Delaunay Reconstruction from Multiaxial Planar Cross-Sections

C. R. Dance and R. W. Prager

CUED/F-INFENG/TR 273

January 1997

Department of Engineering
University of Cambridge
Cambridge CB2 1PZ
England

1 Introduction

"Given a set of cross-sections of an object, make an approximate representation of its surface." This is known as the *surface from cross-sections* problem. Here, the approach selected to solving this problem is to generate *triangulated* surfaces.

Many solutions to this contour interpolation problem are possible and for the case of parallel planar contours many solutions have been proposed. However, this work focuses on the harder and more general case of arbitrarily oriented (*multi-axial*) planes, which has received much less attention. Indeed, it appears that only one algorithm for this case has previously been published (by Payne and Toga [46]).

Ideas from [46] are incorporated in the procedure developed here, but the restrictive assumptions on the shape of the contours which are required by their method are removed. This is achieved by generalising the ideas of Boissonnat and Geiger [5, 7, 24] on Delaunay reconstruction to the multi-axial case. The latter authors use an efficient two-stage procedure for triangulating between parallel planes, which they have been able to generalise to treat two but not more non-parallel planes in [6]. Here, this is replaced by a remarkably simple alternative developed by the same group [8].

Reconstruction from contours is an attractive problem since it provides an opportunity to use computational geometry and graph theory with visually appealing results. More importantly, good solutions to it have great practical applicability, as the following examples show:

- **Biological Morphology.** Reconstructions from contour data obtained by CT and MRI are exploited for surgery and radiation therapy planning, volumetry, prosthesis milling and finite element simulation [14, 24]. They have also already been used for volumetry in ultrasound [27] and in the microscopy of cells [33].

- **Industry.** In computer aided design, lofting techniques specify and then generate the geometry of an object as a sequence of sections [47] [1]. Robotics and reverse engineering use laser range imaging which often leads to reconstruction from sections. Non-destructive evaluation sometimes employs interpolation of contours from ultrasound or X-ray slice data.

- **Geology/Geography.** Applications have been made in the reconstruction of terrain from topographic elevation data and of invertebrate fossils from serial grinding data [26]. Sedimentology and seismology should also find the methods useful.

Here, non-parallel reconstruction is motivated simply by the fact that the data has been sampled this way. However, the non-parallel case generally offers a number of advantages over the parallel case:

- **Connectivity.** With parallel planes, all connectivity between slices must be guessed.

[1] Ships can be made by solving the *surface from cross-sections* problem physically: first laying out a set of transverse ribs to guide the shape and then nailing longitudinal strips to these. This takes up much space and therefore used to be done in the lofts of shipyard buildings, hence the name *lofting* [68].

Intersecting planes can definitively show that regions are connected: the data may provide absolute constraints on topology between planes.

- **Object Shape.** Objects whose shape variation is not well-captured by parallel planes may be more naturally sampled.

- **Triangulation.** Triangulation of non-parallel sampled data usually involves less points from smaller regions in each step (this is further elucidated in Section 6.6). This means that these steps can be performed faster and that the effects of any inconsistency of a triangulation with the true shape of the object being reconstructed are propagated over a smaller region.

A quick reference to this report is as follows:

- Section 2 examines the relation between the *triangulated surface from cross-sections* problem and some alternative reconstruction methods, so providing more motivation for the approach adopted here. Some closely related problems which share fertile common ground are pointed out.

- Section 3 defines and explores the problem. The emphasis is on different ways of viewing the problem, special aspects of the non-parallel case, the existence of solutions and how fast solutions may be obtained.

- Section 4 surveys previous solutions to the parallel and non-parallel planar problems.

- Section 5 provides an overall picture of the reconstruction algorithm and the theory behind it.

- Section 6 details the data-structures and algorithms used to implement the ideas of the previous section.

- Section 7 describes some approaches to noise problems with contours from intersecting planes.

- Section 8 demonstrates the performance of the algorithm on artificial data and real data from CT and ultrasound. The ultrasound models obtained are applied to estimation of organ volumes.

- Section 9 draws conclusions on the work presented in this report.

- Section 10 suggests some further work along the lines of this report.

2 Related Problems

2.1 Voxel Reconstruction

Surfaces may be obtained from 3D data in ways other than reconstruction from contours. In *voxel* approaches, it is assumed that data is available as a 3D grid (or cuberille), each cell of which is known as a voxel. The simplest such approach gives a surface as a set of faces of voxels [63]. Often the grid is more widely spaced in one direction than another, for example to ensure minimal radiation dosages in CT. The set of voxels lying on the boundary of the object of interest would be a rather blocky representation of that surface. Therefore an interpolation method, such as the Marching Cubes algorithm [32], is employed.

The main disadvantage of such methods relative to reconstructions from contours is the larger amount of data to be processed in reconstruction and the larger number of small tiles which are produced in the reconstructed surface, even when that surface is smooth. Furthermore, it has been shown that appropriate contour reconstruction methods give better estimates of surface normals [24]. It is harder to intervene in voxel approaches to improve erroneous connections or aliasing in their results. Finally, they lack an obvious extension to cases where data does not lie on a regular grid and resampling would greatly destroy resolution, or where boundaries are not well-described by iso-value surfaces for some image property.

2.2 Smooth Surfaces

Smooth surfaces can be obtained in two ways: as mappings from a pair of parameters to 3D space (parametric surfaces) or as the zeros of a scalar function with 3D argument (implicit surfaces).

B-splines are the most widely used parametric surfaces. Nonrational B-splines [44] and their rational counterparts [47] have recently been used for reconstruction from contours. Both of these works provide interesting references on the *smooth surface from cross-sections* problem. Thin-plate splines [9] allow minimisation of curvature, but require computationally expensive optimisation since they perform global interpolation. Fitting of splines requires a set of *location parameters* to be determined. This often requires an existing underlying surface which is normally generated by triangulation [34, 64] or an expensive optimisation procedure. It is difficult to guarantee that splines are smooth in 3D space (visual continuity) and that they do not self-intersect, although solutions to both these problems have been devised and the problems are quite rare in practice [64].

Implicit surfaces do not require the location parameter step but rather need an *a priori* parametrised model. This can limit their representational power, unless the model is itself selected on the basis of an approximate underlying surface. Such complex implicit surfaces are difficult to transform and it is hard to ensure their boundedness [36]. At the same time, simple implicit surfaces make it fairly easy to treat noise, missing data and boundedness. For this reason the simplest implicit surfaces, such as quadrics, quartics, and hyperquadrics [30] are frequently used.

Triangulations are also preferred to smooth surfaces here since they enable simpler, more well-developed and more rapid computations on the surface. Specifically, they facilitate the finding of lengths, volumes and areas. They also allow rapid rendering and hence visualisation, point location (especially finding closest points and checking whether points lie inside or outside the reconstructed surface), surface simplification and finite element computations. Furthermore, triangulations may be computed for parallel planes, by considering only a pair of sections at a time. More sections must be treated simultaneously to ensure curvature constraints are satisfied.

Finally, the use of other linear surfaces constructed from higher order polygons is possible [49]. Such surfaces approximate rather than interpolate the data because of their linearity requirement.

2.3 Related Topics

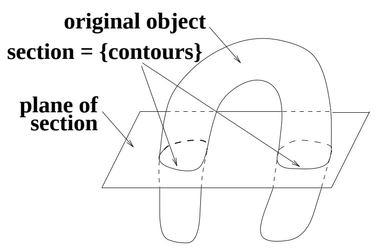
Several important but less constrained problems are closely related to reconstruction from contours. An excellent overview of reconstruction from sets of points alone using methods connected with those proposed in this report may be obtained from [17, 64]. In some situations, such as laser range imaging, the points or contours are occluded, so multiple views of the object must be combined [61]. The problem of reconstruction in stereo vision is harder still since only poor knowledge of point depth is available [21, 20].

A more constrained problem which is intimately related to the approach to reconstruction taken here is that of meshing a specified volume. This is usually performed for finite element analysis [65], computer graphics or computer aided design.

Finally, intermediate sections of reconstructions from parallel sections are interpolations of the shapes of the contours in those sections. Thus they may be used for *morphing* in computer graphics [51] and for the averaging of random shapes. Similarly, techniques from those fields may come to be useful in reconstruction.

3 The Problem

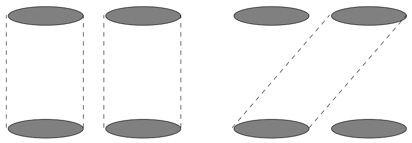
3.1 General Considerations



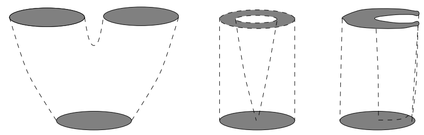
The *surface from cross-sections* problem is defined in terms of a set of *contours*. These are taken to be simple closed polygons representing the intersection of a surface with a plane of section. A *section* (or *slice*) is the set of contours which lie on a single plane. It is assumed that the contours in a section do not intersect each other. These concepts are illustrated by the adjacent figure.

The problem is to generate a surface which has a given set of sections as cross-sections. That is, cutting the reconstructed surface with the original planes gives the original contours. In the case of parallel planes, such a reconstruction may be formed by considering the space between one pair of adjacent planes at a time and creating the part of the surface lying entirely between those planes. For the non-parallel case, this may be generalised to the individual consideration of each of the gaps (chunks) between the planes. The first task with non-parallel planes is the generation of a suitable representation of this subdivision (see Section 5.2).

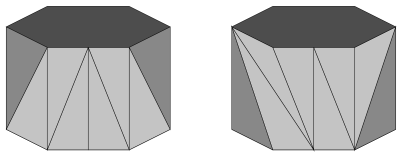
Reconstruction may be decomposed into several subproblems [34]:



- **Correspondence.** Which contours on one plane should be linked to which contours on which other plane?



- **Branching.** What should be done when sections to be connected have different numbers of contours, or when a contour has a concavity or hole that its neighbour does not? Some form of bifurcation is necessary.



- **Tiling.** Given a pair of contours to connect, which vertices and edges should be joined to make triangles? What is an appropriate method of capping off contours at the ends of branches?

It is not necessary to break the problem up this way to solve it, but it helps to illustrate the kinds of situations to be confronted. The method proposed in this report follows a different paradigm. This is known as *volume reconstruction* to distinguish it from the above *surface reconstruction* [24]. In this paradigm a mesh of tetrahedra is constructed between slices, with contour points as vertices. New vertices are then added to ensure that the contours are part of the mesh. Finally, some tetrahedra are eliminated to make the volume consistent with the contours. The reconstructed surface is then just the surface of the remaining volume. Thus, there are also three subproblems:

- **Meshing.** What tetrahedra should be selected?
- **Conforming.** Where should points be added so that all contour edges appear?
- **Sculpting.** What tetrahedra should be removed?

The non-parallel case has all the problems of the parallel case and some new ones. In particular it is possible for regions to tile onto themselves (if intersecting regions from intersecting planes are considered to be merged into a single region). Furthermore, it is important that contours on intersecting planes intersect accurately on the line of intersection of those planes: it is impossible to make reconstructions consistent with data which is inconsistent with itself.

Whether the surface or volume paradigm is chosen, there are many ways of solving each subproblem. The general principles which should guide any shape reconstruction mechanism have been examined in [66]. For many applications, a successful solution may only be judged by its consistency with human perception and knowledge. In other cases, performance is quantifiable, for example by comparing distances in shape between a reconstruction and a known true object, by comparing reconstructions from different sets of planes or by assessing the quality of the triangulation in terms of the accuracy and efficiency it allows for any later computations [24].

It has been pointed out in [66] that a reconstruction algorithm should be invariant of net rotations or scalings of its input. Three other natural criteria are that a reconstruction preserve convexity [67], that it be self-consistent and that it be fair. Preservation of convexity means that if the slices consist of convex regions, the resulting reconstruction should also be convex. Self-consistency means that reconstructions made from planar sections of an interpolated pair of slices be the same as the original reconstruction. Fairness means that reconstruction should treat regions inside and outside an object equally: a reconstruction should be the same if the subdivision of the input slices into regions inside and outside the object is replaced by its complement. These criteria are sometimes not possible to meet and when they are, they are hard to enforce when a reconstructed surface consists of triangles alone. Nonetheless, it might be hoped that the triangulation at least satisfy them approximately.

As always, it is desirable that the algorithm perform efficiently and have an underlying unity and simplicity. It is often forgotten that an algorithm should also be robust: many beautiful procedures of computational geometry although perfect from a mathematical viewpoint, are rendered unusable by precision problems unless stiff measures are taken [35]. In the present case, an incremental approach can aid human interaction: for example, if automatically generated solutions to the correspondence problem are checked and edited by eye. An incremental approach can also enable reconstruction as data is acquired: for example, in a 3D ultrasound scenario where a model is generated and visualised in real time and the person performing the scan decides to increase the sampling density in a particular region.

It is interesting to consider the generalisations of the problem to other dimensions. In 2D, the equivalent problem becomes the reconstruction of contours from line segments (see Section 5.1). It is possible to consider a fourth dimension as a time dimension and so have the reconstruction of deforming objects from their surfaces at different times or when their sections in a number of planes are viewed continuously.

3.2 Limits on Triangulation

Given a pair of contours on parallel planes, it might be imagined that a triangulated surface linking them with only contour vertices as triangle vertices could always be found[2]. A counterexample to this intuition may be found in [25] (consisting of a triangle and a spiral with 63 vertices). Fortunately, appropriate placement of at most one extra point on each contour is sufficient to guarantee the existence of such a surface [24] for any pair of single contours. A consequence of the latter constructive proof, not mentioned by its author, is that a polyhedral surface between an arbitrary pair of contours with n vertices in total can be obtained in $O(n)$ time[3]. This is interesting since it implies that the reconstruction problem is easier than triangulating a set of n points in a plane, for which $O(n \log n)$ is optimal [48]. In a weaker sense it is also easier than triangulating a simple n-gon since for this problem only formidably complex algorithms $O(n)$ are known [41].

[2]Strictly, it is required to find a simple closed polyhedron with the simple closed polygonal contours as planar sections.

[3]The surface consists of a pair of cones plus a patch over their common edge to force the result to be a polyhedron. The apices of the cones are the leftmost vertex of one contour and the rightmost vertex of the adjacent contour. The definition of left is arbitrary. The apices and extra patch take $O(n)$ to find and each cone takes $O(n)$ time to construct.

The author knows of no analogous results for n contours per slice. However, it should first be noticed that it is not possible to make a polyhedral surface simply by adding points on the planes of the contours: consider the case of one circle connecting to two circles - there must be a split which is not homeomorphic to a ball on one of the surfaces. If however, points may be added between planes, the n contours of a plane can be connected into a single loop, points added just below (or above, as appropriate) the plane on the lines linking contours and the above arguments applied to the resulting loop. However, the process of connecting n loops to form a connected planar graph takes $O(n \log n)$ time. For example, the contours may all be identical polygons centred along a line. The problem of finding non-intersecting connections between contours is then equivalent to sorting them along the line.

4 Previous Solutions

The need for automatic solutions to the *surface from cross-sections* problem was recognised years before any were found. In early electron microscope studies of nervous system morphology [31], stereo pairs of stacked contours were visualised and graph models of fibre bifurcations (solutions to the branching problem) were automatically generated. However, the first solution to the triangulation problem was not given until 1975 [29]. This started a series of investigations of optimal tiling for the case of one contour per slice. Soon after, ways of improving reconstruction speed by heuristic tiling [11] were introduced and generalisation to multiple contours began [11, 53]. Volume based approaches [5] and reconstruction from non-parallel planes [46] have only been studied in the last decade. Reviews of the *surface from cross-sections* problem may be found in [50, 34, 37].

4.1 Surface Reconstruction from Parallel Sections

Optimal Tiling. Tiling has been addressed by constrained optimisation of various costs. The earliest solution maximised the volume of the reconstruction for convex contour segments and minimised it for concave segments [29]. This was soon followed by a minimal surface area [22] solution. This has physical plausibility when modelling liquid boundaries which tend to minimise their surface tension. Other criteria have included minimal radii of circumscribed circles of triangles (a variant of Delaunay triangulation of the contours when projected onto a single plane – see Section 5.4.2) [34] and minimal sum of absolute value of angle between the contour edge parts of successive triangles [67]. When the cost is decomposable into a sum of terms from each triangle, the problem reduces to a search in a directed toroidal graph, for which divide and conquer techniques [13] can be applied to improve efficiency [22].

Various constraints have been applied to define acceptable paths through this graph. Principally, these are that each contour segment be present in at least one tile and that each tile contain at least one contour segment [29, 22]. Attempts to stop self-intersection have also been included. Prevention of crossing of neighbouring junctions between paired contour lines [29] and enforcement of the *angle consistency* criterion [67] are necessary but not sufficient for this purpose.

Although these methods were designed for the case of a single pair of contours, it is possible to generalise them by techniques outlined below.

Heuristic Tiling. Optimal tiling is slow despite attempts to make it faster [22, 56]. Therefore, heuristic approaches have been suggested, based on first transforming contours into more standardised shapes, then rapidly triangulating by local criteria (*local advancement rules*) and finally mapping back to the original contours. In [11] contours are first scaled so their bounding rectangles fit the unit square and translated so their centroids coincide. Triangulation commences by connecting the closest pair of contour vertices and proceeds by sequentially choosing the shorter of the two possible edges to extend the existing triangulation.

Other local advancing rules have been suggested. The local minimum edge length criterion of [11] has been extended to minimise the sum of squares of the successor triangle's edges in [2] and to connect all vertices of one contour to their nearest vertices on an adjacent contour in [19]. Meanwhile, [23] advances along each contour being tiled at as near as possible the same rate and [12] increments a triangulation to give the maximum degree of alignment between the directions of triangulated points from the centroids of their contours.

Such approaches strongly depend on the coherence (similitude) of adjacent contours. Other means of transformation to increase this coherence include the normalisation of contours to have unit perimeter [23] and the mapping of each concave portion of a contour onto the line joining the endpoints of that portion [19].

Instead of enforcing coherence, a more sophisticated approach is to search for similar contour portions. This is known as *matching* and may be performed by segmenting contours on the basis of extremes of approximate curvature [28]. More recently, a voting procedure for matching has been described [2]. This maximises the lengths of pairs of ordered subsequences from contours which match sufficiently. Sufficient matching is taken to mean that there is less than some user-defined threshold distance between the vertices of the subsequences.

Unfortunately, most of the above heuristics are neither rotationally nor scale invariant. All of these tiling techniques, including the optimal ones, will fail when confronted with contours which cannot be interpolated without the addition of extra vertices, like the example of [25]. The fact that most authors appeared not to suspect the existence of such shapes might be considered adequate evidence that they are sufficiently rare to be of no practical concern. However, it has been shown for heuristic tiling that non-intersecting transformed triangulations may become intersecting when the transform is reversed [42]. In fact, *all* of the above methods may lead to self-intersecting tilings even when the shapes to connect *are* tilable [67].

Branching. The earliest branching methods concatenated all contours in one layer into a single contour using new edges called *bridges* [11, 53]. This artifice has been rendered unnecessary by methods which reduce branching problems to a series of one-to-one cases by generating imaginary new slices between the input slices [19, 10]. The necessary addition of extra vertices within a contour to provide suitable splitting points for branching was first addressed by projection of the medial axis of one plane (see Section 5.4.3) onto the adjacent plane [53]. Methods of approximating this axis have been used since then [10].

Correspondence. Decisions about which regions to connect in multiple contour cases (the correspondence problem) have only been addressed recently, with the exception of [57], where a rule-based system using generalised cones was proposed. Previously it was assumed that the topology of the object is either obvious or needs to be determined by user interaction [26].

The simplest automatic method is to connect contours when the area of overlap of their bounding rectangles exceeds some threshold [19]. The dependence of this criterion on total contour area has been removed in [10] by suitable normalisation. An orientation insensitive approach is to use a minimum spanning tree [34]. First, elliptical approximations of contours are obtained, then connections are selected to minimise the sum of the squared distances between the ellipse centroids and difference between their principal axis lengths. Methods guaranteeing topological consistency have been developed which build trees describing the inside-outside relations of contours [54, 55] and combine them with minimum spanning tree heuristics [26].

4.2 Volume Reconstruction from Parallel Sections

As described in Section 3 volume reconstruction methods form a mesh over the region of interest and then sculpt away parts of the mesh to make it agree with the data. The sculpting concept existed prehistorically and has been applied computationally for years in the control of numerical milling machines [15]. However, volume reconstruction from contour data grew out of studies of reconstructing models from scattered point data.

Early work on reconstructing models from point data by O'Rourke [40] and Boissonnat [3] proposed the construction of the convex hull of the points. The first of these works then proceeded to remove and "flip" triangles to bring all the points onto the surface of the reconstruction while attempting to minimise surface area. In later papers, these authors started from the Delaunay triangulation (see Section 5.4) of the points: Boissonnat [4] sculpted to minimise the change in surface area at each step, while O'Rourke [43] minimised the length of the Voronoï skeleton (see Section 5.4.3) of the reconstructed object.

Eventually, Boissonnat applied the Delaunay triangulation to the reconstruction from parallel contours [5]. A clever and efficient procedure for combining the Delaunay triangulation of each section into triangulations between planes was proposed and sculpting was designed to retain the largest volume of the triangulation consistent with the contours. This approach simultaneously handles the correspondence, branching and tiling problems. However, poor results were obtained when sections on neighbouring planes were dissimilar.

These branching problems were corrected by Geiger [24], who stressed the importance of the relation between the nearest neighbour surface (see Section 5.4.1) and the Delaunay reconstruction, and thus showed how the addition of extra points to the triangulation could improve its quality. The extra points were essentially projections of the Voronoï skeleton of the contours, as had been proposed in [53]. However, Geiger indicated how these points could be efficiently obtained from the Delaunay triangulations of individual sections. Other authors have since implemented the same ideas, for example [60].

4.3 Surface Reconstruction from Non-Parallel Sections

Several authors on reconstruction from parallel cross-sections have suggested that their methods might be extended to work in the non-parallel case [23, 24, 39], but apparently, only the work of Payne and Toga [46] makes such a generalisation.

