Synthesis of Bent Sheet Metal Parts from Design Features

Roger Bush and Carlo Sèquin

Computer Science Division
University of California

Abstract

We consider the problem of automatically synthesizing a 2.5 *D* sheet-metal part in the presence of other parts in an assembly. The part is to be fabricated by cutting or stamping a single piece of sheet metal, and then bending it along straight folds. The inputs to the problem are flat regions (design features) arranged in general 3D position. These regions represent the important interfaces to other parts or to the world. The synthesized part must be realizable from a single sheet of material by cutting and bending; it must also avoid other parts in the assembly, and incorporate the flat regions.

We present a three stage algorithm that solves this synthesis problem. The first stage computes a 3D global *connectivity graph* that indicates which pairs of features should be connected to one another by synthesized, uniform-width spars of material. The second stage calculates a *folding plan* that determines how the 3D shape can be folded flat. Executing this plan transforms the 3D problem into a simpler 2D one. The final stage is the spar synthesis stage which calculates constant-width, flat spars to join each pair of features that the connectivity graph indicates should be connected. These spars have constraints on their shape that arise due to obstacles, topological consistency, fold geometry, and minimal weight.

We have built a simple, 3D CAD system that allows the designer to directly manipulate the design features for a single part and create simple obstacles representing other parts that must be avoided. The system synthesizes, at interactive speeds, the outline of the 2D flattened part as well as the 3D part.

CR Categories and Subject Descriptors: 1.3.5 [Computer Graphics] Computational Geometry and Object Modeling - Geometric algorithms, languages, and systems; 5.6 [Computer-Aided Engineering] Computer-aided design (CAD).

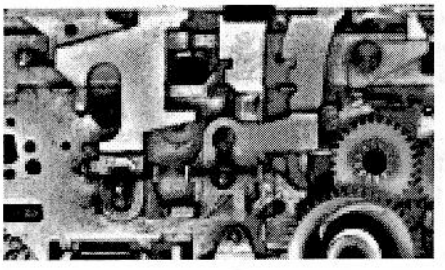
Additional Keywords: sheet metal part design, automated design.

Fifth Symposium on Solid Modeling Ann Arbor MI

1 PROBLEM STATEMENT

Today's mechanical CAD programs require the user to express a design through explicit geometric shapes. However, in many applications, this is an over-specification. For example, many of the member parts in large assemblies often serve a simple structural function in which the exact shape is of little importance. In these situations, using a CAD program as a design tool is very tedious since routine adjustments may require many detailed geometric changes to the parts which have to be specified through explicit geometric edits.

FIGURE 1.



Internal details of a walkman type cassette player. Most of the parts fulfill a simple function, such as translating the force of the play or stop buttons into the appropriate internal action. Routine removal ofpart-to-part interferences involves much tedious work, especially with a traditional CAD system. The parts are *fairly arbitrarily shaped and probably many times stronger (and heavier) than the job requires. Regardless of their function, all distinct parts in a machine must not intelfere, must connect their important portions (features), and must be strong enough to perform without failing.*

For these design problems, a better CAD system would allow specification of the functionally important aspects of the shape. These are typically known by the designer in the earliest stages of design and are not likely to change. For example, in the conceptual phase of design, important part interfaces are generally known, e.g. holes, mating faces, contact surfaces and spatial orientations of the *important portions* of the related parts. Further it is probably much easier to describe a change to these higher-level entities than to describe explicitly all the corresponding geometric changes. In order to be truly useful, such a system must be able to take the high-level description and generate an explicit geometric model that fulfills the specification.

The subject of this paper is an algorithm that takes high-level, *feature-based* input and automatically generates a reasonable geo-

metric part. The problem domain is parts that may be made by taking a flat piece of sheet metal, cutting (or stamping) out a single flat shape, and then bending it into position along straight folds. The inputs to the algorithm are flat part regions, also called *design features*, represented by polygonal boundaries in general 3D position. The other major inputs to the algorithm are 3D obstacles that represent other parts in the assembly or the swept volume of a moving part. In order to be a valid solution for the inputs, the synthesized solution part must connect all of its design features, avoid the obstacles, and unfold into a flat region that may be cut from a single sheet of material. The bulk of this paper is dedicated to describing our algorithm. We also describe a simple, interactive 3D editing system that was implemented to demonstrate and test the algorithm.

2 PREVIOUS WORK

There is an area of CAD research that is fairly broad called the *design-with-features* approach to CAD. Surveys of the CAD literature on features can be found in [9]. Design-with-features can be seen as an attempt to move interaction with the design tool to a higher level of abstraction. In the design-with-features approach, parts are composed of design features that represent the functional geometry of the part.

A popular design-with-features approach focuses on parametrically dimensioned parts that generate explicit geometry. Parametric approaches are useful for designing families of parts that are slightly different from each other, and well understood. However, the parametric approach can't change a part's topological characteristics, it only dimensions the topology that is already provided by the designer.

The parametric approach is not broad enough, since topological changes are required to resolve fairly simple, routine problems. For example, if two parts intersect in space, a natural solution might be for one part to bend and go around the other part. It is unlikely that changes to the existing dimensions of the part would be able to produce such a bend. Using a subtractive strategy, i.e. cutting a notch out to avoid an intersection, is also ill-advised, since it may compromise the strength or functionality of the part.

Our work can be seen as an augmentation to the design-with-features approach. While design-with-features concentrates on the design process by using local, geometric features, our synthesis algorithm concentrates on filling in the details *between* these features. Many parts can be seen as some mixture of special interfaces that have a very particular geometric nature, and the *rest of the part*. Specifically, our algorithm synthesizes the intervening material that holds the whole part together, defining its detailed shape and topology as necessary, while also observing some global criteria such as its connectedness, and its non-interference with other elements in the overall assembly.

Most commercial CAD systems have special sheet metal modules that perform a variety of design tasks. Typically, they provide an approach in which the entire shape is composed of parametrically dimensioned features. Changing the specification forward propagates any dimension specifications in the model. The 3D part geometry may be automatically flattened taking into account deformation and material properties to produce a 2D pattern suitable for stamping. Any bend angles may be checked for bending allowances to insure the part will not be overly fatigued.

We see our algorithm as providing a complementary strength to these fully developed packages. To our knowledge, none of these systems varies the topology of the part automatically. Incorporating the ability of a part to automatically change its shape to avoid obstacles would be a powerful addition to any of these packages.

