IIT Delhi

COP290

Assignment 1

Modeling and Analysis

Submitted By:

Madhur Singal
2015EE10908

Garvit Gupta
2015EE10447

January 22, 2018

1 Creating Orthographic Projections from a Polyhedral Solid

1.1 Introduction

Orthographic Projections is a technical drawing used to represent 3D objects in 2D. This involves projecting three different views of an object on orthographic planes. Using this any solid can be represented on a 2-D plane like paper without missing important information about the solid. This is a widely used method of communicating ideas and objects. We propose an algorithm that compares every 3D edge with all the plane faces of the polyhedral solid and stores its visibility information. We then project every line to the viewing plane according to whether it is hidden or visible.

1.2 Preprocessing the Input Data

A polyhedral solid can be represented by vertices, edges and polygon faces formed from these edges. Suppose the 3D object is to be projected on three orthographic planes p_1, p_2 & p_3 given by $a_1x+b_1y+c_1z = d_1$, $a_2x+b_2y+c_2z = d_2$ & $a_3x+b_3y+c_3z = d_3$ respectively. It is assumed that no plane intersects the 3D model. To project on these planes we rotate and translate our coordinate system such that the normal of p_1, p_2 & p_3 aligns with the x-axis, y-axis and z-axis of our new coordinate system respectively. The intersection point of the three planes will be our new origin. If a point (x, y, z) is represented as $(x', y'z')$ in our new coordinate system, then we get:

$$x\hat{i} + y\hat{j} + z\hat{k} = (x' - x_0).\hat{n}_1 + (y' - y_0).\hat{n}_2 + (z' - z_0).\hat{n}_3$$

where $\hat{n}_1$, $\hat{n}_2$ & $\hat{n}_3$ are normals of planes p_1, p_2 & p_3 respectively such that $\hat{n}_i.(x\hat{i} + y\hat{j} + z\hat{k}) \geq 0 \ \forall \ i \in \{1, 2, 3\}$ and (x_0, y_0, z_0) is the intersection point of three planes. Solving the above equations we get a unique solution for (x', y', z') as the planes are orthogonal. We will transform every vertex of the 3D object to this new coordinate system. The edges and planes will remain unaffected as they are defined with respect to the vertices. The vertices can now be simply projected to the three planes by putting one coordinate as zero. The coordinate system is changed back to the original after generating the projections for all edges. For the rest of the sections we will consider the three orthographic planes as x-y, y-z and z-x.

1.3 Adding New Edges and Vertices

After transformation of vertices to the new coordinate system, we add some new vertices and edges, which are formed due to the intersection between the projected line segments on the orthographic planes. For doing this the algorithm is as follows:

1. Make a new set of edges S containing all the edges of the 3D model.

2. Let (e_i, e_j) be two edges in S with end points (v_{i1}, v_{i2}) and (v_{j1}, v_{j2}) respectively. We will project this edge in one of the three orthographic planes, as line segments and look for intersection or overlap (refer to section 2.2.1). If projection of both the edges correspond to the same line segment then try choosing different plane of projection or another pair of edges.

3. If the line segments intersect at p, we will get two points p_i and p_j in e_i and e_j respectively whose projection is p. The points can be found by putting p in the line equations of 3D edges and solving for the 3^{rd} coordinate. Now we remove edges (v_{i1}, v_{i2}) & (v_{j1}, v_{j2}) from S and add edges (v_{i1}, p_i), (p_i, v_{i2}), (v_{j1}, p_j) & (p_j, v_{j2}) to S.

4. If the line segments overlap as shown in Fig. 1b, the edges e_i & e_j are removed from S and the 3D edges corresponding to the overlapping and non overlapping part of the lines are separately added to S. In Fig. 1b, non overlapping part are *ac* and *db*, and the overlapping part is *cd*.

5. The steps from 2-4 are repeated for all possible combination of edges and plane of projection. We move to the next step when no more intersections (or overlap) can be found.

1.4 Projecting 3D Edges on Three Orthographic Planes

The set S was generated to effectively detect hidden parts of edges. For all edges in S, its projection on a orthographic plane, is either completely hidden by a face or not hidden at all. If a edge $s \in S$ is partially hidden by a face, then the projection of s will intersect with the projection of atleast one of the edges of the face in some orthographic plane. This is not possible, as in this case this edge should have been replaced by smaller edges in the process

of generating the set S. The algorithm to take 2D projection of a 3D edge is as follows:

1. For every face f, we store its projections as f_1, f_2 & f_3 in set F. If any of f_1, f_2 or f_3 is a line, we delete it from P.

