STYLIZED SILHOUETTE RENDERING USING PROGRESSIVE MESHES

Sung-Soo Kim, Seung-Keol Choe

Spatial Information Technology Center
Computer & Software Research Laboratory
Electronics and Telecommunications Research Institute
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350, Republic of Korea
sskim@camis.kaist.ac.kr, skchoe@etri.re.kr

ABSTRACT

Silhouette information has been used to enhance artistic rendering of 3D objects. We present a new method for progressive silhouette rendering of triangle mesh of arbitrary topology by using parameterized brush functions in various styles.
The proposed progressive silhouette rendering framework is consist of two major steps; one is mesh simplification for silhouette feature preservation and the other is the stylized silhouette edge rendering. We also improve the mesh simplification algorithm that can preserve silhouette and volume of arbitrary mesh for silhouette rendering. First, for a given mesh, we can obtain progressive mesh(PM) through our proposed mesh simplification algorithm at the preprocessing step. Then, we can render silhouette edges by various kinds of effects progressively by performing the refinement of a given PM from a base mesh.
Results demonstrate that progressive silhouette rendering which use the progressive mesh and brush functions can cause effect that a person sketches an arbitrary object gradually.

Keywords: non-photorealistic rendering, mesh simplification, progressive sketching, silhouette rendering

1 INTRODUCTION

Recent advances in three-dimensional acquisition, simulation, and design technologies have lead to the growing complexity of the scenes and models being rendered. In order to cope with large polygonal meshes, *mesh simplification* is often used, in which a highly detailed mesh is approximated with a smaller number of triangles. While the simplified mesh generally isn't an exact replica of the original mesh, a small loss in fidelity is often considered an acceptable tradeoff for the elevated display rates, reduced storage requirements, and improved processing speeds afforded by less detailed meshes.
Among the varied goals of artistic *Non-Photorealistic Rendering* (NPR) is the pursuit of perceptually efficient images. A perceptually efficient visual representation emphasizes important features and minimizes extraneous detail and is essential for making comprehensible artistic images. Computer-generated 3D line drawings borrow from centuries of artists' techniques and have recently received significant attention in the NPR community.

It has been recognized that silhouettes are an important visual cue that humans use to determine shape and recognize objects. In the computational vision community there is a large body of work studying how shape information can be computed from silhouette data. In computer graphics, silhouette information has been used to enhance the expressive rendering of 3D objects. However, the combination of these two techniques, *mesh simplification* and *silhouette rendering*, has not been introduced yet.
In this paper, we combine mesh simplification with silhouette rendering to render silhouette edges of a triangle mesh progressively. We present a new silhouette rendering algorithm which can express silhouette edges according to parameterized brush functions, can progressively render 3D objects in various styles by using the *progressive mesh* (PM).

2 RELATED WORKS

First, we discuss related work on mesh simplification and LOD computation methods. Then we discuss previous work on silhouette edge rendering and nonphotorealistic rendering.

Mesh Simplification
Level-Of-Detail (LOD) is concerned to the possibility of using different representations of a geometric object having different levels of accuracy and complexity. A number of algorithms have been proposed for computing LODs in the last few years. The goal of mesh simplification is to represent a polygonal mesh with fewer vertices, but to keep a good approximation of the original mesh. These include algorithms based on vertex clustering, edge collapses[Hoppe96], vertex removal, quadric error metrics[Heckbert97], simplification envelopes, memoryless simplification[Lindstrom98].
We present a mesh simplification algorithm to preserve volume and boundary of triangle mesh and to improve processing speed / memory efficiency by using edge collapses.
A popular method of implementing LOD is to use a data structure known as a *progressive mesh* (PM)[Hoppe96]. The progressive mesh consists of a coarse base mesh and a set of refinement information (sequence of *vertex splits*) that can be used to increase the complexity of arbitrary mesh.
The surface simplification using quadric error metrics method is based on the iterative contraction of edge pairs, which also allows to join unconnected regions of the model[Heckbert97]. The atomic operation, *vertex pairs contraction*, may be conceived as a less general vertex clustering operation.
As simplification proceeds, a geometric error approximation is maintained at each vertex of the current model. The error approximation is represented using quadric metrics. The main advantages of this approach are *computational efficiency* and *generality*. It also allows the simplification of *disconnected* or *non-manifold* meshes.
We compare our proposed algorithm with this algorithm in terms of processing speed and maximum geometric errors of simplified mesh.

