

Creating solid models from single 2D sketches

I. J. Grimstead¹ and R. R. Martin¹
University of Wales College of Cardiff

ABSTRACT

We describe a method of constructing a B-rep solid model from a single hidden-line removed sketch view of a 3D object. The main steps of our approach are as follows. The sketch is first tidied in 2D (to remove digitisation errors). Line labelling is used to deduce the initial topology of the object and to locate hidden faces. Constraints are then produced from the line labelling and features in the drawing (such as probable symmetry) involving the unknown face coefficients and point depths. A least squares solution is found to the linear system and any grossly incompatible equations are rejected. Vertices are recalculated as the intersections of the faces to ensure we have a reconstructible solid. Any incomplete faces are then completed as far as possible from neighbouring faces, producing a solid model from the initial sketch, if successful. The current software works for polyhedral objects with trihedral vertices.

CR Descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling - *Geometric algorithms, languages and systems*; I.2.10 [Artificial Intelligence]: Vision and Scene Understanding - *Perceptual reasoning*; I.3.6 [Computer Graphics]: Methodology and Techniques - *Interaction techniques*; J.6 [Computer Applications]: CAE - *CAD*.

1 INTRODUCTION

We present in this section an overview of the problem that we wish to solve, and an outline of our solution to the problem.

1.1 Problem Statement

In computer aided design there is a need to transfer 3D objects from the minds of designers into modelling systems. Current systems are extremely slow and unwieldy, frustrating designers and interrupting the cognitive process; what is needed is a quick, natural method to input 3D shapes into a CAD system.

Ullman et al. [25] showed that current systems, using a tablet with rubber banding or icons and menus, are detrimental to the de-

sign process; users are continually interrupted from drawing by the need to change their focus of attention from drawing to menu selection, breaking concentration. The importance of sketching in design is covered in Jenkins [14], where current input methods are reviewed and found still not to be natural to the user; he states that current CAD systems do not take into account the special nature of sketching during the conceptual stage of design. This is the most impulsive stage of design, where users want to create drawings as fast as their inspiration; this is definitely not possible with current CAD systems, so designers are held back by the user interface to the computer system.

A system is required whereby sketches may be input quickly and freely with a natural “pencil and paper” style interface, thus moving the machine closer to the designer. Human operators can produce sketches of a 3D object quickly and easily, which we wish a CAD environment to be able to understand directly in the form of a solid model, which may then be further processed. This involves a CAD tool that can interactively input a 2D line drawing of a 3D object drawn using a pen and tablet, and interpret it as a 3D object in real time as the designer sketches, effectively replacing the pencil and paper.

We will take a single, orthographic view of a trihedral object with hidden lines removed and without any holes or connected parts that could be considered to be independent objects (we discuss this in the next section). Our program will use artifacts in the drawing to recover the missing depth information and make simple assumptions about the structure of the hidden parts of the drawing, producing a solid model complete with the hidden lines.

1.2 System Principles

We will use the term *junction* to refer to a point in 2D, *line* to refer to a line connecting two junctions in 2D, *region* to define an area of the drawing, *region boundary* to refer to the loop of lines and junctions bounding a region. We use the term *vertex* to refer to a point in 3D, *edge* to refer to a line connecting two vertices in 3D, *face* to refer to a 3D surface of the object and *face boundary* to refer to the loop of edges and vertices bounding a face. A block diagram of the following system is presented in Figure 1.

The initial stage of our system takes the user’s sketch from the digitising tablet, and converts it into a series of straight lines. The lines in the drawing are then analysed, and 2D tidying is carried out; for instance, lines are connected together, tested for parallelism and so on. We assume that the user is sketching an opaque, trihedral, planar solid; the opacity making the sketching easier for the user. The trihedral assumption makes the reconstruction problem less general than a less constrained planar object, assisting us in the generation of a solid model. The planar assumption is again for

¹Department of Computer Science, University of Wales College of Cardiff, PO Box 916, Cardiff CF2 4YN, U.K.

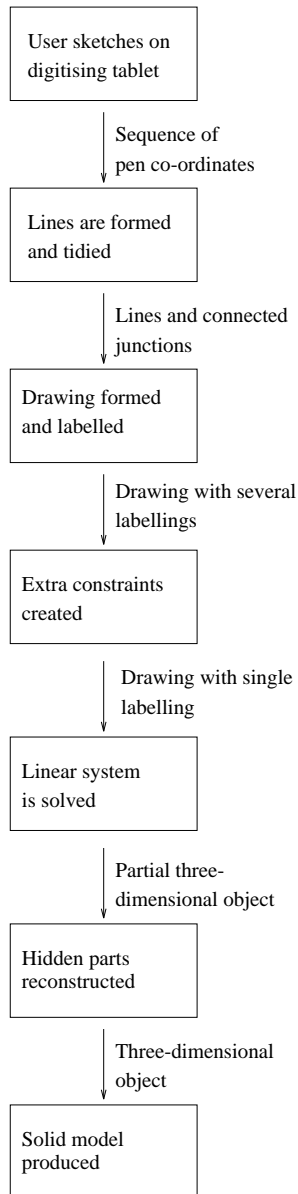


Figure 1: Block diagram of the overall system

simplicity; the assumption that the object has no holes or connected parts reduces the problem of multiple solutions to the line labelling problem. These are not inherent limitations of our approach, but were imposed to keep the initial system simple. This issue is discussed further in Section 5.

Once the drawing is tidied and complete, 3D information is derived from it. Initially the drawing is line labelled with the Huffman-Clowes labelling, and the drawing is divided into regions (which are the forerunners of 3D faces), giving the basic topology of the visible part of the object. Note that if several consistent labellings are possible, then the interpretation system records all labellings and selects an arbitrary labelling to work with. The drawing is then converted into a precursor solid model, with the depths of vertices and face equations still unknown. The line labels are examined, and any hidden faces that intersect visible faces in the drawing are added to the precursor solid model as incomplete faces.

From the precursor solid model, a linear system is produced that constrains the vertices as lying on the intersections of their parent faces. Further constraints are then produced from artifacts in the drawing (such as skewed symmetry and parallelism), and are added to the linear system. A least squares solution is then found giving the unknown geometry of the object (grossly incompatible constraints are detected and rejected). Positions of vertices are refound by intersecting faces to ensure we have a reconstructible object.

The incomplete hidden faces are then completed as far as possible, producing the rest of the topology of the shape, and also its geometry, and the final solid model is produced where possible. A new linear system is generated from this solid model, and a least squares fit found, with extra constraints relating to the newly created hidden faces. The produced solid model is presented to the user for verification and further editing. If multiple solutions exist to the line labelling problem, then the user is prompted to verify if the produced solid is correct. If it is not, then the system selects the next possible labelling and generates a solid model from it in turn.

2 RELATED WORK

In this section we present an overview of work by other researchers attempting to solve similar problems to the one outlined, and of other workers techniques which we use. Some of this work comes from the computer vision community, being aimed at reconstructing objects from line drawings extracted from images, rather than sketched by a user. These are different problems, as images of physical objects contain extra information that can be used to reconstruct the object (such as shading).

2.1 Similar Work

Many researchers have addressed the problem of inputting a 3D object from 2D data; the creation of 3D models from several orthographic views (usually scanned in from an original technical drawing) is a popular area, due to the amount of drawings that users would like to convert into computer models. Orthographic views in Chen [3], Dutta [5] and Lequette [19] are converted to a solid model, with Kobori [17] working with a wireframe model to produce a solid model. These papers aim to produce a solid model given a complete drawing of the target object which contains depth information, and concentrate on matching vertices between views (as each vertex appears once in each view), or just producing face information from a bare line and vertex model. An unusual but related approach is by Suffell [22], where the designer draws two views of the object and the program combines these via stereo to form the required solid model. However, we do not wish for the user to have to draw several views to input a quick sketch and our aim is single view reconstruction.

The approach we wish to use is to convert a single view into a solid model, such as the work by Leclerc [18] and Marill [20].

These methods accept a complete wireframe drawing as input, and use an optimisation approach by gradually disturbing the depth of each vertex from an initially flat drawing to a 3D wireframe, minimising the standard deviation of angles between connected lines. Being based on an optimisation scheme, these methods require a complete wireframe picture of the whole object and do not take into account drawing errors, nor do they attempt to tidy the drawing. Our system accepts a hidden line removed drawing, tidies it and produces a complete solid model as output rather than a wireframe.

The research of Kanade et al. [15, 16] concentrates on the usage of gradient space and skewed symmetry to infer the missing depth information; drawings and real images are used, with the final output being a solid model. Our method also uses skewed symmetry, except that we use additional features in the drawing to estimate the gradient of parts of the image. Kanade works directly with digitised images of real objects, and hence does not investigate the problems of maldrawn objects.