3 EDITING SYSTEM

The interface to the shape synthesis program is a 3D direct-manipulation editing system. Its purpose is to demonstrate the shape synthesis algorithm by facilitating interactive positioning of the design features and obstacles. As such, the emphasis is on qualitative manipulation rather than precise editing. Most objects are edited by grabbing them and dragging with the mouse; in response to this, the synthesized shape bends and twists in real time to accommodate the changing input specification, and at crucial transition points will snap into different topological configurations.

3D graphical objects called *pedestals* facilitate the positioning and grouping of the design features. These pedestals have an orientable planar top in which the flat design features will be embedded. The bases of the planar pedestals all rest on a *ground plane* and may be dragged with the mouse directly. The flat, planar tops of the pedestals are displayed as a bounding box around the features embedded in it. Handles on its sides may be grabbed to swivel the top on different axes to reach a particular orientation.

The geometry of a design feature is created by choosing a pedestal and then building up a polygonal boundary by mouse clicks that add vertices in the pedestal top. Completed design features may be repositioned within its pedestal top plane by dragging it by an edge (rigid body translation), or reshaped by moving individual vertices. Any edits to a flat design feature are constrained so that the feature is always embedded in the pedestal's orientable top.

Simple geometrical obstacles may be created to reserve 3D space for other parts of an assembly. These obstacles are not embedded in the pedestals, but are edited relative to the ground plane. The editing system deals only with prism-shaped obstacles whose profiles are swept perpendicularly to the ground plane. The prism is considered to extend to infinity in the direction of the ground plane's normal. This restriction on the allowable obstacle shape is part of the editing system and not the synthesis algorithm. Arbitrary obstacles can be passed to the algorithm as long as there is some mechanism for finding a polygonal contour that represents the outermost boundary's intersection with a given plane.

The synthesis algorithm and editing system can resynthesize the solution part at interactive speeds when the number of input features and obstacles is small.

4 ALGORITHM OVERVIEW

Our algorithm is composed of three stages that solve the synthesis problem for bent sheet metal parts from feature-based inputs. Input to the algorithm consists of *design features*. Design features are flat polygonal regions that must be incorporated into the final shape. These represent some important geometry that often forms an interface to other parts. The algorithm also supports negative regions so that slots and holes may be represented.

The algorithm's first stage computes a 3D global *connectivity graph* that indicates which pairs of features should be connected to one another by synthesized, uniform-width spars of material. This connectivity graph is a tree graph that connects the flat regions to one another using line segments to form a minimal connected structure. This idealized skeleton is used to determine which features to connect to each other to minimize the final part's size.

The second stage calculates a *folding plan* that determines how the 3D shape can be folded flat. For each pair of connected features in the connectivity graph we calculate fold lines and angles that will unfold the features flat with respect to each other. At this point the final 3D part is not determined; only the planes that the part will occupy and which pairs of these are *connected*. A connection indicates that the feature in one plane will be connected by material to a feature in the other plane. This intervening material will be synthesized in the final stage of the algorithm.

The goal of the folding plan is to find a plan that folds all of the design features from their 3D positions into a single plane, without overlapping each other. This transformation from 3D to 2D simplifies the final stage of the algorithm. If no such plan can be found, the algorithm backtracks to stage one to make necessary modifications to the connectivity graph.

The folding plan attempts to use the natural intersections between the embedding planes of the flat design features as folding lines to flatten the part. However, many times the natural intersection between neighboring design features is far away from the features themselves. Since our solution restricts the part to its set of planes, a connection between these two features necessitates going to this distant fold line and back just to join two features that are very close to each other in space. For this reason the folding plan calculation may introduce an intermediate bridge plane between the two planes allowing the connection to be much shorter at the expense of an extra fold in the part.

Once a folding plan is calculated, it is used to reduce the 3D problem to a 2D problem by folding the features into a single plane which represents the sheet-metal from which the part will be cut. Working in this 2D space is simpler; for example, we may more easily insure that the part doesn't use material twice. Obstacle information must also be computed for the 2D space by calculating the intersection regions between the obstacles and each plane to be occupied by the final part shape. An obstacle may represent another part or the swept volume of a moving part.

The third stage is the spar synthesis stage which calculates constant-width, flat spars to join each pair of features that the connectivity graph indicates should be connected. These spars have constraints on their shape that arise due to obstacles, topological consistency, fold geometry, and minimal weight. Once these flat spars are calculated, their union forms the connected region that is the flattened part shape. The folding plan parameters are used to cut the flattened shape into pieces that each occupy a single plane and can be folded up into the final shape using the inverse of the rigid body transformations that flattened them (i.e. parameters of the folding plan).

5 STAGE 1: THE CONNECTIVITY GRAPH

The first stage in our three stage algorithm is to calculate a *connectivity graph* which indicates which design features are to be connected to each other. In addition to purely topological information, the connectivity graph includes geometric information that can be viewed as a lower bound on the size of the part - a *skeletal* part. The connectivity graph is the minimal weight structure that connects all of the design features into a single connected structure. Because it is of minimum weight, it is a useful plan to connect the design features and keep the part size small.

The nodes in the connectivity graph represent the flat design features while arcs represent a desired connection between them. In addition, arcs have a line segment associated with them that represents the shortest distance from one design feature boundary to another.

We concentrate our efforts on finding graphs that are *tree graphs*. A *tree* is a connected graph with no cycles. All trees over a given set of N nodes have $N-1$ edges. If we remove any edge from a tree, the graph will be divided into two separate connected components. Thus a tree topology has the minimal number of edges required to join our features into a single connected part. A connected non-tree topology can be constructed by adding edges as a post-processing step. Such a step might be warranted if part of the tree needed additional reinforcing edges for added part strength.

5.1 Minimum Spanning Trees

Among the many possible tree graphs, many will be poor choices for specifying the connectivity of our part. Our choice of graph must be made by considering the geometric relationships of the input planes embodied in the graph. Input features that are considered *close* to each other should be connected to each other.

Our inspiration for the calculation of our connectivity graph comes from the *Minimum Spanning Tree* problem from graph theory and the interesting properties and relationship between the *Euclidean Minimum Spanning Tree (EMST)*, *Voronoi diagram*, and *Delaunay triangulation* from computational geometry. The Minimum Spanning Tree problem is typically formulated as a problem in graph theory, using only the graph's topology and edge weights for its expression:

> **Minimum Spanning Tree (*MST*) problem:** Given a graph G with N nodes and E weighted edges, find the shortest subtree of G that includes every vertex [4].

