Construction of 2D Orthographic Projection Views from 3D Figure and vice versa

Sushant Rathi, Shashwat Shivam

February 24, 2018

Abstract

This paper deals with the mathematics involved in converting 3D Figures to their orthographic projections and also the methods of reconstruction of 3D figures using the given orthographic views.

Contents

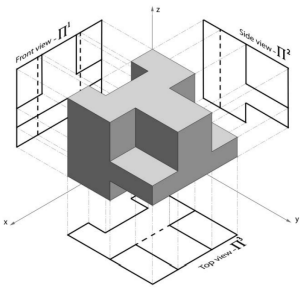


Figure 1: Taking Projections onto different planes.

1 Converting 3D Figure to 2D views

We will be converting 3D figures to 2D views by taking orthographic projections of the constituent points of the figure, and using properties of an affine transformation to connect those points. Let us first understand what the relevant terms mean.

1.1 Orthographic Projections

Orthographic projections are a a form of parallel projection in which the projection lines are orthogonal to the projection plane, resulting in every plane appearing as an **affine transformation** on the viewing surface. (Refer to fig) One useful property of such transformations is that it preserves straight lines, which means that an edge in a 3D projection will be a line segment connecting the projections of its end points after the transformation (unless the points are overlapping). Since we are considering polyhedral solids which can be represented by their constituent edges, being able to project vertices is all we will need to obtain the 2D views.

1.2 Representation of 3D object in Input

We assume the object description is given in terms of the 3D coordinates of the constituent points, and the edge relations (as a wireframe model). While this information is enough for obtaining 2D projections, to obtain **hidden lines** we will need information regarding which points constitute a face as well. We are assuming that the wireframe representation is enough to determine faces, using our algorithm described in the section 4.3 .

2 Straight Lines

Considering the 2 end points of an edge, the projection can be easily taking using a projection matrix. Multiplying both the end points by projection matrices for the specific plane , we can obtain projections onto the 3 planes.

Considering P_{xy} , P_{yz} and P_{xz} to be the projection matrices onto the respective planes and $\boldsymbol{A}$ to be the point to be projected the calculations are as follows :-

$$\boldsymbol{P_{xy}A} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{X_i} \\ \boldsymbol{Y_i} \\ \boldsymbol{Z_i} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x_i} \\ \boldsymbol{y_i} \end{bmatrix} \tag{1}$$

$$\boldsymbol{P_{yz}A} = \begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{X_i} \\ \boldsymbol{Y_i} \\ \boldsymbol{Z_i} \end{bmatrix} = \begin{bmatrix} \boldsymbol{y_i} \\ \boldsymbol{z_i} \end{bmatrix} \tag{2}$$

$$\boldsymbol{P_{xz}A} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{X_i} \\ \boldsymbol{Y_i} \\ \boldsymbol{Z_i} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x_i} \\ \boldsymbol{z_i} \end{bmatrix} \tag{3}$$

Projections can also be taken onto any general plane by using a projection matrix $\boldsymbol{P}$ corresponding to that plane :-

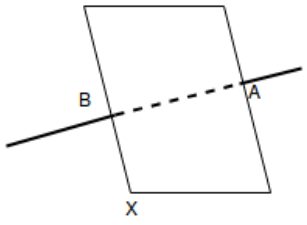
$$\boldsymbol{PA} = \boldsymbol{A'} \tag{4}$$

After taking the projections the points are connected by a straight line.

2.1 Obtaining Hidden Lines

Hidden lines, represented as dotted lines traditionally, are lines that have been occluded in a view by other faces. To determine if a line is hidden or not, we need a method to determine whether a particular face occludes a particular edge or not, and if so what portion of it. We propose the following method for this purpose- If the end points of an edge are enclosed inside the face projection (which will be a polygon), consider only those points for comparison. If an end point of an edge is not enclosed , instead of the end point consider the corresponding point of intersection (Refer to figure) . Now that we have two representative points of the edge (say A and B), our task is to check if these two points lie on the opposite side of the face plane (opposite to the projection plane). To do so, we take any point on the face and consider its projection on the projection plane, call it X. We will now check if A and X are on opposite sides of the face plane or not. Since the edge is not intersecting the face, all points on line segment AB lie on the same side of the plane, and hence we can say that if A and X are on opposite sides, the line segment AB is occluded

To determine if two points A and X are on same side of plane or not, we plug in the coordinates of the points in the LHS of equation of plane $\boldsymbol{ax + by + cz - d = 0}$ and compare signs- the points lie on opposite sides iff the signs are different. For every edge, we iterate over all faces to determine the length of segment occluded, and take a union over all lengths.