Nonphotorealistic Rendering
There has been extensive research for illustrating surface shape using non-photorealistic rendering techniques. Adopting a technique found in painting, Gooch *et al.* developed a tone-based illumination model that determined hue, as well as intensity, from the orientation of a surface element to a light source[Gooch98].
Silhouette edges are particularly important in the perception of surface shape, and have been utilized in surface illustration and surface visualization rendering. The extraction and rendering of silhouettes and other expressive lines has been addressed by several researchers [Salisbury94, Gooch99].
Silhouette edges can be rendered using hidden line removal methods, which are typically batch processes. There are also a number of algorithms described for extracting silhouettes from arbitrary polygonal models. The method proposed by Markosian *et al.* is real time and works on static polyhedral models with known adjacency information[Markosian97]. They use a probabilistic method to identify silhouette edges. The method proposed by Rossignac *et al.* is image precision and does not need adjacency information[Rossignac92]. The visibility of silhouette edges is computed using modified Appel's hidden-line algorithm assuming the view is generic[Appel67].
Raskar *et al.* describe a simple general-purpose method to combine all three operations for any scene composed of objects that can be scan-converted[Raskar99]. Gooch *et al.* describe a hierarchical Gauss map for quickly rejecting edges that are not on the silhouette[Gooch99].

Contributions
The main objective of this work is to contribute to non-photorealistic rendering of multiresolution models. The contributions of this paper are:

- We introduce the idea of progressive silhouette rendering algorithm in various brush styles.
- We describe a concrete mesh simplification algorithm that can preserve the volume, boundary and silhouette of arbitrary triangle mesh.

3 PROGRESSIVE SILHOUETTE RENDERING ALGORITHM

Our proposed algorithms for progressive silhouette rendering are composed of mesh simplification algorithm and silhouette rendering algorithm. First, we create a progressive mesh for silhouette rendering which can be obtained from original mesh through our proposed mesh simplification algorithm at the preprocessing step. Then we progressively render silhouette edges in various styles for the input PM by using parameterized brush functions. The Figure 1 shows processing flow of progressive silhouette rendering system that we proposed.

3.1 Volume and Boundary Preservation

In this paper, similar to most contemporary mesh simplification methods, we use the edge collapse operation to iteratively coarsen the mesh (See figure 2). It does this by repeatedly selecting the edge with a minimum cost, collapsing this edge, and then reevaluating the cost of edges affected by this edge collapse. Therefore, there are following two important processes to achieve mesh simplification.

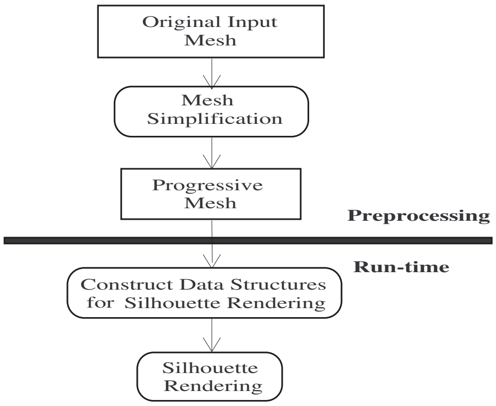


Figure 1: The flow of progressive silhouette rendering

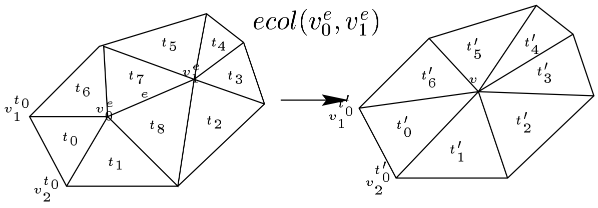


Figure 2: The edge collapse operation[Hoppe96]