The direct input of depth whilst creating the object is investigated by Bier [2], Forrest [6], Fukui [8] and Pugh [21]. The solutions presented rely on lines being snapped into the same depth as previous lines whilst they are being drawn, or use a certain orientation of the input device (usually a mouse) to infer movement in depth. These systems produce solid models directly, but not in a way natural to the user.

Perspective sketching is another method of entering an object (the above methods rely on isometric projection); this type of projection is difficult to sketch, but contains much more information than an isometric drawing. Hale [9] and Ulupinar [26] both use the extra semantics from perspective input; Hale lets the user sketch directly in 3D by drawing cross-sections of the object (being aimed at the car design industry). Ulupinar derives depth from skewed symmetry in much the same way as Kanade [15], but uses vanishing points to help determine the surface gradients. Perspective projection is a useful medium, especially with real images, but is too error prone to be used for quick sketching by hand (as a slight difference in line orientation changes the depth interpretation of the drawing).

Expert systems have also been used by Hwang [11] and Iwata [12], allowing the user to sketch their required object and have it interpreted as a solid model. The restriction with the use of an expert system, however, is that every object that is to be recognised must be present in the knowledge database. In many cases, this is simply too large a problem to consider; a designer may be creating a one-off object that does not consist of off-the-shelf parts in the database, so it would not be recognised.

Moving on to the solving of constraints produced by examining the drawing, Sugihara [23, 24] has worked with linear systems to produce a solid model from single view images from which edges have been extracted. He uses linear equations to constrain the vertices to lie on intersections of faces, with the depths of vertices and gradients of faces being the unknowns in the system. The light intensity data from a real image is then used with a minimisation system to produce a solution to the linear system. In our case, we do not have access to the lighting of an object, so we instead use other constraints to provide us with depth information (skewed symmetry, for example).

2.2 Review of Techniques

In this section we give a brief outline of existing techniques used in this paper.

2.2.1 Line Labelling

Line labelling is a method for interpreting the 3D structure of a 2D line drawing, in which labels are placed on lines to indicate their relative position in space. There are four labels used, indicating that

a line is convex or concave with respect to the viewer, or occluding something to the left or right of the line; as shown in Figure 2, a ‘-’ is used to denote a concave edge, a ‘+’ denotes convex edge, and an arrow denotes an occluding edge. Note that an arrow is directional, such that the solid object is lying to the right of the line and empty space or hidden detail is visible to the left of the line—see the cube occluding the two distant cubes in Figure 2 for example.

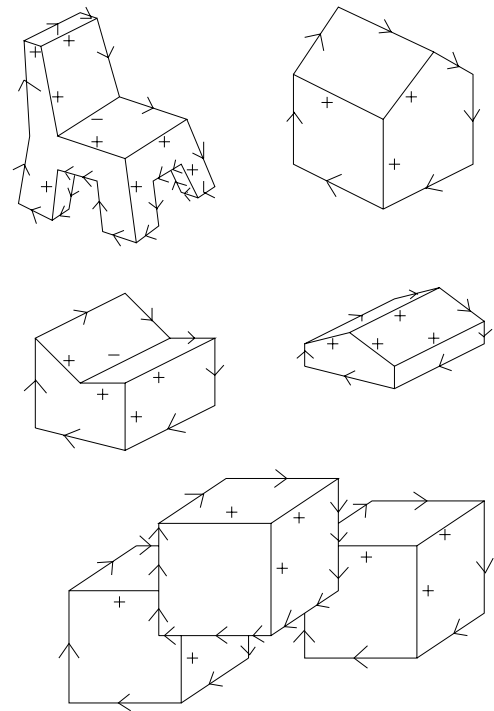


Figure 2: Example labelled drawings

Given these four labellings, there is only a finite number of ways in which lines classified in this way can meet at a junction; i.e. there is a finite number of ways lines can be labelled when they meet at a junction. These finite outcomes were initially derived independently by Clowes [4] and Huffman [10].

Waltz [27] introduced a constraint-based propagation to reduce the set of possible labellings at each junction to form a consistently labelled drawing (or several such drawings if more than one possibility exists).

2.2.2 Linear Systems for Spatial Structure

Sugihara [23] uses a linear system to represent the spatial structure in a drawing (namely, which vertices lie on each face), and to constrain the vertices whilst finding a solution that matches some physical property of the image (in this case, the light intensity across an object). Equations in the linear system are of the form presented below, with x and y being the 2D point co-ordinates and z the unknown depth of the corresponding vertex. We assume that the user will sketch an object in “generic position”, such that faces in the object are not parallel to the line of sight. With this assumption, we take p , q and c as constants for a face, giving constant equations of the form in Equation 1 for each vertex lying on a face.

$$px + qy + z + c = 0 \quad (1)$$

Problems arise in that the resulting set of equations is not necessarily linearly independent, and slight errors in the drawing or calculation can drastically change the solution found; this is the problem

of generic reconstructability, which is covered below; also in general there will not be enough equations to find all the unknowns.

2.2.3 Skewed Symmetry

Kanade [15, 16] has examined the use of skewed symmetry to derive the gradient of surfaces from a single orthographic view. Skewed symmetry occurs when a real symmetry is viewed in an image, where the viewer is not directly in front of a symmetrical face, causing the symmetry to appear “skewed”; see Figure 3.

By detecting the skewed symmetry axes, we then constrain the orientation of the face to lie on a hyperbola in gradient space, where gradient space is the 2D space of ordered pairs (p, q) in the face equation $px + qy + z + c = 0$. The hyperbola arises as the two axes of symmetry must be perpendicular to each other when the face is viewed head-on, and this constrains the possible gradient (p, q) of the symmetrical face (see Figure 4). There are an infinite number of (p, q) pairs on the hyperbola; points G_1 and G_2 are used, as these gradients represent the least slanted orientation that could produce the skewed symmetry in the image from a real symmetry in 3D. Thus, each face that displays skewed symmetry produces a pair of possible gradients, (p, q) and $(-p, -q)$. Line labelling is used to determine which of these are appropriate.

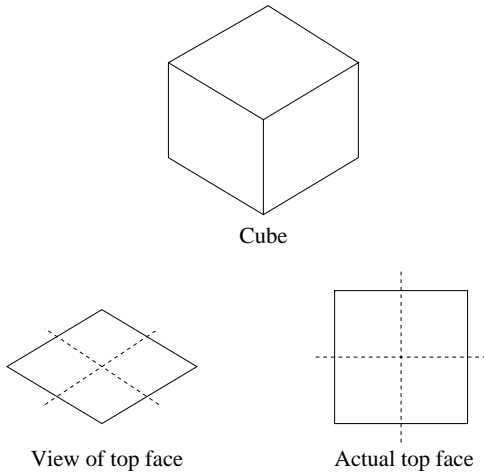


Figure 3: Skewed symmetrical figures

2.2.4 Generic Reconstructability

The generic reconstructability problem arises when the correctness of a drawing may be affected by small errors in the vertex positions; the typical example of a truncated pyramid is shown in Figure 5. With this image, the object is only reconstructible as a solid if the lines joining the outer triangle to the inner triangle meet at a common point (as three planar faces must meet at a common vertex). Clearly, if the original drawing is a sketch by the user it is to be expected that the lines will not exactly meet.

Sugihara [24] presents a method for ensuring that a drawing is generically reconstructible; this involves a linear system, and removing enough equations from the linear system such that vertices in the drawing have enough freedom to be moved to fit the planar interpretation. The linear system is then solved, and the vertices are computed as the intersections of the newly solved faces; thus their original x, y co-ordinates may be altered.

Wang [28] uses an alternative method, which checks several consistency constraints. If the drawing is not generically reconstructible, further planes and vertices are added as necessary (in the case of the

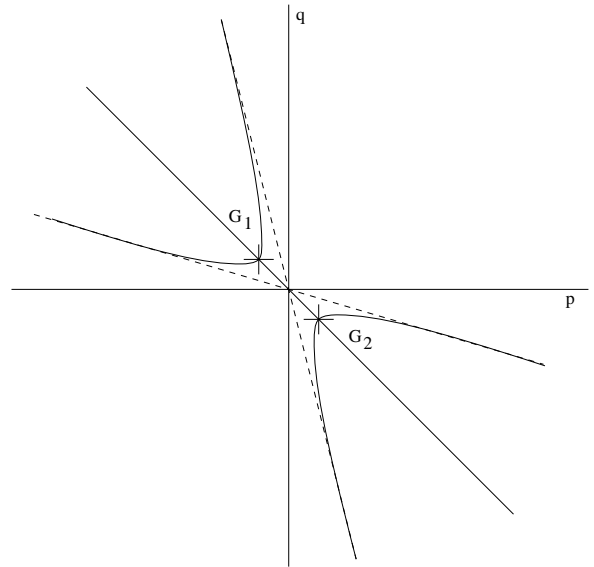


Figure 4: Gradient constraints in a skewed symmetrical figure

truncated pyramid, existing vertices remain unchanged and one extra plane and vertex are introduced to ensure reconstructability).

