{123}Apex}$. The ambiguity is eliminated as shown in figure 6.3.1.



(a) Delone vertex DV_D elected as apex



(b) Delone vertex DV_B elected as apex

Figure 6.2: Special cases in the creation of the Joint Voronoi Diagram: Solutions for Voronoi vertex vs. Voronoi edge case

Identification of a valid configuration

Algorithm 7 Solving Voronoi vertex vs. Voronoi edge Case

$[T_i, T_12]$ = **solveCaseVertexVsEdge**(VV, VE)

Input:	VV: Voronoi vertex
	VE: Voronoi edge
Output:	T_i tetrahedron related to VV
	T_12: set of at most two T_{12} tetrahedrons
Precondition:	level of VV is not the same level of VE
	the projection of VV lies inside VE
Postcondition:	the tetrahedrons in T_i and T_12 form a valid configuration

1: $Apex$ = elect left or right vertex of VE
2: T_i = new T_i using the Delone triangle related to VV and $Apex$
3: **for** every Voronoi edge VE_k related to VV **do**
4: **if** half-plane of VE_k is not the same half-plane of $Apex$ **then**
5: t_{12} = new T_{12} created with the Delone edges related to VE_k and VE
6: add t_{12} to T_{12}
7: **end if**
8: **end for**

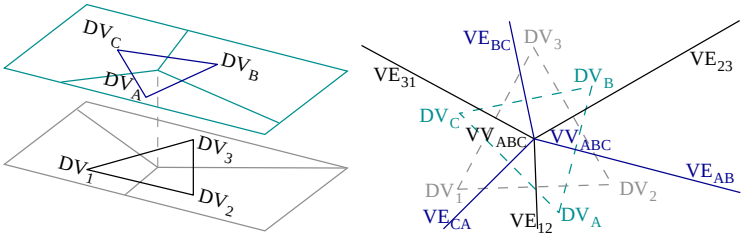
The complete sequence of steps is given in detail in algorithm 7. The infinite version of the Voronoi edge on which the vertex is projected, VE_{BD} in figure 6.3.1, divides the plane into two half-planes, each of them containing one of the Delone vertices related to the edge and one or two projected Delone edges belonging to level i.

Due to the virtual displacement done by electing the apex (line 1), the edges contained in the half-plane where the apex lies, do not longer intersect VE_{BD} but the edges in the second half-plane properly do it (line 4). This "intersect and no-longer-intersect" status on each found intersection leads to a proper creation of the nodes related to the T_{12} tetrahedrons that complete a valid configuration (lines 3-7).

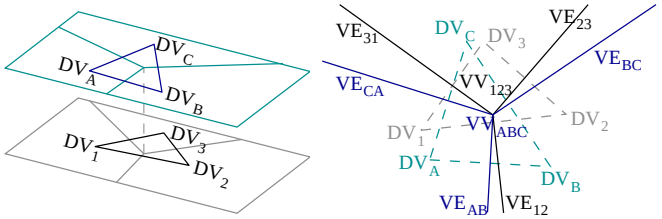
6.3.2 Case 2: Voronoi vertex vs. Voronoi vertex

This case is generated when six Delone vertices are co-spherical, leading to a Voronoi vertex belonging to level i be projected on a Voronoi vertex belonging

to level j.



(a) *1a2b3c* sub-case



(b) *1ab23c* sub-case

Figure 6.3: Special cases in the creation of the Joint Voronoi Diagram: Voronoi Vertex vs. Voronoi Vertex case

In this case nine intersections are identified, VE_{12} vs. VE_{AB}, VE_{12} vs. VE_{BC}, VE_{12} vs. VE_{CA}, VE_{23} vs. VE_{AB}, VE_{23} vs. VE_{BC}, VE_{23} vs. VE_{CA}, VE_{31} vs. VE_{AB}, VE_{31} vs. VE_{BC}, VE_{31} vs. VE_{CA}. As in the previous case, the construction of these nine tetrahedrons, leads to an inconsistent graph. The "election of apex" problem is also present, with the detail that there are two T_i tetrahedrons to elect an apex, and three possible apices for each tetrahedron. When an apex for any of the T_i tetrahedrons is chosen, it restricts the election of the apex for the second T_i tetrahedron and the creation of the complementary T_{12} tetrahedrons. Because of this, this case allows three configurations as solutions.

As it happened in the previous case, the election of an apex could be translated into a displacement of the levels and the elimination of the ambiguity by the assumption that the distance between the apex and the Voronoi

vertex is the smallest.

Algorithm 8 Identifying Vertex vs. Vertex sub-cases

subcase = **identifyVertexVsVertexSubcase**(VV_i, VV_j)

Input: VV_i: Voronoi vertex on level i
VV_j: Voronoi vertex on level j

Output: *subcase*: Flag indicating the sub-case type, its possible values are $1a2b3c$ or $1ab23c$

Precondition: level of VV_i is not the same level of VV_j
the projection of VV_i lies on the projection of VV_j

Postcondition: A sub-case is identified

```
1: Edges[ ] = angular order of all edges related to VV_i and VV_j
2: Subcase = 1a2b3c
3: for every Voronoi edge VE_c in Edges do
4:    set VE_n as the edge next to VE_c in Edges
5:    if level of VE_c is level of VE_n then
6:       Subcase = 1ab23c
7:    end if
8: end for
```

Two different sub-cases are identified for this case, both keeping the same characteristics described above. Algorithm 8 identifies the sub-cases. The sub-cases are determined by the distribution of the edges on the "intersecting star" created when all the edges are projected on the same plane (see figure 6.3.2). There are only two possible distributions, the edges are intercalated or they are not. When two consecutive edges belong to the same level, the sub-case is identified as the *1ab23c* sub-case (lines 5-7). Otherwise, if there are no two consecutive levels belonging to the same level, the sub-case is identified as the *1a2b3c* sub-case (the cycle in lines 3-9 never falls inside lines 5-7).

Identification of a valid configuration for the *1a2b3c* sub-case

For this sub-case, each Voronoi region related to a Voronoi vertex contains a Voronoi edge related to the other Voronoi vertex (figure 6.3(a)). The three solutions for this sub-case are symmetric; the election of the apex for the first T_i tetrahedron does not change the fact that two T_{12} and two T_i tetrahedrons

Algorithm 9 Voronoi vertex vs. Voronoi vertex $1a2b3c$ sub-case

$[T_i, T_{12}]$ = **solveVertexVsVertex1a2b3c**(VV_i, VV_j)

Input:	VV_i: Voronoi vertex on level i
	VV_j: Voronoi vertex on level j
Output:	T_i: set of TWO T_i tetrahedrons related to VV_i and VV_j
	$T_1 2$: set of TWO T_{12} tetrahedrons
Precondition:	level of VV_i is not the same level of VV_j
	the projection of VV_i lies on the projection of VV_j
Postcondition:	the tetrahedrons in T_i and $T_1 2$ form a valid configuration

For this sub-case, each Voronoi region related to a Voronoi vertex contains a Voronoi edge related to the other Voronoi vertex (figure 6.3(a)). The three solutions for this sub-case are symmetric; the election of the apex for the first T_i tetrahedron does not change the fact that two T_{12} and two T_i tetrahedrons are created. In figure 6.4 the *joint Voronoi Diagram* with no ambiguity is shown, and also its physical tetrahedron representation.