- The precedence decision of the edge collapse
- New vertex placement after edge collapse operation

The first step in the simplification process is to assign costs to all edges in the mesh, which are maintained in a priority queue. For each iteration, the edge with the lowest cost is selected and tested for candidacy; if the edge is not a valid candidate, it is removed from the queue.
The second step is to place new vertex after edge collapse operation performed. The vertex placement scheme has a number of desirable properties; it is volume-preserving, it attempts to minimize the aggregate deviation, expressed in terms of tetrahedral volumes, between the two surfaces in each edge collapse, and it also explicitly accounts for changes made to surface boundaries by minimizing similar integral measures of deviation.
A *two-manifold surface* is a topological space where every point has a neighborhood topologically equivalent to an open disk of the two-dimensional Euclidean space E^2. Intuitively, two-manifolds are nonselfintersecting, closed surfaces. One important geometric characteristic of a closed surface is the amount of volume it encloses. In order to preserve the volume enclosed by the surface, we use the following cost function for placing a new vertex as in [Lindstrom98]. However, we don't use volume optimization procedure similar to memoryless simplification. One of the reasons why volume preservation is desirable is that it keeps cascading errors to a minimum. If the vertex placement were not volume-preserving, we may inadvertently end up accumulating errors along a single direction, for example by repeatedly inflating the model.
Thus, volume preservation ensures that here is no bias in the error introduced by a single edge collapses, and that any inflation is always offset by an opposite deflation. We will need to compute the signed volume V of a tetrahedron formed by the vertex x and the three vertices of a triangle t_i.
We here derive a matrix form for this quantity:

$$\sum_i V(v, v_0^{t_i}, v_1^{t_i}, v_2^{t_i}) = \sum_i \frac{1}{6} \begin{vmatrix} v_x & v_{0_x}^{t_i} & v_{1_x}^{t_i} & v_{2_x}^{t_i} \\ v_y & v_{0_y}^{t_i} & v_{1_y}^{t_i} & v_{2_y}^{t_i} \\ v_z & v_{0_z}^{t_i} & v_{1_z}^{t_i} & v_{2_z}^{t_i} \\ 1 & 1 & 1 & 1 \end{vmatrix}$$

We can get the cost function for volume preservation as the following:

$$F_{vol}(e, v) = \sum_i (v_0^{t_i} \times v_1^{t_i} + v_1^{t_i} \times v_2^{t_i} + v_2^{t_i} \times v_0^{t_i})^T v$$
$$= \sum_i {n_i}^T v = \sum_i \left[v_0^{t_i}, v_1^{t_i}, v_2^{t_i} \right]$$

where, n_i denotes the triangle, t denotes the normal vector of the triangle.

When dealing with boundary edges, we will use a similar notation for computing the area vectors associated with these edges.
We define a cost function for boundary preservation as the following:

$$F_{boun}(e, v) = \sum_i A(v, v_0^{e_i}, v_1^{e_i}) =$$
$$\left\| \sum_i \frac{1}{2}(v \times v_0^{e_i} + v_0^{e_i} \times v_1^{e_i} + v_1^{e_i} \times v) \right\|$$