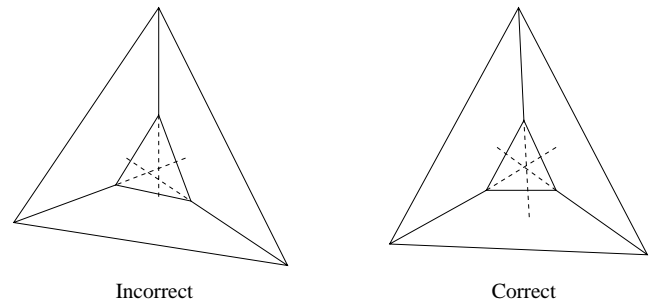


Figure 5: Sample generically unreconstructible figure: the truncated pyramid

2.2.5 Hidden Face Recovery

This subject is touched on by Sugihara [24]. He restricts the problem to objects with exactly four degrees of freedom, and determines hidden faces from three or more vertices that are visible and belong to the hidden face. Intersections between hidden faces can then be found to form edges between them, and the solid can be completed.

3 OUR SYSTEM

In this section we present the details of each component module of our system.

3.1 Line Tidier

This module works entirely in 2D, and works incrementally as the designer sketches. Our sketch tidying process is taken from Jenkins [13], where many techniques for tidying a drawing are given. We only use line straightening, line connection (to ensure close lines actually connect to each other) and parallelism enforcement.

3.2 Line Labelling

We use the Huffman-Clowes labelling system for our tidied drawing; firstly, the module examines each junction in turn and labels its type accordingly. We then assign a list of possible labels of each incident line to each junction, and use Waltz's [27] constraint-based propagation system to remove inconsistent labellings from each junction. We use an exhaustive tree search to produce all distinct labellings of the complete sketch. One is chosen and worked on by subsequent processes; if the user rejects the final result, solids are created from each of the others in turn. We have also developed an incremental line labeller so that this can be accomplished as the user sketches, for extra speed.

3.3 Assignment of Faces to the Drawing

In this section we describe how we derive the visible faces from the labelled drawing, and how we deduce the orientation of hidden faces.

3.3.1 Assignment of Regions

Firstly, we label the exterior of the object as region 0 (representing a "background face", which is assumed to be infinitely far behind the object); we start by finding the junction with the largest y coordinate. We then follow the border from this junction, until we are back at it (see Figure 6). All regions are recorded as a junction loop (a linked list recording each member junction of the region in a clockwise or anti-clockwise direction; see Figure 7), which we refer to as the region boundary.

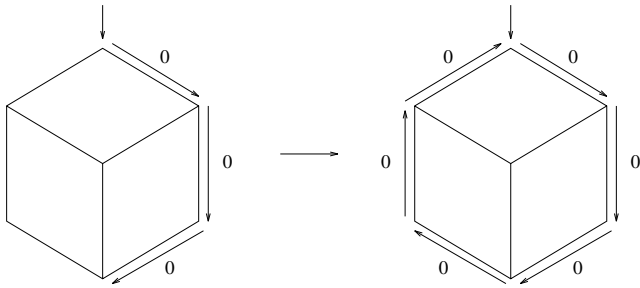


Figure 6: Marking the "Background" face 0 on an object

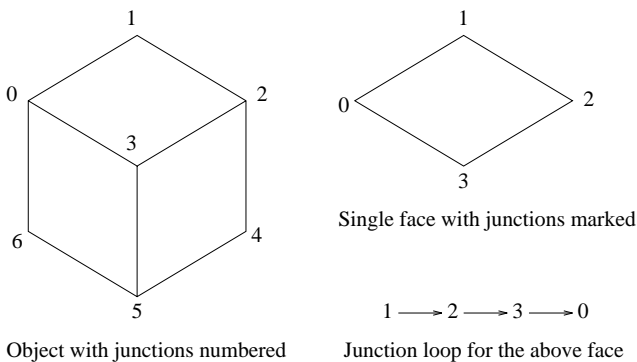
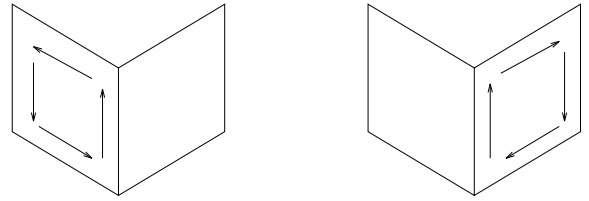


Figure 7: A sample junction loop

Having marked the background region, we then move on to the interior of the object. We examine each line in the object in turn, checking for existence of a region label on the left and right hand side of the line in question. If a region is not assigned, we create a

new region and start to mark the objects by following loops of lines in a consistent orientation (see Figure 8).



Left-hand side marking of face

Right-hand side marking of face

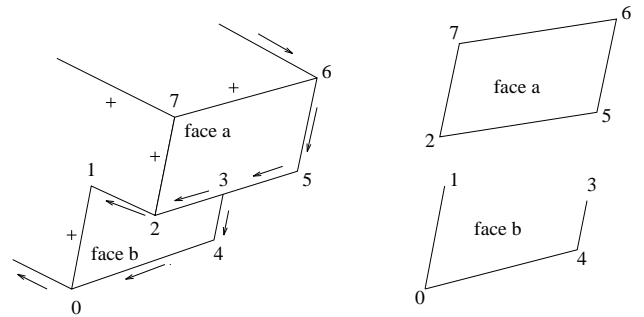
Figure 8: Following each side of a line

3.3.2 Creation of an Intermediate Face Structure Model

We now create an intermediate face structure model from the region information that we have deduced; the model will become a solid model later (at this stage, none of the faces in the object have their equation defined, nor do the vertices have their z coordinate fixed). 2D regions and line labels are examined to produce a 3D face structure, in which occluding lines formerly associated with two 2D regions are now represented as an edge connected to only one 3D face. Occluded faces are then dealt with in the next section.

Initially, we create a 3D vertex for each 2D junction in the drawing, leaving the z coordinate undefined but taking the x and y coordinates from the drawing.

We then create 3D faces from the 2D regions. We follow each region boundary and assign each associated vertex from the junctions in the junction loop to the appropriate face. If a junction has both incoming and outgoing occluding lines that occlude the current face, then it does not correspond to a 3D vertex (see Figure 9) and hence is to be skipped and not added to the face. Note that region 0 in the drawing (the "background") is not physically part of the drawn object, and is hence ignored when creating 3D faces.



Line labelled object with vertices numbered

Faces present in drawing

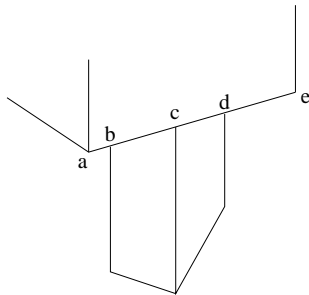
face a: 2 ← 3 ← 5 incoming & outgoing occluding edges; skip vertex
face b: 2 ← 3 → 4 both outgoing occluding edges; use vertex

Actions to be taken when considering vertex 3 from two different faces.

Figure 9: When a vertex has both incoming and outgoing occluding edges

With the face structure and vertices created, we now form the corresponding 3D edges.

We examine each vertex loop (face) in turn, and search for a 3D edge linking each pair of vertices; if one exists, then we note that this edge lies on this face. Otherwise, we search for a 2D line linking the associated junctions of the two vertices in question. If such a line exists, or if several “T” junctions connect the two junctions in a straight line (see Figure 10), then we create a new 3D edge connecting the two vertices (again noting the face it lies on).



Vertex ‘a’ will be joined to vertex ‘e’ by a single edge

Figure 10: A line of “T” vertices connecting two vertices

We have created a partial 3D topological model from the 2D drawing; we also now assign some of the hidden faces to the model.

3.3.3 Creation of Hidden Faces

Given the intermediate face structure and the labelled drawing we check for any vertices connected to only two 3D faces, and create additional faces to form a trihedral model. These additional faces are not complete, and have remaining vertices added to them in Section 3.7.

Using the assumption of trihedral vertices, we know that exactly three faces must meet at each vertex; using this fact, we must have one hidden edge at each two-edge “L” vertex (which corresponds to an “L” junction). An “L” vertex forms the border between two consecutive faces, and we must create a new face as we come across each “L” vertex.

