Journal of Korea Multimedia Society Vol. 11, No. 6, JUNE 2008(pp. 755-763)

A Hybrid Visibility Determination Method to Get Vector Silhouette

Xuemei Lu[†], Kijung Lee[††], Taegkeun Whangbo[†††]

ABSTRACT

Silhouette is useful in computer graphics for a number of techniques such as non-photorealistic rendering, silhouette clipping, and blueprint generating. Methods for generating silhouette are classified into three categories: image-based, object-based, and hybrid-based. Hybrid-based method is effective in terms of time complexity but spatial coherence problem still remains. In this paper, we proposed a new hybrid-based method which produces 3D data for silhouette and also guarantees no spatial coherence problem. To verify the efficiency of the proposed algorithm, several experiments are conducted for various 3D models from simple to quite complex. Results show that our algorithm generates no gap between any two consecutive silhouette lines when the silhouette model is magnified significantly.

Key words: Silhouette, Vector Silhouette, Visibility Determination, Object Space, Depth Buffer

1. INTRODUCTION

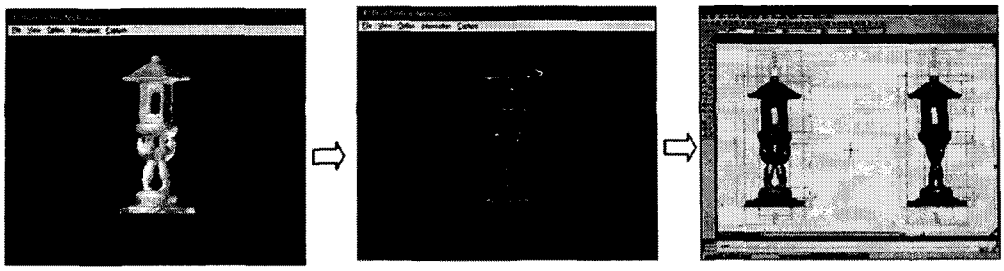
Silhouette lines have played an important role in the visual effect. Especially for non-photorealism rendering (NPR), which including cartoons, volumetric shadows, toon shading, technical illustrations, generating object silhouettes is an important component. Many reverse engineering works and architectural designs also need silhouette lines to represent the basic shape of the model. Silhouette extraction as the first step of these applications, has attracted more and more researchers work on it.

※ Corresponding Author : Taegkeun Whangbo, Address : (461-701) 5-7 Saeromkwan, Kyungwon Univ., Bokjeong-Dong, Sujeong-Gu, Seongnam-Si, Gyeonggi-Do, Korea, TEL : +82-31-750-5417, FAX : +82-32-757-9508, E-mail : tkwhangbo@kyungwon.ac.kr
Receipt date : May 1, 2007, Approval date : Nov. 12, 2007
† Dept. of Computer Science, Kyungwon University (E-mail :bingqing_0703@hotmail.com)
†† CTI of Kyungwon University (E-mail : jcm5758@empas.com)
††† Dept. of Computer Media, Kyungwon University
※ This research was supported by the Ministry of Cultural & Tourism and Korea Culture & Content Agency, Seoul, Korea, under Supporting Project of Culture Technology Institute.

Silhouette extraction algorithms include two stages: silhouette detection and visibility determination. Methods for generating silhouette are classified into three categories: image-based, object-based, and hybrid-based [1]. Image-based method, which is pixel-based, is fast but less accurate, and no metric information available. Object-based produces 3D information for silhouette lines but time-consuming, and spatial coherence problem may be occurred when removing invisible silhouette lines. Spatial coherence problem is a gap between two consecutive silhouette lines. Hybrid-based method, thus, has been proposed to solve these problems, and is effective in terms of time complexity but spatial coherence problem still remains.

In Fig. 1, a reverse engineering sequence to generate blueprint for the heritages is shown. In the first step, we need silhouette lines to depict the basic shape of the 3D heritage models, and they should have no disconnected lines.

