# Data Reduction for Finding Silhouette Edges on 3D-Animated Model

Nucharee Thongthungwong and Rajalida Lipikorn

*Abstract*—**Finding silhouette edges on 3D animated model is resource and time consuming due to its need of calculating and comparing every edge. This paper presents a novel idea for reducing data and repeated calculation by organizing 3D animated model data and getting rid of the unnecessary data for calculation. The approach would save time and resource consuming. Moreover it would create relationship of line between each frame, which later can be stylized.**

*Index Terms*—**-Component, non-photorealistic rendering, 3D animated object, animation, silhouette edge.**

## I. Introduction

There's an attempt to create stylized rendering in animation industry. When talking about rendering, most are aiming for photorealistic look. However, for stylized rendering, there's an attempt to do non-photorealistic look. One approach of non-photorealistic rendering is to imitate traditional drawing, which has feature line for representing shape of character or model.

Choudhary [1] has defined feature lines as linear manifold that can define 3D object on 2D space. There are 4 types of feature lines that are:

- Silhouette lines: edges that lie against background, which appears as the boundary of the objects.
- Self-occluding silhouette lines: edges that lie against the same object portion of which is further behind
- Intersection line: line that marks the intersection of two objects.
- Crease line: line that defines the discontinuity of surface normal or defines the sharp edge.

In this paper, we are focusing on Silhouette lines and self-occluding silhouette lines, which from now we would call them as silhouette edges.

Finding silhouette edges is time and resource consuming, because it needs to compare every faces and edges. Moreover, when dealing with animated model, edges and faces are changing as each frame passes. It also needs continuity or relationship of the line between each frame.

Our approach will organize 3D data structure, which would help creating relationship of each silhouette edges between individual frames. Also it would reduce repetition of calculation and get rid of unnecessary calculation, which result in faster rendering.

Manuscript received March 17, 2013; revised May 20, 2013.

The authors are with Machine Intelligence and Multimedia Information Technology Lab, Department of Mathematics, Faculty of Science, Chulalongkorn University, Bangkok, Thailand (e-mail: Nucharee.T@Student.chula.ac.th, rajalida.l@chula.ac.th).

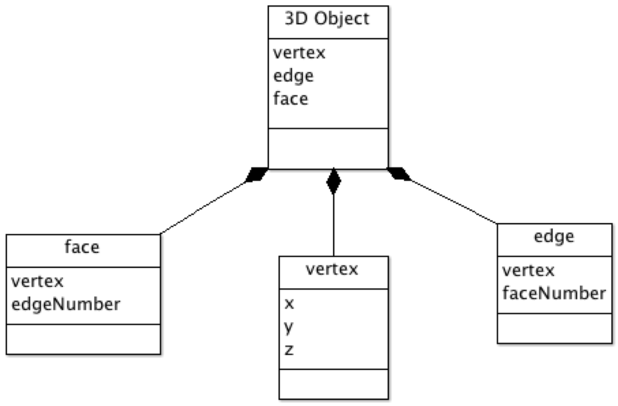
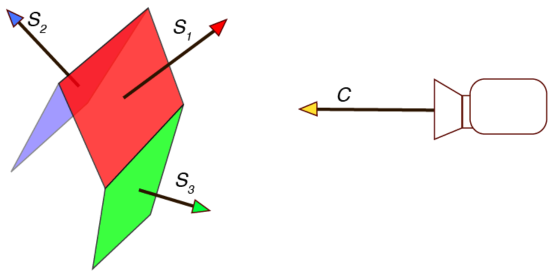


Fig. 1. Collecting 3D data as object.