Thus, we consider occluding edges (identified by having one associated face); if such an edge is found, then we continue to follow any edges connected to this edge that are also occluding until we meet a 2-edge vertex; see Figure 11. Note that we can follow a pathway through the edges in this manner as the only time we will have an edge with one associated face is when the 2D line was marked as occluding; three-line junctions can only have two or no occluding lines, and two-line junctions do not matter as we halt as soon as we find its related two-edge vertex. If we have a hidden face all of whose edges are completely visible (such as the base of a pyramid viewed from above), then there will not be any two-edge vertices on this face; we detect this by ceasing the search as soon as we come back to the initial vertex. The complete vertex loop we have just traversed is then added as a new face, and the connecting edges associated with the new face accordingly.

We otherwise create a new face when arriving at an “L” vertex, and assign this vertex, the edge attached to this vertex and its other terminating vertex to the face (we have found a new hidden face). We now follow the attached one-face edges around the object, assigning them to the new face as we go, until we meet another two-edge vertex (see Figure 12), in which case we go back to looking for any remaining edges associated with only one face, until we have examined all of the edges in the face structure model.

We have now added the appropriate hidden faces that intersect any of the visible faces in the object.

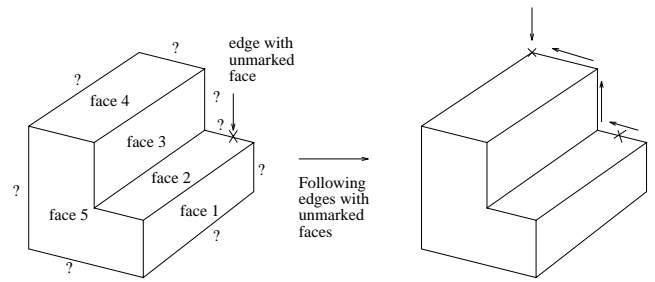


Figure 11: A two-edge vertex, attached to an unmarked face

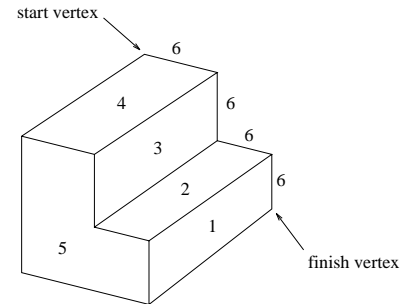


Figure 12: The end of the hidden face—another 2 line vertex

3.4 Constraint Creator

This module creates a linear system from the face structure model, and also adds constraints to the system from artifacts in the drawing such as skewed symmetry and parallelism. This system is then solved, using the artifacts to produce the unknown depths and face coefficients. We will cover the creation of the bare linear system initially, and then move onto the extra equations added by artifact identification.

3.4.1 Creation of the Least Squares System

We examine the vertex loops stored with each face to create the set of equations that constrain the appropriate vertices to lie in each face; each equation is of the form in Equation 1. This is presented as a matrix A and two vectors, \underline{x} and \underline{b} in the form $A\underline{x} = \underline{b}$, where A and \underline{b} contain the constants and \underline{x} the unknowns.

If we have m vertices we thus have m unknown z coordinates, and if we have n faces we have $3n$ unknown coefficients for each face, resulting in $m + 3n$ unknowns. In general, each vertex lies on 3 separate faces, leading to a total of $3m$ equations.

Once all vertices have been added, we have constrained the face structure model but not actually added any information about the unknown depth; this is produced by examining artifacts in the drawing and is discussed in the next section.

3.5 Artifact Identification

In our case, an artifact is any combination of lines or junctions that enable us to constrain the depth of the solid model by making assumptions about the user’s intentions. The artifacts we use are parallelism, skewed symmetry, and right angle fit; these are explained in the following sections.

3.5.1 Parallelism

With planar models of machined objects, there is a high tendency for designers to use parallel faces, for reasons of ease of manufacture and assembly. Parallel edges are hence common and are an important artifact for constraining an object, and a technique for detecting, tidying and constraining them is now presented.

A set of buckets is created at one degree intervals from 0 to 179 degrees and a weight is associated with each bucket, initially 0. Each line in the drawing is examined, and its angle to the x axis calculated to the nearest degree. The line is placed in the corresponding bucket, and the weight for the bucket is increased by the length of the line (a longer line is more likely to be accurately drawn than a shorter one and is thus given greater importance).

After all lines have been added to the buckets, we find those buckets with high weights, and transfer into them the contents of nearby buckets with lower weights on the assumption that the lines in them were poorly drawn. Thus, the positions of the junctions at the ends of these lines must be altered to give them the desired angle.

We use several passes in which we attempt to move the most constrained junctions first, fixing them in position. We then work through the less constrained junctions, moving them to ensure that their incident lines are parallel with respect to previously fixed lines, making unconstrained junctions more constrained until they are fixed in position. This approach may fail if there is a high proportion of junctions associated with three regions, as the drawing is too constrained to allow us to adjust all of the junctions desired. If this happens, some lines will not be adjusted to be parallel as required; however, this does not stop a reasonable solid model from being created. A method is required to enforce parallelism on all desired lines and is the subject of ongoing research.

Once all the lines and junctions have been moved as required, the related vertices are updated to reflect their new position. We then produce equations to keep appropriate pairs of edges parallel in 3D, viz:

$$l_2(z'_1 - z_1) - l_1(z'_2 - z_2) = 0 \quad (2)$$

where l_1 and l_2 are the gradients of the lines in 2D and z_i, z'_i are the z coordinates of the endpoints of the corresponding edge i taken in an appropriate sense.

Note that given n edges to be parallel, we only need generate $n - 1$ equations; for instance, given that edges 1,2,3,4 are parallel, then we generate equations to fix pairs 1,2; 2,3 and 3,4.

3.5.2 Skewed Symmetry

We have already explained how to recover the gradient of a face given the skewed symmetry axis in Section 2.2.3, but not how to detect the axis in the first place; we outline this below.

For a region to contain a skewed symmetry, one axis must pass through the mid-point of a line or through a junction. The axis must also be centred on the centre of gravity of the object for similar reasons. (See Friedberg [7] for explanations of these assumptions and an implementation of a skewed symmetry detector with bitmap images).

We now investigate these possibilities for each line or junction in turn for one axis (thus for a region with 4 member edges, we will have a total of 8 test axis; 4 from junctions and 4 from mid-points of lines). The other axis of symmetry is calculated directly from the orientation of the initial axis and the moments of inertia of the region (see Friedberg [7]). Note that if a reconstructed face has a rotational symmetry (such as a triangle), then there will be multiple solutions to the skewed symmetry test; in which case, we just select the best fitting solution. If we are incorrect in our selection of axes, this assumption will be rejected later when we remove inconsistent equations after finding a best fit solution for the linear system.

Given the two axes, we then reconstruct the face by applying an inverse skew transformation to it, thus producing the original face as if it were viewed face-on (with its symmetry axes being the x and y axes). We then directly test the reconstructed face for symmetry by “mirroring” the face across its major axis of symmetry and measuring the difference in position of each vertex compared to its putative mirror image. These differences are then totalled up, and the candidate axes of symmetry that produce the smallest mismatch are selected as being the best axes for the region.

If the mismatch is within a certain threshold (validating our assumption that the face was indeed symmetrical), we then compute the gradient from the axes as described in Section 2.2.3. The gradient is added in the form of two equations:

$$p = \text{constant} \quad q = \text{constant} \quad (3)$$

to the set of linear equations.

3.5.3 Right Angle Fit

This artifact occurs when edges of a face mainly meet at right angles, as such faces are common in engineering. We select an axis as being parallel to a line in the face, and test all possible pair of axes. For each pair of axes, we reconstruct the face-on view as before, except that this time we now test for the number of right angles in the reconstructed face instead of mirror symmetry. The deviation of each angle between adjacent edges from 90 degrees is calculated for each vertex in the face and totalled. The pair of axes that produces the least total deviation are then selected as giving the most probable orientation of the face if it is bounded by many edges meeting orthogonally.

If the deviation is below a certain threshold, and produces a better fit than the skewed symmetry, then this is used instead to give the gradient of the face which is then added to the linear system in exactly the same way as before.

3.5.4 Other Artifacts and Degrees of Freedom

We realise that the current set of artifacts is by no means exhaustive, and we intend in future to make use of other artifacts. One such artifact is the human assumption that three faces meeting at a “Y” junction are perpendicular to each other if all three angles between the 2D lines are obtuse (Biederman [1]), although this assumption is probably too strong to be generally useful.

