Fast 3D Modelling Using Orthographic Views

Kuldip Deelip Wagh
Department of Computer Engineering
MCT's Rajiv Gandhi Institute of Technology
Mumbai (400053), India
kanhatop@hotmail.com

***Abstract*—3D modelling is very essential step in product design and animation and video games designing process. Most of the modelling softwares require GPU enabled system due to very large number of complex mathematical calculations like 3D vector arithmetic, complex equation solving, and large matrix multiplications etc. which are involved in 2D to 3D conversion algorithms which are used during the modelling process. The proposed algorithm here works without performing a single of such mathematical calculation to convert 2D to 3D. Hence results into efficient, fast and GPU free conversion. Algorithm is easy to understand and implement due to absence of any such arithmetic operations.**

Keywords—orthographic, projections, modelling, non curvilinear, reconstruction, 3D, engineering drawing

I. INTRODUCTION

In the field of machine design, before manufacturing the object, designers model the object using one of the modelling softwares like CAD. Orthographic views are the perpendicular projections of an object on 3D coordinate planes, hence retaining the original dimensions of an object. For most of the objects with no curved surface, at least three orthographic views i.e. front view, side view and top view are required for reconstruction. Only outline edges are necessary to represent any such object. The process of conversion of orthographic views to 3D model consists of four basic steps (figure 1),

- Taking orthographic input in terms of interconnected segments formed by connecting 2D points in any of the views.
- Extracting 3D points from orthographic 2D points.
- Finding the connectivity of obtained 3D points in 3D space i.e. finding the 3D edges of an object.
- Projection of 3D edges to 2D plane for displaying purpose.

The algorithms used in this paper for processing these steps have been derived by observing and analyzing the human thought process while drawing isometric views from orthographic views in engineering drawing.

978-1-4799-3197-2/14/$31.00 ©2014 IEEE

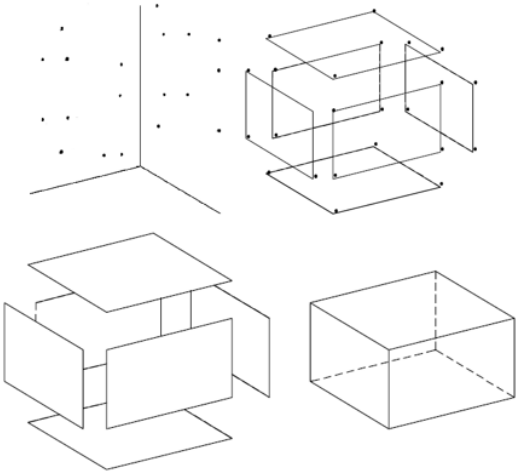

Fig. 1. Steps in modelling

One of the earlier methods for this conversion used coordinate geometry in which each of the line is represented by matrix form. Hence, the matrix multiplications are complex tasks in that approach.

The other method [2] uses very old approach which is not accurate and involves vector algebra e.g. finding normals, finding various products of vectors and algorithms like Minimum internal angle algorithm, Decision chaining algorithm etc.

II. METHOD

A. Conventions

In this paper, front view is drawn in X-Y plane, side view in Y-Z plane and top view in X-Z plane (figure 2). That means, each segment drawn in front view will have end vertices with Z coordinate equal to zero. Similarly, side view having X coordinate equal to zero and top view having Y coordinate equal to zero. These coordinates are completely logical and one can assign different coordinate axis to different views.

