Mathematical Modelling

COP 290 - Design Practices

April 5, 2018

1 Isometric projection to Orthographic projection

Our aim is to take a 3D or its isometric projection and effectively apply transformations to make its orthographic projections. For this we have stated the required theorems/Formulas and then have stated and proved our working algorithm.

1.1 Theorems/Formulas used

Theorem 1.1.1. Matrix for shifting/translation a point (x,y,z) in space with (a,b,c) to (x-a,y-b,z-c) is :

$$S = \begin{bmatrix} 1 & 0 & 0 & -a \\ 0 & 1 & 0 & -b \\ 0 & 0 & 1 & -c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Proof. A translation is applied to an object to re-position it along a straight-line path from one coordinate location to another. Take a point (x,y,z) $\in \Re^3$. We need to translate it by (a,b,c) respectively. Consider the multiplication

$$\begin{bmatrix} 1 & 0 & 0 & -a \\ 0 & 1 & 0 & -b \\ 0 & 0 & 1 & -c \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} (1*x)+(0*y)+(0*z)-(a*1) \\ (0*x)+(1*y)+(0*z)-(b*1) \\ (0*x)+(0*y)+(1*z)-(c*1) \\ (0*x)+(0*y)+(0*z)+(1*1) \end{bmatrix} = \begin{bmatrix} x-a \\ y-b \\ z-c \\ 1 \end{bmatrix}$$

This shows that the matrix S can effectively translate a point in space through a distance (a,b,c).

□

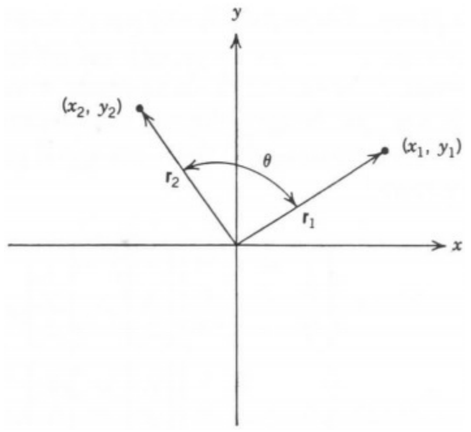
Theorem 1.1.2. Rotating a point (x,y,z) about z axis uses a rotation matrix :

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

Proof. Take a point (x,y,z) $\in \Re^3$. We need to rotate it by Θ anticlockwise about Z-axis. We would first get a 3*3 matrix and then add a column and a row to make it 4*4. Note that z coordinate will remain unchanged so we fix Row 3 and column 3 as:

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

Now problem of finding the missing four coordinates can be reduced to 2 coordinate system. Take 2 points P(X_1,Y_1) and Q(X_2,Y_2) and an angle θ between them. We need to rotate (counterclockwise) and transform point P to point Q.



From the figure we see that-

$$X_2 = X_1 \cos\theta - Y_1 \sin\theta$$
$$Y_2 = X_1 \sin\theta + Y_1 \cos\theta$$

This thing can also be achieved by:

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} * \begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} = \begin{bmatrix} X_2 \\ Y_2 \end{bmatrix}$$

Total matrix in 3*3 will become:

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

Extending it to 4*4 will make it :

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

This matrix was for Anticlockwise rotation and we can change it to clockwise rotation simply by changing θ to -θ By similar measures we get rotation matrices about X and Y axis as:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

□

Theorem 1.1.3. Rotating a point (x,y,z) about an axis in anticlockwise direction by angle θ is same as rotating the coordinate system about that axis in clockwise direction by the same angle θ

Proof. Consider two systems. In system 1 a point is rotated about the Z axis anticlockwise by an angle θ. In system 2 the same point is rotated about the Z axis clockwise by the same angle θ.

Consider the final state of the systems. The angle subtended by the radius vector of the point and the Z axis remains same in both the cases, as that didn't change by the transformation. It is easy to see that the angles by the X and Y axes also remains the same as **relative** radial displacement is same in both the cases.