2. For every edge s, we store its projections as s_1, s_2 & s_3 in set P. If any of s_1, s_2 or s_3 is a point, we delete it from P.

3. Let $p \in P$. For p to be hidden, the conditions are:

   (a) There exist a 2D polygon $f \in F$, such that the midpoint of p lies inside the polygon. To check this condition the following algorithm is followed:

      i. Let the coordinates of p be (x', y'). Draw a line y=y' and find it's intersection with each edge s of polygon f. Initialize left count and right count to 0.

      ii. Let the intersection point with edge s be (x'', y'). If 1 of the end points of s has y-coordinate $> y'$ and $x'' > x$, increment right count. otherwise if 1 of the end points of s has y-coordinate $> y'$ and $x'' < x$, increment left count.

      iii. If left count and right count are both even, p lies outside f. Otherwise, p lies inside f.

   (b) Let s be the 3D edge whose projection is p. Let m be the point on s whose projection coincides with the midpoint of p. For p to be hidden m should be between the plane of projection and the 3D face, whose projection is f. To check this condition the following algorithm is followed:

      i. Let m be $(x", y", z")$. Let the 3D face whose projection is f be k. Let the equation of plane passing through k be $ax + by + cz = d$.

      ii. Let v be the vertex of k closest to f. Let (x', y', z') be the midpoint of v and its projection on plane. Then m is between plane and k if, $(ax' + by' + cz' - d).(ax" + by" + cz" - d)) > 0$

4. After classifying all line segments in P as solid or Hidden. We now look for repeated lines in P. If all repeated lines are of same type (solid or hidden), only one is kept and rest are deleted. If repeated lines contain both solid and hidden lines, we only keep one solid line and delete rest.

2 Creating Polyhedral Solid from Orthographic Projections

2.1 Introduction

Three-View engineering drawings are widely used to represent 3D solids in 2D. Existing products are often presented on blueprints using these engineering drawings. To manufacture these products we need to create a 3D model of them. We propose a step by step bottom up approach to create solid polyhedral objects from 3 orthographic views. The steps broadly comprise of generating 3D points from 2D points in orthographic views, creating the probable wireframe model and finally finding faces to construct the solid while eliminating pseudo edges and faces via a Decision chaining algorithm.

2.2 Preprocessing the Input Data

Before we get into the mathematics, we need to stick with some standards. We are assuming that we have three orthographic views of the solid namely front, side and top view. All the 2D edges and vertices in the three views are known in the form of a graph. Also, there is no compulsion on user to provide correspondence information for the vertices in the 3 views and the end point information of the edges. So the first step before we can start with the actual process is to process input data. Since the user is not asked to enter the intersection points of the edges, the intersection points (if any) are added as vertices in the graph. Next, new 2D edges are added to the graph according to these vertices.

2.2.1 Adding Intersection Points

To find the intersection points for two line segments, we make use of parametric equations. Any 2D line segment in parametric form can be represented as:

$x = x_1 + t(x_2 - x_1)$
$y = y_1 + t(y_2 - y_1)$
where $0 < t < 1$ and (x_1, y_1) and (x_2, y_2) are two points of the line segment.

If we have two line segments l1 and l2, represented as:

$x = x_1 + t(x_2 - x_1)$
$y = y_1 + t(y_2 - y_1), 0 < t < 1$
$x = x_3 + t(x_4 - x_3)$
$y = y_3 + t(y_4 - y_3), 0 < s < 1$

We get two linear equations in s and t from the above equations. On solving these equations we get the following cases:

1. If on solving them we get a unique solution such that $0 < s < 1$ and $0 < t < 1$, then the two lines intersect (Fig. 1a). In this case we add the point of intersection e and edges ce, eb, ae & ed to our graph.

2. If $s = 0/1$ or $t = 0/1$ (not both), as shown in Fig. 1c, we add the point of intersection d and edges ad & db to our graph.

3. If we get infinite solutions of s and t, then the two lines overlap as shown in Fig. 1b. In this case edges ac, cd & db are added.

