Mathematical Model on Engineering Drawing

Mohit Gupta(2016CS50433) and Anoosh Kotak

January 15, 2018

The objective of this paper is to propose a mathematical model describing the construction of orthographic views from a given 3D model and vice-versa.

1 Problem Description and Introduction

Three Dimensional *CAD Models* are usually used by designers because of their multiple uses like visualization, simulation, 3D printing etc. Also, *multi-orthographic view* drawings are been widely used by architects and engineers.

Thus there comes the need of a tool to generate the 2D orthographic projection of a given *3D* Model and also recovering the *3D* object given two or more projections.
The **algorithm and mathematical analysis** of the above problem is briefly reviewed in this report

2 Computing Projections given the 3D Description

Aim:

1. Given any 3D model description we should be able to generate projections to any cross section or cutting plane.

2. Given any 3D model description we should be able to generate orthographic views on any three coincident-orthogonal planes.

2.1 INPUT

We interactively input the 3D object model and store it into a graph as set of vertices, set of edges as vertex pairs, set of faces as set of vertices. The projection plane is taken in an appropriate format as normal vector(n), one point on the plane(r0). Here, the set of vertices also contains the co-ordinates of each vertex.

2.2 METHOD

To project the 3D object onto the plane we need simply need to do the following:

1. Project each vertex on the plane.
2. Join projected vertices whose corresponding original vertices form an edge.
3. Filter hidden lines from these and make them dotted.

So, The problem now boils down to projecting 3D vertices to a 2D plane.

<u>**Theorem 2.2.1:**</u>

Let P be a point for which we have the 3D coordinates $r_p = (x, y, z)$ and we need to project it to a plane with normal $n = (n_x, n_y, n_z)$. If e_1 and e_2 are two orthogonal axis on the plane with origin as $r_O = (o_x, o_y, o_z)$, then the 2D coordinates are given by:

$$t_1 = Dot(e_1, r_p - r_o)$$

$$t_2 = Dot(e_2, r_p - r_o)$$

First we need to fix two coordinate axes for the plane and an origin. Origin can be simply taken as the given point on the plane and axes e_1 and e_2 following the conditions $e_1.n = 0$, $e_2.n = 0$ and $e_1.e_2 = 0$.(*In case of a particular view, the user needs to input one other coordinate axis on the plane so that we get there orthogonal planes by calculating the third axes as cross product of two vectors*).

- **Definition of Cross Product:** $e_2 =$ **Cross**(e_1, n)

$$\begin{pmatrix} e_{x_2} \\ e_{y_2} \\ e_{z_2} \end{pmatrix} = \begin{bmatrix} i` & j` & k` \\ e_{x_1} & e_{y_1} & e_{z_1} \\ n_x & n_y & n_z \end{bmatrix}$$

Let origin is at $r_O = (o_x, o_y, o_z)$ and your two coordinate axis in the plane are defined by:

$$e_1 = (e_{x_1}, e_{y_1}, e_{z_1})$$
$$e_2 = (e_{x_2}, e_{y_2}, e_{z_2})$$

Now taking e_1, e_2 and n as the three coordinate axes we can write any point as

$$r_p = r_o + t_1 * e_1 + t_2 * e_2 + s * n$$

where t_1, t_2 are the distances from e_2 and e_1 respectively ***(and hence are the desired 2D coordinates of projection)*** and s is the perpendicular distance of point from plane.

's' is computed as,

$$s = Dot(n, r_p - r_o)$$

And we get the projection coordinates as,

$$t_1 = Dot(e_1, r_p - r_o)$$

and

$$t_2 = Dot(e_2, r_p - r_o)$$

Conclusion: So, in this way we can project all the 3D-vertices of the object in the graph.

The projected vertex pairs can be simply joined for the ones whose original vertex pairs were also part of edge set.

If all the three orthographic projections are to be shown then similarly, the other two projections can be obtained by taking n as e_1 and n as e_2 and performing the same mathematical analysis.

Hence, we have obtained all the 2D projections.

Filtering Hidden Lines:

- Now to filter hidden lines we check for each vertex if the line joining the vertex and its projection cuts any face of the solid with both P and P' on the same side of the face.
  - ☐ If yes then its a hidden point. We have to check this for each face of set of faces given.
- To check for a particular face, we check for each triangle (each possible set of 3 vertices) formed by vertices of the face and see if the line passes through it. If we find even 1 triangle for which the condition is true then we stop and conclude that P is hidden.
- **Theorem 2.2.2:**

  For each triangle :

  Let P and P' be the vertex and its projection respectively and t_1, t_2 and t_3 be the vertices of triangle.
  - ☐ To check if line passes through the triangle we assume r is the point on the triangle where PP' meets the triangle.
  - ☐ Then we solve for r,

$$Dot((r - t_1), Cross((t_2 - t_1), (t_2 - t_3))) = 0 \tag{1}$$

$$(r - P) * (P - P') = 0 \tag{2}$$

$$(P - r).(r - P') = -1 \tag{3}$$

$$r = (1 - u - v) * t_1 + u * t_2 + v * t_3 \tag{4}$$

where,

  - ∗ (1) is to ensure that r lies on the plane containing the triangle (t_1, t_2, t_3).
  - ∗ (2) to constraint r on line PP'.
  - ∗ (3) to satisfy that P and P' are on the same side of the face.