With our artifacts, we hope to have produced sufficient equations to solve all of the unknowns. The estimation of the gradient of a small proportion of faces will enable us to solve the least squares system, as (for example) once the gradients of two adjacent faces are known, the gradient of any face joining these two faces can be found from the two lines shared by each of these faces and the third face. Our examples have shown in practise that in most cases we do indeed have sufficient equations, but there are cases where this is not so (for instance, when an object has no symmetry, right angles or parallel lines). Further investigation of when such cases may arise, and other artifacts which may add further information is required.

The system as it stands still has one degree of freedom; vertex depth can only be computed relative to other vertices, but we have no measure of absolute depth. To constrain this, we assume the depth of one vertex and add an equation containing this information to the linear system prior to its solution. The vertex whose depth is fixed is the (or a) vertex for which we have the greatest amount of gradient information for its associated faces. This will lead to vertex depths for other vertices on the faces, and so on, when a least squares fit is found for the linear system. If we were to fix the depth of a vertex for which no gradient information is available for its associated faces this would not be sufficient by itself, as the associated faces could

change their gradients to accommodate various values of depths for their other vertices.

3.6 Solving the Linear System

In this section we describe how we use a least squares fit to solve the system, and how we solve the generic reconstructability problem.

3.6.1 Least Squares Solution Method

With the equations we have produced, there is no exact solution; due to drawing and digitisation errors the solid model is inaccurate, and we cannot guarantee that the extra constraints we have added are consistent. With this in mind, we selected a least squares fit algorithm to produce the most consistent solution.

We now have to deal with any bad choices in our selection of equations. The linear system is passed through the least squares algorithm, and the standard deviation from the result is then calculated. If any equations produce an error above a certain threshold (we use three standard deviations), then the equation is deleted from the least squares system as it is inconsistent with the others.

After any such equations are removed due to excess error, the least squares solution is recalculated from the remaining equations. This is repeated until no more equations are rejected, or the least squares algorithm fails to produce a non-trivial solution.

If a solution cannot be found, then the system produces an error message and informs the user that it cannot create an object from the given drawing. If there are multiple possibilities for the labelling of this drawing, then the system selects another labelling and tries again; otherwise, the user can adjust the drawing, or start afresh with a new drawing.

Once a solution has been found, the faces equations and vertex positions are known.

3.6.2 Generic Reconstructability

An object may be inaccurately sketched by a user, and result in an impossible object—the object cannot be created from planar faces, as discussed in Section 2.2.4.

Neither of Sugihara’s nor Wang’s solution is particularly elegant or intuitive to the designer. With Sugihara’s method, the adjustment in the drawing is concentrated in a very small area of the drawing, usually around one or two vertices which results in one part of the drawing being badly distorted to form the correct object (see Figure 13). Wang’s method is unacceptable as it adds extra faces to a designer’s object; this is an unreasonable approach as a designer will not accidentally miss out an entire face of an object.

Instead, with our least squares solution method, we spread the error throughout the drawing and then produce a generically reconstructible object afterwards. The least squares produces the best fit for a constructable object (it must be constructable if it obeys our constraints) from the drawing. After the best fit has been found, the vertices will only lie approximately on the faces. Thus the vertex information found from the fit is now discarded, and all vertices are recomputed as being the appropriate intersections of the faces to ensure they lie exactly on the faces. An example of a reconstructed object using our method is presented in Figure 14.

3.7 Completion of Hidden Faces

In this section we follow the stages required to complete the occluded and partially occluded faces in the object. We first create extra edges at each two-edge vertex by intersecting neighbouring faces, thus adding to the boundary of the incomplete faces. These newly created edges are then intersected with each other to further

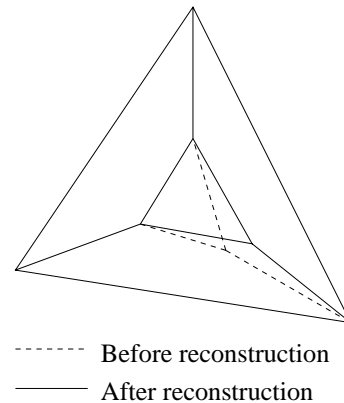


Figure 13: A truncated pyramid after Sugihara’s generic reconstructability algorithm

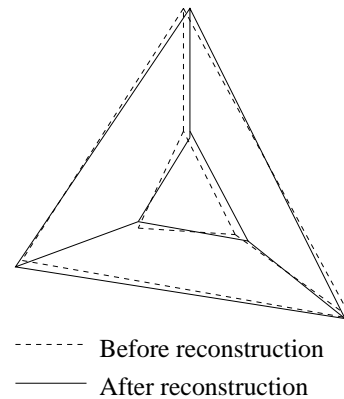


Figure 14: A truncated pyramid after our generic reconstructability algorithm

complete the boundaries, forming more parts of the boundary of the solid model. If some boundaries still remain incomplete, we provide two further methods in an attempt to find further boundary elements of the incomplete faces, based on the assumption that faces are likely to be parallel to each other, and based on searching for partial boundaries which apparently belong to different faces but actually belong to the same face. These processes are then reapplied in turn if necessary, as each may be able to make use of further information provided by the others, with iteration continuing until no extra information is obtained.

3.7.1 Creation of 3-Edge Vertices

As the face structure model still has some vertices that only have two incident edges, we must intersect the associated faces to form additional edges and join them to the two-edge vertices to form a complete trihedral model. We now describe our method to reconstruct the missing edges; note that we will consider the missing faces later.

We search for any two-edge vertices; we examine the two incident edges. Out of the two edges, there will be references to a total of three faces; one face will be common to both edges, leaving the other two faces without a connecting edge; see Figure 15. We calculate the edge intersecting the two faces if (and only if) both of the faces are of known gradient; if they are not, then we skip this vertex and continue the search.

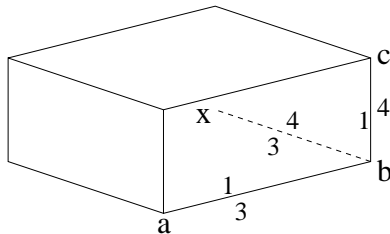


Figure 15: Faces meeting at a two-line vertex

We now create a new vertex (vertex x in Figure 15) to lie on the intersecting edge and position it at an arbitrary distance along the edge from the current vertex. A new edge is created that links the current vertex (vertex b in Figure 15) to the newly created vertex.

The two faces are now marked as lying on either side of the edge and the edge in turn is added into the boundaries for these two faces. This is continued until there are no more untested two-edge vertices. We have now completed as many two-edge vertices as faces with known gradients have allowed us; note that further uncompleted two-edge vertices may be solved later by another pass through this code as more of the object is reconstructed.

We now need to complete the face boundaries of the hidden faces; at this stage we take the face structure model and examine it for any face boundaries that are open; i.e. two consecutive edges in the boundary are not connected, such as the face in Figure 16(i). We attempt to close these boundaries by intersecting unconnected edges where appropriate and reflect the changes in the face structure model.

We search each incomplete face in turn, examining the face borders looking for an “open” edge connecting two vertices (an open edge is where one vertex on the edge has only one incident edge, such as vertex 5 in Figure 16(i)).

Once one is found, we follow the face border over this one-edge vertex to the next vertex, calling the first vertex the original vertex and the latter the destination vertex; see Figure 16(ii). We now have two unconnected vertices; if we are fortunate, the two edges attached to the vertices can be extended to meet to complete the face

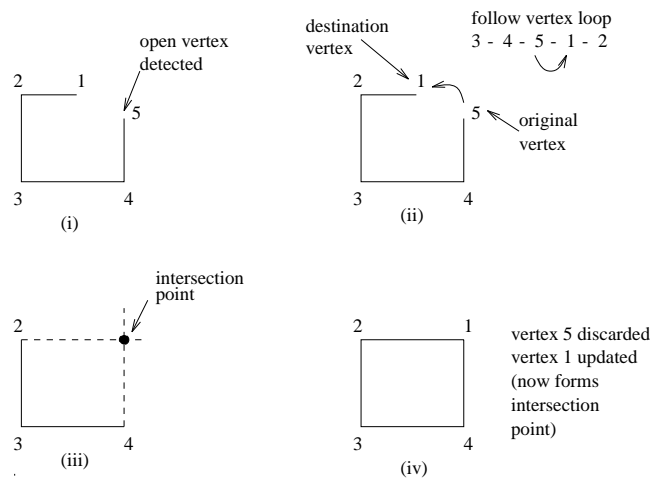


Figure 16: The completion of a face boundary

border, as they both lie in the same plane. In practise this intersection point may be at a position in space well away from the rest of the object, and if so, is unlikely to represent the users’ intended completion of this hidden part of the object. If so, we ignore this intersection, skipping this pair of vertices; otherwise the boundary loop is update as shown in Figure 16. We must also appropriately update the boundary loops of the other faces associated with the edges which have been extended.