1: $Edge_j$ = any Voronoi edge related to VV_j
2: $Region_j$ = Voronoi region to the left of $Edge_j$
3: $Apex_1$ = Delone vertex related to $Region_j$
4: t_i = new T_i using the Delone triangle related to VV_i and $Apex_1$
5: add t_i to T_i
6: $Edge_i$ = Voronoi edge whose projection lies inside $Region_j$
7: $Region_i$ = Voronoi region not bounded by $Edge_i$ on the level of $Edge_i$
8: $Apex_2$ = Delone vertex related to $Region_i$
9: t_i = new T_i using the Delone triangle related to VV_j and $Apex_2$
10: add t_i to T_i
11: DE_i = Delone edge related to the Voronoi edge to the left of $Region_i$
12: DE_j = Delone edge related to the Voronoi edge to the right of $Region_j$
13: t_{12} = new T_{12} using DE_i and DE_j
14: add t_{12} to T_{12}
15: DE_i = Delone edge related to the Voronoi edge to the right of $Region_i$
16: DE_j = Delone edge related to the Voronoi edge to the left of $Region_j$
17: t_{12} = new T_{12} using DE_i and DE_j
18: add t_{12} to T_{12}

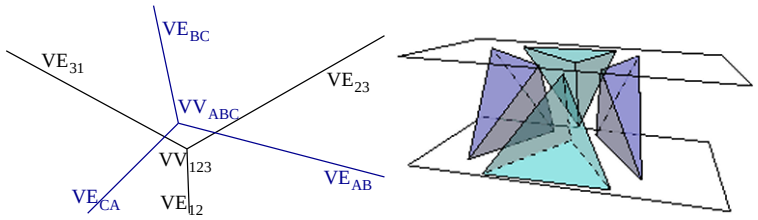


Figure 6.4: Special cases in the creation of the Joint Voronoi Diagram: A solution for *1a2b3c* sub-case, where DV_3 was elected as apex for the T_i tetrahedron related to VV_{123}

are created. In figure 6.4 the *joint Voronoi Diagram* with no ambiguity is shown, and also its physical tetrahedron representation.

Algorithm 9 implements the solution for this sub-case. The election of the apex for the first T_i is done in line 4. The vertex that lies in the region that is opposite to the first elected apex in the consecutive level is chosen as apex for the second T_i tetrahedron (line 6-8). The T_{12} tetrahedrons that complete the valid solution are created using the edges that bound the corresponding Voronoi regions of the elected apices (lines 11-13 and 15-17).

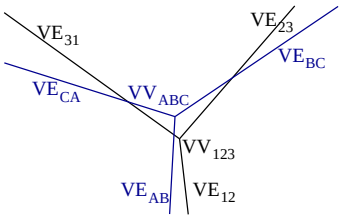
Identification of a valid configuration for the *1ab23c* sub-case

For this sub-case, the solutions are not symmetric as they are in the previous sub-case. The solutions are shown in figure 6.5.

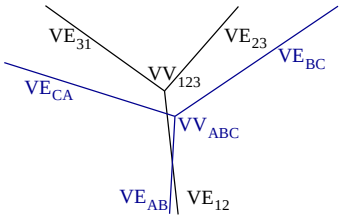
The simplest solution is shown in figure 6.5(c), where just one T_{12} tetrahedron is created. To construct that solution, some elements must be identified: the *Full Region* and the *Lone Edge*.

Full Region: The Voronoi region that contains two Voronoi edges belonging to the other level is named the *Full Region*. In figure 6.5(c), the *Full Region* for level i is the Voronoi region VR_1, bounded by VE_{12} and VE_{31}, and for level j it is the Voronoi region VR_C, bounded by VE_{BC} and VE_{CA}. The identification of this region is done in algorithm 12.

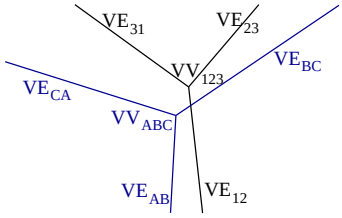
Lone Edge: The Voronoi edge that is alone in a Voronoi region belonging to the other level is called the *Lone Edge*. In figure 6.5(c), the *Lone Edge* for level i is VE_{12}, and for level j it is VE_{BD}. The identification of this edge is done in algorithm 11.



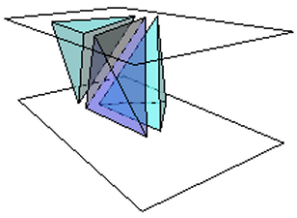
(a) Solution with three T_{12}



(b) Solution with two T_{12}



(c) Solution with just one T_{12}



(d) Tetrahedrons for solution shown in (c)

Figure 6.5: Special cases in the creation of the Joint Voronoi Diagram: Solutions for the *1ab23c* sub-case

Algorithm 10 formalizes the solution for this sub-case. The valid construction is composed by the T_{12} tetrahedron related to the intersection of both Lone Edges (line 15), and the T_i tetrahedrons created by each Delone triangle related to a Voronoi vertex and the Delone vertex related to the Full Region of the other level used as the apex (lines 7-11). In figure 6.5(d) the tetrahedrons related to this solution are shown.

Algorithm 10 Solving Vertex vs. Vertex 1*ab*23*c* sub-case

$[T_i, T_{12}]$ = **solveVertexVsVertex1ab23c**(VV_i, VV_j)

Input:	VV_i: Voronoi vertex on level i
	VV_j: Voronoi vertex on level j
Output:	T_i: set of TWO T_i tetrahedrons related to VV_i and VV_j
	$T_1 2$: a T_{12} tetrahedron
Precondition:	level of VV_i is not the same level of VV_j
	the projection of VV_i lies on the projection of VV_j
Postcondition:	the tetrahedrons in T_i and $T_1 2$ form a valid configuration

1: $Level_i$ = level of VV_i
2: $Edges[\,]$ = angular order of all edges related to VV_i and VV_j
3: $[Lone\_edge_i, Lone\_edge_j]$ = **findLoneEdges**($Edges$, $level_i$) alg.11
4: $[Full\_region_i, Full\_region_j]$ = **findFullRegions**($Edges$, $level_i$) alg.12
5: $base$ = Delone triangle related to VV_i
6: $apex$ = Delone vertex related to $Full\_region_j$
7: t_i = new T_i using $base$ and $apex$
8: add t_i to T_i
9: $base$ = Delone triangle related to VV_j
10: $apex$ = Delone vertex related to $Full\_region_i$
11: t_i = new T_i using $base$ and $apex$
12: add t_i to T_i
13: DE_i = Delone edge related to $Lone\_edge_i$
14: DE_j = Delone edge related to $Lone\_edge_j$
15: T_{12} = new T_{12} using DE_i and DE_j

Algorithm 11 Finding the Lone edges for the 1ab23c sub-case

[$Lone\_edge_i$, $Lone\_edge_j$] = **findLoneEdges**($Edges$, $level_i$)

Input: $Edges$[ ]: angular order of the edges
$level_i$: level used as reference

Output: $Lone\_edge_i$: Lone edge in level i
$Lone\_edge_j$: Lone edge in level j

Precondition: $Edges$ contain all the edges related to the Voronoi vertices
$Edges$ form an intersecting-star.

Postcondition: Only two lone edges are found.

```
1: for every Voronoi edge VE_k in Edges do
2:   set VE_p as the edge previous to VE_k in Edges
3:   set VE_n as the edge next to VE_k in Edges
4:   if level of VE_p is level of VE_n then
5:     if level of VE_k is Level_i then
6:       Lone_edge_i = VE_k
7:     else
8:       Lone_edge_j = VE_k
9:     end if
10:  end if
11: end for
```

Algorithm 12 Finding the Full Regions for the 1ab23c sub-case

$[Full\_region_i, Full\_region_j]$ = **findFullRegions**($Edges$, $level_i$)

Input: $Edges[\ ]$: angular order of the edges
$level_i$: level used as reference

Output: $Full\_region_i$: Full region in level i
$Full\_region_j$: Full region in level j

Precondition: $Edges$ contain all the edges related to the Voronoi vertices
$Edges$ form an intersecting-star.

Postcondition: Only two Full regions are found.

1: **for** every Voronoi edge VE_k in $Edges$ **do**
2: set VE_p as the edge previous to VE_k in $Edges$
3: set VE_n as the edge next to VE_k in $Edges$
4: set VE_{nn} as the edge next to VE_n in $Edges$
5: **if** level of VE_k is level of VE_n **then**
6: **if** level of VE_k is $Level_i$ **then**
7: $Full\_region_j$ = Region bounded by VE_p and VE_{nn}
8: **else**
9: $Full\_region_i$ = Region bounded by VE_p and VE_{nn}
10: **end if**
11: **end if**
12: **end for**

6.4 Elimination of tetrahedrons

The elimination of all the faces of the T_i tetrahedrons belonging to a non-solid connection leads to the creation of a incomplete surfaces. This defect is fixed in the version implemented as part of this project. When a T_i tetrahedron belonging to a non-solid connection is eliminated, a hole results in the place where its base stood. To avoid such holes, the horizontal triangles (bases) of the T_i tetrahedrons eliminated by non-solid connections are kept and included into the reconstructed surface.

The rule of elimination of T12 tetrahedrons is changed. The T12 tetrahedrons that are kept are 1-solid and 2-solid at the same time.

