Mathematical Modelling and Analysis
COP290

Parth Shah 2016CS10333

Yash Raj Gupta 2016CS10337

January 2018

Contents

Abstract

Three-dimensional CAD models are usually used by designers because of their multiple uses (visualization, simulation, machining). However, nowadays, multi orthographic view engineering drawings are still widely used. Accordingly, a conversion tool for obtaining 3D CAD models from 2D drawings (known as the "reconstruction problem") and vice versa is a very useful approach in a broad range of applications. In this paper we present a matrix based method for reconstructing solids from it's orthographic projection. First we develop some basic concepts of Matrix and Vector Algebra. We propose a systematic algorithm for converting any general rotated 3D model to it's 2D orthographic projections.Then we explore how to reconstruct a 3D model with it's 2D orthographic projection, and after preparing it's wireframe model we fill it.

Keywords - Objects Orthographic Projection, 3D Reconstruction, Wire-Frame, Pseudo Faces and Edges, Matrix Representation

Introduction

3D CAD modelers are recognized as one of the mostly used tool in engineering design. Both solid and surface CAD models have become crucial for a large number of CAE techniques (e.g. visualization, simulation, CNC machining, ...). Anyway, multi orthographic view engineering drawings have been widely used up to latest decade and still are, so they play an essential role in traditional engineering. Actually, many products are still designed by means of orthographic views. Moreover, many engineering tasks involve modification of existing design, thus an automatic tool for reconstructing a 3D CAD model, starting from multi orthographic view engineering drawings, would prove to be particularly useful in many applications. In addition to its research significance, this kind of tool would ease a number of practical issues, mostly in the field of automatic conversion of digitized engineering drawings into 3D CAD models. So we are building a software which converts a 3D model to it's 2D orthographic projection and also solves the problem of reconstruction of 3D objects using it's 2D views and in this paper we explain all the fundamental ideas behind it.

Before moving forward we want to inform the reader about the assumptions we are going to build this project on.The first assumption is that all the object and the views provided will be polyhedral i.e. no curved edges will be present.Also for the reconstruction of the 3D object problem we work on the assumption that the 2D views given will be correspondingly labelled and no vertex or edge will be unlabeled.

In this paper we first develop a solution to create the Orthographic Projection from 3D model by using the concepts of Projection Matrix and Rotation Matrix as discussed in the class. We think of including interaction of rotation an object in 3D in however manner one like and be able to see it's orthographic projection.

We then discuss about an efficient way of reconstructing a 3D object by it's correspondingly labeled 2D views. We first think of how many 2D views are necessary for the reconstruction problem.For vertices, whenever we project any vertex we drop one of it's coordinate but the other two are preserved. So we can extract coordinate information of the vertex from 2 of the views and could successfully tell the exact coordinates of it in the 3D space.So 2 views are both necessary and sufficient for extracting coordinates of all the vertices present in the 3D prototype. Now all we need to know is which all vertex are connected in the 3D space i.e. what all edges are present.

All edges present in one 2D view will also be present in the 3D prototype. But their are some cases we need to focus on. In a 2D view many vertex may become coincident (whose projections are same) and between 2 pairs of coincident vertices if an edge exist, then we won't know what the actual end points of the edge are and it may even happen that the edge may overlap.So to work through this problem we add all possible permutation of the edge possible and so some pseudo edges may be added which we will remove in the next stage of the algorithm.

It may also happen that there is an edge present between coincident vertices, those won't be visible in projected view. Mathematically speaking all normals to projecting plane will be seen as points on the projecting plane rather than an edge.One alternative to the problem is that we may add all possible permutation of the edges from the set of coincident vertices(pseudo edge) and then remove them in the next step of the algorithm but a more efficient way of solving the problem exist.We were till now just using 1 view for edges but we know that at least 2 views are necessary to plot all the vertices so we may better use it for edges too.

It's not possible for any edge to be projected as points in two non-parallel views because for it to be true that edge need to be normal to both the planes which is not possible for planes that are not parallel.So on actually seeing two views we can see all edges present. Therefore two correspondingly labeled views are necessary to successfully find all the vertices and edges present in the 3D problem and no more views are required.

Vector Algebra

1.1 Vectors

An n-tuple (pair, triple, quadruple, ...) of scalars can be written as a horizontal row or vertical column. A column is called a vector. In analytic geometry it ís convenient to use columns of coordinates for points.