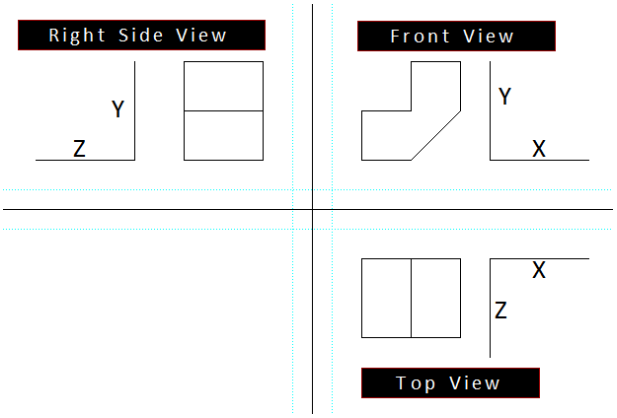

Fig. 2. Convention of 3D coordinates for each view

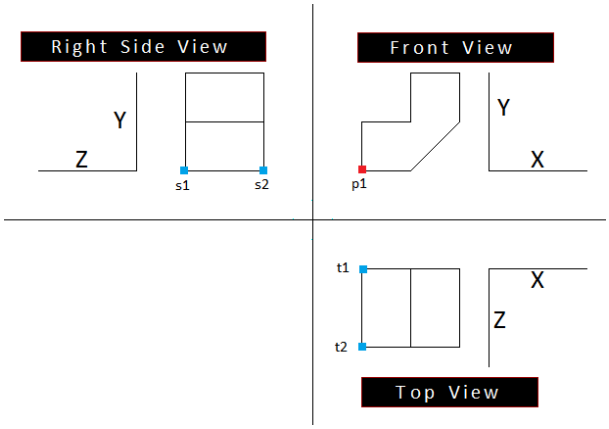

Fig. 4. Example

B. *Conceptual Explanation*

Orthographic views are projections of any 3D object on X, Y and Z planes. If we draw normal from the vertices of all the views then they all intersect into various points in 3D space (figure 3). These new points are actual 3D vertices of the object in space. In similar way, one can find all 3D points for object by only using comparison and equality checking.

C. *Logical Working*

By considering the conventions, first select any one point from front view, suppose p1 (figure 4). Then find the associated points for p1(x, y, 0) in side view with same Y coordinate as p1(x, y, 0), since Y axis is common in both the views as per the convention. The associated points in side view for p1(x, y, 0) are s1(0, y, z1) and s2(0, y, z2). Now, similarly find the associated points for p1 in top view with matching X coordinate. The obtained points are t1(x, 0. Z3) and t2(x, 0, z4).Now among the obtained points find those points in side view matching their Z coordinate with Z coordinates of points in the top view. In the example, s1 matches t2 and s2 matches with t1 i.e. z1 and z4 are equal and z2 and z3 are equal. Hence for point p1 we got two associate points in 3D having coordinates as (x, y, z1) and (x, y, z2). In similar way process all the points in front view and find all the 3D points of the object. The reason to choose the front view vertex points is that, mostly the front view contains more no. of points than the other views hence extracting all the 3D points easily. If one selects side view then the result will still be same but the number of comparisons will increase.

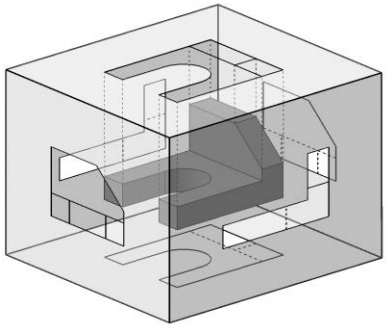

Fig. 3. Intersection of normals from vertices of all views

D. *Finding Edges*

Next task is to find the connectivity of points in 3D space. It is simple observation that most of the vertices in 3D are intersection points of exactly three edges each in 3D. That means three edges pass through each vertex in 3D, in most of the cases.

- Each edge which is parallel to any of the three axes in 3D is represented by the segment of same length in any two orthographic views.
- Each edge with length L, which is not parallel to any of the axis and is perpendicular to any one of the axis in 3D i.e. lying in one of the coordinate planes, is represented in one of the orthographic view as a segment with the end points having the same coordinate as the coordinates of end points of the line in 3D, except the coordinate for axis to which it is perpendicular.
- For the line which is present in neither of planes, is displayed by the segment with length same as projection of 3D line on any of the coordinate planes.

These observations are sufficient to determine all the edges and also to verify them as written in algorithms.

E. *After Processing and Display*

The points if contributing exactly three edges verifies the correctness of the points. If any of the points is contributing less than three edges represents an error. But there can also be some points contributing to more than three edges. There are several ways to display a model to screen depending on the type of view chosen.