The priority decision of edge collapse is important process in the mesh simplification. The edge collapse criteria to remove edge in a mesh are the length of edge, $L(e)$, the sum of degrees of two end-vertices, $d(v)$ which is derived from the *degree method*[Jünger98].
We define the cost function which calculates cost that correspond to each edge according to above criteria. In case of we apply only the cost function $L(e)$, the simplified result of the mesh has equal triangles being removed edges equally. If we increase simplification level, it becomes hard to keep volume and the boundary characteristic of the mesh. So, we should consider neighborhood information of edges. Therefore, we must include information that can preserve volume and boundary at the priority order decision of edge collapsing.
Based on these error measures, the best edge collapse in each iteration is the one that yields the smallest amount of change. Formally, the edge cost F_C is written as

$$F_C = \lambda F_{vol} \cdot d(v) + (1 - \lambda) L(e)^2 \cdot F_{boun} \qquad (1)$$

The parameter λ provides the user with explicit control over the tradeoff between surface and boundary fidelity. In general, $\lambda = \frac{1}{3}$ trends to work well.
A *progressive mesh* is created by starting with an input mesh and applying a series of edge collapses. Because a vertex split and an edge collapse are inverse operations, one can apply a series of edge collapses, and then apply the corresponding vertex splits (in reverse order) to recover the original mesh. The data structure for the PM representation corresponds closely with the tuple(M^0, $\{vsplit_0$, ... ,$vsplit_{n-1}\}$). By using this property, the simplified mesh, which is produced by the proposed cost function, is stored PM form as the following data structure[Hoppe96].

```
struct CPMesh {
    CMesh base_mesh;// base mesh M^0
    Array<Vsplit> vsplits;//{vsplit_0, ... ,vsplit_{n-1}}
    int full_nvertices;// number of vertices in M^n
    int full_nwedges;// number of wedges in M^n
    int full_nfaces;// number of faces in M^n
};
```

The *base_mesh* field stores M^0 using the CMesh structure which means an abstract data structure for any triangulated mesh. We can apply vertex split operations to recover the original mesh progressively. Figure 3 shows an example of the progressive silhouette rendering result of the cow model using PM.

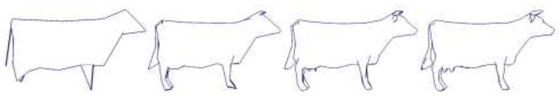


Figure 3: An example of progressive silhouette rendering (100 faces, 300 faces, 600 faces, 2,024 faces)

3.2 Silhouette Preservation

To preserve object silhouette, we should preserve the details of polygons with view orthogonal direction and surface boundary of the silhouette edge normal with view direction and refine the polygons on the silhouettes.
Now we describe *multiresolution view sphere* structure[Kim98]. We assume that an object is represented as set of polygons. View sphere is a Gaussian sphere that maps the normal direction of polygons to the point in the view sphere. Points on the sphere (x, y, z) maps the unit direction vector (x, y, z). Polygons that have the direction vector (x, y, z) are mapped into the point (x, y, z) on the view sphere. Each point in the view sphere represents the set of polygons that have the direction of the point represents. Distance of two direction vectors on the object surface is represented as distance of two points in the sphere. Using view sphere, polygons that have similar directions with the view are easily identified. If view direction is given as v, visible polygons with similar directions are located at the neighbor part of the point v on the surface of the view sphere.
To give efficient access to the view sphere, we have partitioned the sphere into discrete view cells. Each cell occupies part of the view sphere. Each cell contains a set of regions, which represents simplified representation of polygons mapped into that part of the sphere.
In the example of the silhouettes, object silhouette parts have more detail. Conversely, parts that have similar normal directions with view direction and back faces have less detail.
We define a view cell that covers the view direction as a *view directional cell* and view cells with orthogonal normal as a *view orthogonal cell*. Our algorithm chooses a simplified cell for a view directional one and uses refined cell for view orthogonal ones. We assumed that object simplification is occurs when the object is far enough to the viewer. So, polygons in the view orthogonal cells form a silhouette of the object to the view.

3.3 Silhouette Edge Rendering

The most straightforward way to find silhouette edges is to calculate the dot product of the two face normals that share an edge with the view vector.
The equation (2) describes the condition between face normals N_1, N_2 and eye vector V for the silhouette edge detection (See the figure4).

$$(N_1 \cdot V) * (N_2 \cdot V) \leq 0 \tag{2}$$

If the above condition is satisfied then the edge shared by front face and back face is a silhouette edge. This implies at run time we need to examine all edges of an object.

