COP 290
Mathematical Formulation

Ansh Sapra - 2016CS50392
Arshdeep Singh - 2016CS50625

Contents

3D to 2D

Given a 3D model description of any polyhedron we need to generate projections of the model on to cutting plane. So we would need an efficient representation for the 3D model which can be easily manipulated and then used to find projections of the model on to any cutting plane.

1.1 3D Representation

In the 3D world, there are 3 dimensions so we need 3 parameters to represent any point in a 3D space. So we can use the Cartesian Coordinates to store or access any vertices of the 3D drawing provided to us. So we can represent any point using 3 parameters namely x, y and z coordinates. We can store the information about the edges of the 3D drawing by forming an edge set that stores information about the vertices that have an edge between them. We will use this the coordinate system to store the information of the 3D drawing as it is very easy to manipulate/transform these parameters as required for different actions like taking projections of the object, translating or rotating the object, scaling the object etc. by elementary matrix multiplications as shown later in the report.

1.2 Transformation

Transformation is the process of mapping a shape or object to another according to a well-defined rule which is intended to be performed on the object given in the 3D drawing. We can represent transformation of the whole object by mapping the individual points of the object to a different point (These points here would be the vertices of the 3D object). This is useful as we can transform any line segment according to a desired rule by just applying the desired rule on to the end points of the line segment and then by joining the transformed points together.

We may want to apply multiple simple transformations to a single object so as to perform a complex transformation (eg. rotation about an arbitrary point as shown below). We can do this by writing functions for different simple transformations and storing the sequence of labels of transformations that are to be applied in an array, and later looking at each label and invoking the corresponding function. But this is not an efficient way since the array may consume a lot of memory when complicated

transformation is to be applied. So we will use the Matrix representation for doing the transformations (multiplying a specific matrix to the initial matrix so as to get the transformed coordinates in the matrix resulting from the product of these two matrices) as complex transformations would be the product of the individual transformations.

1.2.1 Translation

Translation refers to the shifting of origin from one point to the other. Translation may not be required directly for the current assignment but it is an essential transformation which is used for rotating any point about any desired line in space, because it is easy for us to rotate about the coordinate axis which pass through the origin. So, we first translate the system such that the origin lies on the line about which the rotation is desired, perform the rotation and translate the system back to the original origin that we started off with.
Translation can easily be implemented by subtracting a 3×3 matrix from the 3×3 matrix containing the coordinates of the point to be translated since it only involves adding constant factors to the coordinates. But this would violate our principle to represent different transformations in a similar fashion and we would not be able to concatenate them to form more complicated transformations. So we design a 4×4 matrix to be multiplied to the 4×1 matrix $\begin{bmatrix} x, y, z, w_0 \end{bmatrix}$ containing the three coordinates and one dummy variable, w_0 (or a constant, eg. 1, as used below).
Considering t_x, t_y and t_z as translation factors for each coordinate respectively, we get the following translation matrix.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

Applying it gives us,

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} = \begin{bmatrix} x + t_x & y + t_y & z + t_z & 1 \end{bmatrix}$$

which is the same as what we had expected after assigning dummy variable w_0 to be 1.

1.2.2 Rotation

Since we need to generate projection on to any arbitrary cutting plane, so we will rotate the given object or 3D drawing according to the plane on to which the projection is required so that we always need to take projections on one plane only (say XY plane for instance). This means that instead of choosing the plane on to which the projection is

to be made we will first rotate the object or drawing such that it's projection on to the XY plane after rotation would be same as that of the object or 3D drawing on to the desired plane.

Rotation about X, Y and Z axis

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

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

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

Where θ is the angle by which the rotation is done in the counter-clockwise direction. R_x, R_y, R_z are rotation matrices for rotation about the X, Y and Z axis respectively.

Rotation about an Arbitrary Line

Rotation about any arbitrary line is done in multiple steps. We first translate the system to one of the points lying on the given line. transform the axis (line around which the rotation is done, not the x, y, or z axis) to some position or orientation where we know how to rotate about the axis. Next, we rotate about the axis and finally use inverse transformations to bring the axis back to its original position and orientation.