In this paper, we proposed a hybrid method to do visibility determination, which can extract vector silhouette for generating blueprint of heritage models. We use z-buffer but produce 3D data in



3D heritage model Silhouette extraction Generate blue print

Fig. 1. Vector silhouette extraction for generating blueprint.

object space. We consider the visibility of each pixel on a silhouette line. The line and triangle have different drawing methods, which leads to different depth values and different projected pixels positions in z-buffer. We use a method with three depth buffers to overcome this problem. Instead of comparing the z-buffer values of an edge and the z-buffer value of mesh model, we compare between an adjacent face and mesh model. We have tested several mesh models with our method to verify the efficiency. We can get the 3D data of visible silhouette for generating blueprint. The vector silhouette extracted using our method is robust, even enlarging the vector data many times, there are no disconnected lines.

In this paper, we have a number of contributions as follows:

- Find out the reason that cause difference between line and triangle depth buffers' values, which is the main problem in image-based and hybrid-based method.
- Propose an approach to erase the difference between line and triangle depth buffers' values and get silhouette edges.
- Propose a whole system in order to extract silhouette lines for generating blueprint for culture heritages.

The following parts of this paper are organized as follows: Section 2 introduces some background of this research. Section 3 gives out the detail of our proposal. Section 4 is the experiment data. And the last section is the conclusion and future work.

2. RELATED WORKS

Lu et al. [2] describe a visibility method to extract vector silhouette for generating blueprint of culture heritages. The system includes four stages: 1. Building edge list from vertices and triangle information when loading mesh model; 2. Extract silhouette edges for a certain view direction with the Brute Force method to guarantee all the silhouette will be detected; 3. Determine the visible silhouette lines and rendering; 4. Save the vector silhouette lines according to user's subscribe. Fig. 2 shows the algorithm sequence:

In this paper, we use the same algorithm sequence with Lu et. al [2], but propose a new hybrid visibility determination method in step 3. We will introduce the existing algorithms, especially Lu's algorithm in the following part.

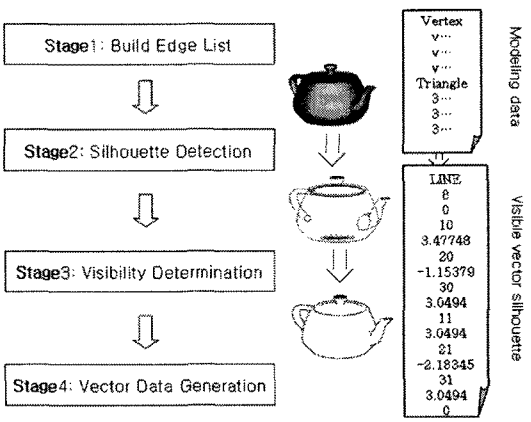


Fig. 2. Algorithm Sequence

2.1 Build Edge List

Since the silhouette definition is view dependent, we need to find all the edges from the vertices and triangle information, which can be used in real-time to extract the silhouette edges according to view direction.

There are two ways to find all the edges [3]. The simple one is a two-passes method. The first pass to find all the edges. The second pass to match the triangles to edges. it is $O(fCount^2)$, where fCount is the number of triangles. Using a hash table to store edges and their indices, we can achieve O(fCount) time with a single pass through the triangles. But we should allocate about twice as much space as we need to for the edges, since each edge is shared by two triangles.

In Lu et al. [2], they create the edge list using two dimensional structure to control the memory usage. The first dimensional of the structure has the same size of vertex number in vertex index sequence. Each element of the first dimensional array is a link list, which contains the edge that coming form this vertex. The space complexity of the final edge list is exactly the number of edges.

They traverse all the triangles and get the endpoint indices and adjacent faces information for each edge. For each triangle, there are three edges which defined with two endpoints of this triangle. We call these two indices as start point index and end point index sequently. If the start point index is less than the end point index, They check it has been stored or not, if not we recognize it as an unreferenced edge, and write it into our edge list, else ignore it.

By the combination of our two dimensional data structure and this one pass method, the time complexity of building edge list is Q(fCount), and the space complexity is the exactly the number of all edges.