## II. Data Collection

Polygon mesh can be defined by at least 2 types of data: vertices and faces. Vertex is the position of *x*, *y*, *z* coordinate in world space, and a face is composed of at least 3 vertices. Standard object like Wavefront OBJ [2] is collecting vertices, faces and texture data, but an Obj file collects data of only one frame. It doesn't contain animation data on surface. Our data collection is doing something similar to Wavefront OBJ, but we don't need texture data. Instead of that, we purpose to collect edge data (Fig. 1). Edge is composed of 2 vertices. A polygon face needs at least 3 edges, or at least 3 vertices. Edge data would be stored as tree data structure, which would help defining continuity of silhouette edges. Moreover, we would collect animation data on surface by collecting vertex position of every frame, but edge and face data would be collected at a frame since edges and faces' relationship doesn't change during animation. Our data collection method could work with both skeleton-based animation and surface deforming animation.



$S_1$= Normal vector of red surface
$S_2$= Normal vector of blue surface
$S_3$= Normal vector of green surface
$C$= Normal vector of camera
If $(S_1 \bullet C) \times (S_2 \bullet C) \leq 0$ : found silhouette or ridge edge

Fig. 2. Finding silhouette edges by doing dot vector of surface normal and camera.

## III. Finding Silhouette Edges

Methods for defining silhouette can be classified in 2 main approaches. The first one is the geometry-based approach [3], [4]. One of the concepts of this approach is to detect silhouette edges by finding the edges, which have both front facing and back facing (Fig. 2). In order to find the silhouette edges with this approach, it is necessary to compare every edge of the geometry. It is time-consuming and complicated. However, it can result the edges on 3D geometry, which later can be defined as vector lines. The second one is image-based approach [5]-[7], which focuses on depict only the portion of the silhouettes that contribute to final 2D image. It uses 2D image-processing technique such as edge detection filter, leveling the contrast of light and shadow, and so on. This technique lacks of the control for the line thickness, and also hard to generate the stylization of the brush stroke.

However, our approach is based on SED algorithm [4], which is geometry-based approach. We suggest using geometry-based approach, because we expect the final result to be vector lines which later could possibly be edited and stylized.

Firstly we need to create an array for checking edges. The size of an array is the same as the quantity of an object's edges. If necessary, we choose the pre-calculation frame by picking the frame that shows the detail of an object the most. Then begin with the first edge. Check if this edge is silhouette edge or not. If yes, pick a vertex from this edge and find connected edge for calculating silhouette edge. Also mark on the position in an array of checking edge. Before checking the new edge, we should check if the edge is already checked or not. By this method, we'll have an array of edges, which has continuous vertices. For the next frame, instead of calculating the whole edges, we calculate only the previous silhouette edge and the edges that connected to the start and end points of each continuous silhouette edge as described in the following algorithm:

```
Calculation for silhouette edge:
For each edge in mesh:
If (edge is marked in Edge Check array)
Else,
Do dot product with camera normal and each face normal in relation to current edge. (C●face1, C●face2)
Do cross product with the result of previous dot products. (C●face1 × C●face2)
If (C●face1 × C●face2 <= 0)
//This edge is silhouette edge.
Mark on Edge Check array
Store in Silhouette Edge array
Pick next continuous edge.   If (edge is not marked in Edge Check array)
Do Calculation for silhouette edge
Else
Mark on Edge Check array
End if
```

On each frame, we calculate and render the silhouette line form the Edge Check array and a bit more on continuous edge in relation to silhouette edge. With this approach we can create animated vector lines, which can represent a 3D object on 2D plane.

The Silhouette Edge array stores the results of silhouette edge calculation. We store this array as tree format so that we can use the tree relation as the continuity of our silhouette edge whereas the tree's branches can define the amount of silhouette lines on the scene (Fig. 3). Moreover, we can use this array as a reference for calculating the next frame, which help reducing the faces for calculating silhouette edges.

On the next frame, we reset the Edge Check array, then do calculate the silhouette edges based on the Silhouette Edge array of the previous frame. After finishing the loop based on Silhouette Edge array, we pick the root and the external nodes of the new Silhouette Edge array for finding the continuing edge and checking if it can be silhouette edge or not. With this method we can reduce the calculation on upcoming frame, and we can also get the continuous line between frames as well.