Suppose the axis is specified by the following equations:

$$\begin{aligned} x &= Au + x_1 \\ y &= Au + y_1 \\ z &= Bu + z_1 \end{aligned}$$

where [A,B,C] is the vector indicating the direction of the line. From the equations, we are sure (x_1, y_1, z_1) is on the line. Therefore we translate the axis by the factors -x_1, -y_1

and -z_1 such that the line passes through the origin. The matrix for doing this is given below and denoted as T:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_1 & -y_1 & -z_1 & 1 \end{bmatrix}$$

and to get back to original position we would take it's inverse matrix T^{-1}. The next step is trying to rotate the axis of rotation around x-axis until the axis of rotations is in xz-plane. However, we do not how much the rotation angle about x-axis is. To figure that out, imagine we make a parallel projection of the line onto the yz-plane. The direction of projection is parallel to x-axis. Since the direction vector of the line is [A, B, C] and we have moved it such that it passes origin, the line segment L1 from point (0, 0, 0) to (A, B, C) is on the axis of rotation. The projected point of (A, B, C) will have coordinates (0, B, C). Denote the line segment from origin to (0, B, C) as L_2. The length of this segment is then

$$V = \sqrt{B^2 + C^2}$$

When we rotate L_1 around x-axis, segment L_2 is also rotated. The angles of each rotation are equal. That is, to find out the amount L_1 needs to rotate about x-axis to land in xz-plane, we can instead find the angle of rotation of L_2. Moreover, we actually do not need to know how much this angle is. Since in all the rotation matrices, only the cosine and sine of an angle is used, we only need to obtain those two values instead of the actual angle.
Therefore, by applying trigonometry in yz-plane we get

$$\sin I = \frac{B}{V}$$

$$\cos I = \frac{C}{V}$$

where I is amount that L_2 needs to rotate about the origin in yz-plane, and since value of B, C, and V are known the matrix R_x is as follows,

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C/V & B/V & 0 \\ 0 & -B/V & C/V & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Suppose the final target matrix we want to achieve eventually is M. Up until now, we have $M = TR_x$. This temporary value of M is the same as saying our axis of rotation is lying in the xz-plane. The next step is rotating about y-axis so that it overlaps z-axis. Then the rotation about the line is the same as rotation about z-axis. Denote the line

segment in xz-plane after rotating L_1 about x-axis as L_3. Its length N does not change after applying T and R_x. Therefore,

$$N = \sqrt{A^2 + B^2 + C^2}$$

If we have to rotate about y-axis for angle J, then,

$$\sin J = \frac{A}{N}$$

$$\cos J = \frac{V}{L}$$

and the matrix R_y for such a rotation would be

$$R_y = \begin{bmatrix} V/N & 0 & A/N & 0 \\ 0 & 1 & 0 & 0 \\ -A/N & 0 & V/N & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Finally, we are going to rotate about z-axis for angle θ. Since now the axis of rotation is the same as z-axis, rotating about z-axis is the same as rotating about the line. The matrix R_z for this is,

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

Putting all of this together we can get the matrix M to rotate about any arbitrary line as

$$M = TR_xR_yR_zR_y^{-1}R_x^{-1}T^{-1}$$

1.3 Projection from 3D to 2D

Equipped with the tools of 3D transformation we can now map a 3D object on a 2D screen. This process can also be called projection. Projection can be broken down into several different transformations. And once we know the matrices for projection, what is left is that by applying matrix algebra, we can get the coordinates for drawing on screen.

Since, on a 2D screen we just need two numbers to specify the position of a point and in 3D we use three numbers to specify the position of any point. One intuition, is that we could use a 3×2 matrix for the purpose of projection.
However, although this approach is doable, using a non-square matrix introduces problem when we want to combine transformations as matrix multiplication requires the

number of columns in the previous matrix being equal to the number of rows in the next. Also, some transformations like translation will pose problems as its matrix is not compatible with a matrix having dimensions other than 4×4. And translation matrix is certainly used in the construction of certain type of projection. Therefore, we will continue using 4×4 matrices as our representation of transformation and projection and we will augment the coordinates in 2D and 3D worlds to have dimension 4. That is, in 3D, we keep using the coordinates with dummy coordinate w_0. And in 2D, we use (x, y, z, w_0) as the coordinates even though z may not be used.

There are two major types of projections, namely Parallel projection and Perspective projection, but the projection used for the purpose of engineering drawings is Parallel projection.

A parallel projection is formed by extending parallel lines from each vertex on the 3D object until they intersect the screen. The point of intersection is the projection of the vertex. We connect the projected vertices by line segments which corresponds to original object.