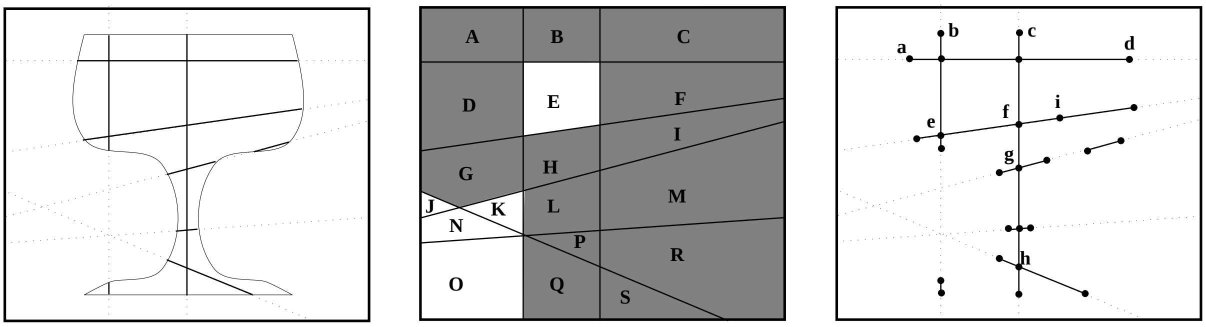
Payne and Toga first construct a representation of the way the space containing the reconstructed object is divided up into chunks by the planes of section. This is a structure known as an *arrangement* (see Section 5.2). Then the intersections of contours on each plane with other planes are found, and the contours are partitioned into fragments which begin and end at such points. It is assumed that the fragments on different planes may be joined into loops around each of the chunks of space between planes: no branching or correspondence problems are allowed. Finally, each of these loops is triangulated by a minimum area method.

These authors recognise the limitations of the assumption of no branching and point out that triangulation can produce poor results for loops which vary in several directions at once. They propose to "thin" loops by removing data points on fragments which do not lie in the "principal direction" of a triangulated loop to improve the visual appearance of the reconstruction when the latter problem occurs. However, the concepts in the latter suggestion are not explained and the idea seems to be working against the goal, which is to produce surfaces which accurately represent the data!

5 The Idea

5.1 Overview and Example

Here, what is to be achieved in the next sections is illustrated with a simple 2D example of reconstruction. In 2D the input data for reconstruction consists of arbitrarily oriented linear cross-sections of a 2D object which are piecewise either in or out of the object. Figure 1(a) shows a grail object with sections shown as solid line segments when in and dotted when out.



(a) The input data and the true boundary. **(b) The arrangement of lines with chunks to triangulate (shaded).** **(c) The set of points to triangulate superimposed on the data.**

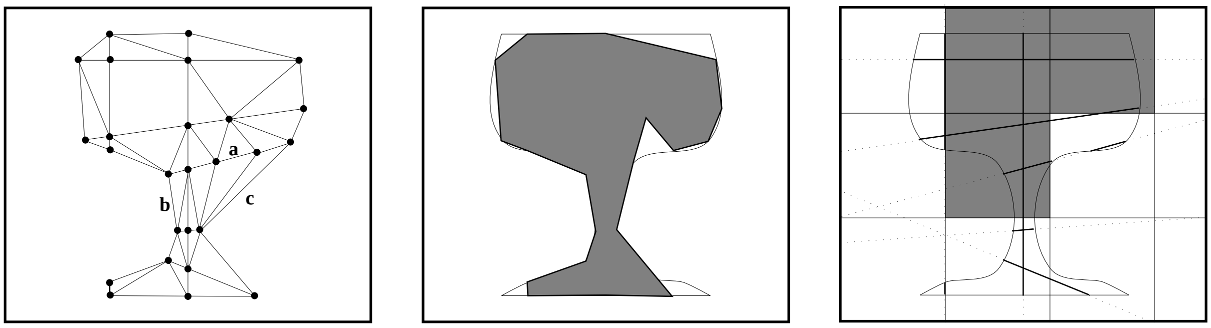
FIG. 1: 2D Example of Delaunay reconstruction from multi-axial contours.

The first stage forms a representation of the way the space is divided up by the lines. The result is a set of chunks which form the natural units in which triangulation may be performed. These provide the analogue of adjacency of planes in parallel planar reconstruction. In Figure 1(b) the chunks are labelled A, B, ..., S. Notice that only the 19 chunks within a bounding box are shown out of a total of 29. The idea described in Section 5.2 uses a bounding box to save on the computation of unnecessary chunks.

Then chunks are labelled according to whether or not they require triangulation and whether any of the points where lines intersect (the vertices of the chunks) should be included in a triangulation. The results of labelling are shown in Figure 1(b) in which the shaded parts need triangulation and the white parts do not. Partitioning and labelling are described in Section 5.3. The decision as to when to triangulate clearly depends on the topological relation of a chunk to the object. A chunk with no part of its boundary in the object clearly does not require triangulation, as is the case for chunks J,K and N. However, "requires triangulation" is not equivalent to a "having some boundary in the object." For example chunk E lies entirely in the object, so a sensible reconstruction is just the entirety of E. More subtly, chunk O only has one of its sides in the object, which leaves nothing to triangulate to.

Next the vertices which will be involved in triangulation are gathered together. These include the endpoints of the line segments which lie in the object: for example, vertices a,b,c,d in Figure 1(c). Also, there are the points of intersection of the lines which were decided on in the labelling stage: for example, vertices e,f,g,h. After a first phase of triangulation, some extra vertices are added to improve the quality of the triangulation, mainly in branching situations (*splitting vertices*): for example vertex i. Finally, in order for all the required edges to be present, yet more vertices may need to be added. This is achieved automatically in a stage known as making the triangulation conform. No such vertices are required by the present example.

Triangulation is performed chunk by chunk resulting in a mesh like that of Figure 1(d). The mesh uses up all the vertices and edges required, but may violate the input data in the sense that regions known to be outside the object get filled: for example edge a. The set of violating edges does not include edges like b and c. Although they lie out of the true object, the input data does not impose this constraint and could equally well have come from an object for which b and c were in.



(d) The triangulation. **(e) The reconstruction (shaded) and the true boundary.** **(f) The result of voxel reconstruction (shaded).**

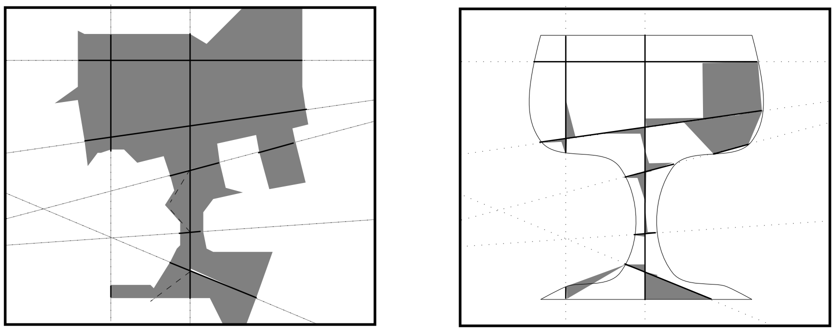
FIG. 1: 2D Example of Delaunay reconstruction from multi-axial contours.

Triangles containing violating edges are removed in a step known as *sculpting*. Once sculpting has removed these edges, it may leave an object with some bizarrely connected portions which are only joined at a vertex (or along an edge in 3D): for example the triangle containing c which remains if both triangles with edge a are removed in Figure 1(d). Such triangles are also removed. Triangulation and sculpting are described in Section 5.4. Sculpting produces a reconstructed object like Figure 1(e). This reconstruction has noticeable differences from the actual object, which is not surprising since only seven lines have been used in total.

It is worth comparing the results with those from a voxel reconstruction in which voxels are made as small as possible subject to the constraint that every voxel should contain some data. In the present example this gives a total of twelve voxels. Voxels are labelled in or out of the object according as more or less of the data they contain is in or out. Such a reconstruction is shown in Figure 1(f).

A truer analogue of the voxel method used in parallel plane reconstruction would be to extend voxels normal to the input lines, giving a result like Figure 1(g). This uses an arrangement to label a point in if its closest data point is in and out if its closest data point is out. It produces a complex, jagged boundary and even goes outside the bounding region. This is because it is a form of piecewise constant interpolation.

A piecewise linear interpolation should be preferable. Geiger [24] views Delaunay reconstruction in the parallel plane case as producing an approximation to the *nearest neighbour surface*. This object is formed by connecting each point in each region of the input data lying in the object (interior region) to its closest points in interior regions on neighbouring planes. Applying the same idea to non-parallel planes, the nearest neighbour surface for the present example is shown in Figure 1(h).



(g) An alternative voxel method: the block nearest-neighbour surface.

(h) The nearest neighbour reconstruction.

FIG. 1: 2D Example of Delaunay reconstruction from multi-axial contours.

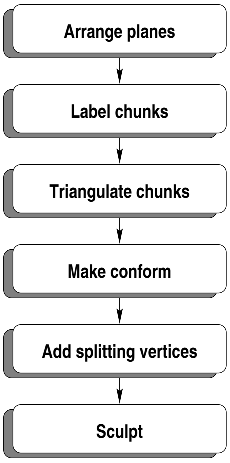


Figure 1(h) shows that in the non-parallel case the nearest neighbour surface is even less desirable than the block nearest neighbour surface! In particular, regions between lines at angles greater than $\frac{\pi}{2}$ do not even get connected: the closest point on a line is the perpendicular projection onto it.

The triangulation method described here is a non-parallel generalisation of Delaunay reconstruction, but it does not treat Delaunay reconstruction as an approximation of the nearest neighbour surface. It may be considered as a form of interpolated voxel reconstruction but instead of considering every voxel in each region, it only considers the region boundaries, making it more efficient.

Finally, the overall process for reconstruction is summarised by the adjacent flowchart.

5.2 Arrangements

This section introduces the subdivision of a space by a set of planes and a means of computing it. Such subdivisions are known as *arrangements*. For all the parallel plane methods of Section 4, triangulation proceeds by considering one infinite gap between a pair of adjacent planes at a time. For arbitrary planes, adjacency of planes is a function of position in space. Here, the arrangement of a set of planes of section provides the natural units in which triangulation may be performed. Arrangements are fundamental structures of computational geometry. For a full introduction to their properties, consult [41] then [16].

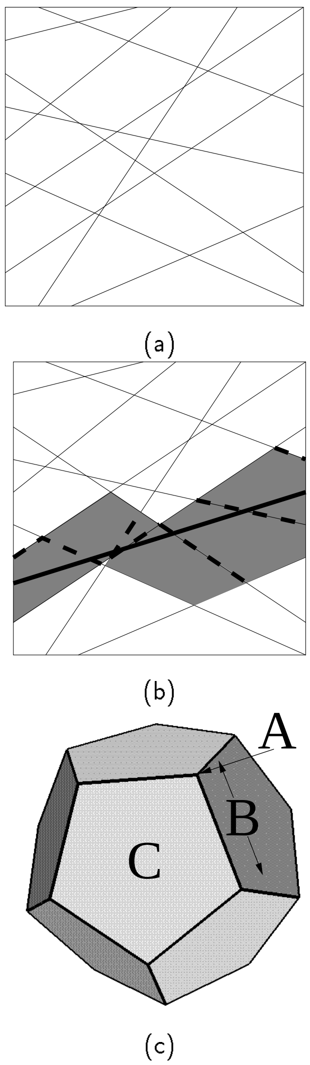


Figure 2: Examples of arrangements.

In 2D the equivalent of a plane is a line. A 2D arrangement is shown in Figure 2. It has three types of region: the points where the lines cross, the line segments between such points and the polygonal regions which the line segments surround. These regions are called the *faces* of the arrangement. Faces of dimension 0 (points) are called *0-faces*, faces of dimension 1 (line segments) are called *1-faces* and so forth.

When a new line is added to an arrangement, like that shown in bold in Figure 2(b), the set of faces which it cuts are called the *zone* of the line. Here the 2-faces of the zone are shaded and the 1-faces are bold dashed.

In 3D, an arrangement has polyhedral regions between planes called 3-faces, in addition to 0-faces, 1-faces and 2-faces. Alternative names for k-faces are as follows:

0-face	$\leftrightarrow$	point or vertex
1-face	$\leftrightarrow$	line segment or edge
2-face	$\leftrightarrow$	convex polygonal region
3-face	$\leftrightarrow$	convex polyhedral cell.

The term face is adopted as it avoids verbosity and confusion with the use of the terms vertex, edge and polygon elsewhere, while capturing the underlying symmetry of this situation.

Frequently in the following sections, it will be useful to talk about the faces lying in the boundary of another face, which are of one dimension lower. Such faces are called the *subfaces* of the higher-dimensional face. In the adjacent figure of a 3-face, the 0-face A is a subface of 1-face B which is itself a subface of 2-face C. Similarly, the faces which have a given face in their boundaries are called the *superfaces* of the lower-dimensional face. For example, the 3-face is a superface of C.

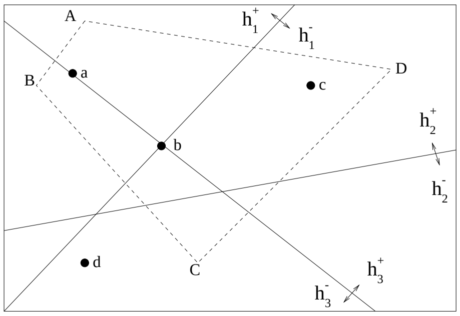


Figure 3: A sub-arrangement of three planes h_1, h_2, h_3 with bounding region $ABCD$. a, b, c are points within the bounding region and have position vectors:

$$\begin{aligned}\nu(a) &= (1,1,0)\\ \nu(b) &= (0,1,0)\\ \nu(c) &= (-1,1,1).\end{aligned}$$

d lies outside the bounding region and has position vector

$$\nu(d) = (-1,-1,-1).$$

To formalise[4] and generalise these notions to a Euclidean space of dimension d, define the *position vector* of a point p with respect to a set of hyperplanes $H = \{h_1, h_2, \ldots, h_n\}$ as the n-component vector with i^{th} component

$$\nu_i(p) = \begin{cases} +1 & \text{if } p \in h_i^+ \\ 0 & \text{if } p \in h_i \\ -1 & \text{if } p \in h_i^- \end{cases} \tag{0.1}$$

where h^+, h^- denote the half-spaces above and below h. (Any fixed direction which is not parallel to any hyperplane will do as a definition of *above*.) Then, a *face* is the set of all points with the same position vector. A face is a k-face if it has dimension k. Notice that all faces are open convex sets.

A face f is a subface of a face g if the position vector of f differs from that of g by the replacement of just one non-zero component by zero. g is a superface of f iff f is a subface of g.

The *arrangement* of H is the set of all $0\ldots d$-faces formed by the hyperplanes in H and a *sub-arrangement* of H is the subset of the arrangement consisting only of faces within some convex region called the *bounding region*. An example of a 2D sub-arrangement is shown in Figure 3.

An arrangement may be optimally constructed incrementally. If A_k is the arrangement of k planes $h_1, \ldots, h_k$, then A_{k+1} is formed by first finding the zone (recall Figure 2(b)) of h_{k+1} in A_k then replacing each face f in the zone by three faces: the part of f above h_{k+1}, the part below h_{k+1} and the intersection of f with h_{k+1}. Since the zone at stage k has $O(k^2)$ faces[5], an arrangement of n planes has $O(n^3)$ faces. For the reconstruction method developed here, it is likely that only a small fraction of these faces fall within the region where triangulation will be performed. Therefore it is useful to construct only the part of an arrangement lying within a volume bounding all the contour data. This also enables a simpler implementation of the construction algorithm, since infinite faces need not be represented. The lack of infinite 1-faces also simplifies the cutting up of contours and labelling of regions described in the next section.

[4] In particular, it is important to state what "in the boundary of" actually means: subfaces are *not* contained in their superfaces.

[5] The proof in [16] is incorrect. A correct version may be found in [18].

5.3 Partitioning and Labelling

Once the space has been partitioned into a sub-arrangement, the contours are split into fragments each of which is contained in a single 2-face. The goal is to be able to rapidly access all parts of a contour on the surface of any given 3-face. This enables the recognition of which faces lie in or out of the object and the formation of loops over any 3-face involving fragments of contours from different planes for triangulation. Also, since triangulation is performed one 3-face at a time, it is necessary to include the points of intersection of contours with 1-faces as triangulation vertices. An example of the partition of a contour is shown in Figure 4.

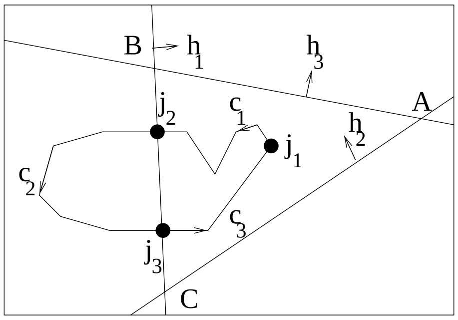


Figure 4: A typical partition of a contour into chains c_1, c_2, c_3, and junctions j_1, j_2, j_3 by intersecting planes h_1, h_2, h_3. j_1 lies in ABC, a 2-face, while j_2 and j_3 lie on BC, a 1-face.

Contour fragments are known as *chains* and the points where they join together are called *junctions*. For consistency, every contour is taken to begin and end at a junction. Thus, junctions may occur both in the middle of 2-faces and on the intersections of contours with 1-faces.

It is now shown that some 3-faces do not need triangulation and that some 0-faces need inclusion in a triangulation in order to provide consistency with the input data. To recognise these cases it is necessary to determine the topological relation of each face in the sub-arrangement to the object to be reconstructed. This process is called *labelling*.

In the parallel case, labelling is not usually considered as a stage in reconstruction at all: since the reconstructed objects are connected, they must be connected from each infinite parallel 2-face to its adjacent 2-face. Also, since parallel planes do not intersect, there are no 0-faces to include in a triangulation.

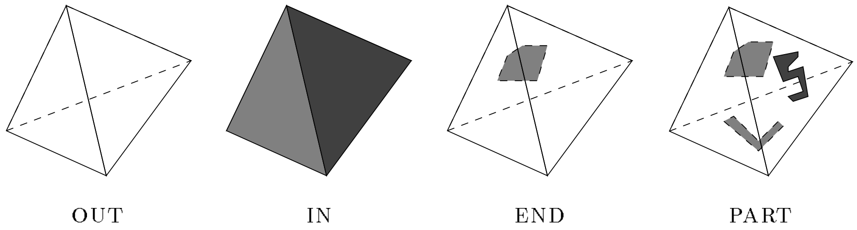
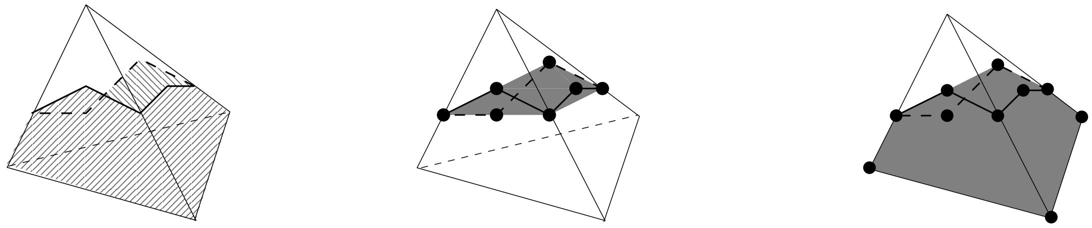


Figure 5: The four labels for a 3-face. Regions inside the object are shaded.

Firstly, consider the situation for 3-faces as in Figure 5. They may lie entirely *out* of or *in* the reconstructed object. There is no point trying to triangulate such faces. Alternatively, they may have just one subface (boundary face) partly in the object, and hence lie at the *end* of some branch of the object. In such cases, the contours have nothing to be triangulated to. Finally, more than one such subface, but only *part* of the total boundary may be in the object. Correspondingly, the labels OUT, IN, END and PART are assigned to these cases.

Figure 6: Example of why it is necessary to include 0-faces as triangulation vertices: a tetrahedral 3-face formed by 4 intersecting planes and their contours (in bold/dashed when occluded by a 2-face).



(a) The area labelled IN by the contours is hatched.

(b) The largest possible region covered by a triangulation when only loop vertices (black points) are triangulated is shown in grey.

(c) The region covered by the triangulation when IN 0-faces are also triangulated is shown in grey.

Secondly, consider any triangulation of the set of contour vertices on some 3-face, like those in Figure 6. The biggest possible triangulation of any set of points is their convex hull. However, the convex hull of the contour vertices alone is not large enough to agree with the information provided by the contours. If the point set is expanded to include any 0-faces which lay *in* the object then this problem is solved. The labels OUT and IN are assigned to 0-faces to distinguish these cases.

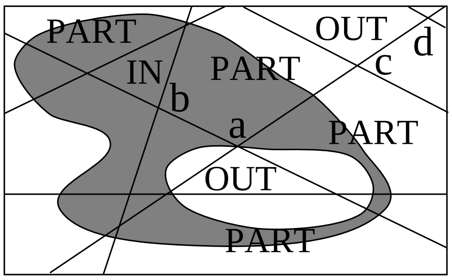


Figure 7: The labelling of $(k < 3)$-faces. The lines are the intersections of planes with a plane coplanar with the page. The shaded region is the inside of the object. The labels of some of the 2-faces are indicated. 0-faces a and b are IN, while c and d are OUT.

Analogous symbols are given to the other faces in order to derive the appropriate labels for 3-faces (see Figure 7). Formally, each set of contours for a plane partitions that plane into a set of points which are inside the object, S, and its complement which lies outside the object, $\bar{S}$. Correspondingly, a $(k < 3)$-face f is labelled with one of the following values:

$$\begin{aligned} \text{IN} &\Leftrightarrow S \cap f = f \\ \text{OUT} &\Leftrightarrow \bar{S} \cap f = f \\ \text{PART} &\Leftrightarrow S \cap f \subset f, \end{aligned}$$

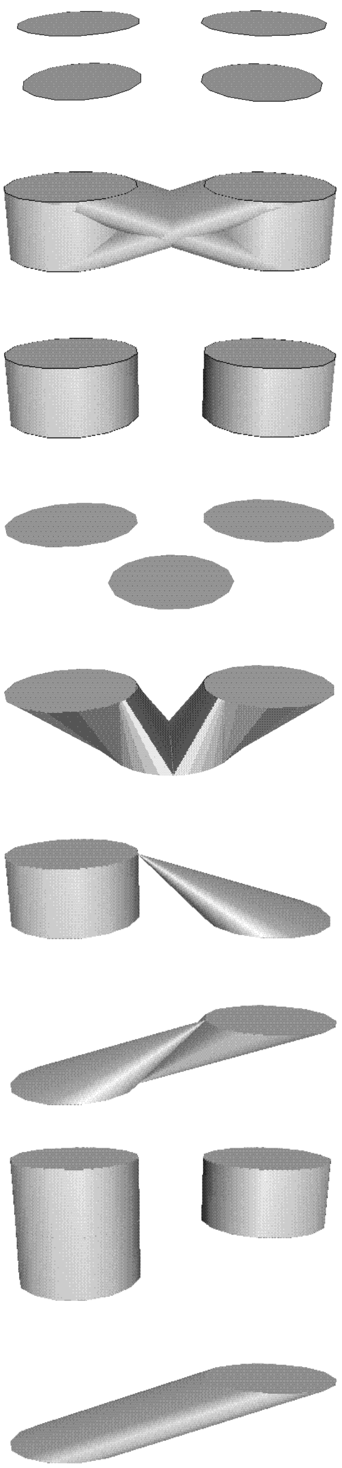
while 3-faces are classified as

IN	$\Leftrightarrow$	all subfaces of f are IN
OUT	$\Leftrightarrow$	all subfaces of f are OUT
END	$\Leftrightarrow$	all but one subface is OUT
PART	$\Leftrightarrow$	at least two subfaces are not OUT but f is not IN.