III. ALGORITHMS

Let F is one of the points in front view, T from top view, S from side view. X coordinate of a point P in orthographic view can be called by xcoordinate[P] can be called as ‘X coordinate of point P’. Similarly Y coordinate by ycoordiante[P]. and Z coordinate by zcoordiante[P]. After converting into 3D, the point has given the index no. and its 3D coordinates are stored

into x[index], y[index] and z[index]. Each of the points is connected to three other points to form an edge. Suppose point p1 is connected to p2 along X axis, then it is stored into data structure by storing index of p2 into xconnection[index of p1] and index of p1 into xconnection[index of p2]. Similarly point along Y axis is stored into yconnection[index] and Z into zconnection[index]. One can use linked list rather than arrays for data structure.

A. Finding the Points

Initialize index

For each point F(x, y) in front view

For each point T(x', z) in top view

If(x' equals x)

For each point S(y', z) in side view

If(y' equals y)

3D coordinates of point are (x, y, z) and are stored into x[index], y[index], z[index].

Increment Index

End if

End for

End if

End for

End for

B. Finding Edges Parallel to Coordinate Axes

For each obtained 3D point p1

For each obtained 3D point p2

If(any two coordinates of p1 matches with respective coordinates of p2 and third coordinate is different)

Store the index of p2 into either xconnection[point] or yconnection[point] or zconnection[point] where appropriate connection is chosen by the axis along which coordinates of two points differ.

End if

End for

End for

C. Finding Edges non Parallel to Every Coordinate Axes and Perpendicular to One of the Axis

Now, check if every obtained 3D point has initialized xconnection, yconnection and zconnection. If every connection is initialized then stop, otherwise perform the following algorithm.

For each 3D point p1 whose connection is not initialized

For each 3D point p2 whose connection is not initialized

If(both points have exactly one coordinate same and there is a segment in any of the views having end points' coordinate matching with remaining two coordinates of p1 and p2)

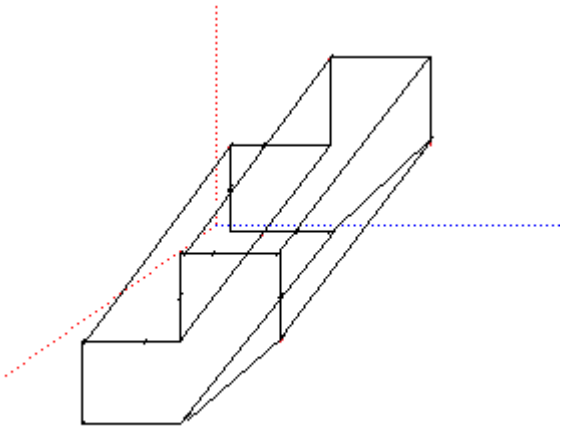

Fig. 5. Output 3D model for above example

Store the index of p2 into either of xconnection[p1] or yconnection[p1] or zconnection[p1] where appropriate connection is the one which is not initialized yet. Similarly, fill the uninitialized connection of p2 by p1.

End if

End for

End for

D. Finding remaining Edges

Now, all the edges are found and connected for almost all objects with linear surfaces. There are some cases where few points are having more than edges intersecting. But since probability of existence of such object is very little, one can directly connect such two points which are still having uninitialized xconnection or yconnection or zconnection.

E. Output of Example.

The figure 2, 4 and 5 are captured during deployment of project. Since efficiency is primary concern, the project is implemented in c++.

F. Analysis results

The implemented project consisted of following steps:

1. Creating interface for user to draw orthographic view.
2. Apply described algorithm on obtained orthographic view and generate 3D model.
3. Display generated model and give flexibility of 720^0 view to user to visualize model.

System specifications used:

- Intel core i7 2760 2.4 GHz processor
- 8 Gb DDR3 1600 MHz RAM

- Windows 8.1 64 bit OS
- Visual c++ compiler

The whole project is programmed single threaded.