Chapter 7

Results for *PSM*

The improved method was applied to diverse sets of contours, and two of them, the most representative, are shown here.

7.1 Skull

The Skull is a set of 258 contours, placed on 63 planes parallel to the XZ plane. The resulting surface is composed by 39.808 triangles. This set of contours present wide m-n branches specially in the levels between the nose and the eye holes. In figure 7.1 a detail of levels 29 and 30 is shown. In figures 7.2 and 7.3 more details may be observed.

7.2 Brain

This set of contours is a complement given with the algorithm of the B+G method[1]. It is composed by 15 parallel levels, with 105 contours. The reconstructed surface has 13607 triangles. Figures 7.4 and 7.5 show more details.

[1] `ftp://ftp-sop.inria.fr/prisme/NUAGES/Nuages/`

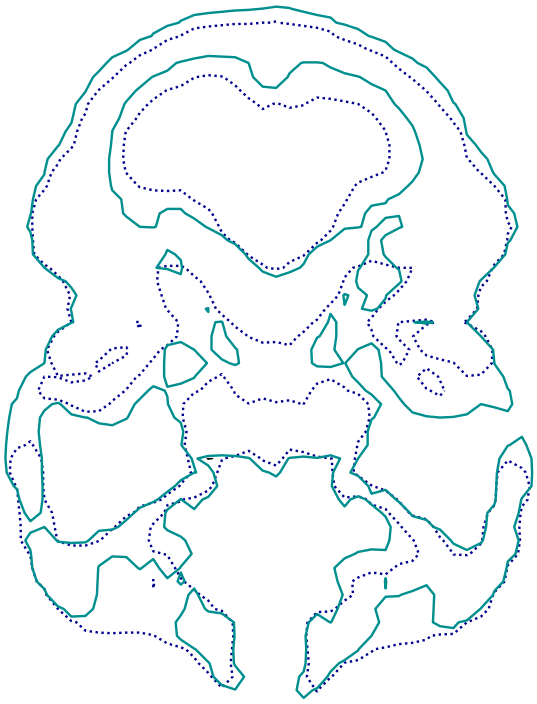
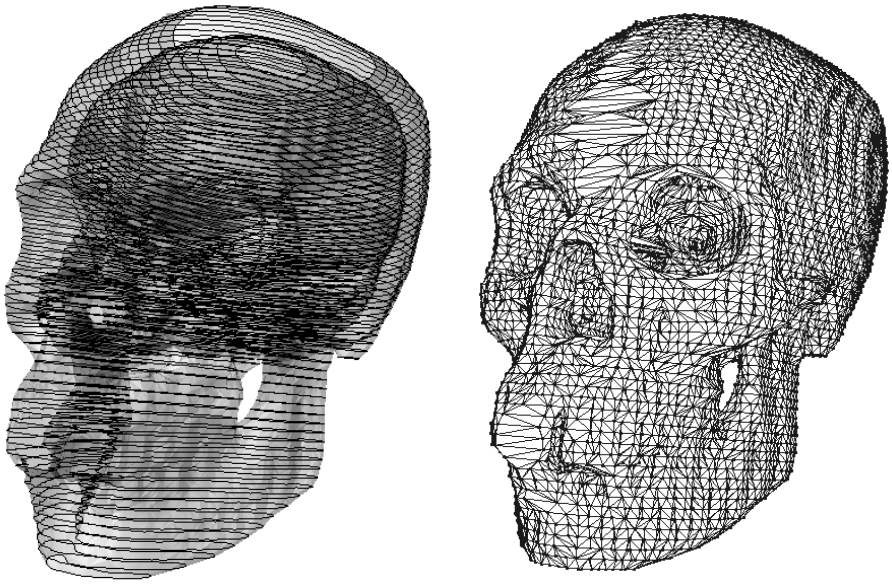


Figure 7.1: Detail of levels 29 and 30 of the set of contours “skull”



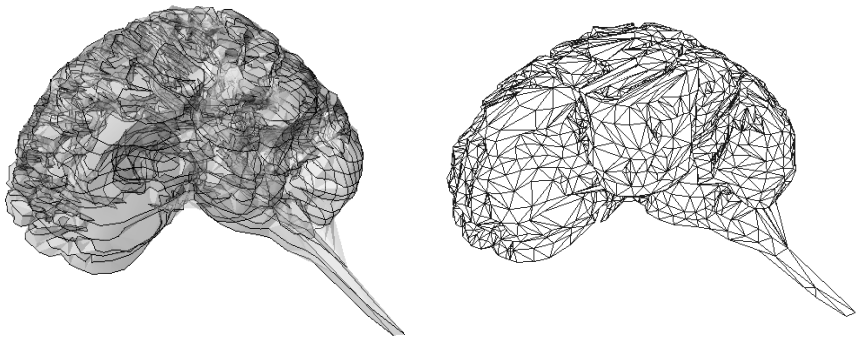
(a) Set of contours (b) Wireframe of the skull

Figure 7.2: Set of contours and the reconstructed surface

(a) Reconstructed surface in a transparent material

(b) Reconstructed surface in concrete

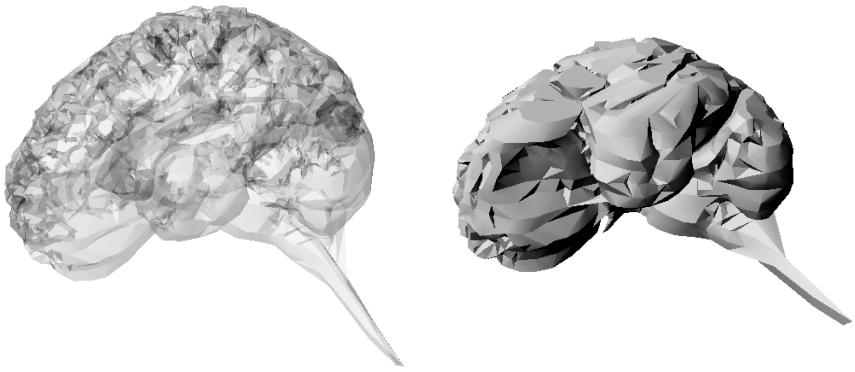
Figure 7.3: Reconstructed surface in different materials



(a) Set of contours

(b) Wireframe of the brain

Figure 7.4: Set of contours and the reconstructed surface



(a) Reconstructed surface in a transparent material

(b) Reconstructed surface in concrete

Figure 7.5: Reconstructed surface in different materials

Part III

Implicit Surface Method

Introduction for *ISM*

The *Implicit Surface Method*, or ISM, interpolates a surface from a set of planar contours with no restrictions about their spatial positioning. The process is based on the calculation of an implicit function and the extraction of the surface from such function. The interpolated surface is a smooth surface, that is convenient is some applications. The ISM is based on Greg Turk and James F. O'Brien's research work at the Georgia Institute of Technology in 1999 ([44]). It was originally used for shape transformation and it is being applied in different fields.

The ISM was implemented and tested in the A7 division - Cognitive Computing and medical imaging - of the Fraunhofer Institute for Computer Graphics - FhG IGD - at Darmstadt[2], during semester 2002-2 for a virtual Simulator of Radiotherapy Treatment Planning. The implementation was introduced as an auxiliary tool to be used by physicians to segment medical images separating different regions and organs of interest. In the classic approach, the physician must create a contour bordering the region or organ of interest on each image where it appears. A large organ may take more than 30 contours to be fully bordered. Using the ISM, an organ, such as the prostate, may be defined by 3 or 4 contours or a femur head by 11 contours. Because of the smooth results given, the resulting surfaces are much closer to the anatomical forms.

[2]http://www.igd.fhg.de/igd-a7/

Chapter 8

Theoretical basis for *ISM*

The Implicit Surface Method interpolates a surface by calculating a function. The interpolated surface is defined as the zero-set of an implicit function, which is calculated using the thin-plate interpolation method.

In the following sections a definition of implicit surfaces is given (section 8.1). Also, the thin-plate interpolation and the determination of the interpolation function are discussed (sections 8.2 and 8.3).

8.1 Implicit Surfaces

Generally a surface may be defined in different ways, but basically a surface is parametrically or implicitly defined. When the surface is defined by a parametric equation, it is called a parametric surface, and may be directly generated by solving such a function over a valid range, such as the upper half of a unit sphere centered at the origin defined in the first column of Table 8.1.