2.2 Silhouette Edge Detection

A number of algorithms exist for extracting silhouette edges from polygonal models [4]. We use the Brute Force method to guarantee that all the silhouette edges can be found. This method requires testing the visibility of the two adjacent faces of each edge. At runtime, for every frame, we traverse the edge list. If one of the adjacent face is back facing and another is front facing, this edge will be a contour edge. If both the two adjacent faces are front facing face, and the dihedral angle is smaller than some threshold, it will be an crease edge. We write these edges into the silhouette edge list.

After building edge list and extracting silhouette with Brute Force algorithm, we need to test the visibility of each edge to extract the visible vector silhouette lines.

2.3 Visibility Determination

There are three general ways exist to attack hidden line removal problem. The visible line algorithm presented by Appel [5], which is frequently used in the context of silhouette algorithms. But these object space silhouette visibility tests are usually expensive [6]. A fast but less accurate way of determining silhouette edge visibility is an image space approach. However, in many applications only pixel accuracy is not enough [7,8]. Thus, combining object space and image space approaches in a hybrid algorithm can achieve significant speedup for the visibility test.

Northrup and Markosian [9] use an ID buffer in addition to regular rendering with a z-buffer to determine silhouette edge visibility. Isenberg et al. [10] use a similar approach, in principal. They note that simply scan converting silhouette edges and comparing the computed pixels with values in the z-buffer is somewhat numerically unstable. Thus, they suggest not only looking at the pixel's exact position but also at its 8-pixel neighborhood for pixels that are further away than the tested. This significantly reduces the numerical problems. But there are disconnected lines in the rendering result.

Actually the unstable of z-buffer caused by different rasterization method between line and triangle [2]. Lu's algorithm calculate the difference between z-buffer values on same pixel position for line and triangle to deal with the different z-buffer values and different pixel position problem caused by the rasterization method. But since for each pixel the difference should be calculated, it has a high time complexity. In this paper, we proposed a more easy way to solve this problem. And the triangle rasterization depends on the projected area, this cause the disconnect problem in z-buffer. Our proposal also deal with this problem. We will explain these problems more detail in section 4, and propose our method to solve them.

2.4 Intersection Coordinate and Vector Data File

After checking the visibility of all pixels on an edge, we can decide the visibility of the whole edge. If all the pixels are visible, this edge is visible; if all the pixels are invisible, this edge is invisible; if partly of the pixels are visible, we need to calculate the intersection coordinates to depart the visible part and the invisible part on this edge. Then we can write the visible silhouette lines into vector image [11].

3. VISIBILITY DETERMINATION

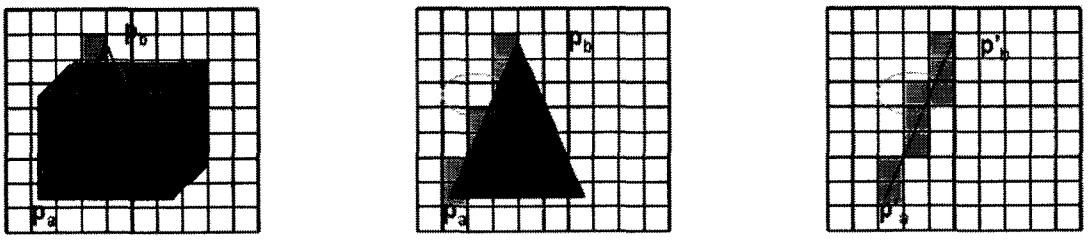
We compare the depth buffer values for mesh model and silhouette lines. We find that the values on same pixel positions are different. So we can not compare them directly. When we rotate the model, if the triangle area is small, there are no depth buffer values on some pixel positions. In this part we research the reasons about this three problems, and propose a new visibility determination method.

3.1 Three Rasterization Problems in Visibility Determination

The frist two problems are caused by difference between line rasterization and triangle rasterization. We explain these two problems firstly.

To examine the visible of segment edge, we take each silhouette edge separately. We compare the depth buffer values in Fig. 3 (A) and Fig. 3 (C). Obviously the line P_aP_b is partly hidden by the cube. So only the visible part of the line in Fig. 3 (A) has the same depth buffer values as in Fig. 3 (C). Unfortunately the values are totally not same. We use a media buffer for an adjacent triangle instead of line. We compare the depth buffer values in Fig. 3 (A) and Fig. 3 (B) on the projected pixel position in Fig. 3 (C). Another problem arises, there are no depth buffer values in Fig. 3 (B) while there are buffer values in Fig. 3 (C) in some case. See the position which indicated by circle in Fig. 3 (B) and Fig. 3 (C).