The *Voronoi diagram* is a geometric construct that partitions the plane to solve the post office location problem. The boundaries of the regions associated with each *post office* are called *Voronoi polygons*, and the post offices themselves are called *Voronoi sites*.

> **Post Office Location Problem:** Given N post offices located in the plane, draw a diagram that partitions the plane into regions that are associated with the post office closest to them [4].

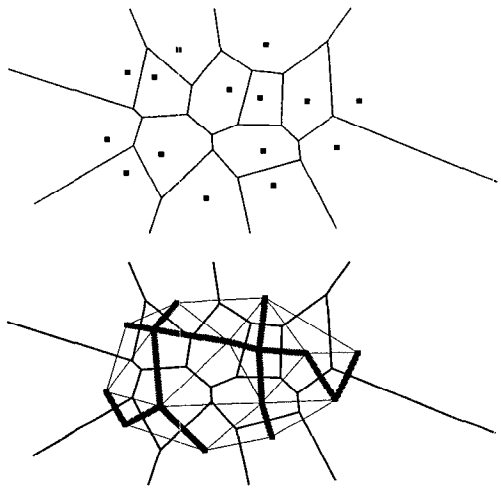
The *Delaunay triangulation* is a dual of the Voronoi diagram, that is, faces correspond to vertices and vice versa in the topological incidence lists for face, vertices and edges of the *Voronoi diagram*. The actual geometric position of the Delaunay vertices is at the *Voronoi sites*. The *EMST* over a set of nodes is a subset of the *Delaunay triangulation's* edges; it can be found most efficiently by taking advantage of this relationship, since the triangulation can be

computed quickly and has far fewer edges than the complete graph over the same nodes [7].

The Voronoi diagram partitions space and in doing so, indicates which sites are appropriate neighbors to be joined by edges in the triangulation. This eliminates overly long edges and edges that cross each other. The 3D version of the Voronoi diagram employs *Voronoi polyhedra* to partition space, and there is a corresponding *3D Delaunay triangulation* and *3D EMST*. The *3D Delaunay triangulation* uses tetrahedrons, rather than triangles, to partition space. Edges chosen between points in the 3D version would not, in the general case, intersect each other, but they may pass close to each other giving rise to a shape that is *tangled up*.

Our algorithm calculates something akin to the 3D EMST. The new construct is a minimum weight graph in the sense that the EMST was, the minimum spanning tree if only edges between points in the input set are allowed. However, rather than joining nodes that are points in space, our algorithm joins nodes that are convex polygons in general 3D position (i.e. the design features).

FIGURE 2. Relationship Between Voronoi Diagram, Delaunay Triangulation and EMST



The top figure is a Voronoi diagram. The bottom figure shows the same Voronoi diagram with the Delaunay triangulation and EMST superimposed. The EMST's edges (thickly shaded) do not intersect each other, except at their endpoints. Note that in the top figure any choice of edge between neighboring sites (those sharing a bounding edge) results in edges that don't cross.

In the quest for an even better connectivity graph, we might consider another type of minimum weight spanning tree -- the *Steiner tree*. Steiner trees are allowed to connect edges at intermediary points that are not part of the input node set, which provides a somewhat superior solution compared to the *EMST* over the same set of nodes. However, the calculation of a Steiner tree is *NP-hard* [5]. For our application, the small improvement is not worth the larger computational cost. Also, the introduction of an intermediate node will add more folds to the shape to gain the corresponding weight advantage.

5.2 A 3D Spanning Tree for Connecting Convex Planar Regions in 3D Position

The inputs to the connectivity calculation are the planar pedestal tops and their embedded design features created by our editing system. For simplicity the algorithm generates a 2D convex hull around each set of planar input features, rather than deal with the complexities of individual, possibly concave shapes. We can still have multiple, discreet features within a single plane by using distinct planar pedestals whose orientable tops are coplanar with each other. These separate pedestals could be interactively bound to each other to form a group, although our editing system doesn't support this.

We first join these convex planar regions in 3D space by straight line segments. We try to find the closest pair of points on the two features' perimeters. We call such line segments the *"shortest, boundary-joining segment"* for a pair of convex, polygonal features. If we calculate such a segment for each pair of features, we obtain a complete graph which can be used as the input to an MST algorithm. The required edge weight inputs for the MST algorithm are the lengths of the shortest, boundary-joining segments. The output of the algorithm is a minimal weight graph connecting all of the features. We call this resulting graph the *Minimum Spanning Tree over Planar Features in 3D*, abbreviated *MST-PF3*. In our implementation we use Prim's algorithm for calculation of the MST [4].

The *MST-PF3* can be seen as an ideal part of sorts. The flat design features could be connected to each other by very thin, rigid wires that correspond to the selected shortest, boundary-joining segments in the solution graph. This represents a sort of minimal skeletal part whose total edge length sum may be thought of as a lower bound on the total path length for real designs. This, of course, neglects any obstacles that potentially may interfere with the segments. Although a lower weight graph may occasionally be obtained by allowing segments to join at the interior of one or more features, this is forbidden for our task. Design features must not have any interfering connections to their functioning surfaces. Therefore we limit our consideration to the perimeters of the convex polygons.

The shortest, boundary-joining segment for two polygons in 3D may be calculated by brute force by finding the closest points on each pair of edges between the two polygons. This can be calculated by noting that the shortest distance from a line to a given point is the length of the line segment from the point to a point on the line forming a perpendicular. Thus the first case will be that the closest points are a pair of points on each edge, not including any vertex. The line segment between these points will be perpendicular to each line. It is possible that the perpendicular line segment's end points are incident to the infinite lines through the edges, but don't hit the finite edges themselves. In this case the vertices must be checked by forming a perpendicular to their opposing segment. Finally, the closest points may in fact be two of the vertices.

When edges or features are parallel there may be more than one shortest, boundary-joining segment for a pair of features. In these degenerate situations it is enough to pick any one of these segments, since at this stage we only care about the distance between features for the purpose of selecting neighbors.

6 STAGE 2: CALCULATING THE FOLDING PLAN

Once the connectivity graph is determined, we must calculate folding parameters to form a valid folding plan. A folding plan consists of steps that take the neighboring features' planes and by one or two suitable unfolding moves, successively flatten out all of the 3D part's bends. The end result of executing the folding plan on the 3D input features is a flattened set of non-overlapping features that could be cut from a single sheet of material. We rely on the spar synthesis stage to then connect these features in the 2D plane.