1.1.1 Some properties of vectors

You can add two vectors with the same number of entries:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ x_3 + y_3 \\ \vdots \\ x_n + y_n \end{bmatrix}$$

Vectors satisfy commutative and associative laws for addition:

$$X + Y = Y + X$$

$$X + (Y + Z) = (X + Y) + Z.$$

Therefore, as in scalar algebra, you can rearrange repeated sums at will and omit many parentheses. The zero vector and the negative of a vector are defined by the equations

$$O = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\text{-Y=-}\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} -y_1 \\ -y_2 \\ -y_3 \\ \vdots \\ -y_n \end{bmatrix}$$

Clearly,

$$-O = O \qquad\qquad X + O = X$$
$$-(-X) = X \qquad\qquad X + (-X) = O$$

You can regard vector subtraction as composition of negation and addition. For example, X - Y = X + (-Y), and you can rewrite the last equation displayed above as X - X = O. You should state and verify appropriate manipulation rules. You can multiply a vector by a scalar:

$$\text{sY=s}\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} sy_1 \\ sy_2 \\ sy_3 \\ \vdots \\ sy_n \end{bmatrix}$$

These rules are useful:
1)**Associative Law**: (Xr)s = X(rs)
2)**Distributive Law**: (X+r)s = Xr+Xs
3)$\text{X}^t \pm Y^t = (X \pm Y)^t$
4)$-(X^t) = (-X)^t$
5)$s(X^t) = (sX)^t$

Finally, you can multiply a row by a vector with the same number of entries to get their scalar product:

$$\text{X}^t Y = [x1, ..., xn]\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = x_1y_1 + ... + x_ny_n.$$

1.2 Projection

In linear algebra and functional analysis, a projection is a linear transformation P from a vector space to itself such that P 2 = P. That is, whenever P is applied twice to any value, it gives the same result as if it were applied once (idempotent). It leaves its image unchanged.

Orthogonal Projection

For example, the function which maps the point (x, y, z) in three-dimensional space R3 to the point (x, y, 0) is an orthogonal projection onto the x–y plane. This function is represented by the matrixFor example, the function which maps the point (x, y, z) in three-dimensional space R3 to the point (x, y, 0) is an orthogonal projection onto the x–y plane. This function is represented by the matrix

$$\mathrm{P}_z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The action of this matrix on an arbitrary vector is:

$$\mathrm{P} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

To see that P is indeed a projection, i.e., $\mathrm{P} = \mathrm{P}^2, we compute$

$$P^2 \begin{bmatrix} x \\ y \\ z \end{bmatrix} = P \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = P \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

$We again verify that the projection of a general point P : (x, y, z) on the XY Plane i.e. z = 0 is O : (x, y, 0).$

The direction ratios of the line joining P and O is 0, 0, 1.

We know that the projection of a point on the plane is same as the foot of the perpendicular dropped so the direction ratios of the line PO should be proportional to the direction ratio of the normal of the plane of projection.

The Plane is z = 0 and so the normals direction ratio are 0,0,1 , which is proportional to directional ratio of the line PO so O is the foot of the

perpendicular i.e. projection of point P on the plane z = 0.

Similarly we argue for the projection matrix for a point on XZ and the YZ plane.

$$\mathrm{P}_y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad P_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

1.3 Projection and Rotation of a Line

To find projection of a line on a plane we project 2 of it's points say P and Q to A and B then the projection of the line PQ is AB.

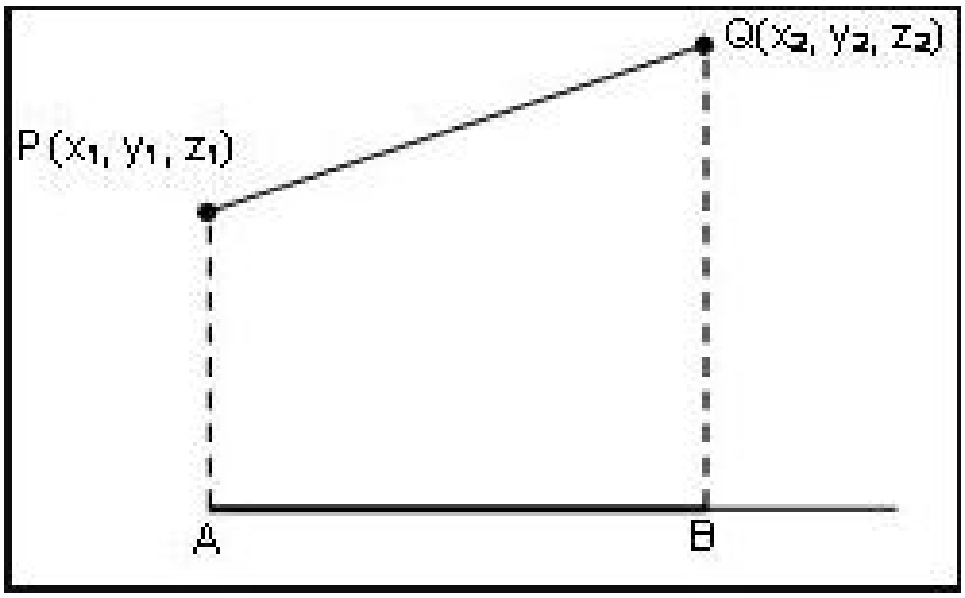


Figure 1.1: Projection of a line

Let the line containing point P and Q be L and it's projection be L‘. As P and Q lie on the line L then the projection of points P and Q i.e. A and B should also lie on the projection of the line L i.e. L‘. So point A and B lie on L‘. We also know that a unique line passes through 2 distinct opoints in the

space so L' is the line joining A and B. So two project a Line L we project two of it's points, P and Q on the pllane and then join the projection of both the points to get the projection of the Line , L‘.