First, the projected pixel positions of line and triangle are different, as shown in Fig. 4. The triangle rasterization uses a scan line filling method. No matter line rasterization uses DDA line drawing



A: Z-buffer for model B: Z-buffer for an adjacent triangle of testing edge C: Z-buffer for testing edge

Fig. 3. Different pixel position between line and triangle rasterization of the testing silhouette edge

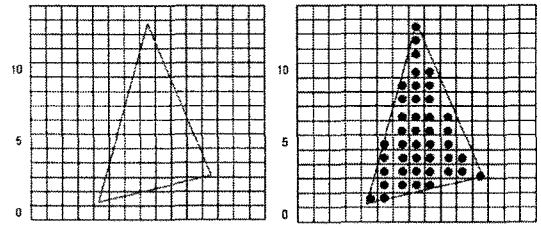


Fig. 4. Different projected pixel position

algorithm or Bresenham's algorithm, the projected pixel will have different positions [12,13].

Second, the depth buffer values on corresponding pixel position are different.

From the OpengGL tutorial [14], we obtained how the value is associated with rasterized line and polygon. For basic line segment rasterization, let the window coordinates of a produced fragment center be given by $P_d=(x_d, y_d)$ and let start point $P_a=(x_a, y_a)$ and end point $P_b=(x_b, y_b)$. Set:

$$t = \frac{(P_d - P_a)\cdot(P_b - P_a)}{\| P_b - P_a \|^2} \tag{1}$$

Note that t=0 at P_a and t=1 at P_b. The associated value on pixel P_d can get by:

$$f_L = (1-t)f_a + tf_b \tag{2}$$

f_L is the interpolated depth values on line, f_a and f_b are the data associated with the starting and ending endpoints of the segment, respectively. For basic polygon rasterization, let any point P_d within the triangle or on the triangle's boundary, we can find α, β and γ as:

$$\alpha = \frac{\Lambda(pp_bp_c)}{\Lambda(p_ap_bp_c)},\ \beta = \frac{\Lambda(pp_ap_c)}{\Lambda(p_ap_bp_c)},\ \gamma = \frac{\Lambda(pp_ap_b)}{\Lambda(p_ap_bp_c)} \tag{3}$$

Where $\Lambda(lmn)$ denotes the area in window coordinates of the triangle with vertices l, m, and n denote a depth buffer value at P_a, P_b, and P_c as f'_a, f'_b, and f'_c, respectively.

The next subsection will give the details to calculate the depth buffer value. Then the value fr of a datum at a fragment p produced by rasterizing a triangle is given by

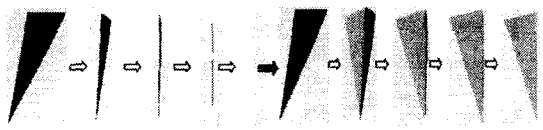
$$f_T = \alpha f'_a + \beta f'_b + \gamma f'_c \tag{4}$$

The third problem arises because that the accuracy of triangle projection depends on the area of triangle.

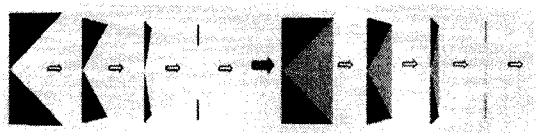
Because the triangle rasterization is based on the triangle area, when project one triangle to the screen, there maybe no rendering result, which means all the depth buffer value of the pixels are same as the background depth buffer values. Fig. 5 shows a case to indicate this situation. (a) The triangle rasterization is based on triangle area, there will no depth values on some projected pixel when rotating the triangle. So we render two adjacent triangles to make sure there are always z-buffer values on the shared edge. (b) For two triangles which share a endpoint only, there will be a disconnects in rotating. (c) By rendering the adjacent riangles of the shared edge, we can make sure there will be a depth value on projected pixel position of this shared edge.