If this intersection would result in a vertex with four incident edges then we cannot directly connect them as this would break the trihedral assumption. Instead, as the edges are connected to two two-edge vertices, we add an edge between the vertices if they are associated with two common faces and this edge will lie on the intersection of the two faces (see Figure 17). Otherwise, we skip this pair of vertices.

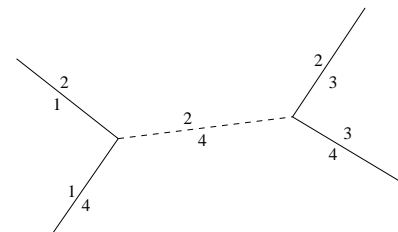


Figure 17: The joining of two two-edge vertices

Once we have attempted to close any incomplete face boundaries, we then need to check if there is any more work to be accomplished, or indeed can be accomplished. We check each vertex in turn to ensure that it is trihedral, and that the parameters of all faces are known. If all vertices and faces are solved, then the object must be trihedral and hence complete; we then move on to the final tidying process mentioned in Section 3.8. Otherwise, we must somehow complete the remaining faces; we now present two methods for solving incomplete faces.

3.7.2 Parallelism of Faces

This can be used to fix the orientation of a face that would otherwise have one degree of freedom; we merely need to fix the angle of rotation of the face with respect to a fixed line that it is attached to. An example of how such a face could arise is shown in Figure 18.

We search for any face that only has a visible edge; if there is such an edge, then we search for a face that is parallel to this single edge, but is not parallel to any of the faces to which the edge is attached (see Figure 18).

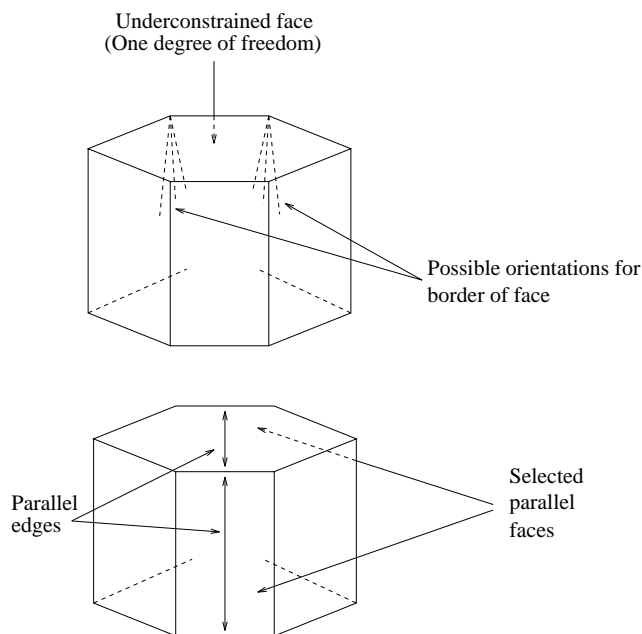


Figure 18: Testing for a parallel face

If there are no candidate faces, then we must skip this face; if there is a single candidate, then we simply select its gradient and assign it to the unsolved face. If we have more than one candidate, then we must select a single face to use; we currently select the nearest face to the unsolved edge.

Now that the unsolved face has a gradient assigned to it, iteration of the reconstruction algorithm will create the missing edges and complete this face.

3.7.3 Confusion Between Faces

This occurs when a single face is labelled as being two separate entities due to occlusion separating the face into two sections (see Figure 19); we check for inconsistencies in the labelling of faces to detect this.

Firstly, we search for any undefined faces (i.e. whose parameters are unknown), and select those which have a single visible, incomplete two-line vertex.

We now check for any three-line vertices in the face that have inconsistent face labels; for example vertex *a* in Figure 19. Such a vertex must originate from a 2D "T" junction, with the partly occluded line being labelled as belonging to an extra face, which is actually just part of another occluded face (faces 9 and 5 respectively in Figure 19). We merge the two incomplete faces to form a single, complete face and update the solid model accordingly.

3.7.4 Iteration of the Reconstruction Algorithm

If we have not yet resolved the object, but have obtained further information using the above methods, then we repeat the steps described in Section 3.7 until we obtain no more information. If we have not completely determined the object and can not get closer to a solution, we simply present to the user as much of the object as we

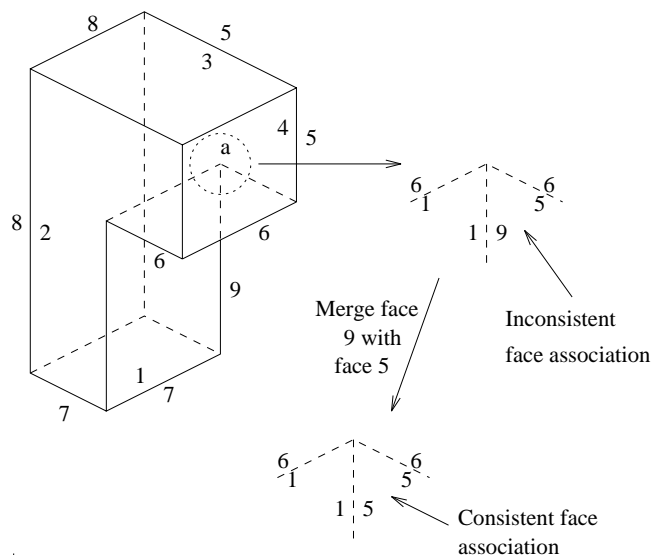


Figure 19: When the labelling of a face is confused

have managed to solve, but warn him that no solid model could be determined; otherwise, we now move on to the final tidying process.

3.8 Final Tidying Process

Having produced a complete solid model, we now construct the linear system again, but this time we include all of the newly created hidden faces and vertices. Solving this allows us to tidy up any remaining anomalies in the new hidden parts of the object.

The same code is used as before to produce the linear system from the model; this time, however, gross errors in the linear system can not arise. The final solid model is now created from the least squares fit calculated for the linear system.

If the solid model is to be used in a solid modelling system, it is helpful to the user to align the object to the world axis, making modelling operations simpler. We assume that vertical lines in the drawing are intended as vertical edges in 3D, and so rotate the solid such that vertical lines are vertical edges in the world space of the solid modeller. If multiple solutions exist to the line labelling problem, then the user is prompted to verify if the produced solid is the intended object. If it is not, then the system selects the next possible labelling and generates a solid model from it in turn. The selection of labellings is an issue for future research, such as detecting features in the drawing to display the most probable labelling to the user, instead of presenting all possibilities.

4 EXAMPLES

In this section we present the results of our system when applied to two simple sketches; a truncated pyramid and an L-block. The drawing is shown before it is tidied, during the interpretation process and a view of the resulting solid model. The entire interpretation process (from initial sketch to final solid model) takes approximately five seconds elapsed time for either example (on a Sun SPARC 1 workstation). These run times are varying due to system load, and could be dramatically reduced by optimising code and removing the additional screen updates used to produce the intermediate figures for testing purposes shown in our example sketches below.

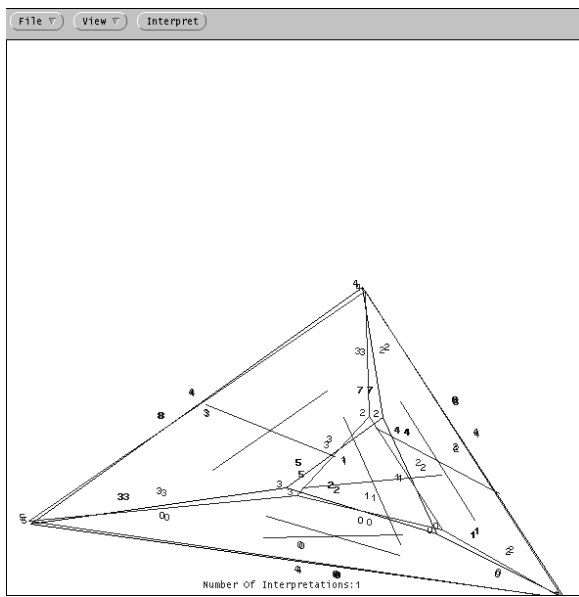


Figure 24: Truncated pyramid—reconstruction from least squares fit of linear system