Each unfolding move consists of a rotation about a fold line, which unfolds a connected design feature relative to a stationary design feature. Each fold line is the intersection of a pair of planes that are either a design feature's embedding plane or an intermediary *bridge plane*. The design feature planes correspond to the orientable tops of the 3D interactive pedestals previously described. The bridge planes are automatically calculated planes which are inserted between two design feature planes when their natural intersection occurs far from the part. Bridge planes form a bridge that allows a synthesized connecting spar to avoid traversing a long distance out to a poor natural intersection at the cost of introducing an extra fold into the design.

Each unfolding move represents a locally valid physical move that could be made by bending a piece of sheet metal stock through creasing along a straight line. The folding plan is not always a valid *assembly plan*. That is, if we execute the folding plan in reverse to fold the shape into 3D position, we may find that distant portions of the part collide with each other while sweeping through space. The folding plan only guarantees that neighboring features won't interfere in this way; we do not solve the global problem. Furthermore, materials may have bending limitations; our application doesn't consider these factors.

6.1 3D Display of the Folding Plan

Providing insight into the calculations that produce the part is at least as important as viewing the part itself. Concerns over what to display influence the choice of algorithm used to perform geometric computations. We find that relying on a constructive geometric approach lends itself toward an intuitive display. We also find that although some geometric calculations are quite simple, consistently displaying something that the user can make sense of in a 3D interactive environment is more challenging.

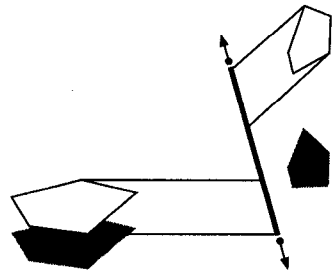
For example, to visualize the folding plan we wanted to provide a view of the two flat features and the place where they would most likely come together. The planes and their intersection are infinite entities. These are particularly challenging to display in a finite viewing volume, since they behave counterintuitively when the view is rotated and they are clipped at constantly changing places.

Our goal was to present a compact and suitably abstracted visualization of the 3D region of space in which a spar would most likely be generated. We chose to project the extents of the polygonal convex hulls around the input features onto the intersection line, and clip the intersection line at the extreme points of these projections. We added small arrows to each end to indicate that the entity is in fact infinite. To further flesh out the shape, we used drop lines that were perpendicular to the intersection line from the extents of the polygonal features. The boxes created by these perpendicular drops give the user a good idea of where the planes are and where they intersect. Most of the display information is not required for the synthesis algorithm to correctly function, however, we found such information indispensable for debugging.

This 3D visualization, when combined with the connectivity graph to show only those intersections that are relevant (those which will be connected), provides a good idea of what the finished part will look like, provided all of the natural intersections between connected features fall in reasonable locations. In general, there will be several of these intersections that are, for practical purposes, at infinity (parallel planes). We will resolve this issue in section 5.3 dealing with bridge planes.

Internally we represent the geometric entities (planes, lines) in the *point and unit vector form*. Typically, determining a good point is the hardest part since there is such a small range of points that will fit in the display. We remedy this by using the vertices of the feature polygons. If we make sure all of our calculations give points that are judiciously chosen with respect to these points, we get good results. For example, in figure 3, we could choose the midpoint of the displayed line segment (the alias for the infinite line) as the midpoint between the projection of the extreme polygonal points to the intersection line.

FIGURE 3. Representation of the Intersection of Two Planes



The above scene shows the intersection of 2 planes. If this natural intersection is deemed fit, the folding plan will use it to fold one of the features away from the other, thus flattening this section.

The calculation of these geometric entities is required for the folding plan. However, the calculations are fairly routine vector math and are thus omitted; more details can be found in [2].

6.2 Unfolding/Folding Calculation

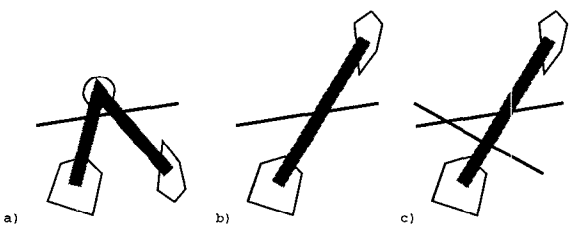
Once an intersection has been determined, we must calculate the parameters for the unfolding operation. These parameters describe a rotation in space about an axis that is the intersection of two planes. Thus we need an oriented axis of rotation and a CCW rotation about that axis which will bring a moving neighbor plane into coincidence with a stationary plane. We call this angle the *unfolding angle* about a folding line. These parameters must in turn be used to calculate modeling transformations that will adjust the local coordinate systems of our features, to both display the unfolded 2D part, and the actual synthesized 3D part. The inverse transformation is used to take a portion of the synthesized part and fold it up into its position in the 3D shape. This helps both to visualize the part and to verify that the folding plan is valid.

Each set of features is specified in its own local coordinate system, which is given by a transformation that takes points in the local coordinate system to the world coordinate system. The feature exists in the x-y plane of the local coordinate system.

There are two sets of parameters that will accomplish this rotation transformation for any two planes. The difference in these sets of parameters are that their rotation angles are in opposite directions and are complementary in their magnitude. The correct parameter set to choose depends on the position of the polygonal features with respect to the intersection line. The way we choose to unfold these features is *away from each other* so that the fold lies between the two features when they lie unfolded flat in the single plane. Figure 4 shows why folding away simplifies calculations. Analysis of the topological consistency requirements shows that the spar must at some point cross the intersection line to move from one plane to another. If the fold line is between the features, we can be assured that any path between the features crosses the fold line at some point; this simplifies the next stage of the algorithm, the spar synthesis stage.

To calculate the unfolding angle, we first assume that the features are not intersected by the intersection line; non-intersection of design features is a requirement for a valid folding plan, and our synthesis algorithm detects such problems and removes them. Given this assumption, we can use two perpendicular vectors, dropped from a vertex of each feature polygon to the intersection line to calculate the unfolding angle. Figure 3 shows perpendiculars that could be used for this calculation.

FIGURE 4. Fold Line Must Fall Between Features



a) Two features whose fold line does not fall between them; the circle represents an "intermediary feature" that would have to be placed for our path planning routine to work. b) The two features folded away in opposite directions, leading to a single crossing of the fold line. c) Two fold lines generated by the insertion of a "bridge plane."