3.2 Our Visibility Determination Method

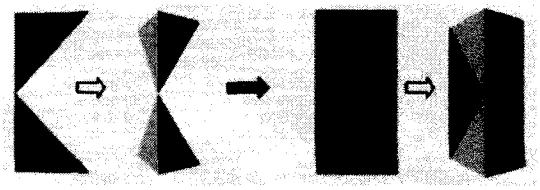
To determine visible silhouette segments, we



(a) The triangle rasterization is based on triangle area



(b) Depth values on the shared edge position



(c) Depth value on the end point position.

Fig. 5. The triangle rasterization problem in rotation

propose a method with the help of three buffers. we take each silhouette edge separately, instead of comparing depth values for mesh model and depth values for line, we use a media depth buffer value for the adjacent faces of this edge.

The visibility determination algorithm we proposed is described as follows:

- ***Step-a***: Render a mesh model and all silhouette lines, and read depth buffer values into array *Buffer1*.
- ***Step-b***: Render a silhouette edge and read depth values into *Buffer2*, and then render the adjacent triangles of the two endpoints of this edge, read depth buffer value into array *Buffer3*.
- ***Step-c***: For each pixel in *Buffer2* which value is not 1, (1 is the depth value for background) compare the value in *Buffer3* with the value in *Buffer1* on same pixel position. If the values are same, we set this pixel visible, else set this pixel invisible.
- ***Step-d***: If the pixels we checked are all visible, this edge is whole visible. If all the pixels are invisible, this edge is whole invisible. If partly visible, we calculate the intersection coordinate between visible and invisible part on this edge.
- ***Step-e***: If all the silhouette edges are checked, stop; else go to ***step-b***.
- ***Step-f***: Save the visible edge or partly visible edge into files.

4. RESULTS

We have tested our framework on some heritage models of Table 1. This table shows the number of vertices, triangles, edges, silhouette edges, visible silhouette edges (including the partly visible ones), ratio of silhouette edges and visible silhouette edges after visibility determination about different models. From these numbers, we can know the model size and have a overview of algorithm complexity.

We compare the time and space complexity of our algorithm with other algorithms in Table 2. For Apple's algorithm, it checks all the edges when considering the visibility of each edge, so the time complexity is $O(eCount^2)$, where eCount is the number of edges. Our hybrid method improves Isenberg's algorithm by exactly comparison instead of check the 8-neighbours. This results in a better rendering result. And the time complexity is also O(eCount), since we do the same process to generate visible silhouette lines.

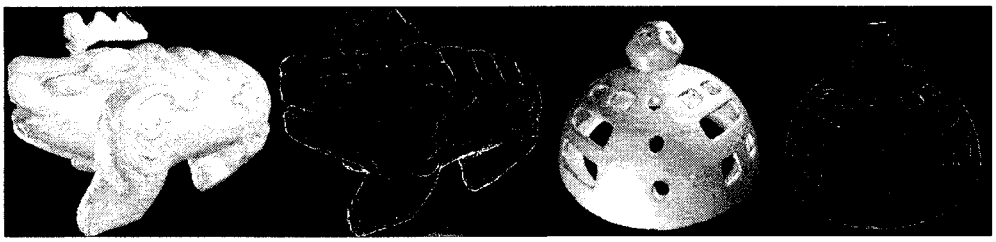
Fig. 6 shows some heritage models and their rendering results of visible silhouette. Fig. 7 is the final vector images of our algorithm. we can enlarge to measure the detail features or zoom out to measure the model size, which can be used di-

Table 1. Number information of models in our experiment

Model	Vertices	Faces	Edges	Silhouette Edges	Visible Silhouette	Sil/Edges %	Visib/Sil %
Vase	530,614	1,061,222	1,591,834	86,820	29,209	5.45	33.64
Totem	496,953	993,902	1,490,853	285,236	11,4893	19.13	40.28
Sculpture	399,774	799,548	1,199,322	35,138	20,065	2.93	57.10
Diagrams	249,284	498,708	748,062	51,778	30,527	6.92	58.96
Tortoise	206,127	412,250	618,375	48,290	19,528	7.81	40.44
Knife	181,515	363,018	544,527	10,798	4,034	10.98	37.36
Characters	35,958	71,916	107,874	7,259	6,064	6.73	83.53
Bunny	34,834	69,451	104,176	6,339	5,983	6.08	94.34