5.4 Delaunay Reconstruction

Now the Delaunay reconstruction method is introduced. The following presentation starts from the *Delaunay surface* which can solve the reconstruction from contours problem in a single 3-face in one step. This does not have a triangulated surface, so it is shown how it may be approximated using *Delaunay triangulations* and *Voronoï diagrams*.

5.4.1 Nearest Neighbour Surfaces and Beyond...



(a) (b) (c) (d) (e) (f) (g) (h) (i)

Consider the *surface from cross-sections* problem for the sets of planar regions (whose interiors are shown in grey) in Figure (a). Perhaps the simplest way to generate a solid object with these sections is to join all points in the lower regions with all points in the upper regions as in Figure (b). Although it solves the problem and resembles a well-known object (a pair of binoculars), this maximum volume solid is not the most intuitive solution since regions which are distant from each other have been connected.

Alternatively, all points in the lower regions could be joined to their nearest points in the upper regions and *vice versa*, as in Figure (c). This corresponds to the heuristic that interpolation is most accurate when based on function values from locations as near to a point as possible.

This is known as the *nearest neighbour solid* and its surface is the *nearest neighbour surface*. It solves the correspondence, branching and tiling problems of Section 3 in one simple step. When solving the branching problem (Figure (d) reconstructs to Figure (e)), one or more splits are introduced in one region. These are known as *splitting points*.

Sometimes the nearest neighbour solid joins rather distant regions, since it assumes that everything must be connected to something. In these cases, the regions are not really solidly linked at all: they are only joined at a point (Figure (f)) or along a line (Figure (g)). Such links are known as *non-solid connections* and solids containing them as *non-manifold solids*.

Often, non-solid connections occur when a branch of an object is encountered for the first time in a sequence of slices, and better results are obtained by removing the non-solidly connected parts, as in Figure (h), where a third slice has been included in the input to Figure (f) (which has also been rotated). An algorithm which works with only one 3-face at a time cannot tell the beginning of a new branch from the alternative very slanted case (Figure (i) could be the correct reconstruction from the input to Figure (g)). The nearest neighbour surface with any non-solid connections removed shall be called the *solidly connected nearest neighbour surface* in the following text.

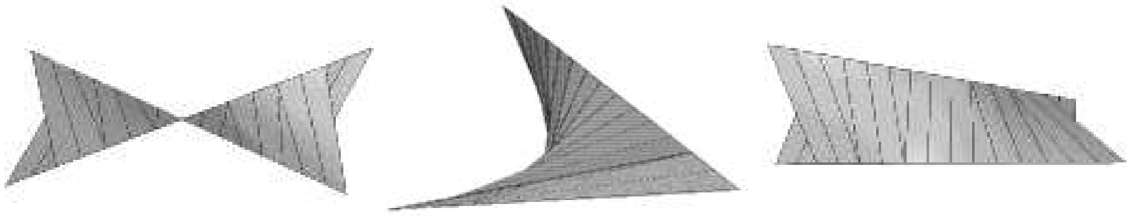
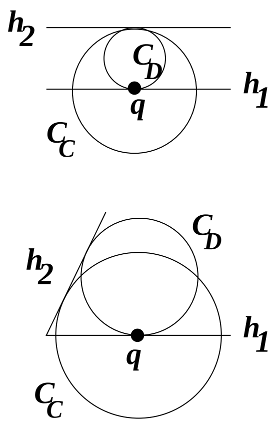


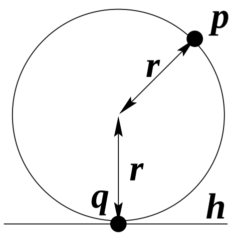
FIG. 8: Three views of the nearest neighbour connection between two lines. The surface consists of two hyperbolic paraboloidal sheets [24].

The Delaunay reconstruction method of [24] generates an approximation of the solidly-connected nearest neighbour surface. Approximation is necessary as this surface is generally curved, even when the regions it connects are straight as shown in Figure 8. It might be hoped that a surface constructed by taking linear combinations of convex regions also be convex. However, this figure also illustrates that the nearest neighbour surface does not preserve convexity.

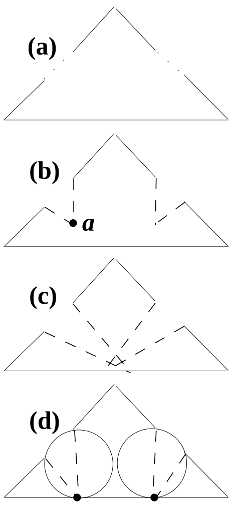


It was shown in Section 5.1 that the nearest neighbour surface has a pathological behaviour for non-parallel planes. Nevertheless, a close relative has good performance even for this case. Consider the adjacent figures of pairs of planes in 2D. The point p in the upper plane h_2 closest to a given point q in the lower plane h_1 may be found by expanding a circle C_C with centre q until it hits h_2. Alternatively, expansion of a circle C_D *tangent* to h_1 at q until it hits h_2 produces the same results for parallel planes, but different results for non-parallel planes. The surface generated by connecting every point in the regions of each plane to the interior points in any other planes which its C_D circle first hits is a simple version of what is known henceforth as the *Delaunay surface*. The same definition may be extended to other dimensions, by using tangent balls instead of tangent circles.

Now the full Delaunay surface is described. It is helpful to imagine that the interior regions on each plane have some small thickness. Then, instead of allowing the balls C_D to be just tangents to the planes, allow them to be tangent to (and not containing any) interior regions: that is, C_D may rest on the side of an interior region and thus penetrate an exterior region of a plane. This full Delaunay surface clearly contains the surfaces described above. Furthermore, it allows the preservation of convexity. This definition is the one used for the rest of this report.



An important advantage of the Delaunay surface for non-parallel triangulation is that when planes are strongly angled, the connections between them are much fatter than with the nearest-neighbour surface. This may be understood by viewing the Delaunay surface as a nearest neighbour surface where the distance between a point q lying in a plane h and a point p is measured along *dog-leg* paths rather than straight lines. A dog-leg path first goes some distance r along the normal to the plane from q then the same distance r on to p, as shown in the adjacent figure.



Another advantage is that when splitting occurs, it always leads to splits which end on the planes rather than between planes. The adjacent Figure (a) shows a set of input planes drawn solid when in and dotted when out. Figure (b) shows the nearest neighbour reconstruction, which requires splitting points (like point a) in between planes. The lines joined to splitting points are shown dashed. It might be imagined that the Delaunay surface leads to splits carved out of the reconstructed volume which intersect each other as in Figure (c). This is not the case here as the Delaunay surface in Figure (d) shows (the two circles for the splitting points are also indicated). In fact, intersecting splits can never happen in a Delaunay surface. Furthermore, unless the boundaries of a split are cospherical with more than one point, which is a rather special case, a single split only generates splitting points on one 2-face[6].

5.4.2 Conforming Delaunay Triangulation

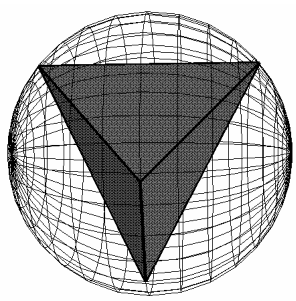
As has just been seen, the nearest neighbour surface is generally curved and is identical to the simple version of the Delaunay surface when reconstructing from parallel planes. Thus, the Delaunay surface should be expected to be curved in general, so it is desirable to find a means of approximating it by triangles. A number of criteria should be satisfied by the approximation:

1. There should be enough triangles to use up all the polygon vertices.
2. There should not be so many triangles that they intersect one another.
3. There should be enough triangles to use up all the polygon edges.
4. Triangles should not fill regions which the contours indicate should not be filled.
5. The surface should have a similar shape to the Delaunay surface.

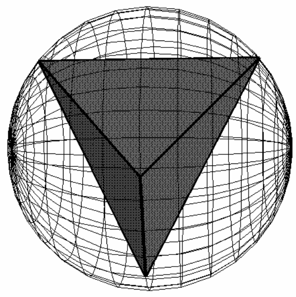
The *Delaunay triangulation* provides a solution which can satisfy requirements 1 and 2[7]. In the following, first the triangulation is defined, then its failure to satisfy requirement 3 is rectified.

[6]This may be seen as the limit of an analogous property of the Delaunay triangulation which is described next.
[7]For a proof that it satisfies requirement 2, see [41].

Requirement 4 must wait until Section 5.4.4. The fulfillment of requirement 5 should be evident at the completion of this section, apart from the non-existence of the splitting points. This shall be rectified in the next section.



In two dimensions, the Delaunay triangulation joins any three vertices to form a triangle when the circle through them contains no other vertices. Such triangles are called *Delaunay triangles*. This is shown in the adjacent figure. Here $\triangle abc$ is a Delaunay triangle as circle abc is empty. $\triangle abd$ is not a Delaunay triangle as circle abd contains c. The circle around a triangle is known as the *circumcircle* and its centre is the *circumcentre*. The circumcircle *circumscribes* the triangle. The vertices of a Delaunay triangle are close because their circumcentre is closer to them than to any other vertex.



In three dimensions, the Delaunay triangulation joins four vertices to form a tetrahedron when the sphere through them (their circumsphere) contains no other vertices[8]. This is shown in the adjacent figure. Notice that the section of the circumsphere of a tetrahedron in the plane of one of its surface triangles (facets) is the circumcircle of that triangle. In fact, the three-dimensional Delaunay triangulation could have been defined just as the set of triangles which have an empty sphere going through them, or even just as the set of pairs of vertices which have an empty sphere through them.

To generalise further and formalise, it is helpful to define the Delaunay triangulation as a *graph*:

Definition 1 *The Delaunay triangulation of V, a finite set of points from $\mathbb{R}^d$, is the graph $(V, \mathcal{E})$, where $\mathcal{E}$ is the set of all pairs of vertices v_1, v_2 from V such that there is some point p which is closer to them than to any other point v_3 in V. That is,*

$$|v_3 - p| \geq |v_1 - p| = |v_2 - p| \text{ for all } v_3 \in V.$$

In d dimensions the locus of points less than a given distance from a point is known as a *ball*. Usually $d+1$ points is sufficient to define a unique ball in d dimensions. $d+1$ points and all the linear combinations of them which lie between them (their *convex hull*) are known as a *simplex* [9]. So, in d dimensions, the generalisation of the notion of a Delaunay triangle circumscribed by a circumcircle is that of a simplex circumscribed by a circumball.

Every vertex has at least one closest vertex. So, a sphere can be drawn through a vertex and made larger until it hits the closest vertex. This sphere can then be moved around until

[8] Some people call this the Delaunay tetrahedronisation.

[9] Strictly speaking,the convex hull of a set of $k+1$ points T for $k \leq d$ in dimension d is known as a *k-simplex*, denoted by s_T. A collection of simplices S is known as a *simplicial complex* or *triangulation* if

1. $s_T, s_U \in S \;\Rightarrow\; s_T \cap s_U = s_{T \cap U}$,
2. every face of every s_T in S is also in S.

(a) (b) (c)

FIG. 9: Conforming Delaunay triangulation.

it hits another vertex. These three vertices then define a Delaunay triangle since they have an empty sphere through them. So, requirement 1, that there should be enough triangles to use up all vertices is satisfied by the Delaunay triangulation.

The Delaunay triangulation *does not* guarantee that every edge of a polygon be present in it. For a given set of vertices, there is usually only one Delaunay triangulation. However, there are usually many polygons which may be drawn through a given set of points. Two polygons through the same set of points are shown in Figure 9(a) and (b).

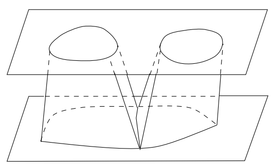
Fortunately, judicious addition of extra vertices to a polygon is enough to ensure that it will be part of a Delaunay triangulation. A Delaunay triangulation with enough extra points that all the required edges are present (possibly as unions of other edges) is known as a *conforming Delaunay triangulation*. An example of a conforming Delaunay triangulation is shown in Figure 9(c), where two additional points have been placed in order to make the triangulation conform to Figure 9(a), the edges of which are shown in bold.

To see that this is always possible, let every edge of a polygon have points added to it at separations less than twice the separation between any two non-concident polygon edges (that is to say, edges which do not share any vertices as endpoints). Then the circle centred at the midpoint of any pair of consecutive vertices is clearly empty of other vertices and so present in the triangulation. This result also shows that when polygon edges are obtained from some pixelised image, and hence the distance between non-coincident edges is less than the interpixel spacing, no more points need be added than there are pixels in the image representation of the boundary. Hence conforming triangulation is always at least as efficient as voxel methods.

Optimal algorithms have been found which add the minimum number of points necessary on the edges of any set of polygons to give every resulting edge a circle empty of other edges in [38]. For this problem it is known that the number of additional points necessary can be made arbitrarily large depending only on the *shape* of the input polygons. In particular, more points need adding as the polygons become thinner and more elongated.

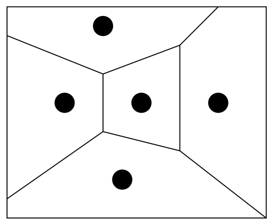
Alternatively, one may aim to find the minimum number of extra points which may be placed *anywhere* to make a Delaunay triangulation conform. For a planar graph with m edges and n points $\Omega(mn)$ points are necessary but the best algorithm to date adds $O(m^2n)$ points [59].

5.4.3 Voronoï Diagrams and Delaunay Splits



It has just been indicated that new points must be added in order that the Delaunay triangulation contain the edges of a polygon to be interpolated. Yet more points need to be included to approximate the Delaunay surface. In branching configurations, as in the adjacent figure, and for some rather bent regions, this surface splits *inside* a region, so triangles will need vertices near these splitting points.

In the parallel planar case, the location of the splitting points is easily quantified in terms of the *Voronoï diagram*. When the Voronoï diagram has been defined, it shall be shown to have such a close relationship with the Delaunay triangulation, that only the Delaunay triangulation need be computed to find appropriate splitting points. Then the location of splitting points shall be generalised to the non-parallel case.

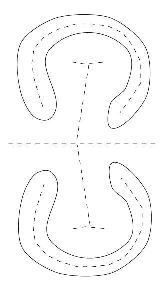


The set of all points closer to a given point in a point set than to all other points in the set is known as the *Voronoï cell* for the point (see adjacent Figure). The union of all Voronoï cells for each of a set of points is called its *Voronoï diagram*. The vertices of Voronoï cells are called *Voronoï vertices* and their edges *Voronoï edges*. Formally, the Voronoï cell of a point q drawn from a set of points S in $\mathbb{R}^d$ can be written as the set

$$\left\{ p \in \mathbb{R}^d \;\middle|\; |q - p| < |q' - p| \text{ for all } q' \in S - \{q\} \right\}. \tag{0.2}$$

Voronoï cell boundaries are segments of the bisectors of pairs of points. In fact, Voronoï cells in $\mathbb{R}^d$ are just unions of d-faces of the arrangment of these bisectors [16].

An equivalent way of defining the Voronoï diagram is as the set of points whose nearest site is not unique. Points in this set are the boundaries of the Voronoï cells and are known as the *Voronoï skeleton*[10]. One use of this diagram is to find the closest point to a given point: all that is necessary is to find which Voronoï cell the point lies in. So, one relationship to the nearest-neighbour surface is already apparent.



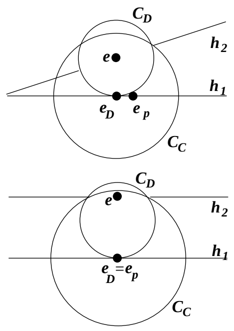
A generalisation immediately points out the splitting points. The set of all points whose nearest site within any *region* in a set of regions is not unique is called a *generalised Voronoï diagram*. A generalised Voronoï diagram is shown in the adjacent Figure. The parts of this diagram lying outside the regions are known as the *exterior Voronoï skeleton* and those inside are known as the *interior Voronoï skeleton*[11]. When interpolating regions on two planes by the nearest-neighbour surface, it is precisely the perpendicular projection of the exterior Voronoï skeleton of regions on one plane onto the adjacent plane which constitutes the splitting points.

[10]The Voronoï diagram is also known as the Dirichlet tessellation, the Thiessen polygons, the Wigner-Seitz zones, the unweighted power-diagram or Blum's medial axis transform.

[11]Further generalisations are of great importance in other contexts, for example, the consideration of metrics

A remarkable fact is that the Delaunay triangulation can be defined as the *dual* of the Voronoï diagram: if each vertex of a point set is connected to every vertex whose Voronoï cell it shares a Voronoï edge with, then the Delaunay triangulation is obtained. The duality between Voronoï diagrams and Delaunay triangulations is illustrated in Figure 10.

This definition of the Delaunay triangulation and the definitions given above are equivalent, since a common boundary point p is equidistant from a pair of vertices v_0, v_1, and nearer to them than to any other vertex, by definition of the Voronoï diagram. Thus a ball centred on p and touching v_0, v_1 is empty of other vertices and hence a Delaunay ball. Similarly, the centre of any Delaunay ball must be equidistant from its defining vertices and thus be a Voronoï vertex. This relationship shows how once a Delaunay triangulation has been computed, points approximating the splitting points may be obtained directly: these points are just the projections of the centres of the intersections of the circumballs of Delaunay tetrahedra which have facets outside the regions.



In the non-parallel case, the adjacent figure shows that direct projection (e_p) of the external Voronoï skeleton (e) no longer provides appropriate splitting points. Instead, splitting occurs wherever a Delaunay ball (C_D) for a tetrahedron with a facet lying outside the contours touches another plane in its object interior regions (e_D). Note that this implies that the position of splitting points must vary not only with the centre of the split but also with its width. This must be true for a satisfactory method in any case, else the triangulation cannot be scale invariant.

5.4.4 Sculpted Triangulations

The outer boundary of the Delaunay triangulation of a set of points is the convex hull of that set. This boundary does not generally meet requirement 4 of page 20 that the triangulation not violate the contour data. The operation of removing tetrahedra/triangles from a triangulation to make the boundary of the remaining set of simplices satisfy some property is known as *sculpting*. This section sets up some terminology for discussing sculpting, says exactly which tetrahedra need removing and shows when this can be done. It closes with some thoughts on tetrahedra which need neither be kept nor removed.

A set of contours divides a plane into *in-regions* which are inside the object and *out-regions* which are outside the object. When a set of tetrahedra does not violate such a subdivision it is said to be *acceptable*. The edges and facets of a tetrahedron in a conforming triangulation cannot cut the boundaries of regions by the definition of a conforming triangulation. That is, the facets which lie in the input planes may only be in in-regions or out-regions. They are therefore classed as *in-facets* or *out-facets*. There are more different types of edges, of which it is helpful to distinguish those lying entirely in out-regions as *out-edges*.

other than the Euclidean distance is important in fluid mechanics and many interesting properties of subdivisions on the basis of the closest pair of points (or trio of points and so forth) known as higher order Voronoï diagrams are described in [16].

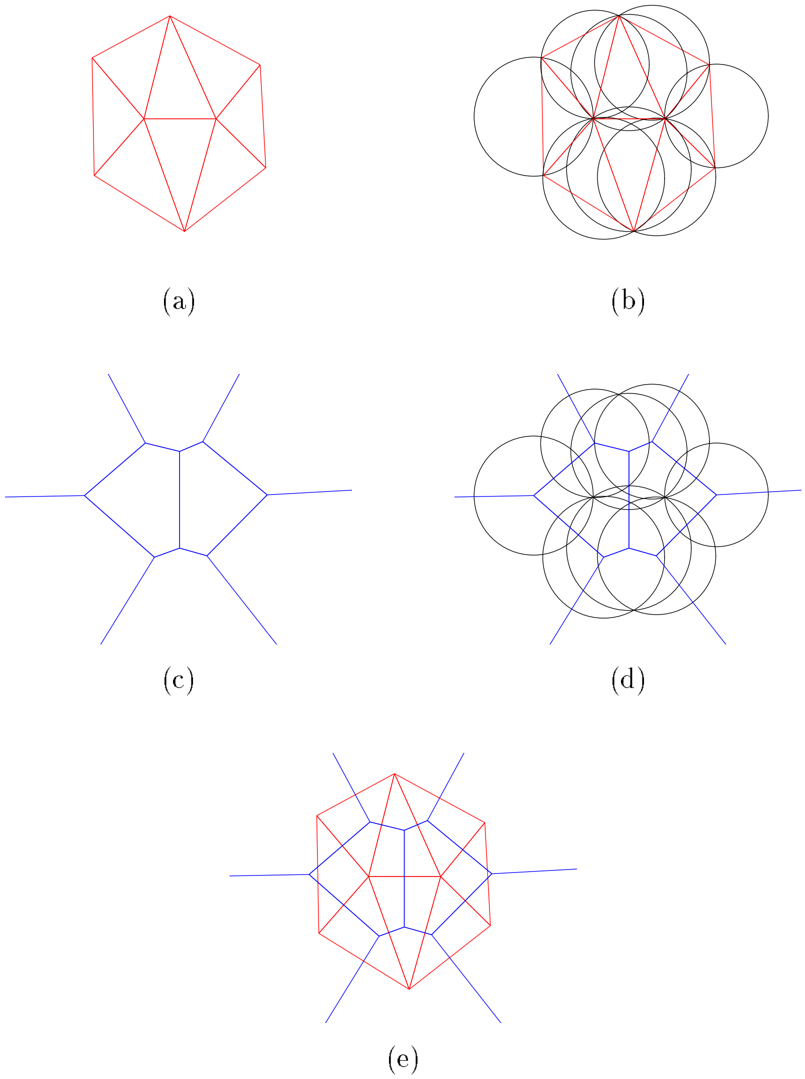
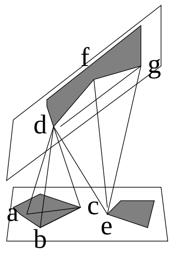


FIG. 10: The relationship between the Delaunay triangulation and the Voronoï diagram. (a) The Delaunay triangulation of a set of points: the outer boundary is the convex hull of the point set. (b) The Delaunay triangulation with finite circumcircles overlaid: each circumcircle is maximal and empty. (c) The Voronoï diagram: the regions are convex but not triangular. (d) The Voronoï diagram with the Delaunay circles overlaid: the circumcentres are the Voronoï vertices. (e) The Delaunay triangulation with Voronoï diagram overlaid: Delaunay edges correspond to Voronoï edges although they may not intersect, Delaunay vertices correspond to Voronoï regions and Voronoï vertices to Delaunay triangles.