Suppose xy-place is the place where we want project the object of interest. So we basically have to drop the z coordinate of any given point to get it's projection on to the xy-plane. As discussed above we will do this by multiplying the matrix containing the three coordinates and a dummy variable with a 4×4 matrix as shown below:

$$P_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Applying it gives us,

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x_p & y_p & 0 & 1 \end{bmatrix}$$

Where x_p and y_p are the coordinates of the point having x, y and z as the coordinates in the 3D space after projection onto the xy plane.

Similarly we may have

$$P_{yz} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P_{zx} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

As stated earlier to get a projection on to any oblique plane we would first rotate the given 3D model such that the required direction of viewing becomes parallel to on of the coordinate axis (say Z axis) and would then take the orthogonal projection of the model onto the corresponding plane (xy plane).

2D to 3D

Given multiple orthographic views of a 3D object we need to reconstruct the 3D model of the object. In this we may consider that we are given 3 orthographic views of the object to be reconstructed.

2.1 2D representation

In this problem we may consider that we are provided with labelled orthographic views with the corresponding points labelled accordingly in the 3 orthographic views (eg A, A', A''; B, B', B'' etc.). So that we may get the 2 coordinates of each point from any of the projections. That is for a point named A, we would get x and y coordinate from the first view, y and z coordinate from the second view, and x and z from the third view. So it can be seen that only two orthographic (any two) are sufficient to get the x, y and z coordinates of all the points of the 3D model.

But in the orthographic views one point may be given multiple labels i.e it may correspond to multiple points having 2 of the coordinates same but a different third coordinate. So if we see a line between 2 points in one of the orthographic views, we may still not be certain about the edges that would be present in the 3D model of the object. But we can first plot all the possibilities according to the individual orthographic views and then compare the corresponding end points of the edge in other two orthographic views and then remove the edges that are not permissible according to any of the other two views (if there isn't an edge between two points in any of the orthographic views, then we can certainly say that there would not be any edge between those points in the 3D object, but if there is an edge between two points having multiple labels in any of the orthographic views, then we cannot certainly conclude as to which edge lies in the 3D object without looking at the other views).

This way we would be able to get a wire-frame description (edges and vertices) of the 3D model whose orthographic projections had been provided.

In order to generate three two-dimensional orthographic views (namely, top view, front view and side view) to convey the form of the three-dimensional objects, one approach that can be followed is

- Transformation of 2D vertices to 3D vertices.
- Generation of 3D line segments from 3D vertices.

- Construction of faces from 3D line segments.
- Formation of 3D objects from faces.

2.2 Algorithm

2.2.1 2D vertices and 2D lines

We take the input in the form of 2D vertices and 2D Lines. Now point of intersections of all the 2D lines are added to the list of 2D vertices. Point of intersection of two 2D lines can be obtained by standard methods. While constructing a 3D object, we consider each quantity as a probable quantity since that quantity may not be present in the final analysis.

2.2.2 Probable 3D vertices

A list of probable 3D vertices is constructed in this step. If any two 2D vertices, belonging to different views, have the same coordinate value for the shared coordinate axis, then the third view is searched for the 2D vertex which has the same values as the remaining two coordinates from the original two 2D vertices under consideration. If the search is successful then a 3D vertex containing the common x, y, z coordinates from three 2D vertices is found. This procedure is carried out for all the 2D vertices.

2.2.3 Probable 3D Edges

This step involves both the generation of probable-edges and checking the validity of probable-vertices. If any probable-vertex is found to be invalid, that probable-vertex is deleted and the procedure goes back to the beginning of this step.

Generating probable edges

A straight line which connects any two probable-vertices is a probable-edge provided the projections of this 3D edge can be found in all the three input views

Checking of probable vertices

The probable-edges are stored in terms of their end points. From this, a table that gives the probable-edge numbers to which each probable-vertex belongs is created. Since the algorithm deals with solid objects, each p-vertex should belong to at least three probable-edges. If a probable-vertex belongs to less than three probable-edges, then it is assumed to be a false probable-vertex and is deleted. Whenever a probable-vertex is deleted, the validity of the probable-edges is no longer guaranteed, hence the process returns to the beginning of this step (i.e.. step 3). If none of the probable-vertices is deleted then the process advances to the next step.

2.2.4 Probable Faces

This can be done in a similar way as that of probable 3D edges. In this step, a list of probable planar faces is constructed. This step is composed of the following sub-steps.

- Determination of planar surfaces.
- Generation of probable-edge closed loops.
- Probable-edge loop relationships.
- Formation of probable-faces.
- Testing of 2D dashed lines.
- Checking of p-edges.