A similar approach is used for finding the rotated version

1.4 Rotation Matrix

1.4.1 Basic rotation in 2D

Rotation depends on an axis of rotation and the angle turned through. Consider first rotation in the plane, about the origin. If a point (x, y) with coordinates

$$\text{x} = \cos\phi, \qquad y = sin\phi,$$

is rotated through an angle θ, then the new position is (x', y'), where

$$\text{x} = \cos(\phi + \theta), \qquad y = sin(\phi + \theta),$$

Expanding these latter expressions, we find

$$\text{x'} = xcos\theta - ysin\theta$$
$$\text{y'} = xsin\theta + ycos\theta$$

or,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Thus the three rotation matrices corresponding to rotation about the z, x, and y axes in $\text{R}^3 are$:

$$R_z = R_z(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{R}_x = R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{bmatrix}$$

$$\mathrm{R}_y = R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix}$$

All three give positive rotations for positive with respect to the right hand rule for the axes x, y, z. If R = R() denotes any of these matrices, its inverse is clearly $R^{-1}(\theta) = R(\theta) = R^T(\theta)$.

Translations and rotations are examples of **solid-body transformations**: transformations which do not alter the size or shape of an object.

1.4.2 General Rotation

Any orientation/rotation can be expressed as rotation about three axis by some particular angle. And so every other rotation matrices can be obtained from these three using matrix multiplication. For example, the product

$R = R_z(\alpha)\, R_y(\beta)\, R_x(\gamma)$

represents a rotation whose yaw, pitch, and roll angles are , and , respectively.

1.4.3 Properties of Rotation Matrix

In this we prove the properties for the basic rotation matrix one could easily prove it general rotation matrix using similar arguments.

$\mathrm{R}(\theta) = \begin{bmatrix} \cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix}$

1)$R(\theta)R(\theta)$ is non-singular with: $R^1(\theta) = R(\theta)$

$detR(\theta) = cos2(\theta) + sin2(\theta) = 1$

$detR(\theta) = cos^2\theta + sin^2\theta = 1$

Because the determinant is not 0, the rotation matrix is not singular

And to see $R(\theta)^{-1} = R(\theta)$ we see calculate it's product,

$$\mathrm{R}(\theta)R(-\theta) = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} = \begin{bmatrix} cos^2(\theta) + sin^2(\theta) & 0 \\ 0 & cos^2(\theta) + sin^2(\theta) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_2$$

2) R$(\theta)^T = R(\theta)$

$$R(\theta) = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

$$R(\theta)^T = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

3)R$(\theta)R(\theta)$ is symmetric for all $\theta = 2k\pi, k\epsilon$.
The diagonal entries are the same, $cos\theta cos\theta$. They pose no constraint.
For the off-diagonal entries, the question is when is $sin(\theta) = sin(\theta)$
The answer is when $sin(\theta) = -sin\theta$ or $2sin(\theta) = 0$? This is satisfied for $\theta = 2k\pi, k\epsilon Z$.

1.4.4 Preservation of a Line between two Points

To find projection of a line on a plane we project 2 of it's points say P and Q to A and B then the projection of the line PQ is AB.