In the case of an implicit surface, it is not possible to directly calculate the points that belongs to it. An implicit surface is is defined by an implicit function and the points belonging to such surface must be tracked and selected depending on their function value. For a given value t, the t-surface is composed by all the points whose function value is t. In the second column of Table 8.1, the upper half of an unit sphere centered at the origin is defined as the 0-surface, or zero-set of f.

Table 8.1: Upper half unit sphere parametrically and implicitly defined

	Parametric	Implicit
Function	$f(x,y) = \sqrt{1-(x^2+y^2)}$	$f(X) = \|X\|^2 - 1$
Range	$x, y \in \mathbf{R}$ and $(x^2+y^2) \leq 1$	$X \in \mathbf{R}^2 \mathrm{x} \mathbf{R}^+$
Surface definition	$S = \{X \| X = [x, y, f(x,y)]\}$	$S = \{X \| f(X) = 0\}$

8.2 Thin-plate interpolation

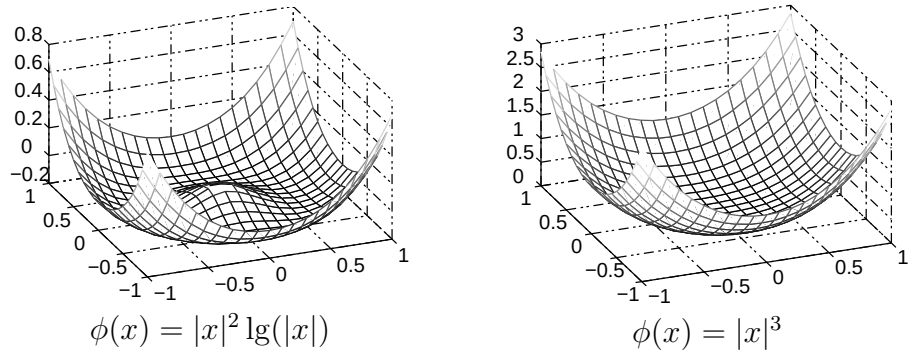
The thin-plate interpolation is a method commonly used to solve the scattered data interpolation problem ([44]), and it is widely used in the computer graphics domain, with sparse surface constraints ([20] and [43]). The target is to find a smooth function that matches a set of k locations $\{c_0, c_1, \ldots c_k\}$ with a set of k values $\{h_0, h_1, \ldots h_k\}$, and interpolates the space remaining in a smooth way. A pair composed by a location c_i and a value h_i is called constraint.

An energy function is used to measure the smoothness, and depending on its definition, different solutions can be reached. In 2D, the energy function is defined by Equation 8.1, which measures the aggregate curvature of f. Notice that the scattered data interpolation problem may be generalized to higher dimensions, using the appropriated energy function for each dimension.

$$E(f) = \int f_{xx}^2(x) + f_{yy}^2(x) + 2f_{xy}^2(x)\mathrm{d}x \tag{8.1}$$

where $f_{xx}(x)$ represents the second partial derivative in x direction, $f_{yy}(x)$ in y direction and $f_{xy}(x)$ in x and y directions.

The scattered data interpolation problem is solved when a function f that satisfies the constraints and has the minimum energy value is found. The family of Radial Basis functions $\phi(x)$ naturally solves this problem. The functions belonging to this family are circularly symmetric functions centered at a particular point. Duchon ([12]) proved that the biharmonic radial basis function, shown in Figure 8.1(a), solves this problem in 2D and the triharmonical radial basis function, shown in Figure 8.1(b), in 3D.



(a) biharmonical radial basis function

(b) triharmonical radial basis function

Figure 8.1: Radial basis functions used for 2D and 3D

8.3 Determination of the interpolation function

Using the appropriate radial basis function $\phi(x)$, the triharmonical radial basis function in the case of 3D, the interpolation function may be written as a weighted sum as it is shown in Equation 8.2.

$$f(x) = \sum_{j=1}^{k} w_j \phi(x - c_j) + P(x) \tag{8.2}$$

where c_j represents the locations of the constraints, w_j the weights, and $P(x)$ a degree one polynomial that accounts for the linear and constant portions of f. Solving for the weights w_j and the coefficients of $P(x)$ subject to the given set of constraints c_i and h_i, it produces a function that both interpolates the constraints and minimizes the energy function.

To calculate the interpolation function, the constraint locations and values are used. It is known that each constraint location c_i, will have h_i set as its function value, so, replacing this in equation 8.2, results in equation 8.3.

$$h_i = f(c_i) = \sum_{j=1}^{k} w_j \phi(c_i - c_j) + P(c_i) \tag{8.3}$$

The use of the whole set of constraints and adding some equations to

assure that a solution may be found, leads to a kxk equation system. The equation system, in matrix representation, is shown in equation 8.4, where ϕ_{ij} is $\phi(c_i - c_j)$.

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1k} & 1 & c_{1x} & c_{1y} & c_{1z} \\ \phi_{21} & \phi_{22} & \dots & \phi_{2k} & 1 & c_{2x} & c_{2y} & c_{2z} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{k1} & \phi_{k2} & \dots & \phi_{kk} & 1 & c_{kx} & c_{ky} & c_{kz} \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 & 0 \\ c_{1x} & c_{2x} & \dots & c_{kx} & 0 & 0 & 0 & 0 \\ c_{1y} & c_{2y} & \dots & c_{ky} & 0 & 0 & 0 & 0 \\ c_{1z} & c_{2z} & \dots & c_{kz} & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_k \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{8.4}$$

Notice that ϕ_{ij} and ϕ_{ji} are equal and that only ϕ_{ii} is zero, so, the matrix to the left of the equation 8.4 is mostly polulated and symmetric. This peculiarity needs to be taken into account when the equation system is being solved.

Chapter 9

The Implicit Surface Method

The *ISM* is based on the calculation of an implicit surface. The surface is defined as the zero-set of a calculated interpolation function.

The input is a set of planar contours, with no restrictions about their spatial positioning between them. The contour points are used to calculate an implicit function, whose zero-set contains the contour vertices and interpolates a smooth surface.

The whole process is composed by three main steps. First, the extraction of the constraints from the given contours is done. Then, the constraints are used to calculate the implicit function and lastly, the contours are tracked or the surface is extracted from the function.

9.1 Definition of constraints

The vertices of the given contours and some other complementary points are used to define the constraint locations and its related h_i values.

Notice that the determination of the interpolation function (section 8.3) only requires a set of locations c_i, and its related value h_i. The input given for this method is formed by a set of contours describing the surface. The contour vertices will be in the zero-set of the interpolation function, implying that its h_i value is zero. If only these points are used to calculate the function, then, there are no restrictions for the function to set the whole space as the zero-set ($f(x) = 0$ for all $x \in R^3$). In order to avoid such trivial function, some complementary constraints locations must be found. Some external or internal points, respect to the surface, must found and added.

Every vertex of the given contours, related with an h_i value of 0, is used to create the set of *boundary constraints*. To find internal and external points respect to the surface, two margins (an outer and inner) are built. The vertices of the inner margin are associated with an h_i value of $+1$. Those points form the *internal constraints*. In a similar way, the vertices of the outer margin, related with an h_i value of -1 compose the *external constraints* (algorithm 13).

Algorithm 13 Definition of the set of constraints

$[c_i, h_i]$ = **contoursToConstraints**(C)

Input:	C: Set of contours
Output:	c_i: locations in the space of the constraints
	h_i: value of the constraint in the given location
Precondition:	every contour in C is planar
Postcondition:	every c_i has a corresponding h_i value
	there are at least two different values in h_i

```
1: for every contour c in C do
2:   for every vertex v in c do
3:     add v to c_i
4:     assign the related value h_i of c_i as BOUNDARY_VALUE (0)
5:   end for
6:   set inner as the internal margin of c to a TINY distance
7:   for every vertex v in inner do
8:     add v to c_i
9:     assign the related value h_i of c_i as INTERNAL_VALUE (+1)
10:  end for
11:  set outer as the external margin of c to a TINY distance
12:  for every vertex v in outer do
13:    add v to c_i
14:    assign the related value h_i of c_i as EXTERNAL_VALUE (−1)
15:  end for
16: end for
```

In some applications, the contours are manually created, for example, they are drawn on perpendicular medial images. It occurs that in some "critical" places -like the plane intersections- the contour points are set really near to each other, with no fixed position. It leads the process to create a surface

with abrupt changes of curvatures that do not abide to the wished result. To avoid these kinds of artefacts, some constraints must be filtered. The filter replaces such inconsistent points for their centroid. To identify these points, the box that bounds all the contour points is found, and the points that fit in a box of 5% of the size of the original bounding box are selected to be replaced.