So it won't be wrong to say that the final coordinate of the point is same in both the cases. □

Theorem 1.1.4. Rotating a point (x,y,z) about a general axis passing through (x_0, y_0, z_0)

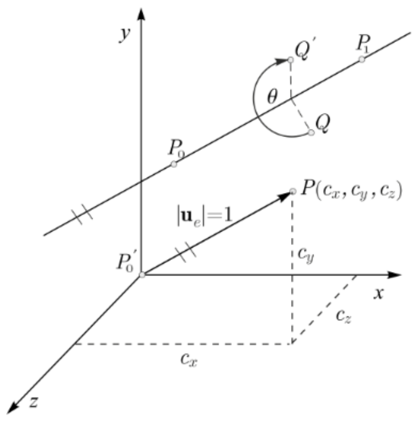


Figure 1: Figure showing shifting of axis

Proof. **Procedure** :

1. Translate (x_0, y_0, z_0) so that the point is at origin
2. Rotate space about the x axis so that the rotation axis lies in the xz plane

3. Rotate space about the y axis so that the rotation axis lies finally along the z axis
4. Rotate about the z-axis by the angle θ.
5. Perform the inverse of the combined rotation transformation.
6. Perform the inverse of the translation.

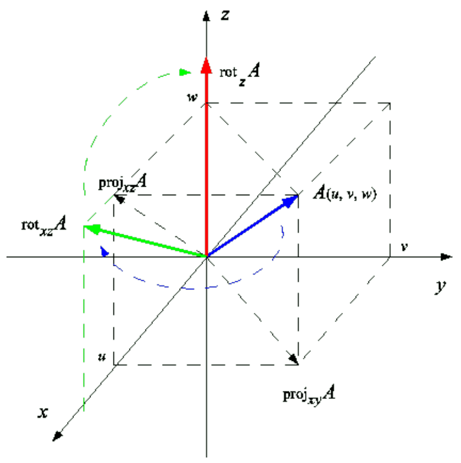


Figure 2: Figure showing rotation of axis to coincide it with Z-axis

For the simplicity we compute the u = $P_1 - P_0$ vector, which after the normalization can give us the direction cosines of the axis:

$$U_e := u/|u| = (c_x, c_y, c_z).$$

Then we have $c_x^2 + c_x^2 + c_z^2 = 1$ and $\cos\theta_x = c_x$, $\cos\theta y = c_y$ and $\cos\theta_z = c_z$.
Let the translation matrix be

$$T(a) = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Next step requires 2 successive rotations of coordinate axes about X-axis by angle θ_x and about Y-axis by θ_y.(anticlockwise)
Using theorem 1.x, it can be stated that the rotation matrix would be same as rotating the point about the corresponding axes CLOCKWISE by appropriate angles.
Let these rotation matrices be R_x(-θ_x) and R_y(-θ_y) and these were defined in Theorem 1.2 where

$$\sin\theta_x = cy/d \qquad \cos\theta_x = cz/d$$

$$\sin\theta_y = d \qquad \cos\theta_y = d$$

Note that the negative sign accounts for the direction change in angle as mentioned above.

Now, we have made the axis along the Z-axis and we can easily rotate the point about z axis by the required amount θ as defined in Theorem 1.2. Let it be defined as R(θ)

The final step is to take the required (R^{-1}(.)) inverse transformations of the matrices to replace the object/point at its original place.
The final transformation we get is:

$$M = T^{-1}(a)R_x^{-1}(-\theta_x)R_y^{-1}(-\theta_y)R_z(\theta)R_y(-\theta_y)R_x(-\theta_x)T(a)$$

□

Theorem 1.1.5. Matrix for projections on X-Y plane, Y-Z plane or Z-X plane

Proof. A projection matrix for orthographic projections is used to project an object on the X-Y/Y-Z/Z-X plane. The basic function is to project the coordinates of the object on a screen behind without any scaling or distortion.