Neighboring features that employ a bridge plane will have two sets of folding parameters, one for each folding line. In this case, we must make sure that our two sets of folding parameters fold away from the bridge plane. In order to use the previous method of finding the folding angle, we require a representative point that lies in the bridge plane and is on the *other side* of each of the folds with respect to each of the two features. From this point we can find perpendicular drops as we did before and represent unambiguously the folding away action. To do this we first find the perpendicular drop to each of the intersection lines (as before). We take the midpoint between these two points to use as the point on the other side of each of the folds. From this midpoint we can once again form perpendicular drops to each of the intersection lines. In this calculation, we must handle the degenerate case of our midpoint falling on the intersection of the two intersection lines. Such a midpoint would generate zero length vectors when projected onto either intersection line. This case can be easily detected and corrected by using another vertex on either feature polygon.

Topological consistency for joining features by a bridge plane, requires that the spar cross the two fold lines in the proper order. That is, in travelling from feature A to feature B, we must first cross the intersection between the plane of feature A and the bridge plane, followed by that of the bridge plane and feature B. Section 6.2 indicates how the algorithm ensures topological consistency between features that employ a bridge plane for their connection.

6.3 Bridge Plane

The folding plan first attempts to use the natural intersection between planes, since adding a bridge plane is a more complicated operation. During the calculation of the folding plan, the natural intersection is examined to determine its fitness. If it fails on any of a number of criteria, a bridge plane is inserted to circumvent use of the natural intersection. One problem that arises often is that the intersection between two planes may be located far away from the actual polygonal features to be joined. This happens whenever two nearly parallel planes' features are close to each other in space. Another problem that may occur is a fold line intersecting a feature. A particularly important problem we must detect and resolve is the collision of features in the material plane. Feature collisions occur when a folding plan is executed and the features unfold onto the material plane and overlap one another. This would correspond to material from the sheet being used more than once, for different features.

Upon detection of any of these problems, we intercede by choosing intermediate planes that *bridge* the two feature planes and resolve the problems caused by using the natural intersection. We consider only a single bridge plane between any two features. Such a bridge plane induces two fold lines, one at each intersection with each of the feature planes.

There are two separate issues concerning the use of a bridge plane. The first issue is deciding when to use a bridge plane instead of the natural intersection. The second issue is calculating the parameters for a particular bridge plane. Solving the second problem is much harder since the bridge plane must satisfy many different types of constraints.

Let us first explore the issue of when to use a bridge plane. Many natural intersections may occur *too far away* from the features. That is, the straight line distance through space between the features is much shorter than the associated shortest distance through the folded space of the two planes.

To form a shorter path through space between the features, it is possible to position a bridge plane so that it contains the shortest, boundary-joining segment. One way to measure the potential gain from this insertion of a bridge plane is to look at the unfolding angle between the feature planes. The greater the unfolding angle, the more the potential gain by inserting a folding plane to bridge the gap. There is an inherent trade-off between two very different, desirable qualities for our shape. The first is that the shape should conserve overall spar length to keep the shape from growing too large. The second is that the shape should have as few bends as

possible. Introducing a bridge plane will reduce spar length at the cost of an extra fold in the material. Our empirical results show that a folding angle of 150 degrees is a reasonable threshold above which a bridge plane should be inserted.

Determining whether the natural intersection cuts through one of our polygonal features is also fairly simple. This can be discovered by examining the signed distances, to the other plane, of all vertices of the feature. If any of the signs are different, the plane cuts the feature. If a fold crosses a feature, we can't be sure that a spar crosses the fold to reach the feature, so this natural intersection will not be used.

Another problem that indicates that a bridge plane is called for is that the unfolded feature collides with other features in the 2D material plane. Our solution for resolving this is given in section 6.6.

6.4 Bridge Plane Parameter Calculations

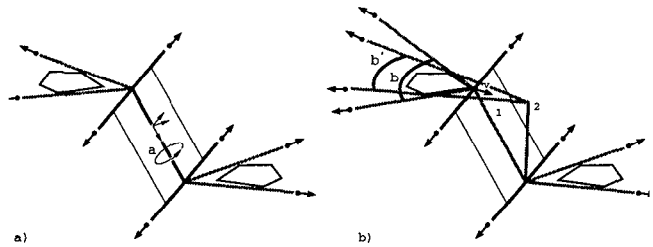
Calculating a well positioned bridge plane is difficult, because there are several different types of criteria a bridge plane must meet. The following is a list of desirable properties for the bridge plane:

1. It can't intersect the features, and there should be some space between the bridge plane and each of the features.
2. In the material plane, the fold lines should fall between the features.
3. The bridge plane should allow short spars; i.e., the (infinite) fold lines must pass reasonably close to the features.
4. When unfolded, the feature must not hit other features. Material can't be used twice.
5. The bridge plane should make it easier to generate spars whose fold lines do not fall lengthwise on the spar.
6. If a bridge plane's fold lines intersect nearby to the spar's location, we must maintain topological consistency. That is, we must cross over the fold lines in an order that corresponds to the order they would be crossed in 3D.

The shortest, boundary-joining segments calculated earlier provide a lower bound on the spar length. If the bridge plane contains this segment, and the path is free of obstacles, the spar synthesis stage will generate a spar that is close to the optimally shortest path between the two features. The set of all planes that include this line segment can be described by using the segment as an axis about which the parameterized plane rotates. A *twist* angle fixes the parameterized plane at a particular orientation.

However, there is a competing concern that prevents us from using the shortest, boundary-joining segment as the actual axis. Placing the fold line coincident with a vertex of the feature polygon may cause the corresponding physical feature to be distorted when a real fold is made to the physical material. To avoid this, we must move the end points of the closest approach segments away from each of the feature polygons. A bridge plane that is *anchored* very close to both polygons may in fact, leave no legal values for the twist angle.

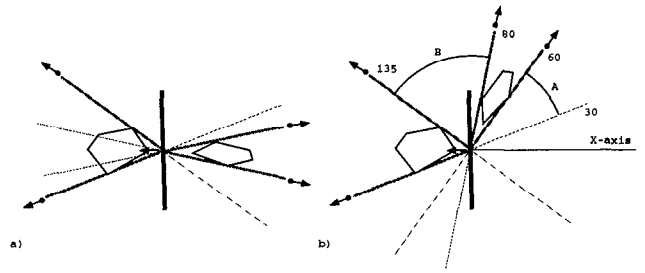
FIGURE 5. Bridge Plane Parameterization



a) The parallelogram represents the bridge plane at a particular twist angle. The thick lines are intersection lines between the planes. The grey lines show the extreme positions of the intersection the bridge plane may make without intersecting the features. b) Moving the endpoints of the closest segment away from the polygons is necessary to provide a greater range of legal positions for the bridge plane.