These definitions are illustrated in the adjacent figure, which shows a pair of non-parallel planes with in-regions shaded. Facet *abc* is an in-facet of tetrahedron *abcd*. Facet *dfg* is an out-facet of tetrahedron *defg*. Edge *de* is an out-edge. As this diagram suggests, tetrahedron *defg* and any other tetrahedron containing *de* should be removed in sculpting.

The central fact about sculpting is:

Property 1 *Under certain assumptions, removal of all tetrahedra with out-edges from a conforming triangulation is necessary and sufficient to leave an acceptable set of tetrahedra.*

The only risk in choosing suitable assumptions for this is that a tetrahedron may contain both in-facets and out-facets. Fortunately, the following assumptions make this impossible and so make Property 1 true:

(i) Contours from intersecting planes meet exactly on the line of intersection of those planes, giving a consistent classification of those lines into in-regions and out-regions.

(ii) No contour passes through the point of intersection of three planes or has a finite (that is, non-infinitesimal) section lying on the line of intersection of two planes.

To see that these assumptions work, observe that any two facets of a tetrahedron share an edge. Consider a tetrahedron with both in- and out-facets. Then the edge between these lies in an intersection of two planes which violates assumption (i) or (ii).

As remarked in Section 5.4.1, it is desirable to approximate the solidly connected Delaunay surface. Therefore, all tetrahedra which are part of a connected set of tetrahedra which is only connected to another plane at a point or along an edge should be removed. It is possible to remove further tetrahedra, however no good reason for doing so is evident. Indeed, the removable tetrahedra enable the surface to preserve convexity and the removal would take extra computing time.

6 The Algorithm

6.1 The Top

Here the idea described in the previous section is formalised in an algorithm called **reconstruct**. The description provided is non-incremental. However an incremental version is possible in which a new reconstructed model is generated as each plane is inserted. This only involves some shuffling of function calls on a level one layer lower than that described in the pseudocode here. Such an incremental approach is more computationally expensive since each time a new 3-face is created from an existing triangulation, it must be retriangulated.

For the description of the algorithm, a number of assumptions are made:

[A1] The intersections of the planes are such that no more than two planes meet at a line and no more than two lines meet at a point within the bounding region.

[A2] Contours from intersecting planes meet exactly on the line of intersection of those planes, giving a consistent classification of these lines into regions inside and outside the object to be reconstructed.

[A3] No contour passes through the point of intersection of three planes or has a finite (that is, not infinitesimal) section lying on the line of intersection of two planes.

The removal of assumption [A2] would require a change in the overall problem formulation (Section 3), which involved producing reconstructions which were consistent with the input data. At the same time, all real data is noisy, so procedures for relaxing this assumption are described (Section 7). A function **merge-junctions** is included in the formal description of the algorithm below as a tag indicating one place where such procedures can be applied. The other assumptions can be removed but it would complicate the presentation to give the completely general algorithm, so only pointers to means of removing them are provided.

At input to reconstruction, it is assumed that every plane has an *upper side* defined in terms of its normal. Also, the vertices of contours are assumed ordered so that any point just counter-clockwise of a contour lies inside the object to be reconstructed and any point just clockwise lies outside the object. Other representations involved in the algorithm will become apparent in the following sections. As motivated in Section 5.2, the algorithm first constructs a sub-arrangment $\mathcal{A}$ of the input planes $\mathcal{H}$ with a function **arrange**. The contours $\mathcal{C}$ are then split with a function **split-contour** and the faces of $\mathcal{A}$ labelled by function **label-faces** for reasons explained in Section 5.3. The following pseudocode subsumes a large number of operations including conforming Delaunay triangulation and sculpting in one function **triangulate** which is discussed in Section 6.5.

Algorithm: **reconstruct**
Input: a set of contours $\mathcal{C}$ on planes $\mathcal{H}$
Output: a reconstruction consistent with $\mathcal{C}$

find a convex bounding region $\mathcal{B}$ containing all contours in $\mathcal{C}$
$\mathcal{A} \leftarrow$ **arrange**$(\mathcal{H}, \mathcal{B})$
for each contour $\mathcal{C}_k$ in $\mathcal{C}$, **split-contour**$(\mathcal{C}_k, \mathcal{A})$
merge-junctions
$\mathcal{P}_2 \leftarrow$ list of all PART 2-faces found in splitting
$\mathcal{P}_3 \leftarrow$ **label-faces**$(\mathcal{A}, \mathcal{P}_2)$
for each PART 3-face f in $\mathcal{P}_3$
    $\mathcal{T} \leftarrow$ **triangulate**(f)
    output$(\mathcal{T})$

The algorithm can be made to function for the lower- and higher-dimensional versions of the problem. In 4D, the major difficulty is the fact that a guaranteed method of making a Delaunay triangulation conform to an arbitrary polyhedral surface is presently an unsolved problem [58]: it is easy to make the edges of a surface conform but a surface may not be present in a triangulation even if its edges are.

6.2 Arrangements

Recall from Section 5.2 that a sub-arrangement is a description of the way the space inside some bounding region is divided by a set of planes into a set of faces. This section explains the computation of sub-arrangements. Attention is restricted to the three-dimensional case and the simplicity assumption mentioned above is made, which may be put more precisely as:

[A1] No $(5-k)$ k-faces meet at a $(k-1)$-face within the bounding region.

However, the algorithm described readily generalises to other dimensions and to the treatment of special cases in the intersections of planes. A colourful presentation of these generalisations is given by Edelsbrunner [16], except that he only considers the case of arrangements with no bounding region. For discussion of the linear algebra involved in finding intersections and interior points of lines and planes the reader is referred to [41, 13, 52].

Recall from Section 5.2 that a sub-arrangement may be optimally constructed incrementally. At each stage, the set of faces in an existing arrangement cut by the new plane (the zone of the plane) is found, then each of these faces are cut into three parts. This leads to the following top level algorithm.

Algorithm: **arrange**
Input: a set of planes $\mathcal{H}$, a bounding region $\mathcal{B}$
Output: the sub-arrangement $\mathcal{A}$ of $\mathcal{H}$ in $\mathcal{B}$

load $\mathcal{A}$ with a representation of $\mathcal{B}$
for each plane h in $\mathcal{H}$
    $\mathcal{Z} \leftarrow$ **find-zone**$(h, \mathcal{A})$
    split-zone$(\mathcal{Z}, h, \mathcal{A})$

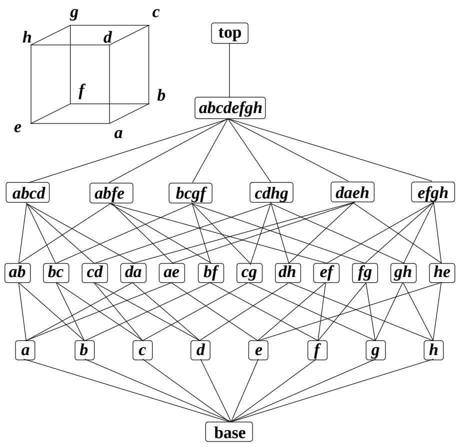
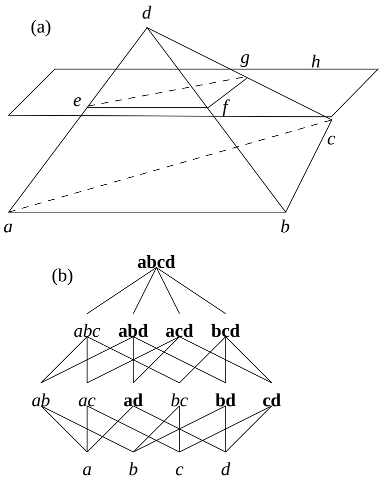


Figure 11: A cuboid and the finite portion of its incidence graph.

To implement this, a representation for an arrangement is needed. The most convenient choice is an *incidence graph*. An incidence graph has the faces of the arrangement as nodes and arcs between two nodes iff one is a subface of the other. For convenience, a *base* node is connected to all 0-faces and a *top* node is connected to all 3-faces.

Figure 11 shows the incidence graph of a cuboid. For the experiments described in Section 8 bounding regions had this shape. When an incidence graph is used in the construction of a sub-arrangement, it is convenient that each face store some arbitrary point internal to it so intersection tests may be performed. It should also contain a label used to indicate the nature of its intersection with the plane currently being inserted.

6.2.1 Computing The Zone



Consider the simplest possible update to an arrangement as shown in the adjacent figure. This is the part of an arrangement corresponding to a single tetrahedron cut by a new plane h. The goal is to find the faces which are shown in bold in the incidence graph in part (b).

Zone computation first finds an arbitrary 1-face in the boundary of the bounding region (the bounding surface) which intersects the new plane h. For example, if the tetrahedron in the adjacent figure is considered to be the bounding region, 1-face ad would do. If the bounding surface is not complex, this may be efficiently achieved by looping over all the 1-faces it had before any planes were inserted until one which is cut (say u) is found.

This is followed by a walk along the actual 1-faces of u until one which is cut is found[12]. If there is no such 1-face then h is entirely out of the bounding region, so the algorithm goes on to the next plane. The zone is then grown out from this face. The key to this is:

Property 2 *The portion of the incidence graph corresponding to the zone is connected. Furthermore, the portion of this subgraph containing 1-faces and 2-faces alone is connected.*

This holds because a plane is connected, and the arrangement fills the bounding region. The 'furthermore' part holds because the intersection with h is itself just a 2D arrangement to which the same argument applies. The property is readily verified for the bold lettered faces in the Figure (b).

[12]Observe that u will generally have already been cut by planes inserted prior to h.

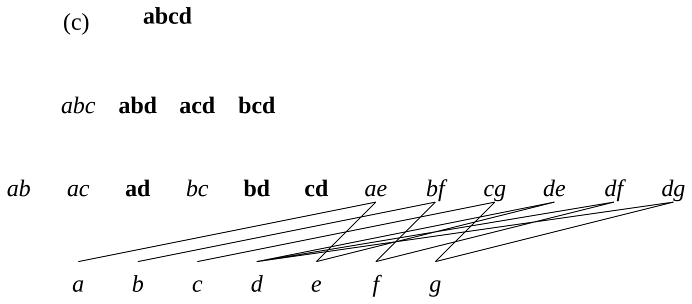
It is easy to tour a connected graph by depth first search. In this case, go to all cut superfaces of a 1-face, then to all cut subfaces of these 2-faces and so on until nothing remains to be visited. A queue $\mathcal{Q}$ of 2-faces and the marking of faces as they are visited prevent cycling in such a search.

Finally, the 3-faces of the zone are just the superfaces of its 2-faces. This gives the following algorithm in which the zone is represented as an array of lists of k-faces $\mathcal{L}_k$. (Each face is assumed to be initially unmarked.)

Algorithm: **find-zone**
Input: a sub-arrangement $\mathcal{A}$, a plane h
Output: the zone $\mathcal{Z} = \{\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3\}$ of h in $\mathcal{A}$

find a 1-face e in the bounding surface cut by h or return
mark an arbitrary superface f of e and append f to $\mathcal{Q}$ and $\mathcal{L}_2$
while $\mathcal{Q}$ is not empty
    $f \leftarrow$ dequeue $\mathcal{Q}$
    for each unmarked subface e of f which intersects h
        mark e and append e to $\mathcal{L}_1$
        mark each unmarked superface g of e and append g to $\mathcal{Q}$ and $\mathcal{L}_2$
for each f in $\mathcal{L}_2$
    mark each unmarked superface g of f and append g to $\mathcal{L}_3$

6.2.2 Splitting The Zone



The splitting of the zone proceeds from low to high dimension: first the 1-faces are split, then the 2-faces and finally the 3-faces. The above figure shows how 1-faces have been split to replace the zone found in the example of the previous page. No removal of old faces has yet occurred and only newly formed links have been shown in order to avoid confusion. An analogous example for a cuboid is shown on the next page.

Pick some direction which is not parallel to h, and define *above* and *below* with respect to it. The part of $\mathbb{R}^3$ above h is denoted by h^+ and that below h by h^-. Then the key to the splitting process may be stated as:

Property 3 *A new plane h splits each face f of its zone into three new faces:*

1. *The part $f_0 = f \cap h$ of f in h. This is of one dimension lower than f. The subfaces of f_0 are (i) any new faces of appropriate dimension which were created by splitting subfaces of f and which lie in h, or (ii) the base if f_0 is a 0-face.*

2. *The part $f_a = f \cap h^+$ of f above h. This is of the same dimension as f. The subfaces of f_a are (i) any subfaces of f which lie above h and were not split, (ii) any new faces of appropriate dimension lying above h which were created by splitting subfaces of f and (iii) f_0.*

3. *The part $f_b = f \cap h^-$ of f below h. This is of the same dimension as f. The subfaces of f_b are (i) any subfaces of f which lie below h and were not split, (ii) any new faces of appropriate dimension lying below h which were created by splitting subfaces of f and (iii) f_0.*

These facts may be readily verified by considering the definition of faces in terms of position vectors as described in Section 5.2. The process for generation of the relevant connections simply applies this proposition, although since it proceeds from low to high dimension, correct connection of some newly created faces to their superfaces may not be performed at their construction but must wait until the iteration at one dimension higher.

Algorithm: **split-zone**
Input: a plane h and its zone $\mathcal{Z} = \{\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3\}$ in a sub-arrangement $\mathcal{A}$
Effects: $\mathcal{A}$ is updated to include h

```
for k = 1...3
    for each f in L_k
        make the new faces
            f_0 = f ∩ h, f_a = f ∩ h^+, f_b = f ∩ h^-
        connect f_0, f_a, f_b according to Property 3
        remove f from the graph
```

6.3 Contour Partitions

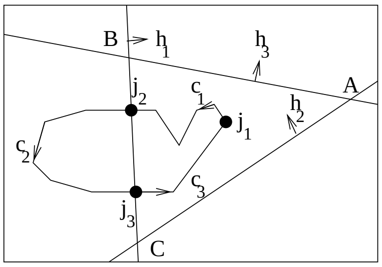


Figure 13: A typical partition of a contour into chains c_1, c_2, c_3, and junctions j_1, j_2, j_3 by intersecting planes h_1, h_2, h_3. j_1 lies in ABC, a 2-face, while j_2 and j_3 lie on BC, a 1-face.

Now the means of splitting the contours into fragments is briefly described. First, recall from Section 5.3 that a *chain* is an open connected portion of a contour which lies within a single 2-face and a *junction* is a point on a contour which lies either on a 1-face or a 2-face and connects together several chains[13]. A contour is split in a single walk around it. At each stage the procedure keeps track of the 2-face currently being traversed and starts new chains and junctions when a 1-face is crossed. Figure 14 shows that a single edge may traverse any number of 1-faces. For later stages of reconstruction it is helpful to make a list of all the 2-faces which are traversed while performing the partition.

[13] It is convenient to have junctions on a 2-face since a chain was defined as being open, while a contour may

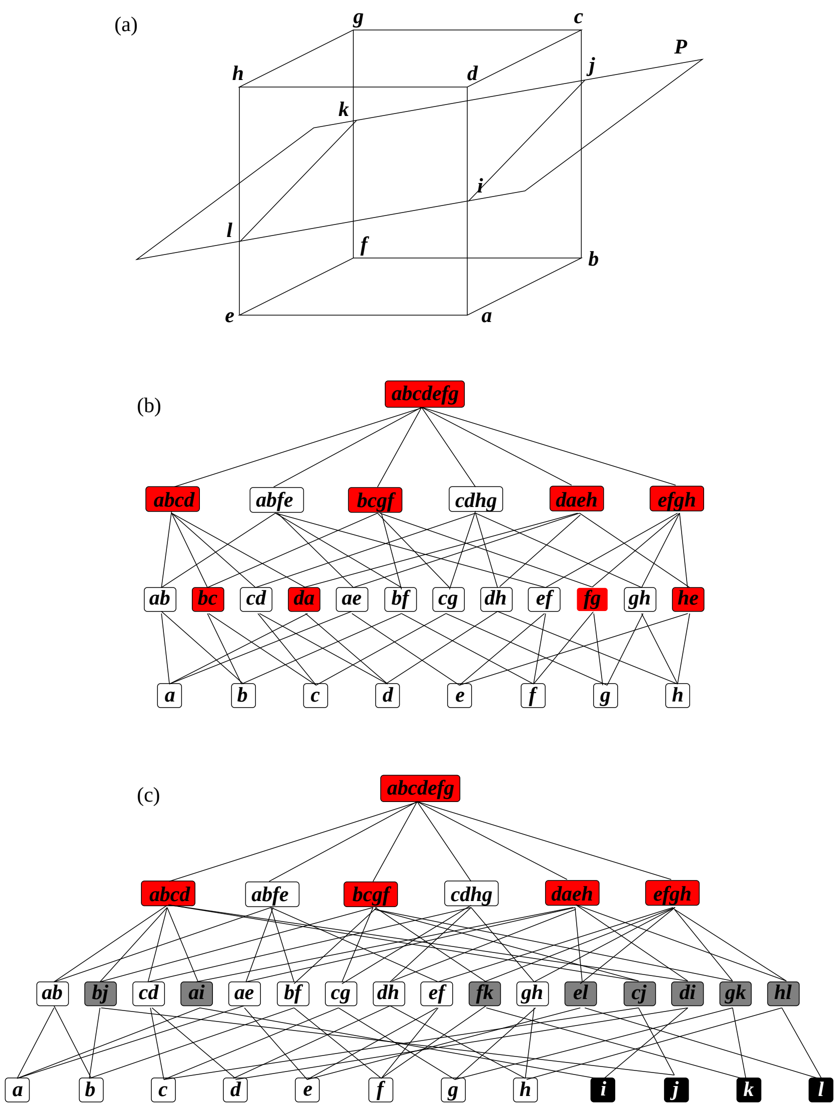


FIG. 12: Addition of the plane P to the finite portion of the incidence graph of a cuboid (compare Figure 11). (a) The geometry of the situation. (b) The incidence graph at the end of **find-zone**. The faces of the zone are shaded. The top and base faces are not shown for simplicity. (c) Immediately prior to the $k = 2$ pass of **split-zone**. Observe how not all 1-faces are correctly connected and how none of the 1-faces involving two new 0-faces have yet been incorporated.

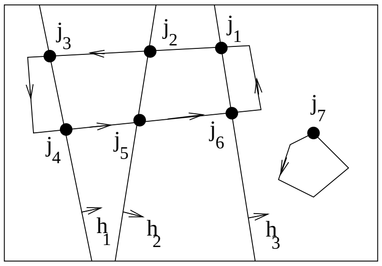


Figure 14: A less typical partition of a pair of contours. This shows that a single edge of a contour may cross many 1-faces before arriving at the next vertex. Alternatively, a contour may be restricted entirely to a single 2-face.

To simplify the subdivision process, the assumption [A3] of Section 6.1 is brought into play. This may now be stated more formally as:

[A3] No contour edge passes through a 0-face, no vertex lies on a 0-face and no contour segment has a finite (not infinitesimal) portion on a 1-face.

The incidence graph of the sub-arrangement makes it easy to test for the requisite intersections: if the 2-face in which one endpoint lies is known, only intersections with the convex polygon bounding that face need be tested for. This polygon is just the set of subfaces of the 2-face. The incidence graph also enables us to trivially locate the 2-face which an edge crosses onto after cutting a 1-face. The incidence graph is useful here since the process is essentially finding the zone of each contour segment. These considerations lead to the following procedure which is assumed to be called once for each contour with all 2-faces initially labelled UNMARKED.

Algorithm: **split-contour**
Input: a single polygonal contour $\mathcal{C}_k$, a sub-arrangement $\mathcal{A}$, a list of 2-faces $\mathcal{P}_2$
Effects: all faces crossed by $\mathcal{C}_k$ are labelled with chains and junctions,
$\mathcal{P}_2$ is updated to contain all 2-faces crossed by $\mathcal{C}_k$ which have
not already been crossed by some other contour

```
f ← the 2-face containing first point a_0 of C_k
start chains and junctions on f
for each edge ab in C_k from a_0
    while ab crosses some subface of f on its way to b
        f ← the neighbouring 2-face crossed into
        if f is UNMARKED, mark f PART and append f to P_2
        create appropriate chains and junctions
    if b ≠ a_0, include b in the current chain
finish the current chain and the first junction
```

6.4 Labelling Faces

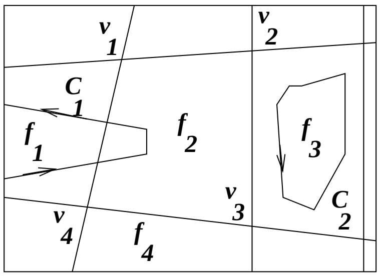
Recall from Section 5.3 that the goal of labelling is to indicate the topological relationship of 0-faces and 3-faces to the object to be reconstructed. To do this, labelling starts with the list of PART 2-faces returned by the procedure described above for partitioning contours. First the 0-faces in the boundary of each PART 2-face are labelled in a walk around that boundary. Then

be confined to a single 2-face. It is also convenient for computation as a junction provides a simple way to start a partition of a contour.

OUT and IN 2-faces are labelled by growing regions using PART 2-faces as seeds. Finally all 3-faces are labelled by counting how many OUT and IN subfaces they have. This gives the following simple overall procedure for labelling.

Algorithm: **label-faces**
Input: an arrangement $\mathcal{A}$ and a list $\mathcal{P}_2$ of all PART 2-faces
Output: all relevant 0-faces, 3-faces and as yet unlabelled 2-faces are labelled
Effects: a list of all PART 3-faces

for each face f in $\mathcal{P}_2$, **label-0-faces**(f)
label-in-out$(\mathcal{P}_2)$
$\mathcal{P}_3 \leftarrow \emptyset$
for each 3-face f in $\mathcal{A}$
  $(n_{\mathrm{IN}}, n_{\mathrm{OUT}}, n_{\mathrm{PART}}) \leftarrow$ the number of subfaces of f of the relevant types
  if $n_{\mathrm{IN}} + n_{\mathrm{PART}} = 0$, label f OUT
  else if $n_{\mathrm{OUT}} = n_{\mathrm{PART}} = 0$, label f IN
  else if $n_{\mathrm{IN}} + n_{\mathrm{PART}} = 1$, label f END
  else label f PART and append f to $\mathcal{P}_3$
return $\mathcal{P}_3$



When labelling 0-faces, care must be taken that all of the boundary of a 2-face may be OUT even though the 2-face itself is PART, as illustrated by the face f_3 which contains contour $\mathcal{C}_2$. Also, although a 1-face may be crossed by a contour, it is not necessarily the case that its 0-faces have different labels. This is the case for the 0-faces v_1, v_4 on the boundary of 2-face f_1 for which the 1-face v_1v_4 is crossed twice by contour $\mathcal{C}_1$. These faces are both OUT.