Take a point (x,y,z) $\in \Re^3$. We need to translate it to by (X,Y,0) for a projection in X-Y plane. Notice that if we directly project the point on X-Y plane we are just diminishing the z value to 0 keeping the X-Y constant i.e (X,Y,0) = (x,y,0)
Consider the multiplication

$$S_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} (1*x)+(0*y)+(0*z)+(0*1) \\ (0*x)+(1*y)+(0*z)+(0*1) \\ (0*x)+(0*y)+(0*z)+(0*1) \\ (0*x)+(0*y)+(0*z)+(1*1) \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

This shows that the matrix S can effectively project a point in space on X-Y plane.
Similarly, there are projection matrices for projecting on Y-Z plane (S_x) and X-Z plane (S_y) too i.e.

$$S_x = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} S_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

□

Theorem 1.1.6. Matrix for isometric projection

Proof. There are eight different orientations to obtain an isometric view, depending into which octant the viewer looks. The isometric transform from a point $a_{x,y,z}$ in 3D space to a point $b_{x,y}$ in 2D space looking into the first octant can be written mathematically with rotation matrices as:

$$\begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} * \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix} * \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}$$

First we rotate the point a along the vertical by α, followed by rotation along the horizontal by θ. $\theta = \arcsin(\tan 30^o)$ and $\alpha = 45^o$
Now all we have to do is to take the projection along the X-Y plane.

$$\begin{bmatrix} b_x \\ b_y \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix}$$

□

1.2 Algorithm

1.2.1 Input

Our model assumes that we will be given a 3-D object. For that object we would have a graph with nodes specifying the (x,y,z) tuple and there would be edges if there is an edge between those two points in 3D object. So, basically we would have a list of nodes with edges between them stored in a data structure. We expect every point to be lablled that is intersection of 3 or more edges. We would also get an input information about the direction of viewing or the front side of the object specified by a plane and an angle.

1.2.2 Output

Output would be again a graph with points labeled with a tuple (x,y)/(y,z)/(z,x) according to the plane of projection. We would also label them with the correspondence of points with the 3D figure which will help maintain the information required to reconstruct 3D object or the isometric view.

1.2.3 Procedure:

1. Loop over all the nodes in 3D space and perform following tasks on them
2. Perform translation to bring global origin to one of the vertices of 3D object.
3. Rotate in such a way to bring the view direction same as front-view direction i.e perpendicular to the X-Y plane. As this is a rigid body all points will go about the same amount of rotation. Rotation will be done as specified in **Theorem1.1.2**
4. Project the point on X-Y, Y-Z and Z-X planes by the method specified in **Theorem1.1.5**
5. Store all the projected points in a graph with labels as a two elemental tuple (x,y)/(y,z)/(z,x) make edge between the vertices that are connected in 3D space too.

1.2.4 Uniqueness

Suppose that the algorithm we follow is not unique. Consider the sequence of matrix transformations to be represented as a a single matrix R as this is nothing but a product of

transformation matrices in some order. Take a vector (x,y,z) = v and apply the transformation R to it to get some v'.
If our algorithm is not unique then it means that there would be another v" that we can get after applying the same transformation.

$$R * v = v' \qquad R * v = v"$$

If this is the case then on subtracting the two equations we get $v' - v" = O$ which is not possible as by assumption v' $\neq$ v" and hence a contradiction. So, our algorithm gives unique projections.

1.2.5 Time Complexity

Say, we have n points in total. Then we are traversing on each point and for each point we are multiplying it to a transformation matrix which is a constant order operation. So, total time complexity will be C*N $\approx$ O(N).

1.2.6 Termination

Algorithm doesn't involve any structure that will hang it in mid process or infinite loop, neither it involves any division of matrices but just involves product of matrices that too for N times. Thus, after projecting each point and taking the information of edges between nodes we would effectively be able to generate a 2D projection for our 3D object.

2 Orthographic projection to Isometric Projection

Our aim is to reconstruct 3D or its isometric projection from its orthographic projections. For this we have stated the required theorems/Formulas and then have stated and proved our working algorithm.
Our algorithm generates a wire-mesh model from input where the corresponding points in different views have been labelled. But we also provide strategies on how we can work with unlabelled data and generate solid model from the wire-mesh. Then we also propose a strategy to target models with dotted or hidden lines.