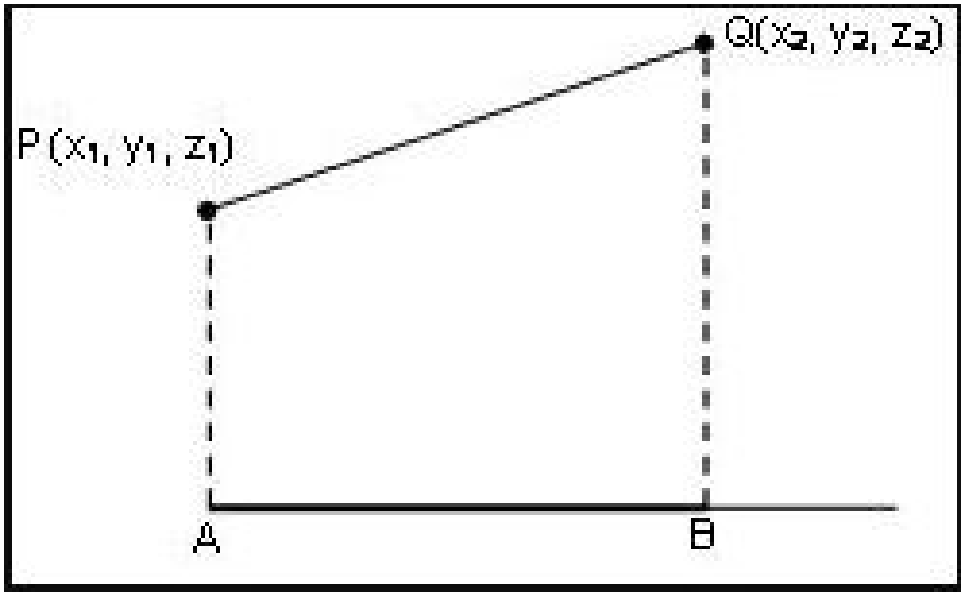


Figure 1.2: Projection of a line

Let the line containing point P and Q be L and it's projection be L‘.
As P and Q lie on the line L then the projection of points P and Q i.e. A and B should also lie on the projection of the line L i.e. L‘.
So point A and B lie on L‘. We also know that a unique line passes through 2 distinct points in the space so L' is the line joining A and B.
Therefore to project a Line L we project two of it's points, P and Q on the plane and then join the projection of both the points to get the projection of the Line , L‘.

A similar approach is used for finding the rotated version of a line L.
If two points A and B lie on a line L then the the rotated coordinates of the points A and B about some axis P, say A‘ and B‘ will also lie on the rotated version of the Line L, L‘.
And as we also know that a unique line passes through 2 distinct points in the space so L' is the line joining A‘ and B‘.
Hence to rotate Line L we rotate two of it's points, A and B around the axis and then join the rotated points to get the rotated version of the Line L, L‘.

Therefore we conclude that to perform projection of rotation of a Line L we perform the corresponding operation on two of it's points and then join them which results to a corresponding operation on the line.

Finding Orthographic Projections from 3D Model

2.1 Modelling of the problem

As described on the previous sections we know how to find projection of point and lines on a plane using elementary Geometry and Matrix concepts.To summarize section 1,

$$\mathrm{P}_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \; \mathrm{P}_y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \; P_z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

are the matrices by which we multiply the position vector of any coordinate C:[x y z] to get it's corresponding projection in YZ , XZ and XY plane respectively.
We have also showed that the projection of a line L containing two points A and B is the same as the unique line joining the projection of points A and B i.e A‘ and B‘ on the plane.
We discussed about the Rotation matrix we need to multiply the position vector of a point to rotate it around any axis as any rotation can be broken down to 3 rotations around the 3 coordinate axis, X , Y and Z. For any general rotation about an axis which can be represented as rotation of angle , , around x, y and z axis respectively, we would multiply it by the three matrices:

$$\mathrm{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos \end{bmatrix} R_y = \begin{bmatrix} cos\alpha & 0 & -sin\alpha \\ 0 & 1 & 0 \\ sin\alpha & 0 & cos\alpha \end{bmatrix} R_z = \begin{bmatrix} cos\alpha & -sin\alpha & 0 \\ sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

And the general rotation will be given by $R_x * R_y * R_z$ multiplied to the position vector of the point we want to rotate.
We also showed that to perform operation on a line we perform it on two of it's points and then join both the points to get the Line.

Now we think of modelling the problem in a mathematically way. From the 3D model we extract all it's vertices and edges present between them.
V is the set of all Vertices in 3D where $V_i = [x_i y_i z_i]^T$ is the i^{th} vertex whose coordinates are $x_i, y_i and z_i$
E is the set of all edges present between the vertices represented as,
$E_{i,j} = [x_i y_i z_i]$ which means an edge between Vertex i and j.
$[x_j y_j z_j]_{2x3}$
We map each view to a number for simple implementation. We map XY plane to 1 YZ plane to 2 and XZ plane to 3.
We have to break this down to 3 sets of Vertices and 3 set of edges for each corresponding view, $V^1, V^2 and V^3$ and $E^1, E^2 and E^3$. Here $V_i^k = [x_i^k, y_i^k, z_i^k]^T$ represents the coordinate of i^{th} vertex in the k^{th} view.
And $E_{i,j}^k = [x_i^k, y_i^k, z_i^k]$ which means an edge between Vertex i and j in the k^{th} view
$[x_j^k, y_j^k, z_j^k]_{2x3}$
Note: For Basic Projection one of the coordinates in V_i^k will be 0 , as the projection are on XY, YZ or XZ plane. All $V_i{}^1, V_i{}^2, V_i{}^3$ will have z, x, y coordinate equal to 0 respectively.