Edge occluded from face

3 Rotation of 3D Figures

Rotation of 3D Figures can be interpreted easily as a multiplication of each point by a rotation matrix. After this the 2D views can be regenerated by simply projecting the new points onto the respective planes. Considering $\boldsymbol{R}$ to be the rotation matrix and $\boldsymbol{A}$ to be the point the new point will now be :-

$$\boldsymbol{RA} = \boldsymbol{R'} \tag{5}$$

The projections can also be taken directly from the original co-ordinate system along with the rotation matrix like this :-

$$\boldsymbol{RPA} = \boldsymbol{A'} \tag{6}$$

3.1 Basic Rotation

A basic rotation is the rotation about one of the axes of the co-ordinate system. Assuming the rotation is of an angle of $\boldsymbol{\theta}$ about the x , y and z axis respectively then the resultant co-ordinates of the rotation are as follows :-

$$\boldsymbol{R_x(\theta)A} = \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\cos\theta} & \boldsymbol{-\sin\theta} \\ \boldsymbol{0} & \boldsymbol{\sin\theta} & \boldsymbol{\cos\theta} \end{bmatrix} \begin{bmatrix} \boldsymbol{X_i} \\ \boldsymbol{Y_i} \\ \boldsymbol{Z_i} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x_i} \\ \boldsymbol{y_i} \\ \boldsymbol{z_i} \end{bmatrix} = \boldsymbol{A'_x} \tag{7}$$

$$\boldsymbol{R_y(\theta)A} = \begin{bmatrix} \boldsymbol{\cos\theta} & \boldsymbol{0} & \boldsymbol{\sin\theta} \\ \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} \\ \boldsymbol{-\sin\theta} & \boldsymbol{0} & \boldsymbol{\cos\theta} \end{bmatrix} \begin{bmatrix} \boldsymbol{X_i} \\ \boldsymbol{Y_i} \\ \boldsymbol{Z_i} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x_i} \\ \boldsymbol{y_i} \\ \boldsymbol{z_i} \end{bmatrix} = \boldsymbol{A'_y} \tag{8}$$

$$\boldsymbol{R_z(\theta)A} = \begin{bmatrix} \boldsymbol{\cos\theta} & \boldsymbol{-\sin\theta} & \boldsymbol{0} \\ \boldsymbol{\sin\theta} & \boldsymbol{\cos\theta} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{X_i} \\ \boldsymbol{Y_i} \\ \boldsymbol{Z_i} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x_i} \\ \boldsymbol{y_i} \\ \boldsymbol{z_i} \end{bmatrix} = \boldsymbol{A'_z} \tag{9}$$

Here $\boldsymbol{R_x(\theta)}$,$\boldsymbol{R_y(\theta)}$ and $\boldsymbol{R_z(\theta)}$ are the rotation matrices about the respective axes.

3.2 General Rotation

Any other general rotation can be obtained using the above three rotations . This can be done in a stepwise manner about the three axes. The final rotation matrix $\boldsymbol{R}$ is obtained as follows :-

$$\boldsymbol{R} = \boldsymbol{R_z(\alpha)R_y(\beta)R_x(\gamma)} \tag{10}$$

where $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ represent the yaw , pitch and roll respectively of the rotation.

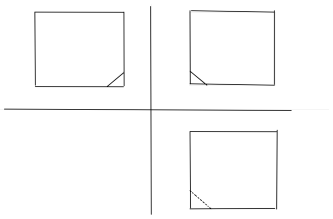


Figure 2: Given Orthographic Projections

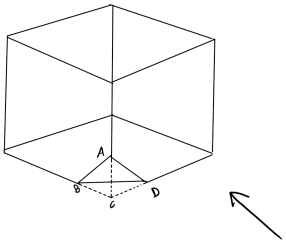


Figure 3: Wireframe reconstructed can have AC,BC and CD present as well

4 Creating 3D Solids using 2D Projections

Our method for reconstruction can be split into two parts :-

1. Constructing a wire-frame model out of the given projections.
2. Obtaining faces out of this wire-frame to define the 3D solid.

4.1 Assumptions

To perform these steps in an efficient and correct manner, we have placed the following restrictions on the input provided :

1. **We will be dealing with polyhedral solids, and hence only planar surfaces**
   This is mainly because we do not have a general method for reconstructing/representing quadratic surfaces. We could however extend our method to simpler objects like cylinders later on.
2. **The input projections will consist of labelled vertices**
   This is because otherwise the wire-frame model obtained after step 1 can contain many pseudo elements, in the sense some of the edges/vertices may not exist in the actual solid (Refer Figure 2 and 3). Hence the algorithm for face reconstruction does not guarantee correctness- it mainly requires using some heuristics like ensuring at least one edge from certain sets exist in the final solid, edges do not intersect faces, etc.

4.2 Step 1: Constructing a wire-frame model out of the given projections

This process is fairly straightforward. Since every vertex is visible in every orthographic view (overlapping vertices as well), one orthographic view gives us 2

coordinates of a vertex. Hence, 2 orthographic views are sufficient to give the 3D coordinates of a vertex. To construct a wire-frame model, however, we need edge relations as well (i.e. which vertices are connected directly). Any edge in an orthographic view can be divided into two categories- either it is perpendicular to the plane of projection (in which case its projection is a point), or it is not (in which case its projection is a line segment). Thus, the existence of an edge AB implies that in each of its projections, A and B are connected either by a line segment or are overlapping, and at most one view can have overlapping points.

To avoid having to check every possible pair of points for edge relations, we will pick a particular view (say top), and consider only those pair of points which have an edge connection in this particular view. We shall then check the other view for existence of line segment connecting these two points (or whether they are overlapping).