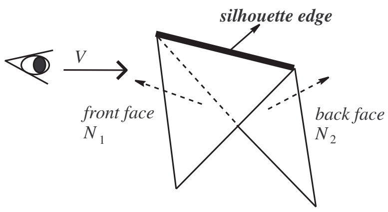


Figure 4: Silhouette Edge Detection

The proposed silhouette rendering system detects and draws the silhouettes of a 3D triangle mesh in real time. In addition to silhouettes, we find edges that mark other key features of a model, such as crease and border edges.
A *crease edge* is detected when the dihedral angle between two faces is greater than a given threshold.

Border edges are those that lie on the edge of a single polygon, or that are shared by two polygons with different materials.
We also find silhouettes in each frame by taking the dot products of the face normals of the two faces adjacent to an edge with the viewing vector, and comparing the product of these two dot products with zero. If the result of this computation is less than or equal to zero, the edge is a silhouette edge and it is flagged for rendering.
The proposed silhouette edge rendering system performs the following steps to achieve effective rendering result of arbitrary polygonal mesh.

1. find all visible silhouette edges from the viewpoint, i.e. boundaries between adjacent front facing and back-facing polygons.
2. solve the partial visibility problem to render only those parts of silhouette edges, which are not occluded by the interior of any front facing polygons.
3. parameterize the silhouette edge
4. define parameterized silhouette brush functions using the distance between tangent and normal vector.
5. render silhouette edges in various style using parameterized brush functions.

The proposed algorithm uses the modified Appel's algorithm, and uses *Half Edge* data structure to improve normal vector calculation speed for two triangles adjacent to one edge. In the preprocessing step, it allocates memory for an edge list including an edge flag for each edge to indicate whether it is a border, crease, or silhouette edge. Iterate over the faces of the model and create a unique edge list using a hash table. The hash function for an edge is the sum of the two vertex indices of the edge. Set border flag for edges with only one neighboring face. For non-deformable geometry, set crease flag for edges for which the dihedral angle between the two neighboring faces is greater than the crease threshold. This algorithm describes in the Algorithm 1.

The runtime algorithm for silhouette rendering describes in the Algorithm 2. First, we calculate face normals if necessary. For deformable meshes, it detects crease edges from face normals and set crease flags. It detects silhouette edges via equation (2) and set silhouette flags in the edge list.
Finally, it renders edges whose edge flag is set through traversing the edge list. Silhouettes are rendered using line segments, the width of which can be adjusted according to lighting parameters, a distance metric, or a user-defined parameter. As an object moves away from the eye, the silhouette line width can be reduced with increasing distance.

Algorithm 1 preprocess_silhouettes

```
edge_list* EdgeList = new edge_list;
initialize EdgeList;
for each edge e_i in EdgeList do
  T_{e_i} = h(sum(e_i.v_1, e_i.v_2));
  if (e_i has one neighboring face f) then
    set e_i.flag = BORDER;
  end if
  A_{e_i} = dihedral_angle(f_1, f_2);
  if (A_{e_i} > crease_angle (e_i)) then
    set e_i.flag = CREASE;
  end if
end for
```

Algorithm 2 runtime_algorithm

```
calculateFaceNormals(n)
detect crease edges from n
e_i.flag = CREASE;
for each edge e_i in EdgeList do
  if (e_i is silhouette edges) then
    renderEdge(e_i);
  end if
end for
```

For a given input mesh, it creates the Half Edge information to render silhouette edges by using brush functions. The process of creating Half Edge requires $O(n \log n)$ time complexity since it has sorting procedure.