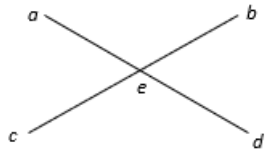

Fig. 1a

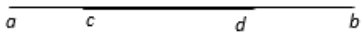

Fig. 1b

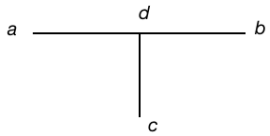

Fig. 1c

2.2.2 Adding New Edges

For our algorithm to find all possible edges, we need to add an edge between all 2D vertices (v_i, v_j) which are connected by a straight line. For this we can implement DFS or BFS and add a edge for all pairs of connected vertices (v_i, v_j) which have a path $v_i, v_{i+1}, ..., v_{j-1}, v_j$ between them, such that vertices $v_i, v_{i+1}, ..., v_{j-1}, v_j$ are all collinear. To check whether two connected edges are collinear or not we consider two connected line segments represented as:

$x = x_1 + t(x_2 - x_1)$
$y = y_1 + t(y_2 - y_1), 0 < t < 1$
$x = x_2 + t(x_3 - x_2)$
$y = y_2 + t(y_3 - y_2), 0 < s < 1$

Then the lines are collinear if:

$\frac{(x_3 - x_2)}{(x_2 - x_1)} = \frac{(y_3 - y_2)}{(y_2 - y_1)}$

2.3 Wire Frame Model

To get the 3D model we first need to create a wire frame model of the 3D object. The wire frame model consists of probable vertices and probable edges joining these vertices. The word probable is used as we get some pseudo elements in our algorithm which don't exist in the 3D model. We can remove these elements once a probable 3D model is complete using Decision Chaining algorithm.

2.3.1 Getting Probable Vertices

As discussed above, to get the wire frame we first need to get the probable vertices of the solid. For this we assume that we have all three orthographic views of the solid and we have all 2D edges and vertices in those 3 views. This was ensured in the preprocessing step. For getting probable 3D vertices any two 2D vertices with same coordinate value for shared coordinate axis are used to search for the third vertex which have the same coordinate values for the remaining two coordinate axis. These three 2D vertices define a probable 3D vertex. The probable points contain all true points along with some pseudo points.

2.3.2 Getting Probable Edges

After finding the probable 3D vertices, the next step is to get all probable edges. For this, we add an edge between every pair of probable 3D vertices which either have a line segment between them in two views and coincide on a point in the third view or have a line segment between them in all the three views. The recently generated edge is compared to every previous edge. If the recently generated edge contains any previously generated edge, then the recently generated edge is not included. If the recently generated edge is contained by any previously generated edges, then all those edges are deleted. To check if a edge is contained in another edge we again write the equations in parametric form. The 3D line is represented as:

$x = x_1 + t(x_2 - x_1)$
$y = y_1 + t(y_2 - y_1)$
$z = z_1 + t(z_2 - z_1)$
where $0 < t < 1$ and (x_1, y_1) and (x_2, y_2) are two points of the line segment.

For an edge e with end points (v_1, v_2) to be contained in the edge of above form, the value of parameter s when v_1 & v_2 are put in the equation should be such that $0 \leq s \leq 1$.

2.3.3 Checking Validity of Probable Vertices

After adding all possible edges, all the probable vertices which have less than three adjacent edges are removed. If no such vertex is detected then the algorithm advances, else the algorithm in section 1.3.2 is repeated. After this algorithm we get all the true edges along with some pseudo edges.

2.4 Getting Probable Faces

After getting the wire frame model the next thing is to get the probable set of faces. To this we use the **minimum-internal-angle** searching algorithm to trace all the planar edge loops in the wire frame. The algorithm is as follows:

1. Start with the leftmost convex vertex (v_i)
2. For vertex v_i, find two adjacent edges (e_i, e_j) of the vertex such that v_k is the other end of edge e_j.

3. Then the adjacent edge of v_k, such that $\frac{e_i \times e_j}{|e_i \times e_j|} = \frac{e_j \times e_k}{|e_j \times e_k|}$ is found, where angle between e_j and e_k is minimum.

4. Now put $e_i = e_j$, $e_j = e_k$ and v_k to be the other end of e_j and repeat step 3 till $v_k \neq v_i$.

5. After $v_k = v_i$, we get a probable face bounded by the edges found in steps 2 and 3.

6. Repeat the steps 2...5 for every vertex v_i.

In Step 3, $\frac{e_i \times e_j}{|e_i \times e_j|} = \frac{e_j \times e_k}{|e_j \times e_k|}$ constraint ensures that the edges the algorithm finds are coplanar and thus the faces are planar. Also, the minimum internal angle constraint reduces the time to find a face as an edge making lower internal angle with the previous edge has a higher probability of closing the polygon.