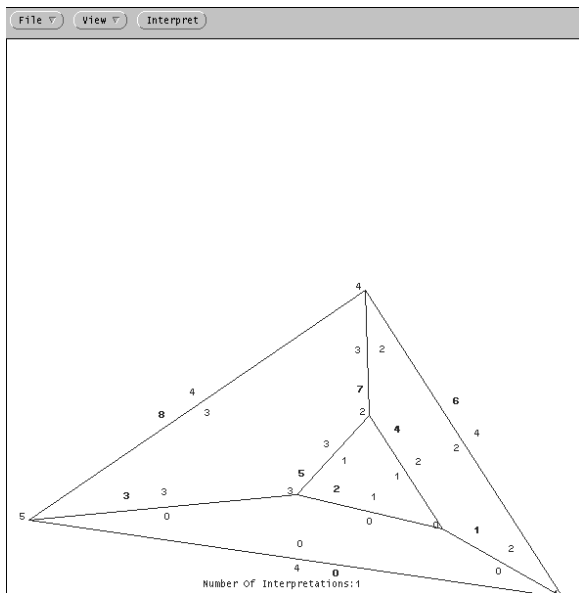


Figure 25: Truncated pyramid—creation of hidden edges and faces

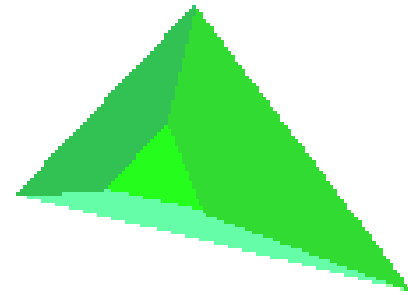


Figure 26: Truncated pyramid—sample view of the reconstructed solid

4.2 L-Block

Example output from our program is presented in Figures 27 to 34, as for the previous example. However, this time, the final figure shows a wireframe view of the resulting solid, so the hidden details which have been reconstructed can be seen.

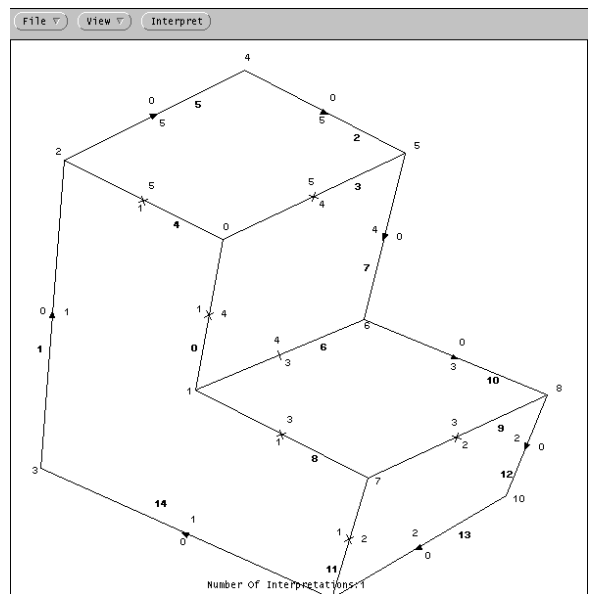


Figure 27: L-block—initial labelling

5 CONCLUSIONS

We have produced a system which takes a roughly drawn, single view, orthogonal hidden line removed sketch of a 3D object and upon making various assumptions about the nature of the sketch, is usually able to produce a B-rep solid model. Information about the hidden part of the object is generated as part of this process, again by making certain assumptions.

The method currently works for many cases, especially if the user sketches the view containing the most information about the

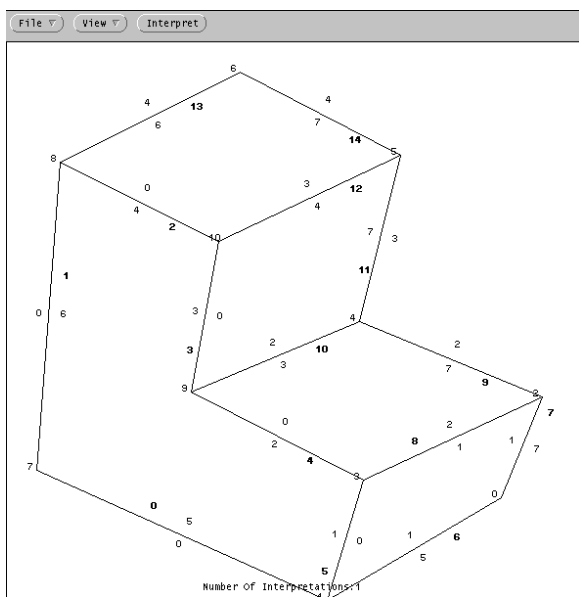


Figure 28: L-block—solid model formation (note hidden faces)