9.2 Calculation of the implicit function

Now, the attention is focused on the generation of the function that defines the surface. The constraints that have been generated in the previous step, locations c_i and values h_i, are set into the matrix in Equation 8.4 and the equation system is solved using LU-decomposition. The calculation of the coefficients is done in algorithm 14. It is important to note that, because of the nature of this function, every constraint influences the whole interpolation, and a change on any of these represents a re-calculation of the coefficients. It is not important in the current context, but for future work or extensions, this condition could be critical and must be taken into account. The coefficients that are calculated here, w_i, P_0, P_1, P_2 and P_3, are used to evaluate the function on any point later.

The *LU-decomposition* ([31]) is a method faster enough for the application requirements. For systems with more than a few thousands of constraints, this method is unfeasible, because of the complexity of this methid ($O(n^3)$). There are some works that have tried and improved this fact, ([24]), where a *compactly supported radial basis function* is used, changing the performance to $O(n^2)$, but making it impossible to use the third step of the method described here.

9.3 Extraction of the surface or parallel contours

As it was said before, the implicit surfaces can not be directly generated. It must be extracted or tracked from the calculated implicit function.

In the case of surfaces, the *Bloomenthal's polygonizer* ([4]) is used. It gives good results and creates surfaces represented as a set of triangles.

Algorithm 14 Calculating the implicit function

$coef$ = **implFunctionCalculate**(c_i, h_i)

Input:	c_i: locations in the space of the constraints h_i: value of the constraint in the given location
Output:	$coef$: coefficient of the Implicit function, composed by the weights w_i and the polynomial coefficients.
Precondition:	At least one external or internal constraint must be defined
Postcondition:	The coefficients defining an implicit function which contains the implicit reconstructed surface.

1: M = matrix to the left in equation 8.4 using c_i.
2: B = vector to the right in equation 8.4 using h_i.
3: $coef$ = solve equation system Mx$coef = B$ using *LU-decomposition*

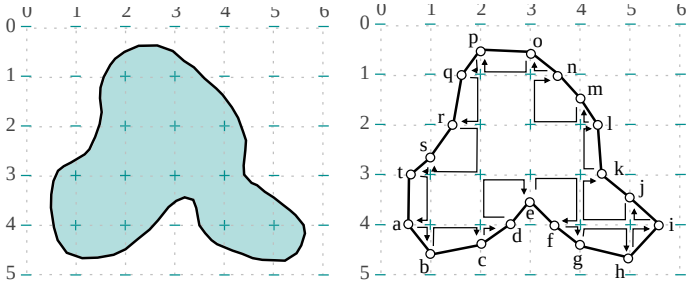
9.3.1 Extraction of parallel contours

A routine has been designed to extract parallel contours from the implicit function. This routine is applied level by level, to the levels where the contours are to be extracted. It is based on the *cell polygonization* of *Bloomenthal's polygonizer* ([4]). Before using this routine on each level, the steps bellow must be followed:

1. Generation of a planar grid in the current level
2. Evaluation of the function at each corner of the grid
3. Contour extraction

The grid is generated as a regular spaced grid, where each corner is a point $[x, y, z]$ lying on the plane of the level being currently processed. Each corner of the grid is then evaluated using the calculated implicit function, as shown in equation 8.2. To extract a contour, any edge in the grid whose end-points have set a function value with opposite sign (+ and −) must be found. Because this function is continuous, the change of sign implies that between the edge vertices there is a point, known as *root*, whose function value is zero. Binary sectioning over the edge is used to localize the position of the *root*. After finding the *root*, the direction along the edge to the positive

corner is selected and it keeps rotating on clockwise sense till another edge with opposite signed end-points is found. This iterates until the contour is closed or the limits of the grid are reached.



(a) Original contour and the evaluated grid

(b) Tracked contour defined by the points $\{a, b, c, ...t\}$

Figure 9.1: Scheme of the algorithm to extract a contour on a level grid

Chapter 10

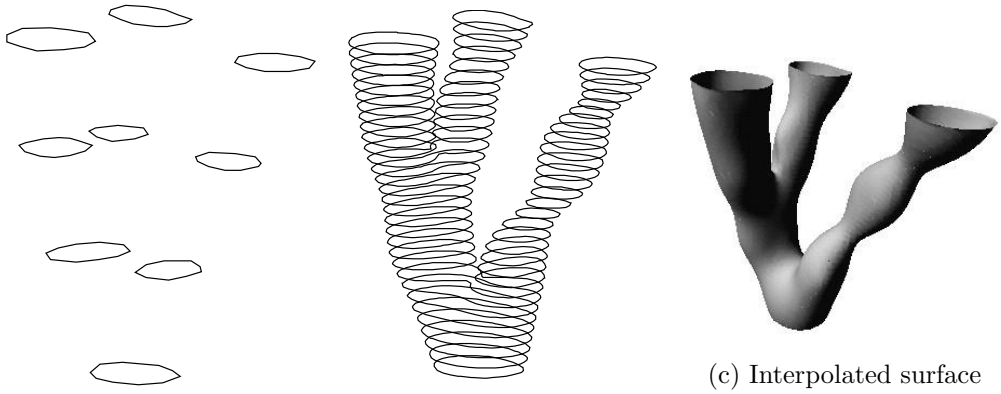
Results for *ISM*

The process described here was applied to several data sets, with positive results. Here three examples are shown, two of them were created by hand and the other one corresponds to real data. Every example presents the contours used as input, a set of interpolated parallel contours extracted with the contour tracker previously described and finally the surface extracted with *Bloomenthal's poligonizer*.

10.1 Synthetic data

The branching example, see Figure 10.1, describes a branching as it could be found in veins. It is composed of a set of 9 planar contours placed on 4 parallel planes with an interplane distance of 5. In total there are 92 contour points, which are translated into 276 constraints (including boundary, internal and external constraints). The interpolated parallel contours run in direction $\mathbf{Z}^+$ with a distance among them of 0.48.

The tamarind example, see Figure 10.2, could describe a tamarind skin. It is composed of a set of 5 planar contours placed on five orthogonal planes, three of them being parallel to the XY plane, one to the XZ plane and one to the YZ plane. In total there are 57 contour points, which are translated into 171 constraints (including boundary, internal and external constraints). The interpolated parallel contours run in direction $\mathbf{Z}^+$ with a distance among them of 0.37.



(a) Input data

(b) Interpolated parallel contours

(c) Interpolated surface

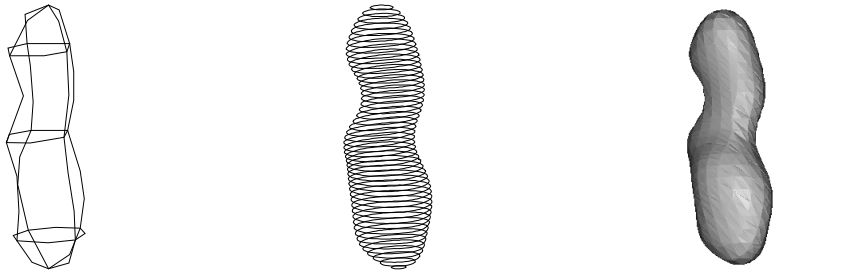
Figure 10.1: Synthetic data: A branching pipe

10.2 Real Data

The last example, see Figure 10.3, is a femur head. The contours were taken from medial images, in EXOMIO, from a CT image. It is composed of a set of eleven planar contours placed on eleven orthogonal planes, five of them being parallel to XY plane, two to XZ plane and four to YZ plane. In total there are 136 contour points, which are translated into 408 constraints (including boundary, internal and external constraints). Filtering is required in this example. The interpolated parallel contours run in direction $\mathbf{Z}^+$ with an interplane distance of 3.13. Every plane matches every level in the CT image, creating a contour that segments the bone head in each image.