The method of labelling 0-faces rests on the following property:

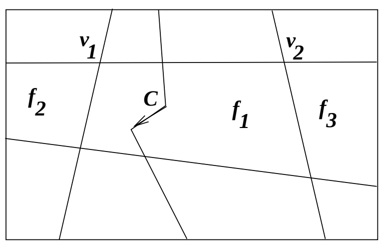
Property 4 *If there is no junction between one 0-face and another 0-face when traversing the boundary of a 2-face in a cycle, then both are identically labelled.*

An example of this property is that v_4 and v_2 have the same label since the path between them via v_3 in the boundary of f_2 has no junctions. This leads to the following procedure:

Algorithm: **label-0-faces**
Input: a PART 2-face f
Effects: all subfaces of f and their subfaces are labelled

find a subface e_0 of f with a non-empty list of junctions
if there is no such 1-face
  label all 0-faces in the boundary of f OUT and return
for each 0-face v in a cycle around the boundary of f starting from a subface of e_0
  if v is UNMARKED
    $v_1 \leftarrow$ the predecessor of v_0 in the cycle around the boundary
    if the 1-face e between v and v_1 has junctions
      $j \leftarrow$ the nearest junction to v
      if v is counter-clockwise of j, label v IN
      else label v OUT
    else label v as v_1 was labelled

To label IN and OUT 2-faces for any plane it is usually sufficient to know at least one PART 2-face with labelled edges for each contour which has been added to that plane. This may not be the case if the set of PART faces contains a connected sequence from one side of the bounding region to another. Therefore labelling is performed starting from the entire set of PART faces for the plane.



Consider the set of all non-PART neighbours of PART 2-faces through 1-faces with an IN subface. These are all IN 2-faces. For example in the adjacent figure, f_1 is a PART 2-face and v_1 is an IN 0-face according to contour C. v_1 is also a subface of the 1-face which f_1 shares with f_2. Therefore f_2 is IN. The analogous argument is true for neighbours of PART 2-faces through 1-faces with an OUT subface. For example in the adjacent figure, f_1 is a PART 2-face and v_2 is OUT so f_3 is OUT.

This argument is simply converted to an algorithm which works with a queue Q of newly-labelled faces which may be either IN and OUT. This queue is first filled with all neighbours of PART 2-faces, which become labelled as they are added, so that it can be assured that no face is added more than once. Any unlabelled neighbour of an entry q in the queue then has the same label as q and may be added to it and so forth until the queue is empty.

Algorithm: **label-in-out**
Input: a list $\mathcal{P}_2$ of all PART 2-faces from any planes to be labelled
Effects: all 2-faces in these planes are labelled

```
for each subface e of each face f in P_2
    g ← the neighbouring 2-face of f through e in the same plane as f
    if g is UNMARKED
        if one of the subfaces of e is IN, label g IN
        else label g OUT
        append g to Q
while Q is not empty
    f ← dequeue Q
    for each UNMARKED neighbouring 2-face g of f in the same plane
        label g as f is labelled and append g to Q
```

6.5 Triangulation

This section shows how the operations of conforming Delaunay triangulation, adding splitting points and sculpting (which were defined and motivated in Sections 5.4.2–5.4.4) can be combined to make the function **triangulate** (alluded to in Section 6.1) which lies at the heart of the reconstruction algorithm.

triangulate acts on a single PART 3-face f. Its first stage must be to find the constraints which f imposes on the triangulation. These are represented as a graph $\mathcal{G}$ and obtained by a function **gather-graph**. Instead of immediately generating a triangulation conforming to $\mathcal{G}$, it is more convenient to first make the Delaunay triangulation of the vertices S of $\mathcal{G}$, using a function **delaunay**. Only then is a conforming triangulation created using a function **conform**.

It is desirable to improve the quality of a triangulation before inserting approximate splitting points (Section 5.4.3). These operations are performed by procedures **delete-obtuse** and **split-lines**. Each of these requires as input a triangulation conforming to $\mathcal{G}$. However, either of these procedures may lead to the triangulation not conforming. Therefore each must be followed by **conform**[14]. Finally, the quality conforming triangulation $\mathcal{T}$ found so far may still violate the contours on the surface of f, so sculpting by a function **sculpt** to produce a subset $\mathcal{T}'$ of $\mathcal{T}$ is needed, giving an overall triangulation procedure as follows:

[14] It is possible to leave out **delete-obtuse**, **split-lines** and all but the first call to **conform**, however results will be inferior when splitting is required.

Algorithm: **triangulate**
Input: a PART 3-face f
Output: a triangulated surface $\mathcal{T}'$ interpolating contour data on f

$\mathcal{G} = (S, \mathcal{E}) \leftarrow$ **gather-graph**(f)
$\mathcal{D} \leftarrow$ **delaunay**(S)
conform$(\mathcal{D}, \mathcal{G})$
delete-obtuse$(\mathcal{D}, \mathcal{G})$
conform$(\mathcal{D}, \mathcal{G})$
split-lines$(\mathcal{D}, \mathcal{G}, f)$
conform$(\mathcal{D}, \mathcal{G})$
$\mathcal{T} \leftarrow$ **output-triangulation**$(\mathcal{D})$
$\mathcal{T}' \leftarrow$ **sculpt**$(\mathcal{T}, \mathcal{G})$

A mysterious aspect of this formal presentation is the presence of the function **output-triangulation** which might seem to rename a Delaunay triangulation $\mathcal{D}$ as a triangulation $\mathcal{T}$. In fact at the current level of description, $\mathcal{D}$ should be considered to be an abstract data structure known as a *dynamic Delaunay triangulation*. Such a structure supports insertion of any set of points at any time using a function **insert-vertices**, while being able to provide output of the Delaunay triangulation itself at any time using a function **output-triangulation**[15]. There are two very different ways in which a Delaunay triangulation can be used in the present application:

- To first construct conforming non-obtuse 2D triangulations of each 2-face of a PART 3-face. Then to efficiently combine these into a 3D triangulation (for example, as shown in Boissonnat [8]).
- To directly construct a 3D triangulation of a whole 3-face.

The 2D way is more efficient since a 2D triangulation need only be computed once for each 3-face in which it is used. It is also less awkward numerically since the fact that points are coplanar is not special in 2D! However, in this report, direct 3D triangulation is used since it makes the overall scheme simpler to implement once numerical problems are overcome. The direct method also enables reconstructions to be made in other dimensions with more minor modifications than are needed to to generalise the combining method.

6.5.1 The Graph to Triangulate

Given a 3-face f, three sorts of vertices must appear in the graph to be triangulated:

1. The vertices of any chains on its subfaces.
2. The vertices at any junctions on its boundary 1-faces.
3. Any boundary 0-faces which are IN.

The reasons for the first two types are obvious, while the reasons for including the third type of vertex have been discussed in Section 5.3.

[15] Strictly speaking this is a *semi-dynamic* method: a dynamic method would also support deletion of points.

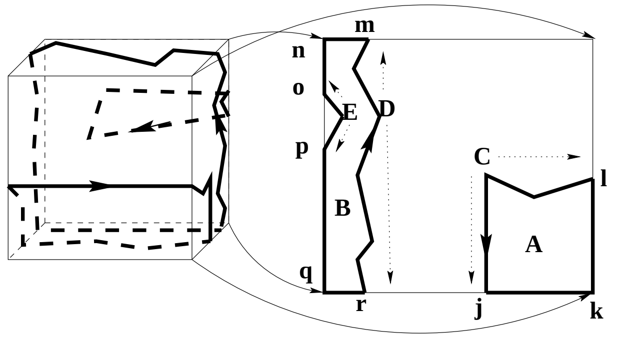


FIG. 15: The graph to triangulate is obtained as one set of loops per face. In loop A, chain C is first followed from junction l to junction j. At junction j, a left-turn is made starting a doubling back portion completed by inclusion of IN 0-face k. Loop B = DmnoEpqr is a more complex example showing how multiple chains (in this case D and E) may be involved in the same loop. At junction o, the correct move is to follow chain E since it classifies line segment op as out of the object.

The set of edges which must be present is simply all edges in all chains on the subfaces of f. However, in order to form a partition of the boundary of the 3-face into regions in and out of the object, which will be seen to be a requirement of sculpting (Section 5.4.4), it is desirable that all these edges are parts of closed planar loops on the 3-face. This can be satisfied by connecting the junctions at the end of chains to neighbouring 0-faces or other junctions and then on to other chains, eventually looping back to the initial chain. At each stage, a turn is made to whichever side preserves the partition of the plane indicated by the contour. This is illustrated by Figure 15[16].

This translates into a simple procedure for graph gathering:

[16]This traversal may lead to certain edges appearing more than once in the resulting graph. As such, it should not strictly be called a graph. Also, it might be imagined that connecting part of a boundary 1-face twice in different loops may lead to errors in the conforming stage as described in Section 6.5.3 if a midpoint were added in order to make such a 1-face conform. This is not the case since these edges lie on the convex hull of the 3-face and so will automatically be present in the triangulation.

Algorithm: **gather-graph**
Input: a 3-face f to triangulate
Output: the graph $\mathcal{G} = (V, \mathcal{E})$ to conform to

```
V ← ∅
ℰ ← ∅
for each subface g of f
    for each unmarked subface e of g
        mark e and append its junction vertices to V
        for each unmarked subface v of e which is IN
            mark v and append v to V
    for each chain C on f
        append the vertices of C to V
        if C is unmarked
            follow a loop starting from C round the boundary of g,
                marking each chain encountered and appending all edges to ℰ
unmark all things which were marked in this procedure
```

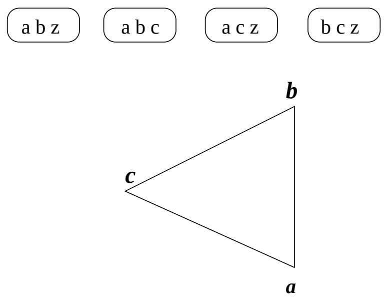
It is possible to perform the edge listing and chain vertex part of graph gathering only once for each 2-face. This is more efficient than the procedure given here and only involves a minor modification.

6.5.2 The Delaunay Tree

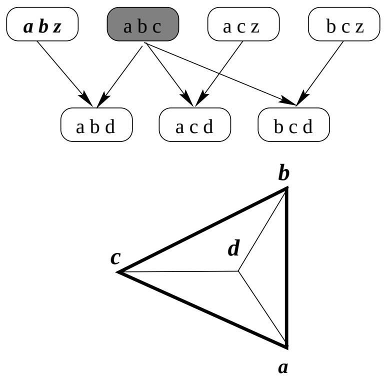
Recall from Section 5.4 that the Delaunay triangulation in 3D is the set of tetrahedra with empty circumspheres. Of many possible algorithms for computing the Delaunay triangulation (see [1] for a survey), the Delaunay tree was chosen as it is incremental, extremely simple and achieves optimal randomised time and storage bounds for any fixed dimension[17]. This method was proposed by Boissonnat and Teillaud [8]. This section first introduces the data structure. Then the central test of the algorithm is described and it is shown how the case of coplanar points is handled: a very important consideration for reconstruction from planar contours. Finally, the algorithm is presented in a top-down fashion.

[17] Randomised time bounds refer in this case to the performance on the worst possible input averaged over random orders of insertion of points in the triangulation. The Delaunay tree algorithm is not output sensitive in some cases as intermediate triangulations can be much larger than the final triangulation [6].

6.5.2.1 Data Structure



Delaunay Trees: (a) The triangulation of points a, b, c and the corresponding Delaunay tree. z denotes the infinite point.



Delaunay Trees: (b) d lies in the circumcircle of the finite root node abc (grey) and therefore kills it. The facets of abc which remain are shown bold: in this case the facets are shared with the infinite simplices.

Every node of a Delaunay tree represents a simplex (recall that this is a triangle in 2D or a tetrahedron in 3D). It is not formally a tree but rather a directed graph with the structure of a family tree: every node has *two* parents. At the roots of the tree lie one node corresponding to a finite simplex and $d{+}1$ corresponding to infinite simplices, each sharing a different facet of the finite simplex. Together, these cover all of $\mathbb{R}^d$. The adjacent Figure (a) shows the roots of a 2D Delaunay tree.

When a new point q is inserted in the Delaunay triangulation $\mathcal{T}$ of a set of points S, it lies in or on the circumball (ball) of at least one simplex (if q is not already in S). This is because $\mathcal{T}$ (including infinite simplices) is a tessellation of all of $\mathbb{R}^d$ and each simplex is within its ball. A simplex s whose ball contains q is said to *conflict*[18]with q. When q is inserted in T, s is said to be *killed*.

By definition of the Delaunay triangulation, a killed simplex is not a Delaunay simplex for $S \cup \{q\}$. If s is adjacent to an unkilled simplex n through some facet f, then the Delaunay triangulation of $S \cup \{q\}$ contains a new simplex fq with vertices $f \cup \{q\}$. In the Delaunay tree, fq becomes a *child* of s and n. All connections in the Delaunay tree arise this way. Figure (b) shows the new connections for Figure (a). The Delaunay triangulation of the set of points which have been inserted in a Delaunay tree is thus just the set of simplices represented by the unkilled nodes.

6.5.2.2 Balls and Conflicts

The parameters of the ball of a finite non-flat simplex with vertices $q_{k=0\ldots d}$ may be trivially derived by considering the conditions that each vertex lies on a sphere (c, r) with centre c and radius r. Shift the origin to q_0, defining $q'_k := q_k - q_0$, $c' := c - q_0$. Then,

$$(q'_k - c')^2 = r^2 \;\Rightarrow\; r^2 = c'^2 \text{ and } q'_k.c' = {q'_k}^2/2 \tag{0.3}$$

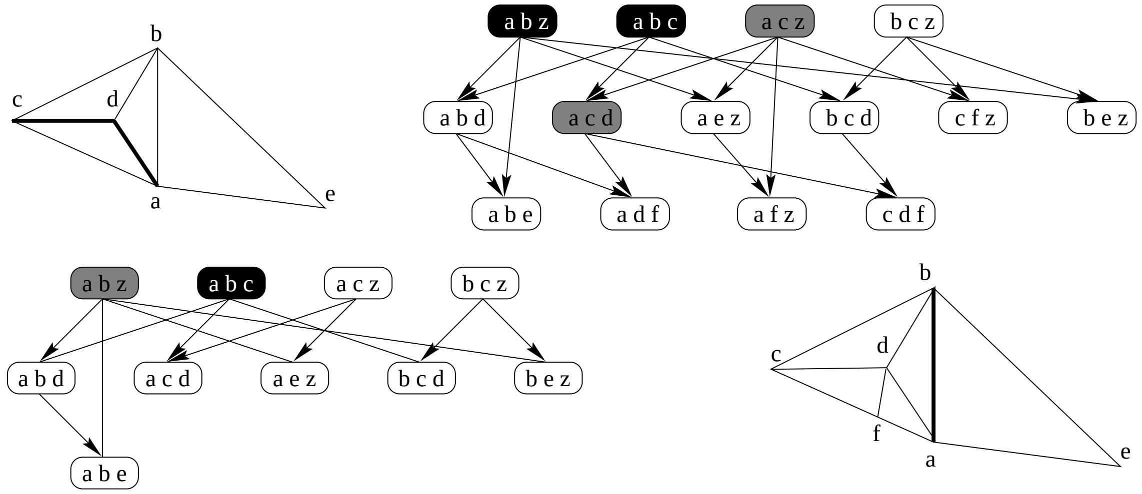
which for $k = 1 \ldots d$ is a square non-singular linear system.

[18]Here the balls are considered *open* and so a point on the ball does not conflict.

To test if a point q lies in an *open* sphere (c, r), one may test whether $(q - c)^2 < r^2$. It is interesting to notice that if q^2 is computed just once for any point and $r^2 - c^2, 2c$ are stored for each simplex, a more efficient equivalent is the point-in-half-space test $(2c).q < (r^2 - c^2 - q^2)$. This is one way of viewing the relation of Delaunay triangulations to convex hulls [16, 41]. Also observe that on division by r^2, this second test implies that a point q should be considered to conflict with an infinite simplex $\{q_{1...d}, \infty\}$ iff it lies in the half-space on the infinite side of the finite face, as long as it is not coplanar with that face. That is, whenever

$$\begin{aligned} \Delta_{v_{1...d}}(q) \quad &:= \quad \text{sign} \begin{vmatrix} 1 & \cdots & 1 & 1 \\ q_1 & \cdots & q_2 & q \end{vmatrix} \\ &= \quad -\text{sign}\,\Delta_{q_{1...d}}(q) \end{aligned} \tag{0.4}$$

for some arbitrary point q which lies strictly outside that half-space. This is illustrated in Figure (c) below. Points coplanar with the finite facet g and hence having $\Delta_{q_{1...d}}(q) = 0$ are considered to conflict iff they lie in the ball of the neighbour through g. This definition is of particular importance when treating data of which large subsets are coplanar and is illustrated in Figure (d).



(c) (d)

Delaunay Trees: (c) Addition of a new point e to the example of the previous page. e lies in the half-plane for the current convex hull-edge ab and therefore kills node abz (grey). The already killed node abc is shown in black. Finite facets of killed simplices shared with unkilled neighbours are shown bold. (d) New point f lies on edge ac which is a convex hull edge and is therefore taken to conflict with it. f also lies in the circumcircle of adc.

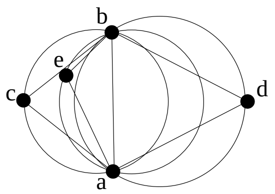
6.5.2.3 Construction

Construction of a Delaunay tree proceeds incrementally, except for the creation of the root simplices, for which any set of $d+1$ linearly independent points is required. At the insertion of a point q the set of simplices $\mathcal{K}$ which are killed by q is obtained by an efficient search procedure named **locate**. Then new simplices are made directly from those in $\mathcal{K}$ and their neighbours in the present triangulation in a procedure called **create**[19]. Thus, the top-level procedure for triangulating a point set looks like:

Algorithm: **delaunay**
Input: a point set S
Output: a Delaunay tree $\mathcal{D}$ containing the Delaunay triangulation of S

construct the roots of $\mathcal{D}$ from a suitable subset $R \subset S$
for each point $q \in S - R$
    $\mathcal{K} \leftarrow \emptyset$
    for each root node s of $\mathcal{D}$, **locate**$(s, \mathcal{K}, q)$
    create$(\mathcal{K}, q)$
return $\mathcal{D}$

The simplices of the triangulation itself may be obtained by traversing the resulting tree and listing all unkilled nodes. If the same loop as in this procedure is made for an arbitrary point set it constitutes the **insert-vertices** function that shall be regularly used in the following sections. In practice other combinations of **locate** and **create** may be used to monitor the appearance and disappearance of particular edges in a triangulation or to simply find which simplex contains a particular point.



The following property is fundamental to the correctness of **locate** [8].

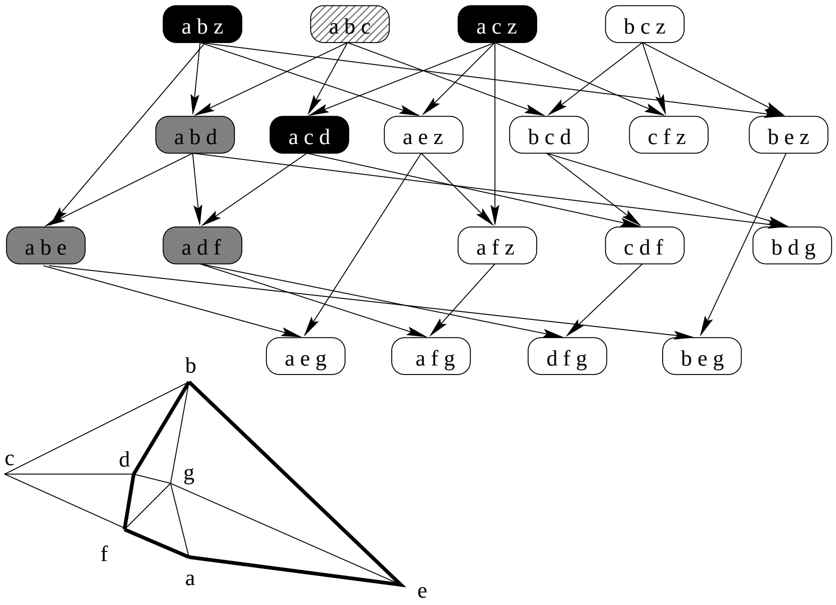
Property 5 *The ball of a simplex is contained in the union of its parents' balls.*

For example, in the adjacent figure the ball of abe is contained within that of its parents abc and abd.

Property 5 implies that if a point conflicts with a simplex, then it conflicts with at least one of its two parents. Applying this argument up the tree it is clear that to locate all conflicting simplices one need only start at the roots and recursively test any child simplex which conflicts. Figure (e) illustrates this. Redundant searches may be eliminated using a *last-visit* entry in each node of the tree, as shown by the following pseudocode[20]:

[19] Observe the similarity with arrangement construction. This is not only the superficial fact that both algorithms are incremental: insertion in each case locates a zone, then splits it to create a replacement.

[20] A minor difference from the version proposed by Boissonnat [8] here is that simplices are not marked killed in **locate**, but only by **create** so that **locate** may also be used purely for point location.



Delaunay Trees: (e) On the insertion of point g, observe how the killed root node abc (hatched) is useful in locating the three new killed nodes.

Algorithm: **locate**
Input: a point q, a simplex s, a list of simplices $\mathcal{K}$
Effects: *last-visit* is updated, simplices are appended to $\mathcal{K}$

if $s[\mathit{last\text{-}visit}] \neq q$
  $s[\mathit{last\text{-}visit}] \leftarrow q$
  if q conflicts with s
    for each child t of s, **locate**$(t, \mathcal{K}, q)$
    if s is not already killed, append it to $\mathcal{K}$

The union of the simplices in $\mathcal{K}$ is a simply connected region, whose boundary consists of a set of facets $\mathcal{F}$. The new simplices are then created in a procedure named **create**. The following property is fundamental to the correctness of **create**:

Property 6 *The new simplices are all simplices of the form fq for each f in $\mathcal{F}$.*

This holds because every Delaunay facet is associated with exactly two simplices (otherwise the Delaunay triangulation would not be a triangulation). Only those facets in $\mathcal{F}$ are missing a simplex after removal of $\mathcal{K}$. The missing simplices cannot be created from any point other than q since otherwise they would still be Delaunay simplices, contradicting the fact that they are missing.