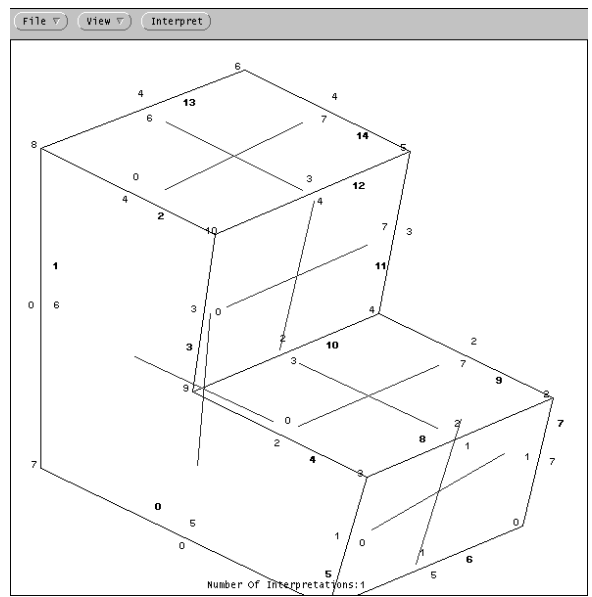


Figure 30: L-block—skewed symmetry and right angle fit

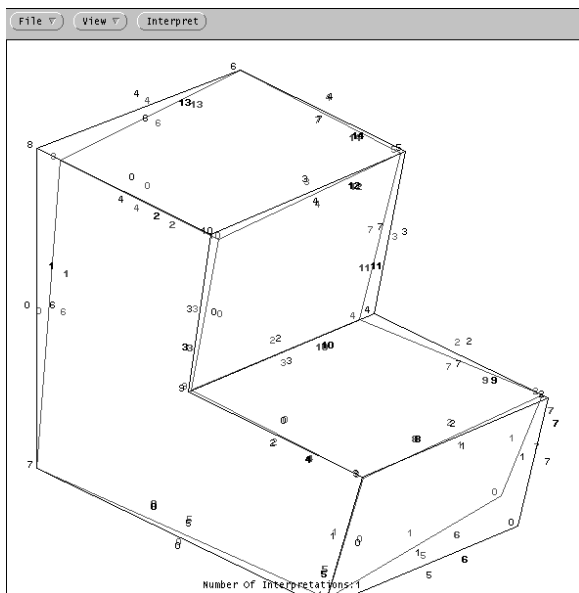


Figure 29: L-block—parallelism

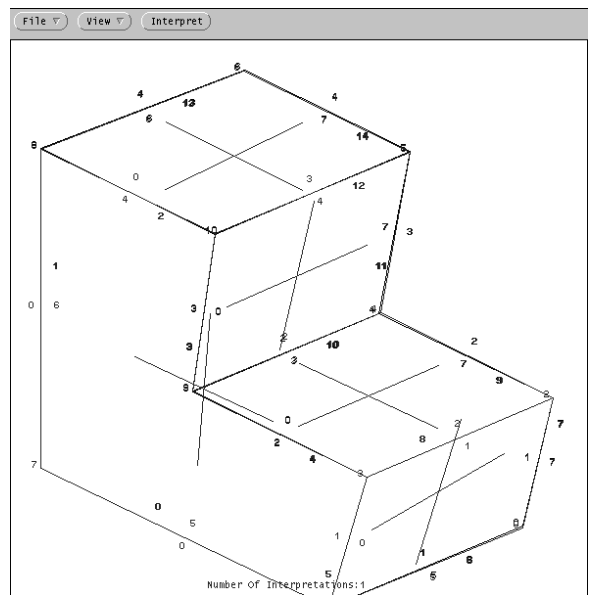


Figure 31: L-block—reconstruction from least squares fit of linear system

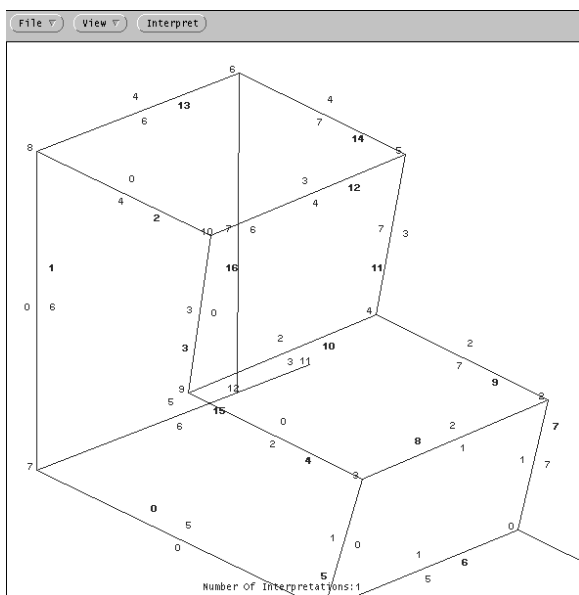


Figure 32: L-block—creation of hidden edges and faces

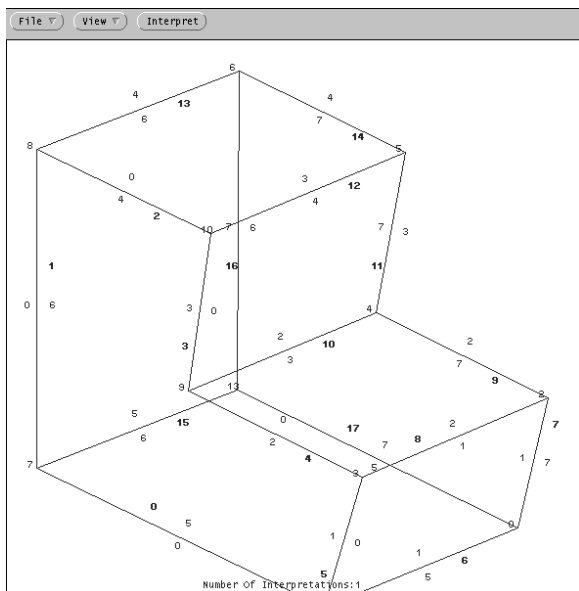


Figure 33: L-block—intersection of hidden edges and faces

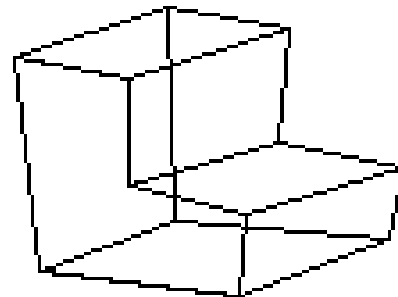


Figure 34: L-block—sample view of the reconstructed solid

object. It is not absolutely guaranteed to give a solid model, however we are looking at improving the algorithm to give a solid model in more cases and to ensure that if a solid is produced, then it is what the user would expect. More work is needed to characterise fully the circumstances when the current system will fail to produce a complete solid model. Once this has been done, it may be possible to identify further artifacts which will help to add further information in such cases. We are also considering algorithms which guarantee a solid is produced in all cases, including when the system cannot create the hidden parts of the object directly from information in the drawing.

Currently the system cannot deal with faces which contain hole loops. Adding these will increase the range of sketches which the system can handle, at the expense of having to consider more cases from the line labelling, and increased difficulty in completing the hidden sections of the object. However, in general, we do not expect the user to sketch very complex objects with our system, at least as single objects. More realistically, because of limitations on hidden details, the user is likely to find it better to sketch the basic object, and “features” (for example, complex bosses), as separate objects which can then be combined using Boolean set operators.

More generally, we also intend to extend the system to cope with simple cases of curved faces. The major changes here will be that new artifact types will need to be considered, silhouette edges will need to be distinguished from real edges, and the system of equations will become non-linear. The system could also be converted to work with perspective drawings if required; this would involve slight alterations to the artifact detection and least squares system. Vanishing point detection would also be required to enforce the parallelism of lines.

Overall, the current system represents a useful step towards the goal of direct creation of solid models from single 3D sketches.

REFERENCES

- [1] I Biederman. Human image understanding: Recent research and theory. *Computer Vision, Graphics and Image Processing*, 32:29–73, 1985.
- [2] E A Bier. Snap-dragging in three-dimensions. *ACM SIGGRAPH Computer Graphics—Special Issue: 1992 Symposium On Interactive Computer Graphics*, 24(2):193–204, 1992.

- [3] Z Chen and D-B Perng. Automatic reconstruction of 3D solid objects from 2D orthographic views. *Pattern Recognition*, 21(5):439–449, 1988.
- [4] M B Clowes. On seeing things. *Artificial Intelligence*, 2:79–116, 1970.
- [5] D Dutta and Y L Srinivas. Reconstruction of curved solids from two polygonal orthographic views. *Computer Aided Design*, 24(3):149–159, March 1992.
- [6] A R Forrest. User interfaces for three-dimensional geometric modelling. In *Proceedings of the 1986 Workshop on Interactive 3D Graphics*. ACM, New York, 1986.
- [7] S A Friedberg. Finding axes of skewed symmetry. *Computer Vision, Graphics and Image Processing*, 134:138–155, 1986.
- [8] Y Fukui. Input method of boundary solid by sketching. *Computer Aided Design*, 20(8):434–440, October 1988.
- [9] B J Hale, R P Burton, D R Olsen, and W D Stout. A three-dimensional sketching environment using two-dimensional perspective input. *Journal of Imaging Science And Technology*, 36(2):188–196, March 1992.
- [10] D A Huffman. *Impossible Objects as Nonsense Sentences*, pages 295–323. Machine Intelligence 6. New York: American Elsevier, 1971.
- [11] T S Hwang and D G Ullman. The design capture system: Capturing back-of-the-envelope sketches. *Journal of Engineering Design*, 1(4):339–353, 1990.
- [12] K Iwata, N Sugimara, and W Lee. Knowledge-based recognition of hand-written drawings for product modeling in machine design. *Expert Systems in Computer Aided Design*, pages 375–405, 1987.
- [13] D L Jenkins and R R Martin. Applying constraints to enforce users' intentions in freehand 2D sketches. *Intelligent Systems Engineering*, 1(1):32–49, 1992.
- [14] D L Jenkins and R R Martin. The importance of free-hand sketching in conceptual design: Automatic sketch input. In *ASME Design Automation Conference in Albuquerque*, 1993.
- [15] T Kanade. Recovery of the three-dimensional shape of an object from a single view. *Artificial Intelligence*, 17:409–460, 1981.
- [16] T Kanade and J R Kender. *Mapping Image Properties into Shape Constraints: Skewed Symmetry, Affine-Transformable Patterns and the Shape-from-Texture Paradigm*, pages 237–257. Human and Machine Vision. Academic Press, 1983.
- [17] K Kobori, N Futagami, and I Nishioka. Automated generation of simply connected solid objects from wire-frame data using operations on graphs. *Visual Computer*, pages 335–341, 1986.
- [18] Y G Leclerc and M A Fischler. An optimization-based approach to the interpretation of single line drawings as 3D wire frames. *International Journal of Computer Vision*, 9(2):113–136, 1992.
- [19] R Lequette. Automatic construction of curvilinear solids from wireframe views. *Computer Aided Design*, 20(4):171–180, May 1988.
- [20] T Marill. Emulating the human interpretation of line-drawings as three-dimensional objects. *International Journal of Computer Vision*, 6(2):147–161, 1991.
- [21] D Pugh. Designing solid objects using interactive sketch interpretation. *ACM SIGGRAPH Computer Graphics—Special Issue: 1992 Symposium on Interactive Computer Graphics*, 23(2):117–126, 1992.
- [22] C Suffell and G N Blount. Sketch form data input for engineering component definition. Technical report, Dept. of Combined Engineering, Coventry Lanchester Polytechnic, 1984/85.
- [23] K Sugihara. An algebraic approach to shape-from-image problems. *Artificial Intelligence*, pages 59–95, 1983.
- [24] K Sugihara. *Machine interpretation of line drawings*. Cambridge, Massachusetts: MIT Press, 1986.
- [25] D G Ullman, S Wood, and D Craig. The importance of drawing in the mechanical design process. *Computers and Graphics*, 14(2):263–274, 1990.
- [26] F Ulupinar and R Nevatia. Constraints for interpretation of line drawings under perspective projection. *Computer Vision, Graphics and Image Processing: Image Understanding*, 53(1):88–96, January 1991.
- [27] D. L. Waltz. Understanding line drawings of scenes with shadows. In P. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, New York, 1975.
- [28] W Wang. *On the Automatic Reconstruction of a 3D Object's Constructive Solid Geometry Representation from its 2D Projection Line Drawing*. Sc.d. thesis, University of Massachusetts-Lowell, Order No.9224569, Department of Computer Science, University of Massachusetts-Lowell, 1992.