2.2 Basic 2D Orthographic Projections(Bottom, Top and Side View)

After knowing the mathematics behind the problem we now think of an algorithm that solves the problem
For projecting all 3D points to 2D we just multiply the Projection Matrix, P_x, P_y and P_z. For projecting each edge on any plane we project both it ends and then store it together in pair as an edge if both don't project to the same point

procedure projectpointXY()
for (each vertex, V_i)*do*
$V_i^1 < -P_z * V_i$
$V_i^2 < -P_x * V_i$
$V_i^3 < -P_y * V_i$
next V_i
end **procedure**

procedure projectedgeXY( )

for (each edge, $E_{i,j}$) do
$V_1 < -E_{i,j}$ [0,] // first row
$V_2 < -E_{i,j}$ [1,] // second row
$V_1^1 < -P_z * V_1$
$V_2^1 < -P_z * V_2$
$V_1^2 < -P_x * V_1$
$V_2^2 < -P_x * V_2$
$V_1^3 < -P_y * V_1$
$V_2^3 < -P_y * V_2$
if ($V_1^1 \neq$V$_2^1$) then
$E_{i,j}^1[0,] < -V_1$ // set first row
$E_{i,j}^2[1,] < -V_2$ // set second row
end **if**
if ($V_1^2 \neq$V$_2^2$) then
$E_{i,j}^1$ [0,] <- V_1 // set first row
$E_{i,j}^2$ [1,] <- V_2 // set second row
end **if**
if ($V_1^3 \neq$V$_2^3$) then
$E_{i,j}^1$ [0,] <- V_1 // set first row
$E_{i,j}^2$ [1,] <- V_2 // set second row
end **if**
next $E_{i,j}$
end **procedure**

2.3 General 2D Orthographic Projection

Our software would include an interaction for rotating a 3D object by any General Angle and we could output the corresponding orthographic projections.
The Rotation of object could be broken to three axis x, y and z axis with some angle say $\alpha, \beta and \gamma$.
To rotate any point we need to multiply it's position vector with corresponding Rotation Matrices Rx , Ry and Rz.So there is just one change in the above algorithm were we even multiply Rotation Matrix, R_x, R_y and R_z along with the Projection Matrix, P_x, P_y or P_z.

procedure projectpointXY( )
for (each vertex, Vi) do
$V_i^1 < -P_z * R_x * R_y * R_z * V_i$

$V_i^2 < -P_x * R_x * R_y * R_z * V_i$
$V_i^3 < -P_y * R_x * R_y * R_z * V_i$
next V_i
*end***procedure**

procedure*projectedgeXY*()
for(*eachedge*,$E_{i,j}$) do
$V_1 < -E_{i,j}[0,]$ // first row
$V_2 < -E_{i,j}[1,]$ // second row
$V_1^1 < -P_z * R_x * R_y * R_z * V_1$
$V_2^1 < -P_z * R_x * R_y * R_z * V_2$
$V_1^2 < -P_x * R_x * R_y * R_z * V_1$
$V_2^2 < -P_x * R_x * R_y * R_z * V_2$
$V_1^3 < -P_y * R_x * R_y * R_z * V_1$
$V_2^3 < -P_y * R_x * R_y * R_z * V_2$
if ($V_1^1 \neq V_2^1$) then
$E_{i,j}^1[0,] < -V_1$ // set first row
$E_{i,j}^2[1,] < -V_2$ // set second row
end **if**
if ($V_1^2 \neq V_2^2$) then
$E_{i,j}^1[0,] < -V_1$ // set first row
$E_{i,j}^2[1,] < -V_2$ // set second row
end **if**
if ($V_1^3 \neq V_2^3$) then
$E_{i,j}^1[0,] < -V_1$ // set first row
$E_{i,j}^2[1,] < -V_2$ // set second row
end **if**
next $E_{i,j}$
end procedure

Reconstruction of 3D Model from Orthographic Views

3.1 Modelling of the problem

First we separate the drawing into three views before further processing can take place and $_i$ is defined as the i^{th} projection (i = 1,2,3) in the orthographic view system.

By using one of these approaches, the result is the creation of two families of sets:

1. V^i, with i=1,2,3 , represent the three sets of vertices, each one corresponding to a projection in the orthographic view system (i.e. Vi is the set of vertices of π^i).
2.E^i , with i=1,2,3 , represent the three sets of edges, each one corresponding to a projection in the orthographic view system (i.e. Ei is the set of edges of π^i).

On the basis of the six sets described above, '$Vi(j) = v^i_j = [x^i_j, y^i_j, z^i_j]$ represents the j^{th} vertex in the ith orthographic view. By definition, each vertex lies on one of the coordinate planes. Consequently, one of the elements of ^i_j is always equal to zero.

The j^{th} edge in the i^{th} orthographic view is identified by its two vertices ^i_h and ^i_k as follows:

$$e^i_j = \begin{bmatrix} \mathrm{v}^i_h \\ v^i_k \end{bmatrix}$$

The three views can be expressed as follows:
$\pi^1 = [e^1_1, e^1_2, ..., e^1_a]^T_a$