Note that there is a possibility that there exist line segments in the given views between two points (say A and B) that are corresponding to different edges, that is, edges whose end-points overlap with A and B in every view. To deal this possibility, we are assuming that the input provided will indicate when there are multiple line segments overlapping in a view, in terms of relative thickness of the lines. Also, in case of overlapping points, the points will be ordered in terms of their distance from projection plane.

4.3 Step 2: Obtaining faces out of the wire-frame to define the 3D solid

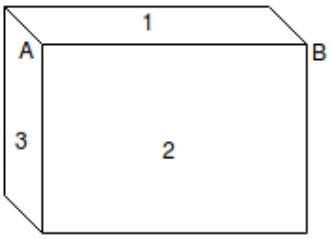
This algorithm, broadly speaking, consists of forming a graph with planar edge loops acting as nodes, and shared edges acting as connections between the nodes (the connection will be labeled by the name of the shared edge between the two faces). Shared edge here, as the name suggests, is an edge that belongs to both edge loops. These edge loops cannot cross each other, that is, the respective edges of any pair of edge loops can either be overlapping, or intersect at their end-points. The nodes of this graph must now be coloured as valid or invalid, with certain constraints. Before stating the constraints, let us define an edge incident set.

Definition 1. *An edge incident set of an edge is defined as the set of valid planar edge loops containing that edge.*

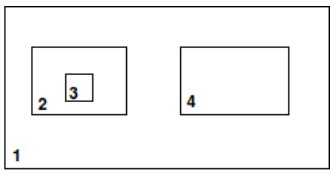
Before defining what valid nodes (edge loops) are, let us recognize the fact that only looking at shared edges is not enough to determine if an area bounded by edge loops is a face or not. This is because a portion of area of a loop can be 'cutoff' by another edge loop inside, due to the existence of a hole/extrusion.

To formalize this, let us make a few definitions regarding edge loop containment.

Definition 2. *An edge loop X is said to be contained inside another edge loop Y iff all vertices of X lie inside the area bounded by Y.*



1 and 2 are valid planar edge loops sharing edge AB



Edge loops 2 and 4 are predecessors of 1, 3 is not

Definition 3. *If an edge loop X is contained in Y, and there exists no edge loop Z such that X is contained in Z and Z is contained in Y, X is said to be predecessor of Y.*

Now let us define what a valid edge loop is.

Definition 4. *A planar edge loop X (graph node) is said to be valid if the region between X and all its predecessors forms a face.*

It is easy to see that if X is valid, all its predecessors will be invalid, as both being valid would imply that the edges forming its predecessors have faces on both sides, which would make that edge redundant. Also, if X is invalid, all its predecessors will be valid, otherwise the third constraint stated will be violated.

Thus, if we construct a poset of edge loops defined on the relation of containment, we can say that for any poset S with a least element, determining the validity of any edge loop belonging to S will determine the validity of every other edge loop belonging to S. [Gal00, see chapter 4]

Given the previous definitions, the constraints we must follow in our reconstruction are-

1. The cardinality of every edge incident set is exactly two.
2. No two edge loops belonging to an edge incident set share the same plane.
3. No two edge loops not belonging to an edge incident set of an edge E, but containing E, can be planar.

These constraints follow from the fact that the faces form a 2-manifold without boundary. [CE00]

Thus, the construction of edge loops will proceed as follows.

1. We will first divide all the edges into sets of co planar edges. After this, for every set we will obtain a set of edge loops. Assuming we have edge adjacency relations (by common vertices), this can be done by simply starting at an edge and running a depth first search from it, and repeating the process for an edge not visited.

2. Once this is done, for the set of edge loops, we construct posets defined on the containment relation as defined earlier. This poset will be useful later as determining validity of any element of a poset determines validity of every other element as well.

3. Now for every edge E, we consider its the set of edge loops it is a part of (call it S(E)). If the cardinality of S(E) is 2, both edge loops must be valid (this follows from first constraint). Thus, we shall first consider only those edges that satisfy this condition. (If the 2 loops are coplanar, we have violated the second constraint, and hence a 3D solid is not possible). For every edge loop X discovered as valid/invalid, we now determine the validity of edge loops of the poset which X is a part of. After every new discovery, we enforce constraints 2 and 3, which means that if two edge loops belonging to S(E) are planar, exactly one of them is valid.

4. We repeat the process for the remaining edges, where we maintain count of the elements belonging to its edge incidence set. If we cannot find any edge for which the condition stated in 3 holds, we will have to perform branching. This means we will consider a particular edge loop to be valid, and repeat steps 3 and 4 from then onwards.

5. If we reach at an impossible state from this branching, we backtrack, reversing the last decision we made. We terminate the process either when we have managed to colour all the nodes (and not violated any constraint along the way), or we have looked at all possibilities but have not arrived at an admissible solution.

References

[CE00] Robert Cipolla and Ralph Martin (Eds). *Proceedings of the Ninth IMA Conference on the Mathematics of Surfaces*. Springer, 2000.

[Gal00] Jean Gallier. *Discrete Mathematics*. Springer, 2000.