Thus **create** connects every facet of $\mathcal{F}$ to q. Each new simplex is made a child of the simplex it killed and of the neighbouring simplices which it did not kill. Finally the adjacency relations between the new simplices are obtained, leaving the Delaunay tree ready for the insertion of a new point, as in:

Algorithm: **create**
Input: a point q and a list of killed simplices $\mathcal{K}$
Output: a list $\mathcal{L}$ of all new simplices
Effects: simplices in $\mathcal{K}$ are killed, their children are created

```
for each simplex s in K, mark s as killed
L ← ∅
for each simplex s in K
    for each unkilled neighbour n of s through a facet f
        create simplex fq and append it to L
        make fq the neighbour of n and child of n and s
for each simplex s in L
    for each unassigned neighbour of s through facet f
        make s neighbours with the other simplex containing f,
          which comes after s in L
```

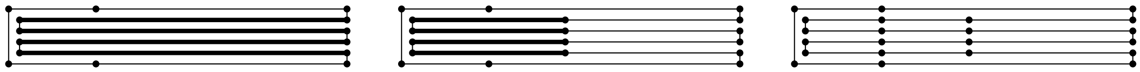


FIG. 16: Insertion of midpoints for conforming triangulation. (a) The original polygon to conform to with vertices marked as little disks. The polygon edges which are missing from the Delaunay triangulation of these vertices are shown in bold. (b) The polygon after one round of midpoint insertion. A shorter total length remains to be conformed to. (c) The polygon after the final stage of midpoint insertion: no edges are missing from the Delaunay triangulation of these vertices.

6.5.3 Conforming Triangulation

A simple way to construct a conforming Delaunay triangulation for a set of edges $\mathcal{E}$ is to repeatedly insert the midpoint m of any edge with endpoints a, b in $\mathcal{E}$ which does not appear in the triangulation, replacing edge ab in $\mathcal{E}$ by the pair of edges am, mb. This process is illustrated in Figure 16. When inserting midpoints, it is important to preserve the sense and order of edges in $\mathcal{E}$ in order that they continue to represent a division of each face into regions in and out of the object being reconstructed.

To implement this, it is necessary to efficiently query a triangulation as to whether or not it contains a given edge. A simple solution would be to use an array where element (i, j) is 1 if triangulation contains edge v_i, v_j, and 0 otherwise. However, the number of edges in a Delaunay triangulation is on the average a good deal smaller than the size of this array. Also, the size of the array is not fixed. Therefore, a hash-table should be used to represent sets of edges for this query.

The overall method is formalised by the following algorithm:

Algorithm: **conform**
Input: a dynamic Delaunay triangulation $\mathcal{D}$ and a graph $\mathcal{G} = (S, \mathcal{E})$ to conform to
Effects: new points are inserted in $\mathcal{D}$, and $\mathcal{G}$ updated accordingly
until the triangulation in $\mathcal{D}$ conforms to the input graph

$\mathcal{E}_{\mathcal{D}} \leftarrow$ the set of edges in $\mathcal{D}$
do
  $S_{new} \leftarrow \emptyset$, $\mathcal{E}_{missing} \leftarrow \mathcal{E} - \mathcal{E}_{\mathcal{D}}$
  for each edge ab in $\mathcal{E}_{missing}$
    append the midpoint c of ab to S_{new} and S
    replace ab by ac, cb in $\mathcal{E}$
  for each point q in S_{new}
    insert q in $\mathcal{D}$, updating $\mathcal{E}_{\mathcal{D}}$ accordingly
while S_{new} is not empty

6.5.4 Obtuse-Segments

In order that the circumcentres of the facets of tetrahedra lying in 2-faces (*2-face triangles*) and containing at least one contour segment as an edge provide a good approximation to the centres of splits, they should lie within their triangles. A necessary and sufficient condition for this is that such triangles be acute. To discuss this it is helpful to define an *obtuse-segment* as a contour segment, one of whose neighbouring 2-face triangles is obtuse.

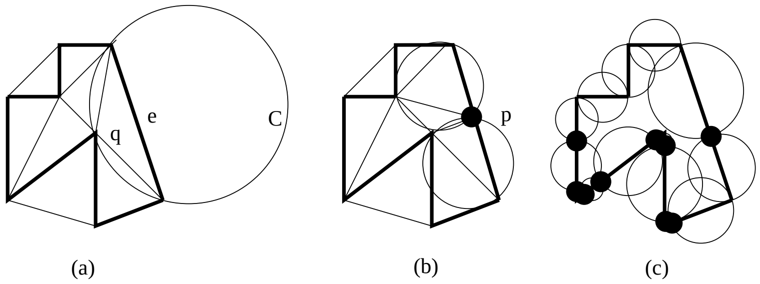


FIG. 17: (a) A Delaunay triangulation conforming to the edges shown in bold. Edge e is an obtuse-segment. The centre of the circumcircle C lies outside its triangle. (b) Removal of obtuse-segments like e by insertion of the projection p of the opposite vertex q makes the centres of the circumcircles of triangles involving parts of the original edge lie within their triangles. (c) A set of edge-free minidisks for the same polygon with extra vertices shown as black disks: the triangulation of these vertices conforms to the graph and is without obtuse-segments.

The natural way to remove an obtuse-segment is to project the vertex in the obtuse neighbouring triangle perpendicularly onto it. This is illustrated in Figure 17 and results in an algorithm like the following:

Algorithm: **delete-obtuse**
Input: a graph $\mathcal{G} = (S, \mathcal{E})$ and a dynamic Delaunay triangulation $\mathcal{D}$ conforming to $\mathcal{G}$
Effects: new points are inserted in $\mathcal{G}$ and $\mathcal{D}$ until no obtuse-segments remain

do
  $S_{new} \leftarrow \emptyset$
  for each edge e in $\mathcal{E}$
    if e is an obtuse-segment for a surface triangle with third vertex q
      $p \leftarrow$ perpendicular projection of q onto e
      append p to S_{new} and update $\mathcal{G}$
  insert-vertices$(\mathcal{D}, S_{new})$
while S_{new} is not empty

This algorithm does not guarantee that all obtuse-segments are permanently removed since when **conform** is repeated after this, obtuse-segments may be re-introduced. Another method is to alternate between **conform** and **delete-obtuse** until no obtuse-segments remain. Geiger [24], could give no proof that this procedure would converge and such a proof is not given here. Instead, it is pointed out that if **conform** adds vertices to contours so that every contour segment has a minidisk (the smallest disk passing through both ends of the contour) empty of other contour segments then there is no need for a **delete-obtuse** stage of the algorithm at all. This is because an obtuse triangle may not be made from an edge with an empty minidisk since the angle in a semi-circle is a right-angle. Such a conforming triangulation may be obtained by the algorithm of [38]. This scheme was not implemented here, but would be a sensible step in any future implementation.

6.5.5 Splitting Points

Now the addition of splitting points is given in detail. First the general principle is described and a means of efficiently implementing it is indicated. Finally, the requisite algebra is provided and the method summarised in an algorithm.

Recall from Section 5.4.3 that splitting points are vertices approximating the locations where the Delaunay surface splits. The planes are divided by contours into *interior* regions and *exterior* regions. The points between such regions on the contours shall be known here as *boundary* regions. Splitting points occur when a ball tangent to an interior point of one plane first hits more than one separate boundary point on another plane in the same 3-face. The extra vertices are then at the points tangent to the balls.

Finding these vertices is readily achieved by first walking around all loops in the graph to which the triangulation conforms and listing all 2-face triangles which are out of the object to be reconstructed. Such triangles are termed *out-triangles* and 2-face triangles which lie in the object are called *in-triangles*. As shall soon be seen, it is useful to label any in- or out-triangles as such in this pass.

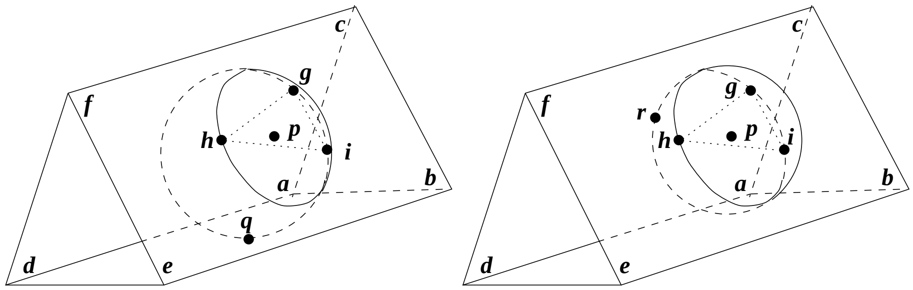
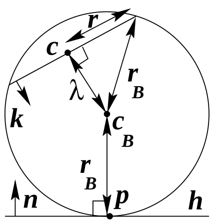
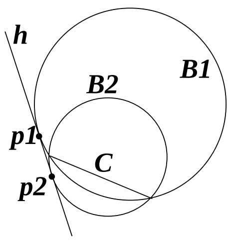


FIG. 18: A pentahedral 3-face $abcdef$ with out-triangle ghi on 2-face $bcfe$ whose circumcircle is shown and has centre p. A loop over 2-faces might first find the ball tangent to 2-face $adeb$ and containing the circumcircle of ghi. This ball hits $adeb$ at q. Next the ball found might be that tangent to 2-face $acfd$ and containing the circumcircle of ghi which hits at r. The projections for the other two 2-faces lie outside the 3-face, so do not receive further consideration. If r is an interior point, it is chosen as the projection since it is nearer to p than q is. Obviously, if r is exterior and q is interior then q is chosen instead.

Given an out-triangle, any splitting point corresponding to it can be found by looping over all 2-faces of the 3-face under consideration (see Figure 18). For each 2-face f the ball tangent to f which contains the circumcircle of the out-triangle in its boundary is solved for, when there is such a ball (it shall be seen below that there are two such tangent balls for a plane, but at most one can hit a given 2-face). It is necessary to find out whether the point of tangency is in the 2-face and is an interior point. This is termed the *interiorness test*. Of those points found in the loop which are interior, that which is nearest to the circumcentre of the out-triangle is chosen.

In order to make this into an efficient procedure, first note that there is no point finding balls for 2-faces which have no interior regions. Secondly, note that the interiorness query is equivalent to finding whether the point of tangency lies in an in-triangle. The answer is clear if the 2-face is IN (see Section 5.3): a point-in-convex-polygon test [41] may be applied. To answer for PART faces, the **locate** function of the Delaunay tree (Section 6.5.2) is used for the tangent point and the returned simplices are looped over to find which has the point in one of its facets. The in-triangle/out-triangle labelling performed in the initial loop then concludes this decision[21]. More efficient methods of finding splitting points for which it is unnecessary to search every 2-face (analogous to ray-shooting in convex polyhedra [41]) could no doubt be devised for situations where large numbers of splitting points are to be found. In the present application, it is unusual to have a large number of 2-faces in a boundary and efficient ray-shooting schemes tend to be rather complicated.





Now a formula for the appropriate ball B (centre c_B and radius r_B) given a circumcircle C (centre c, radius r and unit normal to its plane k) and plane h (unit normal n, distance from origin d) is derived (see the adjacent figure for these definitions). The conditions that B be tangent to h at a point p and that it contain C are

$$(p - c_B).n = r_B, \;\; p.n = d, \;\; r^2 + \lambda^2 = r_B^2, \;\; c_B = c + \lambda k,$$
$$\text{which imply that } r^2 + \lambda^2 = (d - c.n - k.n\lambda)^2 \qquad (0.5)$$

for some scalar λ. The roots of this quadratic are shown in the adjacent figure: there are two balls $B1, B2$ containing C and tangent to h at points $p1, p2$. The correct choice for p is that which lies on the same side of the line of intersection of h with the plane of C as the 2-face for which h was obtained. This is readily computed if each 2-face stores a point internal to it as was suggested in Section 6.2. Taylor expansion is necessary in the solution of this formula when the planes are nearly parallel. In this case, only one root is sensible and the position of p becomes independent of r. Finally, note that once λ, r_B are known, p is obtained as $p = c + \lambda k + r_B n$.

In summary, the algorithm for adding splitting points is as follows:

[21] It might be imagined that performing the interiorness test in order of increasing distance from the circumcentre of the out-triangle would also improve the complexity since fewer faces would require searching. This is not true for worst-case complexity since the sort increases the dependence on the number of 2-faces and in the worst case, all faces need to be examined for interiorness anyway.

Algorithm: **split-lines**
Input: a 3-face g, a graph $\mathcal{G} = (S, \mathcal{E})$ and a dynamic Delaunay triangulation $\mathcal{D}$ conforming to $\mathcal{G}$
Effects: new points are inserted in $\mathcal{G}$ and $\mathcal{D}$ to approximate splitting points

label and list all in- and out-triangles in the unkilled nodes of $\mathcal{D}$
$S_{new} \leftarrow \emptyset$
for each out-triangle t
  $q \leftarrow \infty$ (an infinite point)
  $(c, r, k) \leftarrow$ circumcircle of t
  for each PART or IN 2-face f of g
    compute the tangent point p for f and (c, r, k)
    if f is PART, use **locate** for $\mathcal{D}$ to see if p is interior
    else if f is IN, do a point-in-convex-polygon test for p
    if p is an interior point and $|p - c|^2 < |q - c|^2$, $q \leftarrow p$
  if $q \neq \infty$, append q to S_{new} and to S
insert-vertices$(\mathcal{D}, S_{new})$

6.5.6 Sculpted Triangulation

As described in Section 5.4.4 tetrahedra are removed if they have out-edges or if they are part of a non-solidly connected group.

Tetrahedra with out-edges are easily found by walking around each loop on each 2-face in the conforming triangulation. At each vertex all incident edges which are not part of a loop are considered to find any which lie in the 2-face of the current loop and are clockwise of it. The query of finding all edges incident on a vertex is efficiently supported by a pair of hash tables: one maps from any vertex to a list of edges incident on that vertex and the other maps from any edge to a list of tetrahedra containing those edges. Together these form a *tetrahedral incidence graph*.

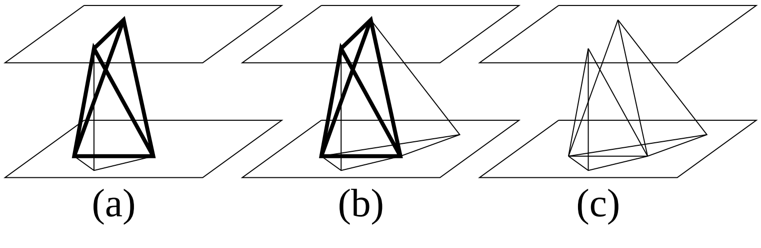


FIG. 19: Types of non-solid tetrahedra, a parallel plane example. (a), (b) Two types of E-non-solid tetrahedron are shown in bold. (c) When the E-non-solid tetrahedron of (b) is removed, two V-non-solid tetrahedra result.

Two types of non-solid tetrahedron are relevant: those which are connected only along edges and those which are connected only at vertices. To formally define these, recall from Section 5.4.4 that a facet in an in-region is called an *in-facet*. A tetrahedron s is said to be *E-non-solid* if

1. s has no in-facets,

2. s has an edge e, in a 2-face,

3. neither of the facets of s incident on e is shared with a tetrahedron which has an in-facet.

A tetrahedron s is said to be *V-non-solid* if there is some 2-face f for which

1. s has a vertex in f,
2. s does not have an edge in f,
3. s does not share a facet with a tetrahedron which has an edge in f.

These definitions are illustrated in Figure 19. They are direct generalisations of those used in [24]. All these definitions can be directly turned into test procedures which work in terms of the planes in which the vertices of a tetrahedron lie and the neighbouring relations of a tetrahedron.

The following obvious property is central to the ordering of the sculpting procedure:

Property 7 *Deletion of tetrahedra with out-edges leads to E-non-solid tetrahedra but not vice versa, while deletion of E-non-solid tetrahedra leads to V-non-solid tetrahedra but not vice versa.*

Therefore sculpting first removes tetrahedra with out-edges, then E-non-solid tetrahedra, then V-non-solid tetrahedra.

When sculpting is complete, although the results contain no edges in violation of the input contours, they may still violate them by not attaching all in-regions to parts of the reconstruction. This is due to the deletion of V-non-solid tetrahedra which contained in-facets. This was seen to be a property of the solidly connected nearest neighbour and Delaunay surfaces in Section 5.4.1.

In summary, sculpting proceeds as follows:

Algorithm: **sculpt**
Input: a triangulation $\mathcal{T}$ conforming to a set of directed loops $\mathcal{M}$
Output: a subset of the simplices of the input $\mathcal{T}$ (returned in $\mathcal{T}$)

```
compute the incidence graph of T
for each loop M_k in M
    for each vertex a of M_k
        for each edge ab incident on a which is still in T
            if ab is clockwise of M_k
                delete all tetrahedra incident on ab
delete all remaining E-non-solid tetrahedra from T
delete all remaining V-non-solid tetrahedra from T
```

6.6 Complexity

This section discusses the worst-case and average-case time complexity of the reconstruction algorithm in 3D. Each procedure called by **reconstruct** is considered, then the total complexity is obtained. The size of the input to **reconstruct** is measured in terms of

$n_{\mathcal{H}}$ – the total number of planes and
$n_{\mathcal{C}}$ – the maximum number of contour points per plane.

The following table summarises the time-complexities demonstrated in the next pages for all calls made to each procedure during one execution of **reconstruct**. It should be borne in mind that the worst case is unrealistic and that the figures given for it ignore **delete-obtuse** and **split-lines**.

Algorithm	$O(\cdot)$ Worst Case	$O(\cdot)$ Average Case
arrange	$n_{\mathcal{H}}^3$	$n_{\mathcal{H}}^3$
split-contour and **gather-graph**	$n_{\mathcal{H}}^2 n_{\mathcal{C}}$	$n_{\mathcal{H}}^2 n_{\mathcal{C}}$
label-faces	$n_{\mathcal{H}}^3 + n_{\mathcal{H}}^2 n_{\mathcal{C}}$	$n_{\mathcal{H}}^3 + n_{\mathcal{H}} n_{\mathcal{C}}$
triangulate, **sculpt** and **output**	$n_{\mathcal{H}}^8 + n_{\mathcal{H}}^2 n_{\mathcal{C}}^6$	$n_{\mathcal{H}}^3 + n_{\mathcal{H}} n_{\mathcal{C}}$
reconstruct	$n_{\mathcal{H}}^8 + n_{\mathcal{H}}^2 n_{\mathcal{C}}^6$	$n_{\mathcal{H}}^3 + n_{\mathcal{H}} n_{\mathcal{C}}$

6.6.1 Worst Case

The worst case of **arrange** depends on the following result [16]:

Property 8 *The number of $(k \leq d)$-faces in a d-dimensional arrangement is $O(n_{\mathcal{H}}^d)$. This is also the time complexity to compute these faces.*

The cost for a sub-arrangement is also this bad since the bounding box may need to be arbitrarily large, so **arrange** is $O(n_{\mathcal{H}}^3)$.

The number of intersections of an arrangement with a set of polygons on one plane is $O(n_{\mathcal{H}} n_{\mathcal{C}})$. This is the maximum number of junctions per plane. Also, if a single contour is entirely in a single 2-face, **split-contour** will take this long to process it. Thus calling **split-contour** for all planes takes $O(n_{\mathcal{H}}^2 n_{\mathcal{C}})$.

label-faces has run-time $O(n_{\mathcal{H}}^3 + n_{\mathcal{H}}^2 n_{\mathcal{C}})$ for the following reasons:

1. All calls of **label-0-faces** will visit a single 0-face at most 12 times by the assumption of Section 6.1 that all 0-faces are the intersection of at most 3 planes. **label-0-faces** always examines every junction in any plane twice. Thus all calls of **label-0-faces** together take $O(n_{\mathcal{H}}^3 + n_{\mathcal{H}}^2 n_{\mathcal{C}})$.

2. **label-in-out** considers any 2-face at most as many times as it has neighbours in the same plane. The sum over all 2-faces of the number of coplanar neighbours is twice the number of 1-faces in that plane. This is $O(n_{\mathcal{H}}^2)$ by Property 8. Thus all calls of **label-in-out** together take $O(n_{\mathcal{H}}^3)$.

3. The labelling of any 3-face requires consideration of all its 2-faces. So in total, all 2-faces are considered twice, giving a run-time of $O(n_{\mathcal{H}}^3)$.

A call of **gather-graph** for a single 3-face collects $O(n_{\mathcal{H}} n_{\mathcal{C}})$ chain vertices, $O(n_{\mathcal{H}} n_{\mathcal{C}})$ junction vertices and $O(n_{\mathcal{H}})$ 0-faces. The numbers of junctions and 0-faces follow from the *Dehn-Somerville relations* ([16] p. 104) which imply the following result:

Property 9 *The number of $(k < 3)$-faces in the boundary of a single 3-face in a 3D arrangement of $n_{\mathcal{H}}$ planes is $O(n_{\mathcal{H}})$.*

Likewise, there may be $O(n_{\mathcal{H}} n_{\mathcal{C}})$ edges in the loops. Obtaining a single vertex or edge is $O(1)$. Thus, **gather-graph** runs in $O(n_{\mathcal{H}} n_{\mathcal{C}})$ time.

The cost of **triangulate** is now considered for the algorithms implemented and then for the best known algorithms.

Given n vertices the 3D Delaunay tree algorithm **delaunay** may take $O(n^3)$ time [8]. As remarked in Section 6.5.3, the number of points added by **conform** depends on the thickness of the graph to conform to rather than the number of vertices and edges. In particular, if the length of edges in the graph to conform to is upper bounded by l and the separation between edges which do not share endpoints is lower bounded by ϵ then $O(l/\epsilon)$ points may need adding. **delete-obtuse** has a similar behaviour.

Let n' be the number of vertices in the triangulation after **delete-obtuse**. **split-lines** considers every out-triangle, of which there may be $O(n')$. This is because the outer boundary of the triangulation prior to sculpting is the convex hull of the input points which has $O(n')$ facets in 3D [41]. **split-lines** then projects onto every 2-face, of which there are $O(n_{\mathcal{H}})$ according to the Dehn-Somerville relations. It also performs a Delaunay tree search to find the projected point. This may take $O(n')$ in the worst case. Thus the implemented version of **split-lines** has run-time $O(n'^2 n_{\mathcal{H}})$ and adds $O(n')$ new vertices. Another conforming operation is performed at this stage. This time any good thickness properties of the input contours may have been destroyed by the addition of splitting vertices.