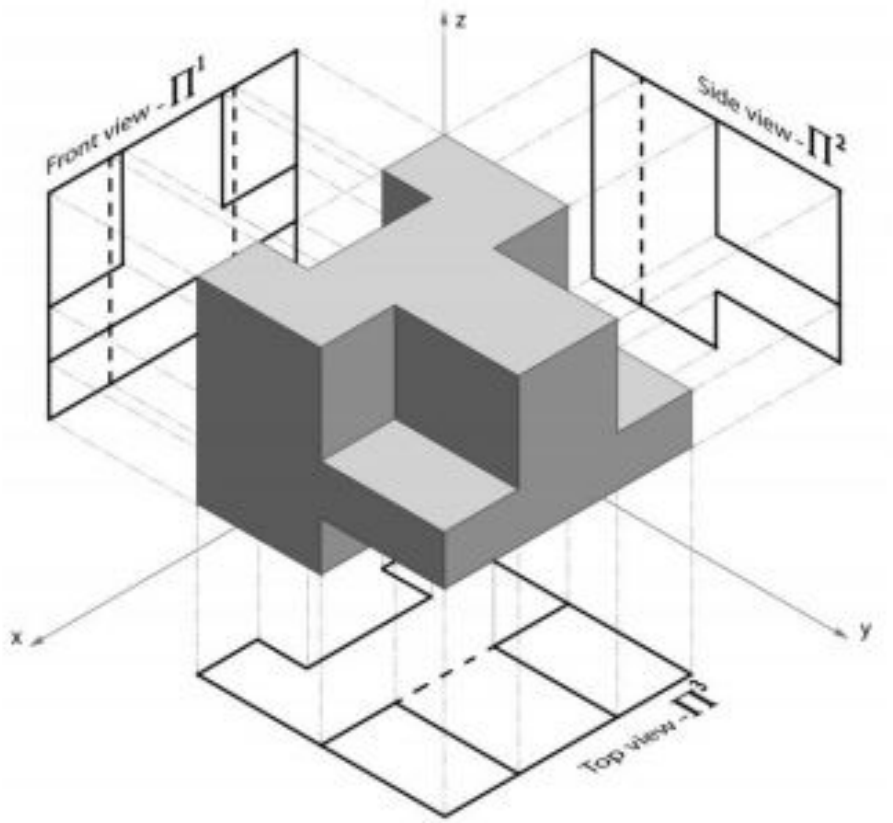


Figure 3.3: 3D Projected Object

$\pi^2 = [e_1^2, e_2^2, ..., e_b^2]_b^T$
$\pi^3 = [e_1^3, e_2^3, ..., e_c^3]_c^T$

where a , b and c are, respectively, the number of edges in π^1, π^2 and π^3 . Finally the set of orthographic projections may be defined as:
OBJ=$[\pi^1\pi^2\pi^3]^T$
where the size of the matrix OBJ is 2a+2b+2c *3.
We also have three more matrices, one for each projection, containing the coordinates of the projection vertices. These structures can be represented as follows:
$V^1 = [v_1^1, v_2^1, ..., v_\beta^1]_{\beta*3}^T$

$V^2 = [v_1^2, v_2^2, ..., v_\gamma^2]^T_{\gamma*3}$
$V^3 = [v_1^3, v_2^3, ..., v_\varphi^3]^T_{\varphi*3}$

Where beta, gamma and varphi are respectively the number of vertices in π^1, π^2 and π^3 .

3.2 Wireframe Reconstruction from Orthographic Views

We have labeled vertices and edges dataset in V1, V2, V3, E1, E2 and E3 from which we need to find the corresponding 3D dataset, V and E.
V_k denotes the k^{th} vertex in 3D space and Ei,j denotes the edge between vertices i and j in 3D space.
The basic idea of the algorithm presented below is that to find the corresponding 3D coordinates of a point say A we find 2 of it's coordinates from two of it's orthographic views and merge them. For example, if point A has coordinates (x, y, z) in 3D space then it's corresponding coordinates in XY and YZ plane will be (x,y,0) and (0,y,z) which when we merge we get it's 3D coordinates back.
We also see that an edge in 2D is reproduced in 3D which will give us all edges but some extra ones too, called pseudo edges which in fact create pseudo faces which we see how to remove in the next section. If in some view we see many points may overlap than many edges may occur between them which can not been seen in this view but in some other view it will be present. It can't happen that an edge is not present in one of the views as it has to be normal to that plane.
As already proved, only 2 views are necessary for reconstruction of 3D object but for now we are considering all three views are given and we cross-check the output.

procedure reconstruct$_v ertices$()
for$(eachvertexin\mathrm{V}^1, V_i^1)$ do
$V[0] < -V_i^1[0]$
$V[1] < -V_i^1[1]$
$V[2] < -V_i^2[2]$//corresponding vertex in YZ plane
$V_1^T < -P_z * V$
$V_2^T < -P_x * V$
$V_3^T < -P_y * V$
if $(V_T^1 V_i^1 or V_T^2 V_i^2 or V_T^3 V_i^3$) then

report Error in Figure
end **if**
next V_i^1
end **procedure**

procedure reconstruct$_e dges()$
for$(eachedgein\mathrm{E}^1, E_{i,j}^1)$ do
$E_{i,j} < -E_{i,j}^1$
if ($E_{i,j}^2$ or $E^3 i,j$ does not exist) then
report Error in Figure
end **if**
next $E_{i,j}^1$