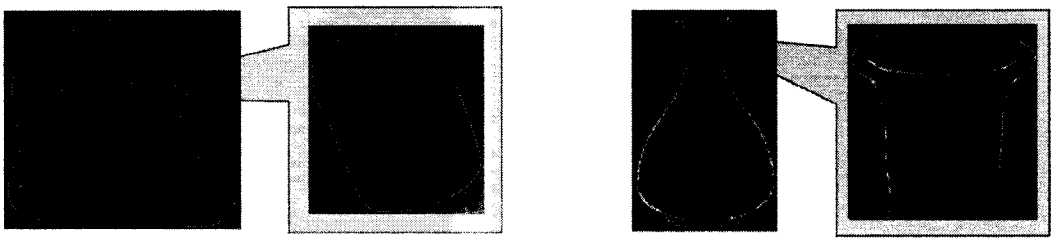
Table 2. Conclusion by comparing with classic algorithms (eCount is the number of edges)

	Appel's algorithm in object space[5]	Markosian's hybrid algorithm[9]	Isenberg's hybrid algorithm[10]	Image Space Algorithms[1]	My hybrid algorithm
Time complexity	$O(eCount^2)$	O(eCount)	O(eCount)	Interactive in realtime	O(eCount)
Accuracy	All visible vector silhouette edges	Cannot guarantee all visible vector silhouette edges, and disconnected lines	Disconnected lines	Pixel matrix	All visible vector silhouette edges



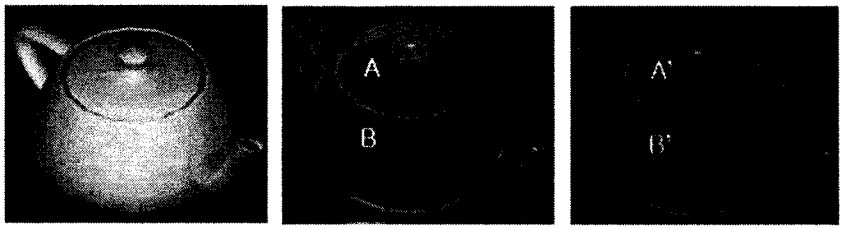
(a) Totem (b) Diagram

Fig. 6. Heritage models and their visible silhouette rendering results



(a) Chinese character and its zoom-in part (b) Vase and its zoom-in part

Fig. 7. Vector Silhouette Images.



(a) Teapot model (b) Isenberg's algorithm (c) Our algorithm

Fig. 8. Compare to Isenberg's algorithm [10]

rectly in industry engineering.

We have focused our effort on optimizing the rendering results. We implemented Isenberg's algorithm to extract silhouette, in order to compare it with our silhouette extraction scheme. In Fig. 8, compare region A, B in (b) and region A', B' in (c) separately, obviously our algorithm has a better rendering result, since the disconnected lines is not

allowed in generating blueprint. Both our algorithm and Isenberg's algorithm have same pixel accuracy.

5. CONCLUSION AND FUTURE WORK

We have shown the framework to extract visible silhouette from heritage models for generating blueprint. And we proposed a new hybrid method to do visibility determination with three depth buffers to generate analytic description of visible silhouette lines.

We use z-buffer but produce 3D data in object space. We consider the visibility of each pixel on a silhouette line. The line and triangle have different drawing methods, which leads to different depth values and different projected pixels positions in z-buffer. We use a method with three depth buffers to overcome this problem. Instead of comparing the z-buffer values of an edge and the z-buffer value of mesh model, we compare between an adjacent face and mesh model. We have tested several mesh models with our method to verify the efficiency. We can guarantee that all visible silhouette edges will be found. The vector silhouette extracted using our method is robust, even enlarging the vector data many times, there are no disconnected lines.

Our hybrid method improves Isenberg's algorithm by exactly comparison instead of check the 8-neighbours. We got a better rendering result which has no disconnected lines compare to Isenberg's algorithm. Both algorithms have the same time complexity, which is O(eCount), since we do the same process to generate visible silhouette lines, eCount is the number of edges of model.