## IV. Reducing Data by Depth map

The silhouette edges that we get now are still having a problem. The silhouette edges, which should not be seen, because they are behind polygon mesh, are still showing. We need to get rid of them as well. We purpose to use depth map to check if an edge in found Silhouette edges is covered by faces at the front. Depth map is an image that contains depth of field data. We create depth map by checking depth of field of each face in object, then render each face as a gray-scale image with the value of normalized depth value. If the face's normalized depth value is less than depth map value at pixel of current rendered face, which means the current face is closer to viewing camera than other previous rendered faces, we change the value of depth map at current rendered face's pixels. If not, we omit the current value and keep the previous depth value. After rendering every face, we would have a depth map image of current frame. Depth map should be done every frame, because as the 3D-model is animated, the depth value of each face is changing depending on motion and viewing camera. Then we compare the edge's depth of field with depth map at edge's position. Edge's depth value can be found by getting the depth value of edge's vertices and calculating for the average value. Then project the edge's vertices' position to camera viewing plane, and use the position on camera viewing plane to get and compare depth value in depth map at the same position. If edge's depth value of any vertex's position value is greater than depth map, we can omit that edge since it is overlaid with faces at the front.

The depth map comparison is done after finding Silhouette edges at the first frame. We use depth map data to find the final Silhouette edges of the first frame. Then on the next frame, we would check if the previous-frame-Silhouette edges are still appearing on this current frame. If it's still appearing on current frame, we would check 1 step up and 1 step down in our edge tree. Normally, if the new Silhouette edges appear, they are the continuity of previous Silhouette lines. Therefore, we check the parent node of the topmost node in each Silhouette lines (or can be defined as starting point of a line). If this node can represent Silhouette edge on current frame, we put the edge of this node in our Silhouette Edge tree and reorder the tree in order to have this node work as the starting point of the line. Then checking the parent node of this adapted Silhouette lines again until we found the node, which cannot appear as Silhouette Edge. Similarly do the same with the bottommost node in each Silhouette line (or can be defined as end point). If found Silhouette edge, put it in Silhouette Edge tree and reorder the tree. Keep doing this until finding the node that cannot be Silhouette edge. However, if the first end point checking result is not Silhouette edge, we delete this edge from Silhouette Edge tree, reorder the tree, and check the adapted end point again until it is Silhouette edge. After checking for current frame Silhouette edge, we should do depth map comparison of current frame so that we could get only the visible lines. This proposed process is only checking the previous and the next edges, which extends from previous frame Silhouette lines. We dedicate the first frame for preparing the data and use it to reduce comparison on the next frame.

## V. Limitation

In order to experiment this approach, we limit some features of our 3D scene. The scene has only an animated 3D object and a still camera. This means we are working on only one polygonal model. It doesn't matter if the animated 3D Object is skeletal animation or deformed/transformed animation since our data collection is collecting the movement at vertex position. Moreover, the 3D object is composed of quad-polygon, which has no manifold. This is the standard polygon in OpenGL. Currently we limited the camera not to be animated. However, when dealing with animated camera, we need to store the camera data, especially the camera vector at each frame so that when doing the Silhouette edge checking on the next frame, we just change the camera vector value upon the current calculating frame.