2.5 Removing Pseudo Elements

For removing pseudo elements, we will use the decision chaining algorithm. Before going to the algorithm, the **Moëbius Rule** needs to be stated, which is as follows:

1. Each edge belongs to exactly 2 faces and the orientation of the edge is opposite on each face.

2. The faces cannot have any intersection except on their edges

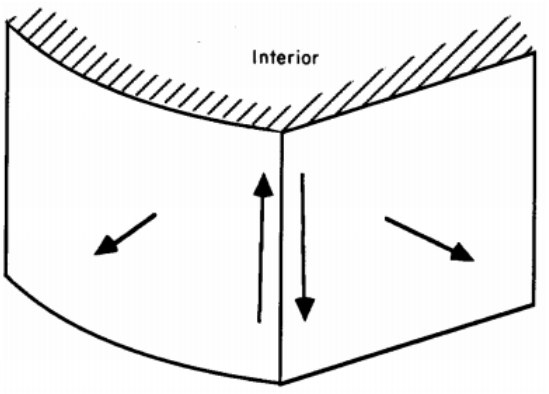

Fig. 2 Moëbius Rule

2.5.1 Decision Rules

The following Decision rules are used to find the pseudo elements:

1. When an edge is adjacent to more than two faces, at most two faces can be true, the rest must be pseudo. (From the Moëbius rule)'

2. 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 Moëbius rule)

3. An edge is false if it projects onto a hidden line in a view, but has no candidate face to occlude it. (From the definition of hidden lines).

4. When an edge is adjacent to only 1 face, both the edge and face are false. (From the Moëbius rule).

5. When an edge is adjacent to exactly 2 coplanar faces, the edge is false and the coplanar faces can be merged. (Because a true edge must be coplanar to 2 non-coplanar faces, otherwise the faces will merge).

6. If a true edge is adjacent to exactly 2 non coplanar faces, then both these faces are true. (If either of the adjacent faces were false, it would contradict rule 4).

7. A face is a false if it is coplanar and adjacent to a true face and their shared edge projects onto a solid line and is not occluded in a view. (The shared edge should project onto the solid line because any other edge would be occluded by the true face. If the candidate face was true, rules 4 and 5 would apply and the shared edge would be deleted).

2.5.2 Decision Chaining Algorithm

The Decision Chaining Algorithm can be used to detect pseudo edges and pseudo faces. It uses the decision rules 1 and 2 to make assumptions, then uses rules 1 to 7 to validate those assumptions and decide all the edges. If the assumption comes out to be invalid, it backtracks and makes other assumptions and continues till all the edges have not been decided.

1. If all the edges and faces obey Moëbius rule 1 and 2 respectively (i.e. each edge is adjacent to exactly 2 faces and no 2 faces intersect), then the solution has been found, otherwise proceed to step 2.

2. Consider undecided edge e_i

3. Consider 2 adjacent pair of faces f_j and f_k of e_i and assume they are true.

4. Apply rules 1...7 on e_i and all it's adjacent faces with the above assumption.

5. If the assumption is valid and some changes have been made to the elements, goto step 1. Otherwise (If either a contradiction arises in the assumption or the assumption is valid but no changes have been made to the elements), repeat steps 3 and 4 for other pairs of planes adjacent to e_i.

6. Repeat steps 2...5 for all undecided edges e_i.

2.6 Containment and Orientation

2.6.1 Containment

Many 3D objects contain holes whose faces are contained in solid faces. In general, faces may contain each other in a hierarchical manner. In such a situation, the outermost face should be a true face and alternately the faces should be hollow and true, i.e., $f_i(face) \supset f_{i+1}(hole) \supset f_{i+2}(face)$.... where the faces are sorted in decreasing order of area.

2.6.2 Orientation

Face Orientation consistency for the solid is maintained using Moëbius rule 1. First, normals to all the faces are stored. We start the verification from the leftmost-vertex. The outward normal to any face adjacent to this vertex must be leftwards. from this, all other faces are oriented iteratively by comparing the direction of traversal of their edge loops. The sign of the normal of the face being considered gets reversed if its face loop traverses the same direction as the previous adjacent face's loop.

3 References

1. "Reconstruction of 3D Objects from three orthographic projections using a decision chaining method" by R.E. Marston and M.H. Kuo.

2. "Construction of 3D solid objects from orthographic views" by U.G. Gujar and I.V. Nagendra.

3. "Automatic construction of curvilinear solids from wireframe views" by Remi Lequette.

4. "Point-In-Polygon Algorithm — Determining Whether A Point Is Inside A Complex Polygon" by Darel Rex Finley