2.1 Theorems

Theorem 2.1. Two nodes are connected in 3D graph if they are either connected in all the 2D views or the line joining them is along the direction cosines of one of the view planes and they are connected in the remaining two views

Proof. If two points are connected in a 3D object, and the connecting line is not along the direction cosines of any of the three view planes, then that edge would be present in all the three views.
If that line is along the direction cosines of some view plane, then the presence of that line would be captured in the remaining two planes. □

2.2 Algorithm

2.2.1 Input

Our model assumes that it receives data of three views. For every view, we have a labelled graph, by which we mean that each node is a tuple with three elements: first two elements define its position in the 2D graph and the third element is the label which helps us to find point to point correspondence among different views.
The information about the edges is assumed to be present in the adjacency list format for each view.

2.2.2 Output

The output is 3D graph, which consists of a set of nodes where each node is 4 element tuple (one for x coordinate, one for y coordinate, one for z coordinate and one for the unique id). The information about the edges is presented using the adjacency list.

2.2.3 Procedure

1. **Store the labels** Each point is present in three different views and so each point has been represented by three different 2 member tuples. Traverse over all the points in each view separately and for each label store its three representations in an array. This will generate an 2D array of Nx3 dimensions.

2. **Generate nodes of output** For each label we have three coordinates of type (x,y),(y,z),(z,x). Each pair of coordinates must have one element common, and so the triplet of pairs gives us the x, y and z coordinates. Note that if they don't have one element common, then there exists some error in the input.

3. **Generate edges of output** For each node i of output graph, traverse over the rest of the nodes. Use theorem 2.1 to update the adjacency list.

2.2.4 Uniqueness

Our algorithm will always give the same solution for a specific input.
Two inputs will give the same answer as long as their 2D graphs are same and correspondence is shown correctly. This is because we finally do the work in such a manner where one node is compared with all the other nodes (to check for connection). Hence the order in which we iterate over the nodes doesn't matter. So two same graphs with different inputs still give the same output.
Our algorithm will give a unique solution even if only 2 views are provided as input. Since we finally output a 3D object, there may exist some more features in the third dimension which cannot be given as input using only 2 views.

2.2.5 Time Complexity

First we traverse over all the points in each view separately doing $3 * N$ operations.
Then for we each node we traverse over the rest of the nodes thereby doing N^2 operations. Hence our algorithm is $O(N^2)$.

2.2.6 Termination

The algorithm first traverses finite number of times over the node set and then for each node traverses over the rest of the nodes which is again finite. It never waits for any condition to be fulfilled and always proceeds in one direction. Therefore it will terminate for sure.

3 Hidden Lines

3.1 Generate Polygons

1. **Generating surfaces** The list of edges is searched for a pair of edges with a common node. The equation of plane passing through these three points is given by

$$\begin{bmatrix} x - x_1 & y - y_1 & z - z_1 \\ x_1 - x_2 & y_1 - y_2 & z_1 - z_2 \\ x_2 - x_3 & y_2 - y_3 & z_2 - z_3 \end{bmatrix} = 0$$

   This equation is divided by the coefficient of x so as to normalize the equation. Duplicate planes are then removed.

2. **Generating boundary loops** The boundaries of each face is determined by examining the co-planar edges. Consider a surface and enumerate the list of all the edges lying on that surface. Choose any one of them as the starting point. Carry out a traversal from one edge to another. If many edges can branch out from the current edge, select the one that makes the smallest angle in the clockwise direction. The loop is complete when the starting edge is traversed in the same direction again.

3.2 Create hidden lines

1. Now that we have information about the faces of the 3D graph, we can easily check whether a vertex is covered by a face or not.

2. Initially consider all edges to be solid.

3. If any point is hidden by a face, we mark all edges connected to that point as hidden.

4. How do we check if a point is covered by some face or not? First we check if the point is behind some plane, for which we take another point with same x and y coordinate and sufficiently negative z coordinate. If these two points when put in the plane's equation yield same sign then our original point is behind the plane. To check if the point is within the boundary of the face, we consider rays emitting from this point. If they intersect the plane at odd number of places, then the point was within the plane.

By –
Atishya Jain (2016CS50393)
Mayank Singh Chauhan (2016CS50394)
IIT Delhi