10.3 Use of constraints

In section 9.1, the external and internal contours were defined using margins. As a testing, the *internal constraints* were not used, and the set of *external constraints* was reduced to just eight points. Those points are the corners of the bounding box of the given contours, and its h_i value was given by the distance of each corner to its closest vertex of the contours. The reduction in the number of constraints is notable, from $3n$ with n as the number of contours vertices, to $n + 8$. This reduction is appreciated when objects with



(a) Input data

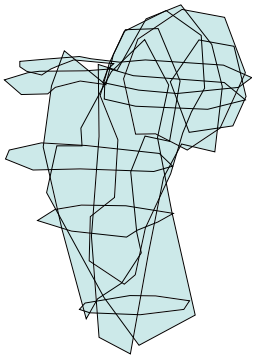
(b) Interpolated parallel contours

(c) Interpolated surface

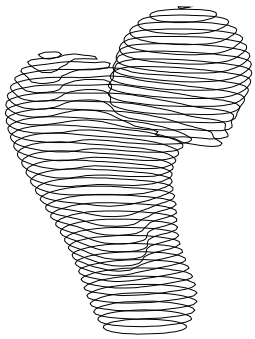
Figure 10.2: Synthetic data: A tamarind skin

large number of contours vertices is processed.

The results of this test showed that the surface reconstructed with this constraints differs from the surface using the whole set of constraints. For simple objects the difference is not evident, and the surface still describes the wished form (figure 10.4). For complex objects with sparse contours vertices, like the femur head, the deformation may cause holes and big changes in the surface which are not wished. (figure 10.5). This test is interesting, because it opens a possible path to improve this method.



(a) Input data



(b) Interpolated parallel contours

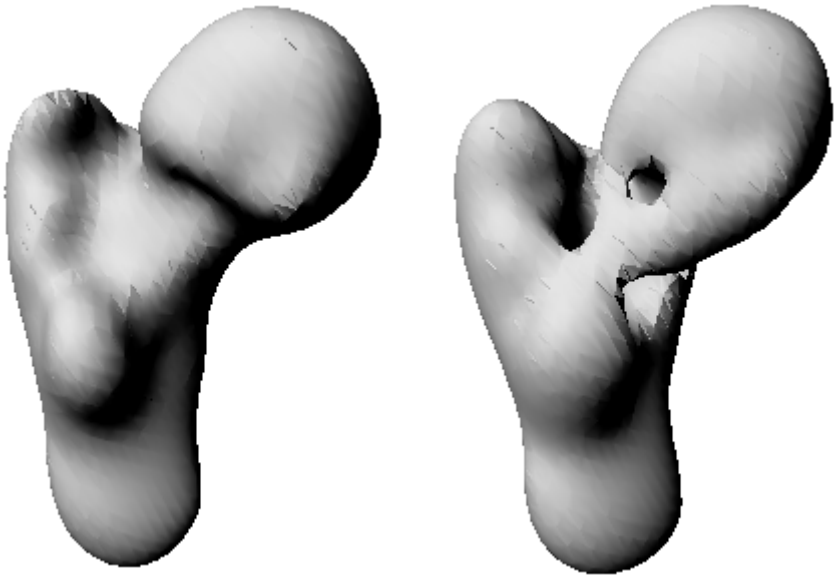
(c) Interpolated surface

Figure 10.3: Real data: A femur head

(a) Using the whole set of constraints

(b) Using only the corners of the Bounding Box

Figure 10.4: Differences in the reconstructed surfaces using different sets of constraints for a simple object



(a) Using the whole set of constraints

(b) Using only the corners of the Bounding Box

Figure 10.5: Differences in the reconstructed surfaces using different sets of constraints for a complex object

Part IV

Conclusions and Future Work

Chapter 11

Conclusions

11.1 For the Implicit Surface Method

The reconstructed surface is smooth, as seen in 10. If this surface is cut using the planes where the input contours were defined, the resulting contours will really differ from the ones used as input. This change is not convenient when the input contours must continue in the resulting surface, like applications where different parts are individually produced to be latter assembled. However, when the input contours are just a raw description of the surface to be reconstructed, such introduced changes to the contours and the surface are really of use. For example when the reconstructed surface represents an organ or any anatomical form, any aesthetic object or any kind of object that must not contain any sharp edge, like toys for children.

The reconstructed surface is optimum to be manufactured by a CAM machine, such as milling machines or a lathe, because the defining function is infinitely derivable.

A drawback is the fewer control present from the user. At the moment, the control of the surface may be done only by handling the points of the contours.

The contraints are limited to a few thousands of them, because an equation system of $n+4$ by $n+4$ unknowns must be solved, where n is the number of constraints.

The surface must be extracted from the calculated function, and an additional step is required. For this project, the *Bloomenthal's polygonizer* is used to extract the surface.

Sets of contours that naturally do not indicate a closing, results in a surface that grows with no control (figure 11.1). To solve such problem, a Bounding Box must be used to cut the reconstructed surface, but the time consumed by the extraction of this surface is not recovered. In figure 11.1 the surface extracted for the branching example before bounding, originally shown in figure 10.1, may be seen.

Figure 11.1: No bounded interpolated surface

11.2 For the Polyhedral Surface Method

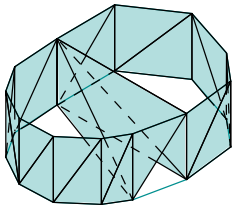
11.2.1 Incomplete surface

In the original method, some horizontal triangles belonging to the reconstructed surface are missing. This triangles are eliminated when the T_i tetrahedrons belonging to a *non-solid connection* are eliminated . To avoid such

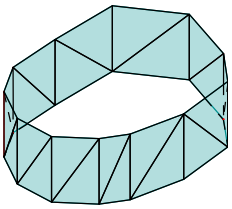
holes, the horizontal triangles corresponding to the T_i tetrahedrons that are eliminated for the *non-solid connections* are kept and used as part of the reconstructed surface. (section 5.5). Such holes creates a surface that is no "watertight". This means, that the reconstructed surface does not allow the classification of the points in the space as internal or external respect to the surface, or in graphical terms, it allows "water" pass in or out.

11.2.2 No-manifold situations

In section 6.3, special cases were considered when creating the *Joint Voronoi Diagram*. These special cases are not considered in the original version of this method ([19]) and when more than four co-spherical points are found, a perturbation is applied to them. This perturbation leads, some times, to non-wished surfaces, like the one shown in figure 11.2(a). The consideration of the special cases does not leave the election of the tetrahedrons to a random perturbation, and improves the reconstructed surfaces (figure 11.2(b)).



(a) Surface reconstructed using the original method



(b) Surface reconstructed taking into account the special cases

Figure 11.2: Differences between a simple surface reconstructed using a perturbation and solving the special cases

However, in some situations, this no-manifold situation may not be eliminated, as the one show in figure 11.3. In these situations, one edge is matched or mapped to more than one edges on the consecutive level, and more than two faces share the same edge.

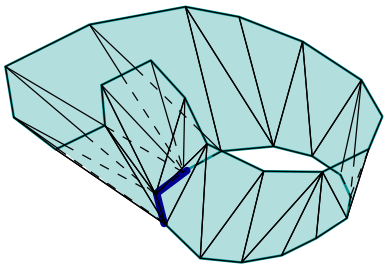
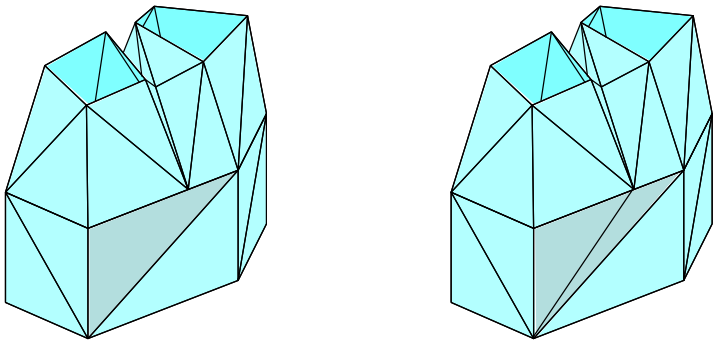


Figure 11.3: A non-manifold situation that may not be eliminated

11.2.3 Watertight surface

Because the method processes each pair of levels at one time, the assembled surface is not watertight, and a final step of homogenization of vertices must be performed. Notice that a contour goes twice to the process, once as level i and the next one as level j. On each of this iterations, different points may have been added to the same contour, and the resulting triangles may not share the same edges. This holes allow the "water to pass in". So, such triangles must be found and divided in two triangles, inserting the non-common vertex in both reconstructions. See figure 11.4(a) and 11.4(b) for details.