The best known algorithms for 3D Delaunay triangulation have run-time of the same order as the number of tetrahedra in the triangulation, which is at most $O(n^2)$ [1]. Recall from Section 6.5.3 that the best known algorithm for conforming triangulation in 2D for a graph with n_e edges adds $O(n_e^2 n)$ new points in $O(n_e^2 n + n^2)$ time [58]. A 2D conforming operation for each 2-face is sufficient for 3D conforming. The author knows of no polynomial time conforming algorithm for **delete-obtuse**. Perhaps the best results yet for small angle conforming triangulation are those of [59] which do not consider Delaunay triangulation. Since no appropriate bound is available, **delete-obtuse** and other later triangulation stages are omitted from the total complexity. However, it is observed that point location in **split-lines** can be improved to $O(n')$ preprocessing and $O(\log n')$ location of a single point using monotone subdivisions and fractional cascading [16].

To combine all the stages of **triangulate**, it is first observed that the number of edges and vertices in the input from **gather-graph** for a single 2-face is $O(n_{\mathcal{H}} + n_{\mathcal{C}})$. Then using the Dehn-Somerville relations, the number of vertices on a single 3-face after the second conforming using the best known algorithm is $O(n_{\mathcal{H}}(n_{\mathcal{H}} + n_{\mathcal{C}})^3)$. This gives a total number of tetrahedra and hence run-time of $O((n_{\mathcal{H}}(n_{\mathcal{H}} + n_{\mathcal{C}})^3)^2) = O(n_{\mathcal{H}}^8 + n_{\mathcal{H}}^2 n_{\mathcal{C}}^6)$.

Sculpting need only consider each edge $O(1)$ times and perform a test taking $O(1)$. The number of edges is at most proportional to the number of tetrahedra. Thus if n'' is the number of vertices at input to **sculpt**, sculpting takes $O(n''^2)$ time for a single 3-face.

Fortunately, the worst cost of triangulating all $O(n_{\mathcal{H}}^3)$ 3-faces is not $n_{\mathcal{H}}^3$ times the worst cost for a single 3-face! Firstly, the total number of chain and junction vertices is $O(n_{\mathcal{H}}^2 n_{\mathcal{C}})$. Consider dividing these equally among $f(n_{\mathcal{H}})$ 3-faces and let the cost for triangulating a 3-face with n

edges and vertices be $C_{\mathcal{T}}(n)$. An adversarial input generator picks f to maximise

$$\text{cost of } \textsf{reconstruct} = f\ C_{\mathcal{T}}\left(n_{\mathcal{H}}^2 n_{\mathcal{C}} f^{-1}\right) \tag{0.6}$$

subject to $f = O(n_{\mathcal{H}}^3)$ and $f = \Omega(1)$. Since the exponent of n in $C_{\mathcal{T}}(n)$ is certainly greater than 1, the cost is maximal for $f = O(1)$. The fact that not only chain and junction vertices, but also 0-faces need triangulation does not change this argument because of the Dehn-Somerville relations. Thus, the total run-time for **reconstruct** is $O(n_{\mathcal{H}}^8 + n_{\mathcal{H}}^2 n_{\mathcal{C}}^6)$ in the worst case[22].

6.6.2 Average Case

To provide average-case results, it is assumed that the object being reconstructed is smooth almost everywhere and is kept fixed while

1. the number of contour vertices is increased by random uniform sampling of the object boundary, and

2. the number of planes is increased by adding random uniformly distributed planes which cut the object.

In the average case **arrange**, **gather-graph** and **sculpt** will perform no better than a constant factor of the worst case when performance is measured relative to the size of their input. However the arguments below show that the input sizes for the latter two are reduced.

Meanwhile, the performance of **triangulate** is greatly improved in the average case. The Delaunay triangulation of n uniformly distributed points has on average $O(n)$ tetrahedra and this is the average time taken to compute the triangulation by **delaunay** [8]. A similar improvement is noted for point location using the Delaunay tree in **split-lines** which takes $O(\log n)$ time per point. At the same time, care may be needed since the triangulation of n points distributed on a pair of lines gives the $O(n^2)$ worst case. So, when the reconstructed object is reasonably smooth, some form of removal of collinear points (*thinning*) has been suggested in [24]. Also, as the separation of consecutive contour vertices is decreased, the chance that the edge joining them will be present in the triangulation is much higher. In fact, asymptotically no extra vertices are required in **conform** or **delete-obtuse**. This is at odds with the idea of thinning.

The average performance of **reconstruct** depends linearly on the number of PART 3-faces. The fraction of these faces decreases as the number of planes grows: a surface occupies a rather small volume. In particular, the zone of a plane has size $O(n_{\mathcal{H}}^2)$ [18], so the number of PART 3-faces should be $O(n_{\mathcal{H}}^2)$ on average. Similarly, the zone of a line in a 2D arrangement has size $O(n_{\mathcal{H}})$. This implies the reductions in cost of **split-contour**, and **label-faces** listed in the table of results above. More importantly, it shows that a single PART 2-face will have $O(\frac{n_{\mathcal{C}}}{n_{\mathcal{H}}} + 1)$ vertices and edges in its graph. Together, these give a total cost for calling **triangulate** for all PART 3-faces of $O(n_{\mathcal{H}}^2 + n_{\mathcal{H}} n_{\mathcal{C}})$.

Combining the cost of triangulation with the cost of computing the sub-arrangement gives a total average-case cost for **reconstruct** of $O(n_{\mathcal{H}}^3 + n_{\mathcal{H}} n_{\mathcal{C}})$. This is certainly an improvement

[22] It is doubtful that this bound is tight since the best known conforming algorithm is, in the author's opinion, not optimal.

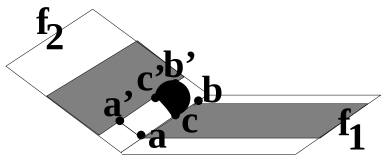
on the worst case! The second term is proportional to the total number of input points and so cannot be improved on. Nonetheless, the cost for computing the arrangement is prohibitive for a large number of planes and one wonders if an alternative scheme could be found which only computes the zone of the object.

7 Inconsistent Intersections

Even for cross-sections extracted from a non-noisy source, it is unlikely that the corresponding contours from different planes will meet exactly on the line of intersection of these planes due to the polygonal approximation involved. However, it is vital for the labelling of faces and formation of loops in the gathering of the graph for triangulation that intersections agree: one cannot hope to produce results consistent with an input when the input is inconsistent with itself. Therefore, consistent intersection was assumed at the start of the algorithm description. This section discusses four methods for enforcing or relaxing this assumption.

7.1 Ignoring Inconsistency.

It is helpful to call each part of a 1-face which is inconsistently described by the input data a *grey segment*. One solution to inconsistency is to ignore grey segments and see what happens.



Consider the pair of regions lying on 2-faces f_1, f_2 in the adjacent figure. Their common 1-face has a grey segment ab. A Delaunay ball tangent to f_1 at c on ab expands to touch c' first. Therefore cc' is part of the Delaunay surface although it is inconsistent with the data on f_2. As c is moved along ab it can be seen that the Delaunay surface will fill the region between aa' and bb'.

Since the Delaunay surface is constructed by adding line segments, it is easy to see that ignoring inconsistency can convert exterior regions to interior ones but never *vice versa*. It is also interesting to note that the size of the inconsistency introduced between reconstruction and data is never greater than the data's original inconsistency. This is a direct consequence of the definition of the Delaunay surface if appropriate measures for these sizes are chosen. For this purpose, the level of inconsistency of a grey segment e could be measured as the maximum distance from any point on e to its closest interior point on the face which classifies e as exterior. The inconsistency introduced could be quantified as the greatest length of any line segment added by the Delaunay surface which lies in an exterior region. Of course, when the face which classifies a grey segment as exterior has no interior regions, it may become entirely filled.

Implementation of this ignoring strategy requires modification of the labelling and sculpting phases of reconstruction:

- In labelling, inconsistency can lead to uncertainty about the choice of label for 0-faces. If a 0-face was really IN and got labelled OUT, there is no chance that the missing space in a reconstruction could get filled later on by existing phases of the algorithm. At the same time, if it was labelled IN but was really OUT, it could get removed by sculpting. Therefore any uncertain 0-face is labelled IN.

- In sculpting, any tetrahedron which would normally be sculpted because it has an exterior edge is not removed if it also has an edge in a grey segment.

7.2 Morphological Filtering.

Another approach is to convolve the data with a ball of appropriate size. This is guaranteed to produce consistent contours. Furthermore, it only causes a level of destruction of the input data in proportion to the size of the errors which are present.

To be precise, consider the input data to reconstruction as a 3D function $B(x)$ of spatial coordinate x with values

$$B(x) = \begin{cases} +\delta & \text{for } x \text{ in the interior regions on planes} \\ 0 & \text{for } x \text{ in between planes} \\ -\delta & \text{for } x \text{ in the exterior regions on planes.} \end{cases}$$

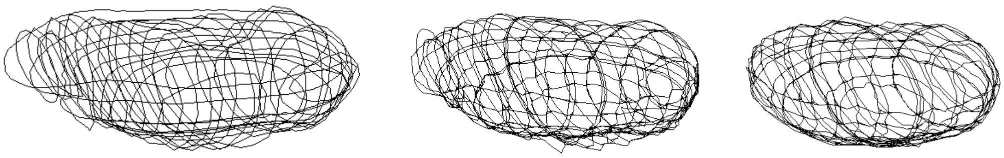
Here δ is a function whose volume integral over a planar region of unit area is unity. Then define the convolution integral

$$J(x) = \int B(x - x')\beta_\epsilon(x')\ dx'^3$$

which involves a function $\beta_\epsilon(x')$ which has value $\frac{3}{4\pi\epsilon^3}$ inside a ball of radius ϵ centred on x' and vanishes elsewhere. Clearly J has value between -1 and 1.

Now consider the locus J_0 of zeros of J on any of the input planes. As $\epsilon \to 0$, J_0 clearly tends to the original contours. However, for any $\epsilon > 0$, J_0 consists of a consistent set of contours.

Figure 20 shows the result of applying this idea to a set of inconsistent gall bladder contours for two different fixed values of ϵ.



(a) (b) (c)

FIG. 20: Convolving contours with a ball to enforce consistency. (a) Inconsistent gall bladder contours prior to convolution. (b) Result of convolution with a ball of radius 1/20 the total length of the bladder. (c) Result of convolution with a ball of radius 1/10 the total length of the bladder.

This example shows that although the results are consistent and resemble the input data, the convolved contours have jagged shapes for small balls and thin sections are strongly eroded for large balls. The issue of how to pick the appropriate value of ϵ for the convolution has not been answered. Furthermore, this convolution took much longer to run than a reconstruction from the contours: it was implemented as a set of planar image operators so every pixel had to be processed independently: this goes against the philosophy of reconstruction from contours.

7.3 Merging Junctions.

A method which adds only one simple stage to the overall procedure and is applicable for low noise levels is to merge inconsistent junctions. An example of this procedure is shown in Figure 21. In this scheme, a loop is made over all lines of plane intersection with junctions.

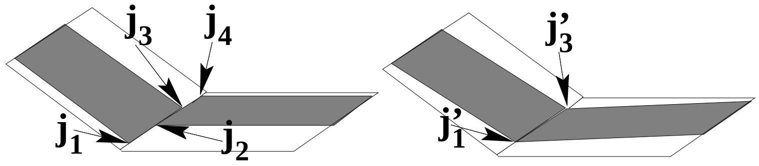


FIG. 21: Merging Junctions. (a) Before merging, the 1-face has 4 junctions, $j_1, \ldots, j_4$ and 2 grey segments. (b) After merging, j_1, j_2 become the point with the average of their positions, j_1', similarly, j_3, j_4 map to j_3'.

Junctions are sorted along these lines and paired with their closest neighbouring junctions from different planes with the correct orientation. A pair of junctions is replaced by a single junction at their midpoint, connected to all four chains that the original junctions had.

It is best to apply this scheme along entire lines of intersection prior to contour partitioning rather than to single 1-faces with junctions after contour partitioning. Firstly, this is because 1-faces may be very small, so the chances of not finding a matching junction on the same 1-face are high. Secondly, if the search for matching junctions were extended to neighbouring 1-faces along a line of intersection, new junctions and a new chains would need to be introduced, since other 1-faces would be crossed by the deformed chain.

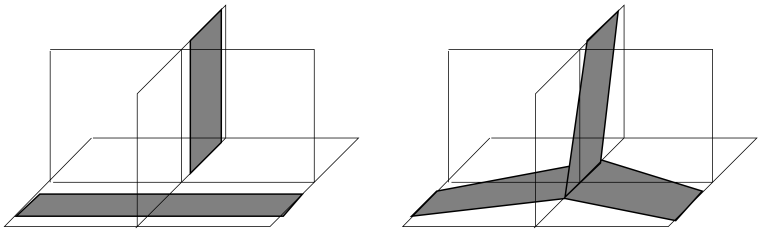


FIG. 22: Merging junctions can lead to crossing of 1-faces which were not previously crossed. This figure shows three transparent planes, two of which have contours whose interiors are shaded. On the left is the situation before merging and on the right that after merging.

This method may fail for a number of reasons. Firstly, not all lines of intersection have the same number of junctions from each incident plane. This may occur even when the error in the contours is arbitrarily small when the line of intersection is nearly tangential to the surface. Secondly, even if suitable pairings can be made, the displacement of a junction may lead to contours intersecting with themselves or with other contours. Thirdly, the act of deforming a contour to make certain lines of intersection consistent may lead to crossings of other lines of intersection and hence new junctions to match. This point is illustrated by Figure 22.

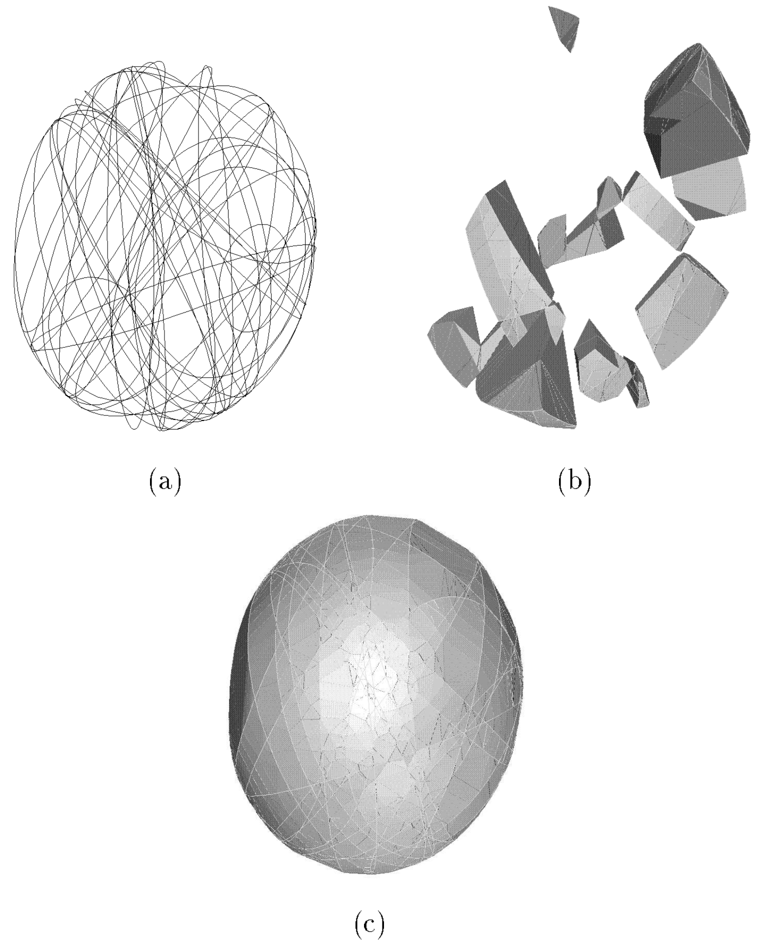
When the method does not fail, results may still not be satisfactory, since sharp jagged parts may be introduced in the contours. However, this can be overcome by a smoothed fading in of the perturbation in the contour (with increased risk of self-intersections), for example weighting with distance along the contour from the junction. Despite these difficulties this scheme has been used successfully for reconstruction from a number of simple mathematically defined objects.

8 Results

This section illustrates the reconstruction method with some artificial data of an ellipsoid and a more complex object with holes. Then some examples with real data are demonstrated: firstly the reconstruction of a pelvis with data from the Visible Human Project [62] and secondly reconstruction of 3D ultrasound data of gall bladders. The latter example is also used for volume estimation.

8.1 Ellipsoid

Consider the reconstruction of an ellipsoid from random planar sections shown in Figure 23. This shows how the reconstruction is performed 3-face by 3-face. No sculpting is necessary here since the object is convex. However, it is necessary to include IN 0-faces in order that the reconstruction fill all IN parts of a 3-face.



(a) (b)

(c)

FIG. 23: The triangulation of 30 random planar sections of an ellipsoid. (a) The input contours. (b) The result when one eighth of the cells have been triangulated. (c) The final object.

8.2 Complex Artificial Object

Figure 24 shows an extreme example of reconstruction for a single 3-face, illustrating how sculpting treats complex objects with holes and non-convexities. First the reconstruction from contours on parallel planes is shown. This could have been achieved by the methods of [24]. Then the input has been rotated onto perpendicular planes: sensible results are still obtained.

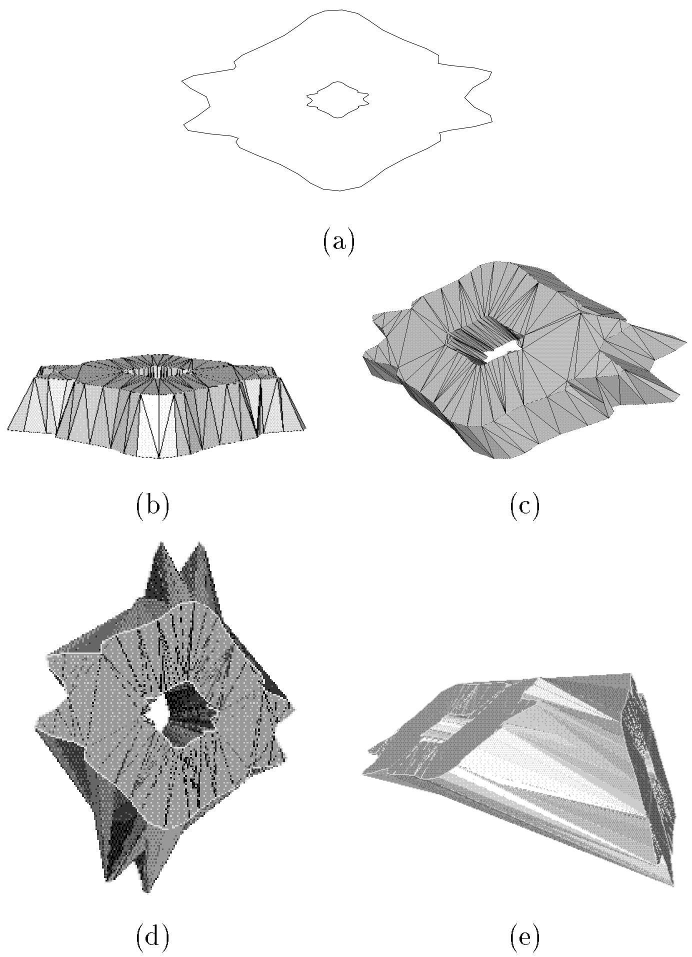
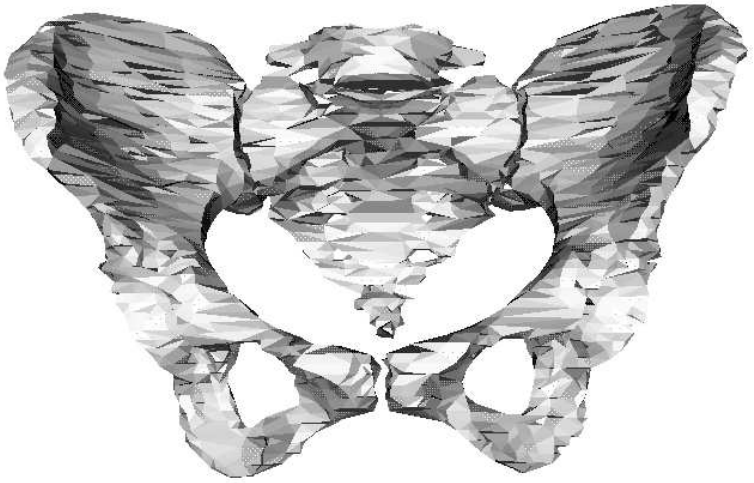


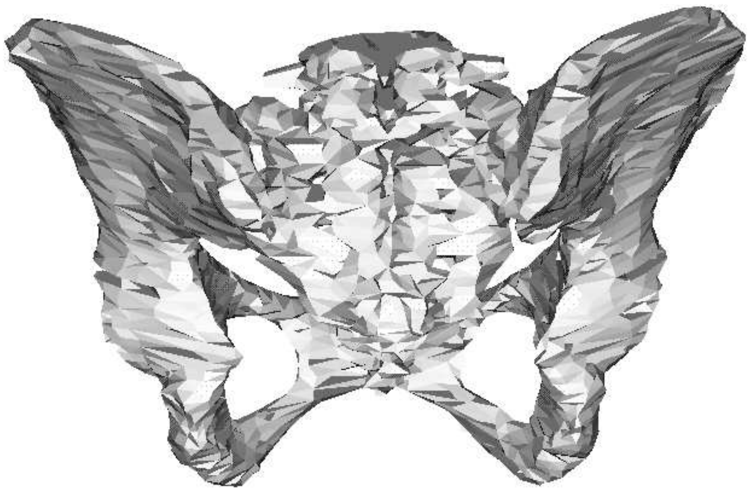
FIG. 24: An extreme demonstration of the approach described: a pair of highly non-convex contours with holes on perpendicular planes. (a) The contours viewed perpendicular to one of the data-planes. (b), (c) Two views of the contours reconstructed on parallel planes. (d), (e) Two views of the reconstructed object on perpendicular planes. Observe the hole right through its centre.

8.3 Pelvis Reconstructions

As part of an investigation of the simulation of childbirth [14], the reconstruction method was used to generate a finite element model of a female pelvis using data from the Visible Human Project [62]. The data consisted of contours extracted from parallel CT slices (numbered 1700-1912 in the Visible Human Project female dataset) by an active contour segmentation scheme. Since parallel slices were used, the same results could have been achieved by the method of [24]. Figure 25(a)-(d) shows four views of the reconstruction.