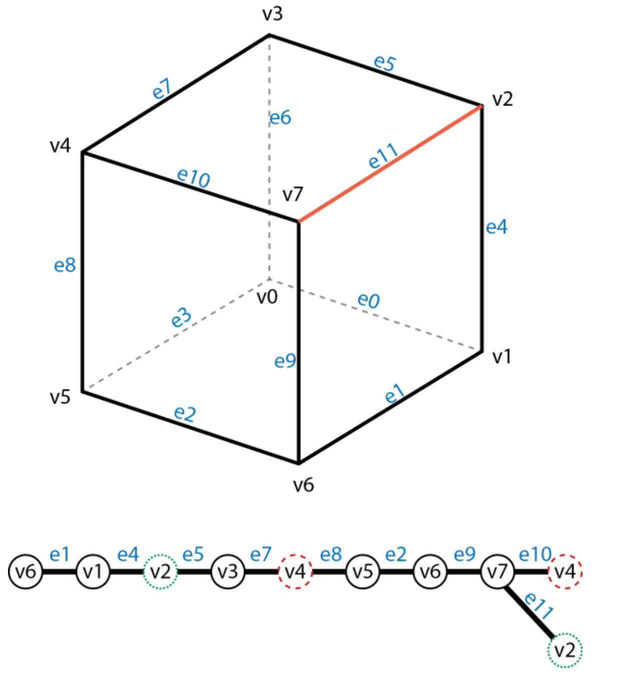


Fig. 3. The calculation of silhouette edges will result as tree data structure whereas the tree's branches can define the number of silhouette lines appearing on the scene.

## VI. Future Work

We have presented the technique for managing 3D data for detecting silhouette edges on a 3D animated model, which may result in losing some silhouette edges in some frame and may miss some possible Silhouette edges that might appear in a few frames. For further experiment, we would find some methodology to cover this issue. We might add function like defining default frame that would be the first frame to calculate. This frame would be selected by manually picking or by finding via the depth map of the frame that has most variation in values. Also we would add more procedure of making vector lines, which have the same starting point as frame goes by, and may be a procedure for line separating and merging since our Silhouette lines might not have the continuity in artist's satisfaction. It would be great if we can enhance some function that can support artist's vision. After that we can do stylization on final rendering.

## References

[1] A. N. M. I. Choudhury and S. G. Parker, "Ray Tracing NPR-Style Feature Line," in *Proc. the 7th International Symposium on Non-Photorealistic Animation and Rendering*, ACM New York, USA, pp. 5-14, 2009.

[2] J. C. Iehl. (February 2012). B1. Object file (*.obj). [Online]. Available: http://www710.univ-lyon1.fr/~jciehl/Public/utiles/maya_obj_spec.pdf

[3] L. Bourdev, "Rendering Nonphotorealistic Strokes with Temporal and Arc-Length Coherence," M.S. thesis, Brown University, Providence, RI, 1998.

[4] D. A. Hostetler, "Silhouette Edge Detection Algorithms for use with 3D Models," Intel Corp., Hillsboro, OR, Graphics Algorithm and 3D Technologies Rep., Feb. 2000.

[5] B. J. Meier, "Painterly Rendering for Animation," in *Proc. the 23rd annual conference on Computer graphics and interactive techniques* (SIGGRAPH '96)., ACM New York, USA, pp. 477-484, 1996.

[6] C. J. Curtis, "Loose and Sketchy Animation," in *Proc. ACM SIGGRAPH 98 Electronic Art and Animation Catalog.*, ACM New York, USA, pp. 145, 1998

[7] A. Gooch *et al.*, "A Non-Photorealistic Lighting Model For Automatic Technical Illustration," in *Proc. the 25th annual conference on Computer Graphics and Interactive Techniques* (SIGGRAPH'98), ACM New York, USA, pp. 477-452, 1998.

**Nucharee Thongthungwong** received B.S. degree in Architecture major in Industrial Design from King Mongkut Institute of Technology Ladkrabang in 2002. She has worked in animation industry for more than 3 years. Currently she is working on her master degree in Computer Science and Information Technology Program, Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University.

**Rajalida Lipikorn** received B.S. degree in Applied Mathematics and M.S. degree in Computer Science from California State University, Northridge, USA, in 1987 and 1992, and received Ph.D. in Bio-Applications and System Engineering from Tokyo University of Agriculture and Technology, Japan, in 2002. She is currently an assistant professor at Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University. Her research involves Image Processing, Computer Graphics, Medical Imaging, Pattern Recognition and Simulation and Visualization.