To remedy this we can compute a better twist axis by taking the shortest, boundary joining segment, and moving each of its endpoints away from the polygon some small amount. We choose to do this by keeping the endpoint in the plane, and moving out in the direction of a vector, that is the average of vectors in the direction of the incident sides of the polygon. Figure 5 a) shows one of the twist axis' endpoint being pulled away from its incident feature. The diagram shows the twist axis in two different positions (1 and 2). The further the point is pulled out, the smaller the angle β is and the greater the range of valid values for the twist angle α. The marginal increase in angle mobility declines rapidly as we move away from the feature, so we avoid picking too small an angle for β. We have found, empirically, that 60 degrees is a reasonable measure for β.

FIGURE 6. Analyzing Twist Ranges



Bridge plane between projected feature set; the thick black line segment with the tiny orthogonal arrow shows the position of the bridge plane, edge on. The grey lines are the projection of the extreme legal positions of the bridge plane as previously described. Each projected range is rotated through 180 degrees to see if it overlaps other ranges. Overlapped regions are joined to find the illegal ranges. The complement of these illegal ranges accommodates legal positions of the bridge plane. In a) there is a single legal angle range. In b) neither the legal ranges nor their complements overlap each other, giving rise to two disjoint legal ranges (A and B). The plane is shown legally positioned within the B range. The X-axis is the zero degree reference.

The actual measurement of β depends on the neighboring sides of the convex hull over the feature polygon's vertices and the axis endpoint. To calculate β we use the technique of bisection starting from an initial interval given by the endpoint at some distance close to zero, and a distance equal to the diagonal of a bounding box around the feature.

Once the twist axis has been determined, the legal ranges for the twist angle α must be calculated. These are the ranges for which the bridge plane does not intersect either polygonal feature. To perform this calculation, we can visualize taking the geometric entities indicated in figure 5 and performing a rigid body rotation on them so that the twist axis is aligned with the y-axis in the new coordinate system. The geometric entities we are interested in are the extreme *intersection rays* attached to the twist axis colored in grey. If we project these rays down into the x-z plane of our new coordinate system and superimpose an edge on view of a plane, we can easily discern the allowable angles for the plane. Figure 5 a) and b) show configurations that give rise to 1 and 2 legal ranges.

This method of examining the projection has degeneracies whenever the twist axis is coincident with any of the intersection rays. In this case the projection of the ray will be a point at the origin. Since the endpoint of the twist axis is chosen to be in the plane, the twist axis is not only parallel, but lies in the particular plane. Such a placement for the bridge plane is problematic since the bridge plane will be coincident with one of the planes for some angle. It is a simple matter to resolve such problems by moving the twist axis' endpoints some small amount in a direction in each plane.

6.5 Bridge Plane Adjustment

The final task is to pick a twist angle in the legal calculated ranges for the bridge plane that does not lead to feature collisions and which enhances our chances of getting a spar that crosses the fold at a 90 degree angle. Our parameterization for the plane already insures that most of the other constraints for picking a good bridge plane will be satisfied. Our approach is to use a numerical method to minimize a function that represents how poorly the fold positions are with respect to a straight spar between the features. We call this function the *fold alignment function*. Once a twist angle is found that minimizes the fold alignment function, we start from this angle and deviate, moving the feature until it no longer collides with any already placed features.

We don't know where the spar will actually cross the fold line, since the spar hasn't been synthesized, however, we need some good estimate as an input to the fold alignment function. Since the spar synthesis stage finds the shortest path between the features, it will tend to generate paths that are a straight line between the features. As an approximation to the actual path, we use the line segment joining the centroids of the features in the material plane. Using this line segment and the two fold lines, we compute the sum of the squares of the dot products between each pair of direction vectors from these lines. A 90-degree crossing will give a dot product of zero between the segment direction and a fold line. Minimizing the sum of the squares of these dot products gives a reasonable compromise between the two angles. Note that our input argument (twist angle) has a limited range, and may be split into two intervals. Brent's method [8] for finding minima (in one dimension) is perfect for our task since we don't have a derivative for this function, only an interval.

Rather than describing the unfolding transformations in all their detail, we provide instead some intuition that is helpful in analysis of these transformations. For this analysis, we will start from Figure 5a) and work in the coordinate system of the bridge plane. Consider an unfolding move applied to either feature, rotating it about the intersection line, until it is flat with the bridge plane coordinate system. A transformation that moves features in their local coordinate system, to this position w.r.t. the bridge plane, is a 2D rigid body rotation about some point in the bridge plane system. Thus, the transformation is a 2D rotation about this point. Thus, from the standpoint of one of the feature planes, the unfolding transformation reduces to two successive 2D rotations about two different points in the plane.

Thus the fold alignment function takes a single input parameter (twist angle), finds the product of two 2D rotations which are applied to the various geometric entities (intersection line directions and line between centroids) and which are also used to calculate the sum of the squares of the dot products.

6.6 Adjusting the Bridge Plane to Resolve Collisions

Once the *optimal* bridge plane twist angle is calculated, a check is made to make sure this position does not cause any feature collisions. The algorithm fixes features in a depth-first order through the connectivity graph, and thus only considers those features that are already fixed for purposes of collision detection.

If the feature does collide with another feature, the twist angle must be varied, moving the feature to the left or right until it doesn't collide any more. To find the best position, both directions are searched and the fold alignment function evaluated to compare the two positions. We search separately for the first optimum to the left and to the right.

To move one feature off of another, the algorithm resorts to another heuristic search. Varying the twist angle from its optimal value will move the feature, but we need some way to detect when we are *off of* another feature, and the function should be well-behaved.

The function we choose is the length of the shortest, boundary-joining segment calculated for the two interfering polygons in the plane. Calculation of this segment can be done as described in section 4.3, or we can take advantage of the fact that the polygons are now in the plane, and use the optimal method described in [3]. The length of this segment is used as the distance between the two polygons, unless their interior regions touch, in which case the distance is zero. We use the technique of bisection to find a value for the twist angle that produces a particular positive value for the distance function. We can use this to provide some amount of distance between the convex polygonal features. This procedure must be repeated if the new twist angle value produces a different collision.

In searching for a particular value for the distance function, we must first find an interval in which the values for the function at the interval's extreme values bracket the desired function value. Since the distance function is continuous, bracketing and bisection can be used to find the value.