for (each Vertex pair, $V_i^1 x V_j^1$) do
make new $E_{i,j}$ s.t. $E_{i,j}[0] = V_i^1$ and $E_{i,j}[1] = V_j^1$
if ($E_{i,j}^2$ and $E_{i,j}^3$ does not exist) then
report Error in Figure
end **if**
next $V_i^1 * V_j^1$
for (each Vertex pair, $V_i^2 x V_j^2$) do
make new $E_{i,j} s.t. E_{i,j}[0] = V_i^2$ and $E_{i,j}[1] = V_j^2$
if ($E_{i,j}^1$ and $E_{i,j}^3$ does not exist) then
report Error in Figure
end **if**
next $V_i^2 * V_j^2$
for (each Vertex pair, $V_i^3 x V_j^3$) do
make new $E_{i,j} s.t. E_{i,j}[0] = V_i^3$ and $E_{i,j}[1] = V_j^3$
if ($E_{i,j}^1$ and $E_{i,j}^2$ does not exist) then
report Error in Figure
end **if**
next $V_i^3 * V_j^3$
end **procedure** ———————————————————————

3.3 Wireframe to 3D Solid

First an object as is recognised in terms of a wireframe of 3D vertices and edges (skeleton), derived from three 2D orthographic projections with point-wise correspondence. That's what we have done in the last section.

3.3.1 Candidate Faces

We use a minimum-internal-angle searching method to trace all planar edge loops in the skeleton. Starting from a convex vertex (at the extreme left), we ensure that loops have no internal edges.

procedure trace-faces( )
for (each vertex, vi)
for (each pair of adjacent edges, ei,e,) do
n t normal to plane, e; x e,
vk t vertex at far end of e,
while ((vkv,) and (ej exists)) do
ek t edge adjacent to vk,
coplanar to n, with minimum-internal-angle from e, e, = ek;
vk = far end of ek
end while
if (vk=v,) then
list e;e,ek, . . . is a candidate face loop
end **if**
next e,q
next v,
end procedure

Some pseudo elements may be detected when loops fail to close, but there are usual1 y many more still to be found.

3.3.2 Decision Rules

Our method of deleting pseudo elements uses the following rules:

When an edge is adjacent to more than two faces, at most two faces can be true, the rest must be pseudo. (From the MoEbius rule)'.
When two faces intersect, only one of them can be true. (If they were both

true, the intersecting edge would have four adjacent faces which contradicts the MoEbius rule).

An edge which projects onto a dashed line in a view, but which has no candidate face to occlude it, is false. (From the definition of dashed lines on engineering drawings)

When an edge is adjacent to only one face, both the face and the edge are false.

When an edge is adjacent to exactly two coplanar faces, the edge is false and the coplanar face can be merged.

If a true edge is adjacent to exactly two non-coplanar faces, then these faces are both true. (If either of the adjacent faces were false, it would contradict point 4)

A face that is coplanar and adjacent to a true face, when there shared edge projects onto a solid line and is not occlude in a view, is a false face.(The shared edge is needed to project onto the solid line because any other edge would be occlude by the true face. If the candidate face was true, rules 5 and 4 will be applied and the shared edge would e deleted.)