Since our algorithm is a hybrid algorithm, the vector silhouette accuracy relates to the window resolution. We will use more precise windows resolution in the rasterization process to improve the precision of our method. In the future, we will use our visible silhouette extraction method to do more work about NPR.

REFERENCES

[1] T. Isenberg, B. Freudenberg, N. Halper, S. Schlechtweg, and T. Strothotte, "A Developer's Guide to Silhouette Algorithms for Polygonal Models," IEEE Computer Graphics and Applications, Vol.23, No.4, pp. 28-37, 2003.

[2] X. M. Lu, S. J. Eun, and T. K. Whangbo, "Vector Silhouette Extraction for Generating Blueprint," In Proceedings of the IEEE International Conference on Automation and Logistics, pp. 2946-2951, 2007.

[3] Eric Lengyesl, "3D Game Programming & Computer Graphics," Charles River Media, 2001.

[4] A. Hertzmann and D. Zorin, "Illustrating Smooth Surfaces," In Proceedings of ACM SIGGRAPH 2000 Computer Graphics Proceedings, pp. 517-526, 2000.

[5] A. Appel, "The Notion of Quantitative Invisibility and the Machine Rendering of Solids," In Proceedings of ACM National Conference, pp. 387-393, 1967.

[6] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Richard Riesenfeld, "Interactive Technical Illustration," In Proceedings of 1999 Symposium on Interactive 3D Graphics, pp. 31-38, 1999.

[7] J. W. Buchanan, M. C. Sousa, "The Edge Buffer: A Data Structure for Easy Silhouette Rendering," In Proceeding of First International Symposium on Non Photorealistic Animation and Rendering (NPAR), pp. 39-42, 2000.

[8] R. Raskar and M. Cohen, "Image Precision Silhouette Edges," In Proceedings of the 1999 Symposium on Interactive 3D Graphics, pp. 135-140, 1999.

[9] L. Markosian, M. A. Kowalski, S. J. Trychin, L. D. Bourdev, D. Goldstein, and J. F. Hughes. "Real-Time Non-Photorealistic Rendering,"

In proceeding of SIGGRAPH '97, pp. 415-420, 1997.

[10] T. Isenberg, N. Halper., and T. Strothotte, "Stylizing Silhouettes at Interactive Rates: From Silhouette Edges to Silhouette Strokes," In Proceedings of Eurographics, Computer Graphics Forum, Vol.21, No.3, pp. 249-258, 2002.

[11] AutoCAD Release 12 DXF Format, "Drawing Interchange and File Formats," Release 12, Copyright (c) 1982-1990, Autodesk, Inc. 1992.

[12] R. Raskar, "Hardware Support for Non-Photorealistic Rendering," In Proceecdings of SIGGRAPH Eurographics Workshop on Graphics Hardware (HWWS), pp. 410-461, 2001.

[13] D. Blythe, B. Grantham, S. Nelson, and T. Mcreynolds, "Advanced Graphic Programming Techniques using OpenGL," In proceedings of International Conference on Computational Science and its Applications (ICCSA2004), pp. 247-256, 2004.

[14] "Basic Line Segment Rasterization," http://www.opengl.org/documentation/specs/version1.1/glspec1.1/node47.html

Xuemei Lu

She received her B.S degree from Sandong University, Sandong, China in 2005. And she received her M.S. degree from Kyungwon University in Seongnam, Korea in 2007. She is currently a Ph.D. degree student of the Dept. of Computer Science in Kyungwon University, Korea. Her research interests include Level of Detail, Point-based Rendering, Non-Photorealistic Rendering.

Kijung Lee

He received his M.S. and Ph.D. degrees from Kyungwon University, Korea in 2003 and 2008. He is currently a researcher of CTI of Kyungwon University, Korea. His research interests include 3D Game Engine, Point-based Rendering, Non-Photorealistic Rendering.

Taegkeun Whangbo

He received his Ph.D. degree from Stevens Institute of Technology, USA in 1995. He is currently a professor of Dept. of Computer Media in Kyungwon University, Korea. His research interests include 3D Game Engine, Level of Detail, Physical Rendering.