6.7 Changing the Connectivity Graph to Resolve Feature Collisions

The goal of the folding plan is to find a set of planes and connections between planes that allow the design features to be folded flat without collisions. We have already discussed operations that eliminate local collisions, that is, collisions between connected features. We must now turn a sequence of these steps into a folding plan that avoids global collisions as well.

To do this we pick a starting node in the connectivity graph and begin making local unfolding calculations with the neighboring design features in depth-first order. If a collision is discovered while working on a particular node in the graph, i.e., unfolding the corresponding feature plane, an attempt is first made to handle this problem geometrically by searching for an appropriate twist angle. If no good angle can be found, the algorithm back-tracks up along the connected branch and tries to resolve the problem by edge swapping at the next ancestor node in the tree. After an edge swap, the algorithm starts down the branch once more.

A given node could in principle be connected to any one of all the other nodes in the tree; thus there are *N-1* possible edge swaps that one might want to explore. Such a set is known in graph theory as a *cut set*, and can be computed efficiently by a simple application of depth first search [4]. However, most of the *N-1* candidate replacement edges will likely be poor choices; it is at this stage that a 3D Voronoi diagram would help in the selection of alternate edges. Such a construction would guide the selection of the next best possible edge that should be evaluated. It would also indicate the number of neighboring features; a number usually far less than *N-1*. However, the calculation of the *MST-PF3* graph didn't use a Voronoi construction, and it is not worth building such a structure just for the problem at hand. Our algorithm thus ranks all potential edges to other nodes by length and tries them out from shortest to longest, thus strongly favoring neighboring sites. There is no point in trying out all possible edges, because the long ones would lead quite obviously to bad folding structures. Our experience has shown, that it is worth trying only the three shortest edges, before referring an unsolved folding problem to the next higher node in the tree.

7 STAGE 3: SPAR SYNTHESIS

Once a folding plan has been calculated, all design feature polygons are unfolded flat onto the 2D material plane. The folding plan specifies which of these features are neighbors that must be connected. The actual joining of two features is accomplished by synthesizing constant width spars that join pairs of neighboring features. These spars cannot simply be straight-line connections; they must avoid the unfolded features lying in the material plane, and they must also avoid 3D obstacles when the part is folded up into its final 3D shape.

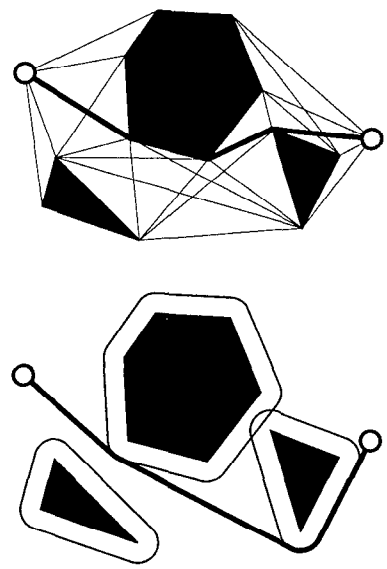
7.1 Connecting Two Features Using Path Planning

Connecting neighboring features using a spar is approached as a 2D path planning problem as used in [1]. Much work on path planning originates in the field of robotics [6]. In the simplest type of path planning problem, a zero dimensional *point robot* needs to find a way through a maze of polygonal obstacles in a plane from a starting point to a goal position.

If the robot's paths are limited to those describable by polylines, then this problem may be solved by performing a graph search of a vertex-connected *visibility graph*. The visibility graph's nodes are the vertices of the obstacle polygons and the start and goal points. All pairs of vertices visible from each other (visibility line), including the edges of the obstacles' polygonal perimeters themselves, are represented by arcs in the graph. These arcs are bidirectional, i.e., vertex A visible from B implies B visible from A (see Fig. 7a).

Computing the shortest path from the start to the goal in the visibility graph does, in fact give us the optimal shortest path from the planning standpoint (Fig. 7a). Thus a path can't be made shorter by taking a path from the middle of an edge, to another point. Nor can a path be made shorter by being curved rather than straight. The shortest path may be computed on a graph using *Dijkstra's* algorithm [4]. However, the 2D polyline path generated by Dijkstra's algorithm can't, in general, be adjusted to form a thickened spar, since such a spar may not be able to squeeze through a narrow gap between obstacles as shown in Figure 7.

FIGURE 7. Visibility Graph



top) *Visibility graph made by connecting all mutually visible vertices with an edge. The bold line shows the shortest path between the start and goal.* ***bottom)*** *The same start, goal and obstacle set. The obstacle boundaries have been offset to create clearance for a uniform width spar. The shortest path is routed differently since there is not enough clearance to squeeze between two of the obstacles.*

In order to solve the problem of finding a path of finite, uniform width, we use a geometric construction known as *Configuration Space* or *C-Space* [6]. All input geometry, i.e., the feature polygons as well as the obstacles, are grown by an offset equal to half the desired spar width. Then we form the visibility graph over

this modified geometry and find the shortest path between start and goal points. Since this path maintains a clearance from the original polygons equal to the chosen offset distance, a spar of twice this width may be routed along this path.

The widths for the various spars is determined by the following, scale-independent heuristic. For any pair of features to be connected, we use the area of the smaller feature as the initial spar width. Thus, if all features were perfectly round, the width of a spar would be the diameter of the smaller circular features. Our system also allows to scale and bias all spar widths by adjustable global constants. The bias can be used to add a small tolerance for object clearance.

7.2 Obstacles

Our application of the visibility graph requires that we make a correspondence between the 3D space in which the final part has to reside and the 2D material plane. One important constraint is that the uniform width spars must avoid all the specified 3D obstacles. In the process of our folding calculations, we compute the relevant 2D intersection contours between the specified 3D obstacles and the given feature planes as well as any new bridge planes introduced. Only those 2D obstacle regions occurring on the same side of any relevant folding line as the original feature itself need to be considered. The relevant 2D intersection contours are then unfolded into the 2D material plane, are clipped to the appropriate fold lines, and merged into joined *obstacle slices* where they overlap one another. These obstacle slices are wedded to a particular arc of the folding plan graph, i.e., they are only relevant for the synthesis of spar geometry in the particular half-plane on the feature-side of the folding line.

When synthesizing different spars, different obstacle slices will have to be considered. Thus, there is no single visibility graph over the material plane; rather, each spar has its own set of obstacle slices, calculates its own offset by half the width of the spar, and then computes its own visibility graph to find the shortest path. The geometry of that spar will of course strongly depend on the way that the features have been unfolded, on the bridge planes chosen, and on the other spars that have been synthesized first.