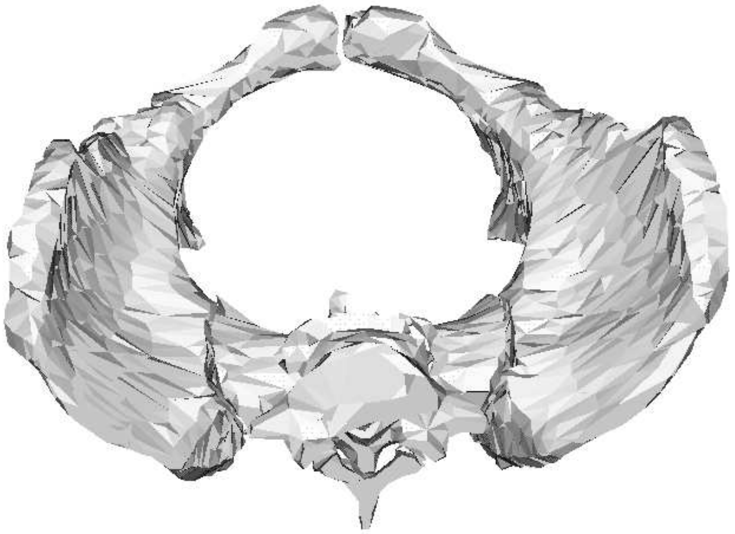
(a)



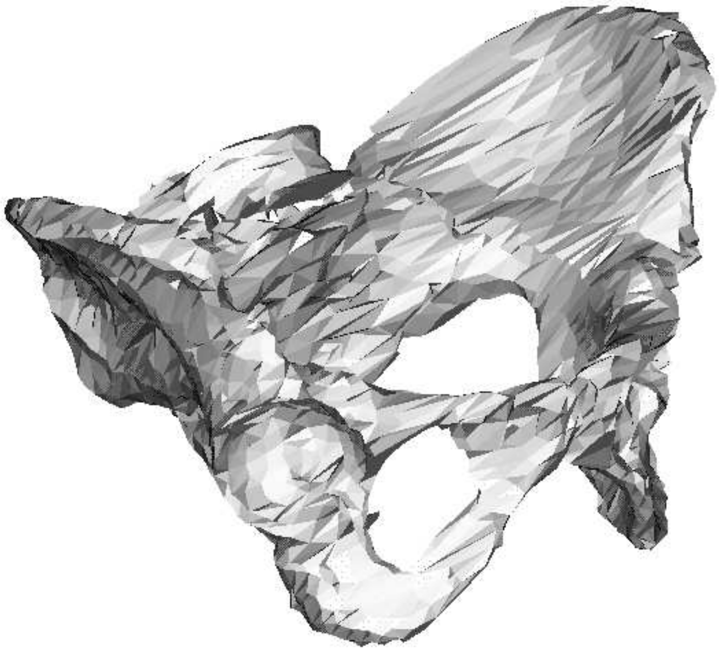
(b)

FIG. 25: The triangulation of CT slices from the Visible Human Project female pelvis. (a) Frontal view. (b) Rear view. Continued overleaf.

The total number of input data points was 5500 and the total CPU time for reconstruction on a SUN Ultra Sparc 1 workstation was 70 s. This reconstruction again illustrates the ability of the method to cope with complex branching data with holes. In several slices, seven different regions needed simultaneous treatment.



(c)



(d)

FIG. 25: continued. (c) The birth-canal. (d) Oblique view.

8.4 Gall Bladder Reconstructions

The gall bladder is a spongy body found within the plane of the major fissure separating the right and left lobes of the liver. Its function is to store bile produced by the liver and bile ducts during fasting and to pass this bile into the duodenum when food is consumed.

The gall bladder was chosen in this study for a number of reasons. Firstly it is fluid-filled and hence easy to segment, while being less subject to internal movement than the arteriovascular system which would otherwise be a natural choice since it is equally easy to segment. Secondly it is usually less occluded by bone and fat than certain other organs, although bowel gas can hinder visibility. Thirdly any entire section of the gall bladder can fit on one scan, and hence there will be no problems with the inapplicability of the formulation of the *surface from cross-sections* problem used above. This is not the case for other regularly scanned organs like the liver and thyroid. Finally, volume measurement, an interesting application for reconstruction, has been well-documented for the gall bladder [45].

All scan data was acquired from a single subject during a one hour session. The subject had fasted for over 6 hours prior to scanning, and continued this fast during the scanning period. This ensured that his gall bladder was full and easy to image. Each scan sequence was taken over a period of approximately 20 s with a 3.75 MHz probe. The subject held his breath and lay as still as possible thoughout.

The clinician moved the probe as steadily as possible and attempted to minimise the variation of pressure on the skin of the subject which might induce motion of the organ. Simultaneously, the organ section was kept as close to the centre of the scan as possible to stay in focus. The sweeps began before one extremity of the organ and terminated after the other extremity, to guarantee that no section of the reconstruction would be missing from the data.

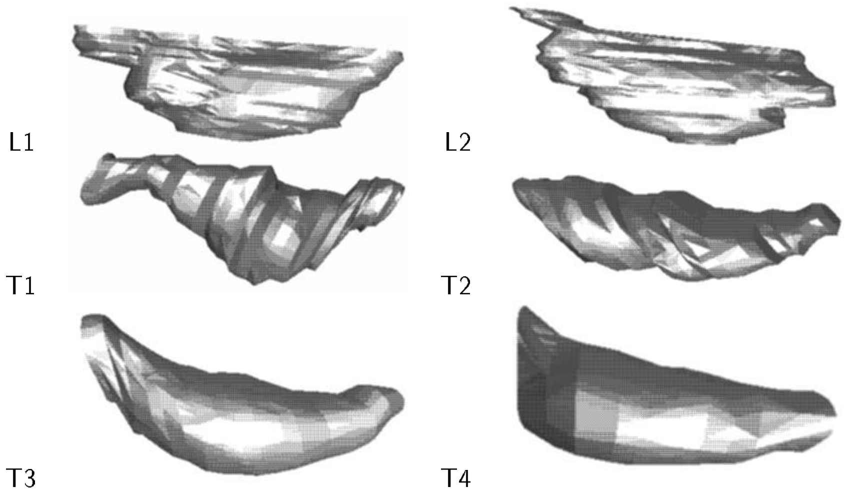


FIG. 26: Gall bladder reconstructions: L indicates reconstruction from longitudinal sections and T reconstruction from transverse sections.

Figure 26 shows the resulting reconstructions. It is interesting how T3 and T4 are much smoother than the other reconstructions. This is not a matter of different numbers of planes: T3 has more planes than the other reconstructions and T4 has less planes than the others.

Additionally, each reconstruction was segmented with the same parameter settings, so this does not provide an explaination either. Furthermore, it seems that the "ends" of the gall bladder occur abruptly. Repeated examination of the scan sequence has shown that this is *not* due to early termination of segmentation – it appears that the gall bladder disappears from the sequence at the end points shown in these figures.

The reconstructions provide an easy means of estimating an object's volume *vol* in terms of the set of all tetrahedra remaining in PART 3-faces after sculpting, $\mathcal{U}$, and the set of IN 3-faces $\mathcal{I}$:

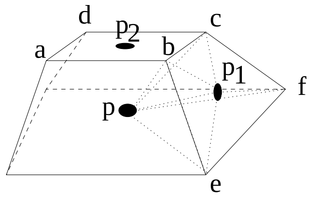
$$vol = \sum_{s \in \mathcal{U}} vol(s) + \sum_{f \in \mathcal{I}} vol(f). \tag{0.7}$$

The volume $vol(s)$ of a tetrahedron s with vertices $v_{1\ldots4}$ (treated as column vectors) is given by

$$vol(s) = \frac{1}{6} \begin{vmatrix} 1 & 1 & 1 & 1 \\ v_1 & v_2 & v_3 & v_4 \end{vmatrix}. \tag{0.8}$$

Either the vertices are sorted or the modulus of this expression taken to ensure that positive volumes are obtained.

The volume of a 3-face f is obtained by decomposing f into tetrahedra. First an arbitrary point p is picked to serve as an apex of all tetrahedra in the decomposition and a point p_k is picked for every 2-face g_k in the boundary of f. p_k should be chosen to be coplanar with g_k. For numerical precision it is preferable that p lie near the centroid of f and p_k near the centroid of g_k. The required volume is then obtained by summing the volumes of all tetrahedra with base triangle composed of p_k and a subface of g_k and with apex p. This well-known decomposition method also works for non-convex polyhedra if signed volumes are allowed.



The adjacent figure illustrates this decomposition of a 3-face. In this figure p is the centroid of the 3-face, p_1 is the centroid of 2-face *befc* and p_2 is the centroid of 2-face *abcd*. The edges of the tetrahedra involved in the sum for *befc* are shown dotted. These tetrahedra are bep_1p, efp_1p, fcp_1p and cbp_1p.

Table 1 gives the volume estimates obtained for the gall bladders illustrated on the previous page for a number of sweeps containing various numbers of planes. These estimates are consistent

Sample	L1	L2	T1	T2	T3	T4
Number of Planes	21	22	35	28	47	18
Estimated Volume (cm^3)	14.7	12.2	17.1	16.1	13.9	16.3

Mean volume 15.1 cm^3, standard deviation 1.8 cm^3.

TABLE 1: Volume estimation of gall bladders using reconstruction from multi-axial planes. L or T in the sample name indicates that the scan was longitudinal or transverse.

with a 10% error in volume estimation on a single sweep (that is, if no averaging over volumes from different sweeps is used). This is with the proviso that no systematic errors are present, which is probably not the case: for one, there should be a systematic underestimation due to the non-triangulation of END 3-faces. Many more scans on different individuals would be needed to increase the level of significance of this error rate. *In vitro* studies would be required to validate the technique to the satisfaction of clinicians.

A 10% standard deviation (fractional repeatability error) in gall bladder volume estimation is of the same order of magnitude as *in vitro* measurements on gall bladder phantoms using the sum of cylinders approach (13.5%) and another 3D method (7.4%) in [45]. Unfortunately this paper does not give fractional repeatability errors for the *in vivo* case although comprehensive *in vivo* studies were performed.

9 Conclusions

In this report, a novel and efficient scheme for reconstruction from arbitrary planar section data has been described. It advances on previous work by combining a guarantee of ability to reconstruct arbitrary data with complex holes and branching (which had previously only been available for parallel plane data), with the ability to treat multi-axial planes (which had previously only been available for data which satisfied a stringent looping criterion). A detailed description of an algorithm implementing the reconstruction scheme has been given.

The technique has been illustrated with a number of complex artificial examples and with parallel plane CT pelvis data. Finally, an application has been made to reconstruction from *in vivo* 3D ultrasound data and subsequent volume estimation.

10 Further Work

There are a number of ways in which the scheme for non-parallel reconstruction could be improved. For example:

1. Presently, the scheme is not *fair* or *self-consistent* in the sense of Section 3. In order to make it so, it is necessary to include splitting points at places other than on the input planes and caps which close off END contour portions. How can this be done?

2. Presently, the scheme generates the part of an arrangment within a bounding box. Is it possible to only construct that part of the arrangment which will need reconstructing?

3. The method of conforming triangulation used can require an arbitrarily large number of points for a fixed number of input points. What is the optimal number of points necessary for conforming Delaunay triangulation? What algorithm can be used for conforming to surfaces in 3D?

4. Is there a more satisfactory approach to the treatment of noise leading to inconsistent intersections?

BIBLIOGRAPHY

[1] F. Aurenhammer. Voronoï diagrams: a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23:345–405, 1991.

[2] G. Barequet and M. Sharir. Piecewise-linear interpolation between polygonal slices. In *10th ACM Computational Geometry Symposium*, pages 93–102, 1994.

[3] J-D. Boissonnat. Representation of objects by triangulating points in 3-D space. In *Proceedings of the 6th International Conference on Pattern Recognition*, pages 830–832, 1982.

[4] J-D. Boissonnat. Representing 2D and 3D shapes with the Delaunay triangulation. In *Proceedings of the 7th International Conference on Pattern Recognition*, pages 745–748, 1984.

[5] J-D. Boissonnat. Shape reconstruction from planar cross sections. *Computer Vision, Graphics and Image Processing*, 44:1–29, 1988.

[6] J-D. Boissonnat, A. Cérézo, O. Devillers, and M. Teillaud. Algorithme dépendant de la sortie pour la triangulation de Delaunay 3D d'ensembles de points contraints. Technical Report RR-1415, INRIA, Sophia Antipolis, B.P. 109, 06561 Valbonne Cedex, France, 1990. Available by ftp from `ftp.zenon.inria.fr`.

[7] J-D. Boissonnat and B. Geiger. Three-dimensional reconstruction of complex shapes based on the Delaunay triangulation. Technical Report RR-1697, INRIA, Sophia Antipolis, B.P. 109, 06561 Valbonne Cedex, France, 1992.

[8] J-D. Boissonnat and M. Teillaud. On the randomized construction of the Delaunay tree. *Theoretical Computer Science*, 112:339–354, 1993.

[9] F. Bookstein. *Morphometric Tools for Landmark Data: Geometry and Biology.* Cambridge University Press, 1st edition, 1991.

[10] Y-K. Choi and K. H. Park. A heuristic triangulation algorithm for multiple planar contours using an extended double branching procedure. *Visual Computer*, 10(7):372–387, 1994.

[11] H. N. Christiansen and T. W. Sederberg. Conversion of complex contour line definitions into polygonal element mosaics. *Computer Graphics*, 13(2):187–192, 1978.

[12] L. T. Cook, P. N. Cook, K. R. Lee, S. Barnitzky, and B. Y. S. Wong. An algorithm for volume estimation based on polyhedral approximation. *IEEE Transactions on Biomedical Engineering*, 27:493–500, 1980.

[13] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, Massachusetts, 1990.

[14] C. R. Dance, R. J. A. Lapeer, and R. W. Prager. 3D finite element model of a female pelvis reconstructed from CT images. In M. J. Ackerman and V. M. Spitzer, editors, *Visible Human Project Conference*, Bethesda, Maryland, October 1996.

[15] J. P. Duncan and S. G. Mair. *Sculptured Surfaces in Engineering and Medicine.* Cambridge University Press, Cambridge, England, 1983.

[16] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Heidelberg, New York, London, Paris, Tokyo, 1987.

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

[18] H. Edelsbrunner, R. Seidel, and M. Sharir. On the zone theorem for hyperplane arrangements. *SIAM Journal on Computing*, 22:418–429, 1993.

[19] A. B. Ekoule, F. C. Peyrin, and C. L. Odet. Surface reconstruction for arbitrarily shaped multiple planar domains. *ACM Transactions on Graphics*, 10:182–199, 1991.

[20] O. D. Faugeras. *Three-dimensional computer vision: a geometric viewpoint.* MIT Press, Cambridge, Massachusetts, 1993.

[21] O. D. Faugeras, E. Le Bras-Mehlman, and J-D. Boissonnat. Representing stereo data with the Delaunay triangulation. *Artificial Intelligence*, 44(1–2):41–87, July 1990.

[22] H. Fuchs, Z. M. Kedem, and S. P. Uselton. Optimal surface reconstruction from planar contours. *Communications of the ACM*, 20(10):693–702, October 1977.

[23] S. Ganapathy and T. G. Dennehy. A new general triangulation algorithm for planar contours. *ACM Transactions on Computer Graphics*, 10(2):182–199, 1982.

[24] B. Geiger. *Three-dimensional modeling of human organs and its application to diagnosis and surgical planning.* PhD thesis, INRIA, Sophia Antipolis, B.P. 109, 06561 Valbonne Cedex, France, 1993.

[25] C. Gitlin, J. O'Rourke, and V. Subramanian. On reconstruction of polyhedra from parallel slices. *International Journal of Computational Geometry and Applications*, 6(1), 1996.

[26] M. J. Herbert, C. B. Jones, and D. S. Tudhope. Three-dimensional reconstruction of geoscientific objects from serial sections. *Visual Computer*, 11(7):343–359, 1995.

[27] S. W. Hughes, T. J. Darcy, D. J. Maxwell, W. Chiu, A. Milner, and J. E. Saunders. Volume estimation from multiplanar 2D ultrasound images using a remote electromagnetic position and orientation sensor. *Ultrasound in Medicine and Biology*, 22(5):561–572, 1996.

[28] N. Kehtarnavaz and R. J. P. de Figueiredo. A framework for surface reconstruction from 3D contours. *Computer Vision, Graphics and Image Processing*, 42:32–47, 1988.

[29] E. Keppel. Approximating complex surfaces by triangulation of contour lines. *IBM Journal of Research and Development*, 19(1):2–11, 1975.

[30] S. Kumar, S. Han, D. Goldgof, and K. Bowyer. On recovering hyperquadrics from range data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(11):1079–1083, 1995.

[31] C. Levinthal and R. Ware. Three-dimensional reconstruction from serial sections. *Nature*, 236(5):207–210, 1972.

[32] W. E. Lorensen and H. E. Cline. Marching Cubes: a high resolution 3D surface construction algorithm. *Computer Graphics*, 21(3):163–169, 1987.

[33] M. Marko, A. Leith, and D. Parsons. Three-dimensional reconstruction of cells from serial secions and whole cell mounts using multi-level contouring of stereo micrographs. *Journal of Electron Microscope Technology*, 9:395–411, 1988.

[34] D. Meyers and K. Sloan. Surfaces from contours. *ACM Transactions on Graphics*, 11(3):228–258, 1992.

[35] V. Milenkovic. Robust polygon modelling. *CAD*, 25:547–566, 1993.

[36] M. E. Mortensen. *Geometric Modelling*. John Wiley, 1985.

[37] H. Müller and A. Klingert. *Surface interpolation from cross sections*. Springer Verlag, 1993.

[38] L. R. Nackman and V. Srinivasan. Point placement algorithms for Delaunay triangulation of polygonal domains. *Algorithmica*, 12(1):1–18, July 1994.

[39] J-M. Oliva, M. Perrin, and S. Coquillart. 3D reconstruction of complex polyhedral shapes from contours using a simplified generalized Voronoï diagram. *Computer Graphics Forum*, 15(3):C 397–C 408, 1996.

[40] J. O'Rourke. Polyhedra of minimal area as 3D object models. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 664–666, 1981.

[41] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, Cambridge, England, 1993.

[42] J. O'Rourke. On the scaling heuristic for reconstruction from slices. *Computer Vision, Graphics and Image Processing*, 56:420–423, 1994.

[43] J. O'Rourke, H. Booth, and R. Washington. Connect-the-dots: a new heuristic. *Computer Vision, Graphics and Image Processing*, 39:258–266, 1987.

[44] H. Park and K. Kim. Smooth surface approximations to serial cross-sections. *CAD*, 28:995–1005, December 1996.

[45] J. Pauletzki, R. Sackmann, J. Holl, and G. Paumgartner. Evaluation of gall bladder volume and emptying with a novel 3-dimensional ultrasound system – comparison with the sum-of-cylinders and the ellipsoid methods. *Journal of Clinical Ultrasound*, 24(6):277–285, 1996.

[46] B. A. Payne and A. W. Toga. Surface reconstruction by multiaxial triangulation. *IEEE Computer Graphics and Applications*, 14(6):28–35, 1994.

[47] L. Piegl and W. Tiller. Algorithm for approximate NURBS skinning. *CAD*, 28:699–706, September 1996.

[48] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, New York, 1985.

[49] M. A. Price, C. G. Armstrong, and M. A. Sabin. Hexahedral mesh generation by medial axis subdivision. *International Journal of Numerical Methods in Engineering*, 38(19):3335–3359, 1995.

[50] L. L. Schumaker. *Reconstructing 3D objects from cross-sections*. Kluwer Academic Publishers, 1993.

[51] T. W. Sederberg and E. Greenwood. A physically based approach to 2-D shape blending. *Computer Graphics*, 26:25–34, 1992.

[52] R. Sedgewick. *Algorithms in C++*. Addison-Wesley, Reading, Massachusetts, 1992.

[53] M. Shantz. Surface definition for branching, contour-defined objects. *Computer Graphics*, 15(2), 1981.

[54] Y. Shinagawa, T. L. Kunii, and Y. L. Kergosien. Constructing a Reeb graph automatically from cross-sections. *IEEE Computer Graphics and Applications*, 11:44–51, 1991.

[55] Y. Shinagawa, T. L. Kunii, and Y. L. Kergosien. Surface coding based on Morse theory. *IEEE Computer Graphics and Applications*, 11:66–78, 1991.

[56] K. R. Sloan and J. Painter. Pessimal guesses may be optimal. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10:949–955, 1988.

[57] B. I. Soroka. Generalised cones from serial sections. *Computer Vision, Graphics and Image Processing*, 15:154–166, 1981.

[58] T-S. Tan. *Optimal two-dimensional triangulations*. PhD thesis, University of Illinois at Urbana-Champaign, 1993.

[59] T-S. Tan. An optimal bound for high-quality conforming triangulations. *Discrete and Computational Geometry*, 15:169–193, 1996.

[60] G. Tracton, J. Chen, and E. Chaney. CTI: automatic construction of complex 3D surfaces from contours using the Delaunay triangulation. In *Mathematical Methods in Medical Imaging III*, pages 86–97, San Diego, California, July 1994.

[61] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *SIGGRAPH 94*, 1994. Available from `http://www-graphics.stanford.edu`.

[62] U. S. National Library of Medicine. The visible human project. `http://www.nlm.nih.gov/research/visible`.

[63] J. K. Udupa. Interactive segmentation and boundary surface formation for 3-D digital images. *Computer Vision, Graphics and Image Processing*, 18:213–235, 1982.

[64] R. C. Veltkamp. *Closed Object Boundaries from Scattered Points.* PhD thesis, Erasmus University Rotterdam, 1992. Available by ftp from `ftp.cwi.nl pub/remco` and published as "Closed Object Boundaries from Scattered Points, Lecture Notes in Computer Science 885, Springer, 1994.".

[65] N. P. Weatherill and O. Hassan. Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. *International Journal of Numerical Methods in Engineering*, 37:2005–2039, 1994.

[66] I. Weiss. 3D shape representation by contours. *Computer Vision, Graphics and Image Processing*, 41:80–100, 1988.

[67] E. Welzl and B. Wolfers. Surface reconstruction between simple polygons via angle criteria. In *Proceedings, 1st Annual European Symposium on Algorithms (ESA)*, volume 726 of *Lecture Notes on Computer Science*, pages 397–408, 1993.

[68] J. Woodwark. *Computing Shape: an introduction to the representation of component and assembly geometry for computer-aided engineering.* Butterworths, Sevenoaks, Kent, England, 1986.