(a) No watertight surface (b) Watertight surface

Figure 11.4: Points added to a surface to ensure it is watertight

Chapter 12

Future Work

12.1 Implementation details in *PSM*

PSM method is based on Voronoi Diagrams and Delone Triangulations. It creates a graph, named the *Joint Voronoi Diagram*, from the intersection of two Voronoi Diagrams. This intersection is done intersecting every edge belonging to one Voronoi Diagram with every edge in the other Voronoi Diagram. Such procedure is time-consuming, and may be improved. Some heuristic and methods exist and may be used, to speed this part of the process.

12.2 Support for internal holes or objects in *ISM*

The *ISM* has some limitations when a model with holes or nested objects is reconstructed. Notice that the internal holes or objects are also surfaces, and they may or may not be defined by the zero-set of the function. As seen, the ISM is not strict in the definition of constraints, so, internal objects or holes may be defined as a k-surface, where k indicates the value of the function on the points of the surface. The value of k must be different for each surface being reconstructed. This implies that the number of surfaces to reconstruct must be known, and the relations between them must be also known, to set a consistent numbering of the surfaces. If this focus is taken, the terms '*external constraint* and *internal constraint*' lose its meaning, and

the k value must be used as the h_i value for each defined constraint. If the internal holes or objects are defined as part of the zero-set of the calculated implicit function, then, the *external* and *internal constraints* must be used. Notice that an internal constraint lies inside the object, so, for a contour that represents a hole, its *internal constraints* are defined as the vertices of the external margin, instead of the internal margin. Notice that, in any case, the relations of the contours, translated to the knowledge of which contours belongs to an internal wall (or hole) and which to an external wall, must be known. To determine this relation between contours that do not share the same plane is also an issue to take into account when extending the *ISM* to support internal holes or objects.

An important point to take into account is the extraction of the surfaces. Actually, it is done using the *Bloomenthal's polygonizer* and requires a seed point to extract one surface. A set of seed points, one per surface to extract, must exist, or the polygonizer must be changed to be able to find by itself the surfaces to extract.

12.3 Use of a different Radial Basis Function for *ISM*

In the case of using *ISM* for larger sets of data, some parts of the method must be changed. The critical part lies in the calculation and evaluation of the implicit function. The calculation of coefficients of the implicit function is done using *LU-decomposition*, as it was seen in section 9.2. The equation system that is resolved in that step is based on the chosen *Radial Basis* function. Using the *triharmonical radial basis function*, the matrix of coefficients of the equation system has only a small number of zero elements. If a different *Radial Basis function* is used, such as the *Compactly-supported Radial Basis Function* ([9]), the number of zero elements may grow enough to create a *sparse* matrix, whose LU-decomposition may be computed in less time. Nevertheless, the use of this radial basis will change the interpolation function, and the *Bloomenthal's polygonizer* may not be adequate to extract the surface.

BRep file Grammar

brep_file	→	**NUM_SHELL_SETS** cs pos_int_n0 cs shell_sets **EOF**
shell_sets	→	shell_set (cs shell_set)*
shell_set	→	**BEGIN_SHELL_SET** cs shell_set_cont cs **END_SHELL_SET**
shell_set_cont	→	**NUM_SHELLS** cs pos_int_n0 cs shells
shells	→	shell (cs shell)*
shell	→	**BEGIN_SHELL** cs shell_cont cs **END_SHELL**
shell_cont	→	vertices_part cs edges_part cs faces_part cs contours_part
contour_part	→	**CONTOURS** cs pos_integer cs contours
contours	→	contour (cs contour)*
contour	→	left_par pos_integer right_par (cs pos_integer)+
faces_part	→	**FACES** cs pos_int_n0 cs faces
faces	→	face (cs face)*
face	→	edge_partner cs edge_partner cs edge_partner
edge_partner	→	pos_integer left_par pos_integer right_par
edges_part	→	**EDGES** cs pos_int_n0 cs edges
edges	→	edge (cs edge)*
edge	→	pos_integer cs pos_integer
vertices_part	→	**VERTICES** cs pos_int_n0 cs vertices
vertices	→	point3 (cs point3)*
point3	→	float_number cs float_number cs float_number
cs	→	(space_char* | tab* | end_line* | comment*)+
comment	→	# λ end_line
tab	→	**'\t'**
space_char	→	**' '**
end_line	→	**'\n'**
pos_int_n0	→	+? **0*** pos_digit digit*
pos_integer	→	+? digit+
float_number	→	(op_sign digit+ (**.**digit+)? (exp op_sign digit+)?) | ((op_sign digit*)? **.**digit+ (exp op_sign digit+)?) | ((op_sign digit*)? (**.**digit+)? exp op_sign digit+))
exp	→	**e**|**E**
op_sign	→	(+|-)?
right_par	→	**)**
left_par	→	**(**
digit	→	**0** | pos_digit
pos_digit	→	**1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

Conventions:
$^{?}$ → zero or one occurrence.
* → zero or more occurrences.
$^{+}$ → one or more occurrences.
| → exclusive or (xor).
( ) → grouping of a subexpression.
λ → Any possible character.
a → a is a keyword.
a → a is an expression defined later.

Example File

The example file represents the surfaces shown in figure 1.

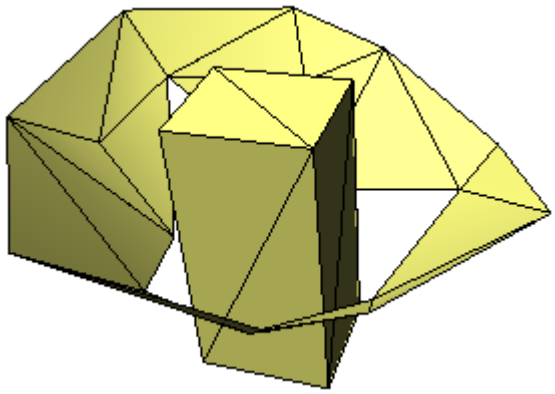


Figure 1: Surface represented by the example file

```
NUM_SHELL_SETS 1 # just one shell_set in this file
BEGIN_SHELL_SET
NUM_SHELLS 2 # two shells for the shell_set
BEGIN_SHELL # shell number 1
VERTICES 8 # 8 vertices in shell 1
2.404294 -2.154735 10 .2404294e-1 -2.154735 0 2.404295 2.056687
10 -2.465704 -2.154735 0 -2.465704 -2.154735 10 -2.465704
2.056687 10 -2.465704 2.056687 0 2.404294 2.056687 0
EDGES 36 # 36 edges in shell 1.  It is 3 times the number of
faces
0 1 1 2 2 0 1 0 0 3 3 1 4 3 3 0 0 4 5 4 4 0 0 5 0 2 2 5 5 0 6 5 5
2 2 6 2 7 7 6 6 2 7 2 2 1 1 7 7 1 1 3 3 7 3 6 6 7 7 3 5 6 6 3 3 5
3 4 4 5 5 3
FACES 12 # number of faces in shell 1
0(3) 1(22) 2(12) 3(0) 4(7) 5(25) 6(33) 7(4) 8(10) 9(34) 10(8)
11(14) 12(2) 13(16) 14(11) 15(30) 16(13) 17(20) 18(21) 19(28)
20(17) 21(18) 22(1) 23(24) 24(23) 25(5) 26(29) 27(31) 28(19)
29(26) 30(15) 31(27) 32(35) 33(6) 34(9) 35(32)
CONTOURS 0 # no contours in shell 1
END_SHELL
# a comment may be placed any where in the file
BEGIN_SHELL # shell number 2
VERTICES 18 # number of vertices in shell 2
-8 6 10 -2 9 9.219544 -6 5 7.810250 -1 7 7.071068 2 8 8.246211
0 4 4 6 2 6.324555 7 3 7.615773 9 -3 9.486833 4 -2 4.472136 5 -6
7.810250 1 -5 5.099020 3 -8 8.544004 -5 -7 8.602325 -3 -4 5 -9 -1
9.055385 -4 0 4 -7 1 7.071068
EDGES 60 # 3 edges/face x 20 faces = 60 edges in shell 2
0 1 1 2 2 0 2 1 1 3 3 2 1 4 4 3 3 1 3 4 4 5 5 3 4 6 6 5 5 4 4 7 7
6 6 4 7 8 8 6 6 7 6 8 8 9 9 6 9 8 8 10 10 9 11 9 9 10 10 11 11 10
10 12 12 11 13 11 11 12 12 13 14 11 11 13 13 14 15 14 14 13 13 15
15 16 16 14 14 15 17 16 16 15 15 17 15 0 0 17 17 15 17 0 0 2 2 17
2 16 16 17 17 2 2 3 3 5 5 2
FACES 20 # 20 faces
0(6) 1(3) 2(52) 3(1) 4(8) 5(57) 6(15) 7(9) 8(4) 9(7) 10(14)
11(58) 12(17) 13(59) 14(10) 15(18) 16(20) 17(12) 18(25) 19(21)
20(16) 21(19) 22(24) 23(13) 24(22) 25(31) 26(28) 27(23) 28(26)
29(30) 30(29) 31(35) 32(34) 33(37) 34(32) 35(41) 36(27) 37(33)
38(40) 39(44) 40(38) 41(48) 42(46) 43(36) 44(39) 45(55) 46(42)
47(50) 48(0) 49(51) 50(47) 51(49) 52(2) 53(56) 54(43) 55(45)
56(53) 57(5) 58(11) 59(54)
CONTOURS 2 # the shell has two contours.
(9) 0 6 15 18 25 31 35 41 48
(7) 43 36 27 23 13 59 54
END_SHELL
END_SHELL_SET
```