There are additional constraints on the way a connector path for a desired spar may be routed. As discussed earlier, spars must properly cross their relevant folding lines, and, in the case of bridge planes, must cross the two folding lines in the proper order consistent with the desired 3D shape. In some cases, the intersection point of the two feature planes and of the bridge plane may occur close to path of the spar. To avoid constructing a path that crosses the two folding lines in the wrong order, we add an extra obstacle in the form of an unbounded, semi-infinite triangle that covers the forbidden 2D region between the fold lines; this will insure that the spar crosses on the correct side of their intersection point.

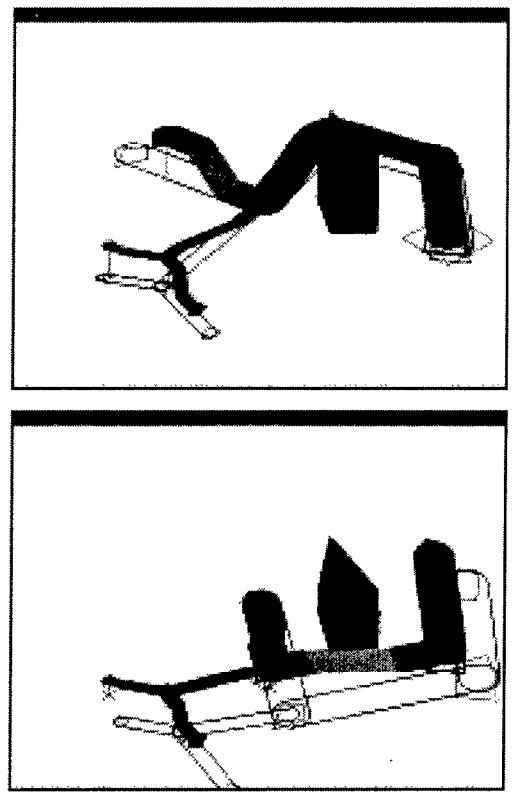
Another constraint on spars is that they must not cross each other in the 2D material plane unless they are part of the same physical feature plane. A synthesized spar is an obstacle in *every* subsequently induced visibility graph. Once material has been allotted to a spar in one feature plane, it can no longer serve another spar in a different folding plane. However, we may overlap spars in regions that belong to the same physical 3D plane. Thus, we don't add the whole spar as an obstacle, only the portion that inhabits other planes. The intersection line is used to clip off the ends of the spar before it is added as an obstacle.

Finally, each synthesized spar will itself become a 3D object which must be folded up and intersected with each of the planes -- just like a "normal" obstacle -- and then will induce new obstacle slices. The intersection of the spar with a plane will generate a very thin rectangle that has the width of the sheet metal stock thickness divided by the sine of the intersection angle.

7.3 Output

The algorithm outputs uniform width spars in the material plane that show the single 2D outline that makes up the flat shape boundary. Each 2D flat spar is split into pieces two or three pieces, depending on the number of planes it will span. These spar pieces are transformed by an inverse folding transformation that folds them up into the actual 3D part shape.

FIGURE 8.



Two screenshots of the demonstration system. The polyhedral prism is an obstacle. The polygonal outlines are features. The flat outline shows where the part will lie if folded flat. The bottom screenshot shows the part in a new configuration, the result of moving one of the features.

8 PROBLEMS WITH THE ALGORITHM

There are several problems with the algorithm. The first problem is that the bridge plane is calculated without any notion of intervening objects. If an object is positioned so that it blocks the shortest, boundary-joining segment, the spar will be forced to bend around the object close to where it must join the feature. This creates a noticeable imperfection in the part. It is possible to factor in consideration of the obstacles which will work in simple situations. However, the reader will note that our synthesis algorithm never did any 3D path planning, but merely picked planes assuming they were wise choices in absence of obstacles and relied on the spars to find a good path. In a very restricted situation, a new approach may be required.

Another problem that is simpler to remedy is the fact that the parts look somewhat skeletal and unfinished. It was our intent to add a post-processing step that would work in the 2D material plane to add material in a pleasing way that encompassed the 2D skeletal pattern. In order to do this, one must be careful about the placement of the bridge planes that are incident to a single node. The fold lines created by these incoming bridge planes constrain how material can be added, and in fact, how good the part actually looks. Since there is so much freedom in choosing the bridge plane, it is likely that our algorithm can be updated to include such considerations.

9 CONCLUSION

We have presented a multi-stage heuristic algorithm that automatically synthesizes flat, bent parts from a terse set of high-level descriptions of input features and obstacles. The algorithm produces reasonable parts at interactive speeds, that can be tuned in appearance somewhat with a few global parameters, such as a multiplier on the spar widths. A strength of the system -- which probably never can be fully automated -- is that it allows easy user intervention through an interactive editor system to detect and easily correct poor solutions resulting from tricky situations.

The algorithm could be used in conjunction with current sheet metal modeling packages to provide more automation and live designs that can adapt to introduced obstacles. Also, although the algorithm as a whole is somewhat specialized, certain aspects of it may be applied to the general 3D problem. In particular, we see the connectivity graph's skeletal representation as a useful abstraction for the general 3D problem.

References

[1] Burgett, S. R., Bush, R. T., Sastry, S. S. and Sequin, C. H., Shape Synthesis from Sparse, Feature-Based Input, in *Concurrent Product and Process Engineering*, ASME, San Francisco (November 1995). pp. 307-318.

[2] Bush, R. T., Masters Thesis, University of California, Berkeley, CA (December 1998).

[3] Chin, F. and Wang, C. A., "Optimal Algorithms for the Intersection and the Minimum Distance Problems Between Planar Polygons," *IEEE Transactions on Computing* C-32(12), pp. 1203-1207 (1983).

[4] Cormen, T. H., Leiserson, C. E. and Rivest, R. L., *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1985. pp. 477, 489, 498, 509, 527.

[5] M. R. Garey, R. L. Graham and D. S. Johnson, "Some NP-Complete Geometric Problems," *Eighth Annual Symposium on Theory of Computing*, pp. 10-22 (May 1976).

[6] Latombe, J. C. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, 1991.

[7] Preparata, P. P. and Shamos, M. I., *Computational Geometry an Introduction*, Springer-Verlag, New York, NY, 1985, p. 230.

[8] Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P., *Numerical Recipes in C*, Second Edition, Cambridge University Press, Cambridge, U.K., 1998. p. 395.

[9] Shah, J. J., "Assessment of Features Technology," *Computer Aided Design*, 23, 5 (June 1991), 331-343.