	Total Time	Self Time
Game::ComposeFrame	23.5286s	0.0539s
Game::ArrangeOrigin	7.1224s	0s
Game::DrawFace	6.0718s	0.0207s
[loop at game.cpp:565 in Game::ComposeFrame]	4.9728s	0s
Game::DrawConventions	2.7946s	0s
Game::DisplayValues	0.8694s	0s
Game::GetCursorPosition	0.6888s	0.0199s
Game::GetPoint1	0.5297s	0.0099s
Game::GetPoint2	0.2394s	0s
D3DGraphics::DrawDottedLine	0.0984s	0s
atan	0.0198s	0s
KeyboardClient::KeyIsPressed	0.0194s	0.0194s
MouseClient::GetMouseY	0.0182s	0.0182s
RTC_CheckStackVars	0.0100s	0s
Game::LengthAndAngleCalculation	0.0100s	0s
pow<int,int>	0.0099s	0.0099s
sqrt	0s	0s

Fig. 6. Analysis Results

The actual algorithm along with calls to various other functions for supportive tasks were coded inside a function named ComposeFrame(). Total time of ComposeFrame() is the summation of algorithm time and other supporting function time. Self time of ComposeFrame() is actual time of algorithm contributing to Total Time.

This means that actual algorithm took only 0.0539 seconds on single core of CPU. Surely, if implemented using GPU with multhreading, can give huge performance increase.

Figure 7 shows average frames per second during complete runtime which is close to 60 FPS.

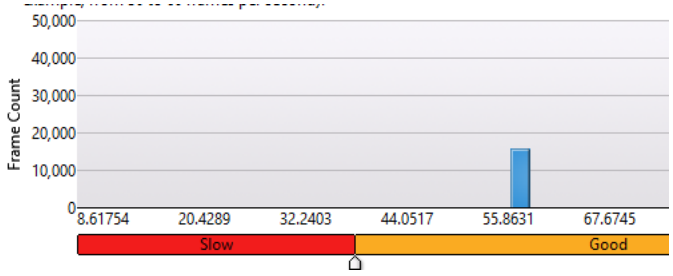

Fig. 7. Average FPS during runtime

G. *Limitations*

- This algorithm works only for objects with no curved surfaces.
- The objects with large number of points intersecting more than four edges through them may lead to the wrong edge detection.
- Flushing due to branching can become one negligible issue when executed on older processor which give large penalty during flushing.

IV. CONCLUSION

Though there are advancement in the techniques for 2D to 3D conversion, still it requires a lot of computational power. High performance GPUs are used for numeric calculations but not everyone can afford it. The presented algorithm is solution to such problem. This approach of comparison, in the modelling process can lower lot of resource requirement in the field of video games and animation; resulting into high performance on low specification machines.

ACKNOWLEDGMENT

I would like to thank Prof. Ekta Upadhyay and Prof. D. S. Kale sir for their assistance and guidance and Prof. D. K. Chakradev sir for valuable help.

REFERENCES

[1] R. E. Marston and M. H. Kuo, “Reconstruction of 3D Objects from Three Orthographic Projections Using Decision-Chaining Method,” IAPR Workshop on Machine Vision Applications, Kawasaki, pp. 13-15, December 1994.

[2] Fahiem, M. A. Haq, and Saleemi, “A Review of 3D Reconstruction Techniques from 2D Orthographic Line Drawings,” Geometric Modeling and Imaging, July 2007, pp.60-66.

[3] R. Furferi, L. Governi, M. Palai and Y. Volpe, “From 2D Orthographic views to 3D Pseudo-Wireframe: An Automatic Procedure,” in IJCA(0975-8887), vol. 5,No. 6, August 2010, pp. 18-24.

[4] W. Wang and Grinstein, “A Survey of 3D Solid Reconstruction from 2D projection line drawings,” Computer Graphics Forum, May 1998.

[5] N. Ahuja and J Veenstra, “Generating Octrees from Silhouettes in Orthographic Views,” IEEE Trans. On PAMI, Vol 11, pp. 137-149, 1989.

[6] J. Y. Zheng, “Acquiring 3D Model from Sequence of Contours,” IEEE Trans. on PAMI, Vol. 16, no. 2, pp. 163-178, 1994