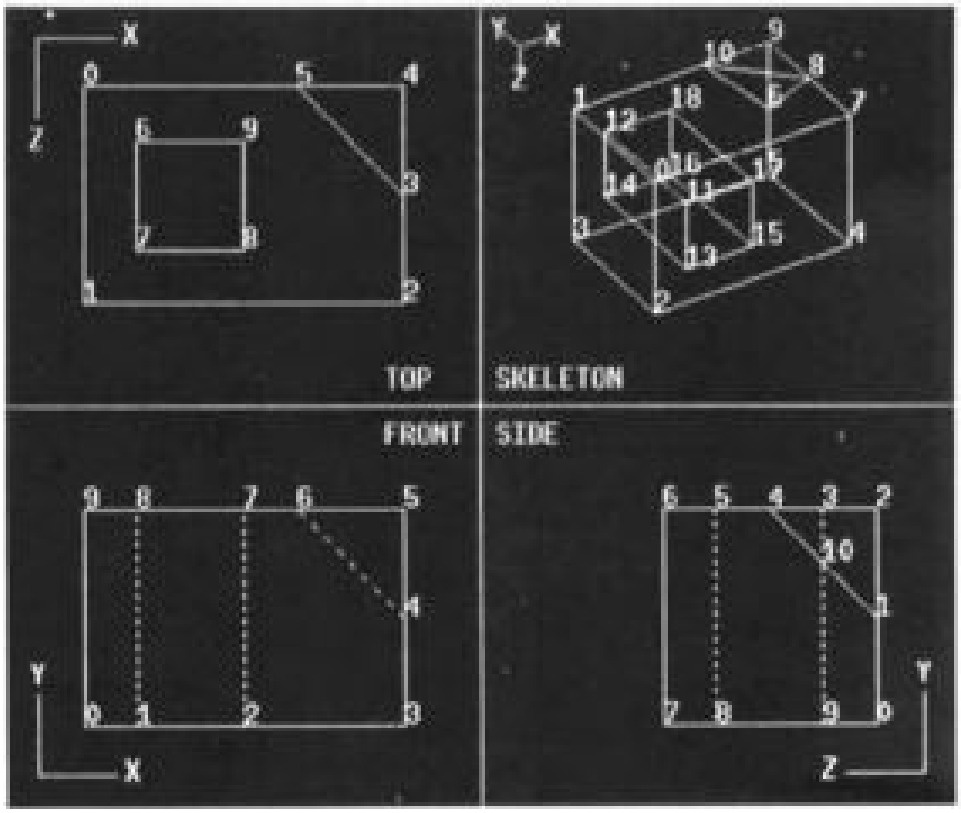
3.3.3 Decision Chaining Algorithm

The decision-chaining method detects pseudo elements in the skeleton and candidate face lists. When undecided edges are found, assumptions are made using rules 1 and 2, then rules 1 to 7 are applied, until either all edges are decided, or no object is found. All possible objects are detected this way, without having to assemble them first. The recursive algorithm is summarised as follows:

procedure decision-chaining( )
if (all edges obey Moebius rule) and (no faces intersect) then
a solution has been found, add it to the list
else
for (each undecided edge, e,) do
for (each adjacent pair of faces, j„ ,fk) do
assume ,fi, ,fk are true;
if (rules 1-7 change anything) then
call decision-chaining( )
end **if**
next,fi,fi

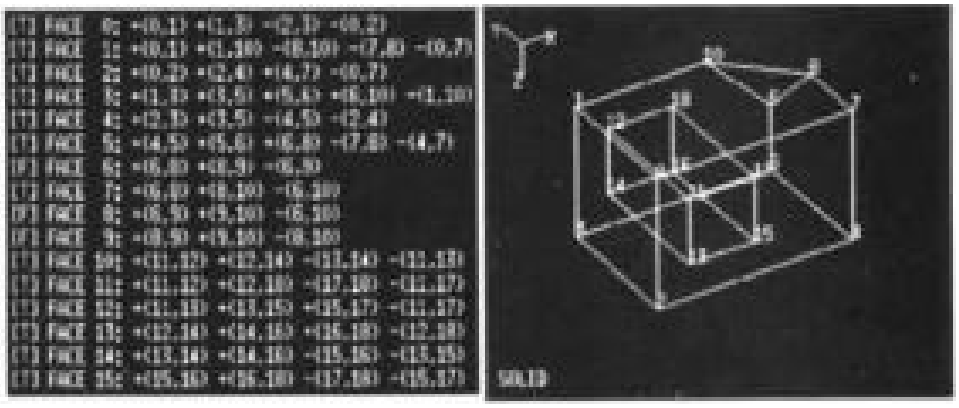
next e;
end **if**
end **procedure**

Using Figure 1(*Page 24*) as an example to illustrate the algorithm, there are sixteen candidate faces found by the minimum-internal-angle method. Initially edge e(6,8) is shared by candidate faces 5, 6 and 7. The algorithm first assumes faces 5 and 6 to be true. However, face 7 cannot be deleted because e(6,8) is a visible edge in the side view, which conflicts with rule 5. Next, faces 5 and 7 are assumed to be true. Now rule 1 says that face 6 is false and the chaining algorithm continues until it detects that faces 8 and 9 are false and a solution is found. Finally, the algorithm back-tracks to assume faces 6 and 7 are true. This fails to find a solution because face 5 cannot be deleted without the loss of visible edge e(4.7) from the side view. Thus a single solution is found, as shown in Figure 2(*Page 24*).



1.JPG 1.JPG

Figure 3.4: Figure 1



2.JPG 2.JPG

Figure 3.5: Figure 2

3.3.4 Containment and Orientation

Sometimes a 3D object includes holes whose faces are contained in another face. This is determined using a sum-of-angles containment test. In general, faces may contain each other in a hierarchical manner. Once the status of one face has been established the remaining can be deduced, they are alternately face and hole, ie, .f,(face) .fi+l (hole) .f,+2(fi cr) 2 . . . , where ,fi is a true face when i is even, and a hole when i is odd. For example in Figure 3, f l2 and f are "hole faces", contained in .f2 and f respectively. Face orientation consistency is maintained by using the MoEbius rule. A face adjacent to a vertex on the convex hull (eg. the extreme left-hand vertex) is used to decide which side is outside. From this, all other faces are oriented by comparing the direction of traversal of their edge loops. The sign of a face plane normal is reversed if adjacent face loops traverse adjacent edges in the same direction. For example, in Figure 3, face ,fo was initially traced using edges in the order vg, vl, vs, v2, but the adjacent facef'l was traced in the order vg, v 1, v 10, vg, v,. Applying the MoEbius' rule to edge e(0.1). results in face fl's plane normal being reversed.