    * In (4) see if u and v that we get are positive and $u + v < 1$. Then r lies inside the triangle.

- If these conditions are satisfied for some r then its a hidden point, and hidden points join to form hidden edges.

This space is intentionally left blank...

3 Constructing isometric view from orthographic views

3.1 INPUT

Labelled projections of the three dimensional object in the form of labelled vertices, edges(as vertices pairs) and hidden lines specified explicitly. This input will then be converted into a graph with the vertices as nodes. The set of vertices will contain the co-ordinates of each vertex. Hence each node will contain a tuple consisting of 3 elements. Also, an edge set containing all the edges between various vertices of the graph is also stored.

3.2 Mathematical Analysis: A Brief Overview

The steps to reconstruct a 3D object from its orthographic projections are as follows:

1. Finding the exact location of 3D vertices and construct super-set of edges.
2. Reconstructing super-set of faces from the super-set of edges.
3. Defining algorithm to delete pseudo edges and faces.

3.3 METHOD

3.3.1 Finding the exact location of vertices and edges:

- We identify triplets of 2D line junctions (one from each view), that have one coordinate in common in each pair of views. All 3D vertices of the object that lie at the intersection of at least three noncoplanar faces are found this way.
- Precisely we iterate over all vertices present in the 3 projections to find the 3-D co-ordinate of the points. From orthographic views we will get 3 tuples of 2 elements each namely (x,y), (y,z) and (z,x). We will find

3 sets such that any 2 have atleast 1 co-ordinate same. If such points doesn't exist then the given input is invalid. If such triplets exist then those triplets will be used as vertices of the 3-D model with (x,y,z) as the co-ordinate. By iterating over all points we can find all the vertices of the 3-D view.

- Then we construct the edge set by taking the intersection of all the edge sets of the orthographic views. Candidate edges are inserted into the skeleton between each pair of vertices that have a 2D line segment connecting their corresponding pair of 2D junctions in each view.
- Each edge is checked for intersection with the other edges and, when this occurs, additional vertices are inserted into the skeleton to subdivide the edges. All the true edges are a subset of these edges.

3.3.2 Reconstructing Faces:

- We use a *an algorithm*(Explained in section 3.3.5) 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.
- All the true edges and faces of the 3D object are a subset of those found so far. Some pseudo elements may be detected when loops fail to close, but there are usually many more still to be found.

3.3.3 Detecting pseudo edges and faces:

1. When an edge is adjacent to more than two faces, at most two faces can be true, the rest must be pseudo.
2. When two faces intersect, only one of them can be true.
3. An edge which projects onto a dashed line in a view, but which has no candidate face to occlude it, is false. (*This holds from the definition of Hidden Lines*).

4. When an edge is adjacent to only one face, both of them should be discarded.

5. When an edge is adjacent to exactly two coplanar faces, the edge actually does not exist while the faces are merged into one in the set.

6. If a true edge is adjacent to exactly two **non-coplanar** faces, then both the faces are included in the set as it is.

7. A face that is coplanar and adjacent to a true face, if their shared edge projects onto a solid line(Seen from input) and is not occluded in a view, the face is indeed not real and is to be discarded.(*Reason: The shared edge is needed to project onto the solid line because any other edge would be occluded by the true face. If the face was true, rules 5 and 4 would apply and shared edge would be deleted which is contradictory.*

We then follow a method where we detect 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.

3.3.4 Filtering Holes:

- Sometimes a 3D object includes holes whose faces are contained in another face.

- In general, faces may contain each other in a hierarchical manner. Once the status of one face has been established, the remained can be deduced.

- They are alternatively face and hole i.e,

$$f_j(face) \supset f_{j+1}(hole) \supset f_{j+2}(face) \supset ...$$

  where, f_j is a true face if j is even and and a hole if j is odd.

3.3.5 Algorithm to trace all planar edge loops in skeleton:

```
• procedure
    □ For each vertex $v_i$
        ∗ we iterate for each pair of adjacent edges $e_i, e_j$ and calculate
          the following:
          $n$ : normal to plane, $e_i$ x $e_j$
          $v_k$ : vertex at far end of $e_j$
            · while $((v_k \neq v_i) and (e_j exists))$ do
              $e_k$ is assigned edge adjacent to $v_k$, coplanar to n, with
              minimum-internal-angle from $e_j$
              $e_j = e_k; v_k =$ far end of $e_k$
            · end while
            · if$(v_k = v_i)$ then
              list $e_i, e_j, e_k, ...$ is a candidate face loop
            · end if
        ∗ next $e_i, e_j$
    □ next $v_i$
• end procedure
```