Bibliography

[1] N. Amenta and M. Bern. Surface reconstruction by voronoi filtering. In *Symposium on Computational Geometry*, pages 39–48, 1998.

[2] F. Aurenhammer and R. Klein. Voronoi diagrams. In J. R. Sack and G. Urrutia, editors, *Handbook of Computational Geometry*, pages 201–290. Elsevier Science Publishing, 2000.

[3] G. Barequet and M. Sharir. Piecewise-linear interpolation between polygonal slices. *Computer Vision and Image Understanding: CVIU*, 63(2):251–272, 1996.

[4] Jules Bloomenthal. An implicit surface polygonizer. In Paul Heckbert, editor, *Graphics Gems IV*, pages 324–349. Academic Press, Boston, 1994.

[5] J. D. Boissonnat. Shape reconstruction from planar cross-sections. *Computer Vision, Graphics and Image Processing*, pages 1–29, 1988.

[6] J. D. Boissonnat and M. Teillaud. A hierarchical representation of objects: the Delaunay Tree. In *Second ACM Symposium on Computational Geometry*, pages 260–268, 1986.

[7] J. D. Boissonnat and M. Teillaud. On the randomized construction of the Delaunay Tree. *Theoretical Computer Science*, 112(2):339–354, 1993.

[8] A. Bowyer. Computing dirichlet tessellations. *The Computer Journal*, 24(2):162–166, 1981.

[9] J.C. Carr, T.J. Mitchell, R. K. Beatson, J.B. Cherrie, W.R. Fright, B.C. McCallum, and T.R. Evans. Reconstruction and representation of 3d

objects with radial basis functions. In *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, pages 67–76, 2001.

[10] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry*, 4(1):387–421, 1989.

[11] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1996)*, pages 303–312, 1996.

[12] J. Duchon. Splines minimizing rotation-invariant semi-norms in sobolev spaces. In *Constructive Theory of Functions of Several Variables*, pages 85–100, 1977.

[13] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, 1987.

[14] H. Edelsbrunner and E. P. Muecke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13:43–72, 1994.

[15] H. Edelsbrunner and N. Shah. Incremental topological flipping works for regular triangulations. In *Proceedings of the 8th ACM Symposium on Computational Geometry*, pages 43–52, 1992.

[16] S. Fortune. Voronoi Diagrams and Delaunay Triangulations. In Ding-Zhu Du and Frank Hwang, editors, *Computing in Euclidean Geometry, Lecture Notes Series on Computing*, pages 193–223. World Scientific, 1992.

[17] G. K. Francis and J. R. Weeks. Conway's zip proof. *American Mathematical Monthly*, 106(1):393–399, May 1999.

[18] M. J. García, O. Ruiz, and C. Cadavid. Syntesis of 1- and 2-pl manifolds. Research report, Universidad EAFIT, Medellín, Colombia, 2003.

[19] B. Geiger. Three dimensional modeling of human organs and its application to diagnosis and surgical planning. Research Report 2105, INRIA, Sophia-Antipolis, Valbonne, France, 1993.

[20] W. E. L. Grimson. Surface consistency constraints in vision. *Computer Vision, Graphics, and Image Processing*, 24(1):28–51, october 1983.

[21] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics*, 2(4):74–123, 1985.

[22] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. In G Goos and J Hartrnarns, editors, *ICALP 90 Proceedings*, number 3 in Lecture Notes in Computer Science, pages 414–431. Springer-Verlag, 1990.

[23] H. Hoppe. *Surface Reconstruction from Unorganized Points*. PhD thesis, Computer and Engineering, U. of Washington (USA), 1994.

[24] B. S. Morse, T. S. Yoo, P. Rheingans, D. T. Chen, and K. R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Shape Modelling International*, pages 89–98, May 2001.

[25] M. Morse. *The calculus of variations in the large*. American Mathematical Society, 1934.

[26] M. Mortenson. *Geometric Modeling*. Wiley Computer Publishing, 2 edition, 1997.

[27] E. P. Muecke, I. Saias, and B. Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. In *12th Annual ACM Symposium on Computational Geometry*, pages 274–283, 1996.

[28] K. Mulmuley. Randomized multidimensional search trees: Dynamic sampling. In *7th ACM Symposium on Computational Geometry*, pages 121–131, 1991.

[29] T. Ohya, M. Iri, and K. Murota. A fast Voronoi-diagram with quaternary tree bucketing. *Information Processing Letters*, 18:227–231, 1984.

[30] F. Preparata and M. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, 1985.

[31] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C: The art of scientific computing*. Cambridge University Press, 2 edition, 1992.

[32] V. T. Rajan. Optimality of the Delaunay triangulation in r^d. In *7th Annual Symposium on Computational Geometry*, pages 357–363, 1991.

[33] Wolfram Research. Eric Weisstein's world of mathematics. Internet Publication. Available from `http://mathworld.wolfram.com/`.

[34] O. Ruiz. Digitlab, an environment and language for manipulation of 3d digitizations. In *Proceedings of the Joint Conference: IDMME'2000 and CSME FORUM 2000*, May 2000.

[35] O. Ruiz. *Understanding CAD / CAM / CG*. American Society of Mechanical Engineers ASME. Continuing Education Institute. Global Training, 2002. ASME Code GT-006.

[36] O. Ruiz and C. Cadavid. Boolean 2d shape similarity for surface reconstruction. In *Visualization, Imaging and Image Processing (VIIP 2001)*, september 2001.

[37] O. Ruiz, C. Cadavid, and M. Granados. Evaluation of 2D shape likeness for surface reconstruction. In *XIII International Congress on Graphics Engineering*, june 2001.

[38] O. Ruiz, C. Cadavid, M. Granados, S. Peña, and E. Vásquez. 2d shape similarity as a complement for voronoi-delone methods in shape reconstruction. Submitted to Computer and Graphics.

[39] O. Ruiz, C. Cadavid, M. Granados, S. Peña, and E. Vásquez. 2d similarity-complemented voronoi-delone methods in shape reconstruction. Submitted to International Journal of Computer and Applications.

[40] Vladimir V. Savchenko, Alexander A. Pasko, Oleg G. Okunev, and Tosiyasu L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14(4):181–188, 1995.

[41] M. Shamos and D. Hoey. Closest-point problems. In *16th Annual IEEE Symposium on Foundations of Computer Science*, pages 151–162, 1975.

[42] M. I. Shamos. *Computational Geometry*. PhD thesis, Department of Computer Science, Yale University (USA), 1978.

[43] D. Terzopoulos. The computation of visible-surface representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(4):417–438, July 1988.

[44] Greg Turk and James F. O'Brien. Shape transformation using variational implicit functions. In *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 99)*, pages 335–342, august 1999.

[45] Greg Turk and James F. O'Brien. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics*, 21(4):855–873, october 2002.