3.4 Stylized Silhouette Edges

Stylized rendering has been a research topic in computer graphics world for a number of years. The motivation is to imitate the stylizations that artists adopt in different media, such as pen and ink work, where, for example, various hatching patterns are employed. The proposed algorithm parameterizes each edges in all silhouette edges at edge parameterization step. All silhouette edges that are found in the previous step are then individually processed. Each silhouette edge is parameterized as a line so that we can offset all of the different points on the silhouette edge. We want to be able to this to provide a random/human sketched look to the silhouette edge. It parameterizes two end-vertices of each edge with regulating to each other size. Here we can obtain effects that person seems to sketch arbitrary 3D object randomly.

3.5 Parameterized Brush Functions

A *brush function* is just a function that will describe what is going on in a particular brush in a very low level. For example, the charcoal stroke's brush function is just a high frequency sawtooth curve. The justification for this can be seen if one tries to draw a charcoal-like scene on paper with a pen; one ends up shaking the pen up and down on the paper in order to achieve similar effects. The same applies here. With parameterized silhouette edges, we can the parameterize the tangent of the edge as well as the normal of the edge. The tangent and normal are used in conjunction with the brush fucnction as vectors offsets for the point. The equation for this is :

$$q(t) = p(t) + v_x(t) \cdot p'(t) + v_y(t) \cdot n(t)$$

where, $q(t)$ denotes the brush stroke's point at t, $p(t)$ denotes the "regular" point's coordinates, $v_x(t)$ denotes the x-value of the brush function, $v_y(t)$ denotes the y-value of the brush function, $p'(t)$ and $n(t)$ denote the edge's tangent and normal at t respectively. Table 1 shows brush functions that we proposed.

Brush	*Brush function*
Charcoal	High frequency sawtooth curves
Lazy	Low frequency parabolic curves
Hand	Low magnitude noise (normal, tangent)
Rough	High magnitude noise (tangent)

Table 1. Brush Functions.

Each brush function that is explained in Table 1 is defined to mathematical expression such as the followings:

$$v_c(t) = t/C * dist_z$$

$$v_l(t) = t/L * dist_z$$

$$v_h(t) = (r * dist_z)/\alpha$$

$$v_r(t) = (r * dist_z)/\beta$$

where, C and L are constants, α and β are threshold values, $r = rand()/MAX_{rand}$, $dist_z = Z_{max} + |Z_{min}|$.

The Algorithm 3 describes the silhouette rendering algorithm using proposed brush functions. The *CalculateDotProduct* function produce the result of dot product of current view vector and normal vector of triangle. If each result of the *CalculateDotProduct* of two triangles that share an edge is different each other or zero, then it performs the silhouette rendering procedure. There are $3n$ edges in a triangle mesh where n denotes the number of vertices in a triangle mesh. Therefore, the number of half edges is $6n$. The time complexity of the Algorithm 3 is $O(n)$ after the half edge is constructed. But, the total time complexity of the proposed algorithm is $O(n \log n)$ since it requires a sorting procedure.

Algorithm 3 RenderWithBrush

```
for each halfEdge i do
    This = CalculateDotProduct(i →origin);
    Twin = CalculateDotProduct(i →twin);
    if (Twin <=0 and This >=0) or (Twin >=0 and
    This <=0) then
        ParameterizeEdge(i);
        CalculateBrushFunction(i);
        DrawBrushEdge(i);
    end if
end for
```

To integrate these two different techniques, PM and silhouette rendering, we maintain each data structures for PM and silhouette rendering. We treat this integration problem in two steps. In the first step we apply vertex split operation for a vertex v_i in PM. Actually, the vertex split operation returns a new edge of given simplified mesh. Therefore we can obtain a new edge e_i of mesh. In the second step we insert this edge e_i to half edge data structure for silhouette rendering. After this second step, we can render stylized silhouette edges by using parameterized brush functions. So, our algorithm requires PM and half edge data structures for achieving progressive silhouette rendering.

The following pseudo-code shows an outline of the integration process for an arbitrary triangle mesh m.

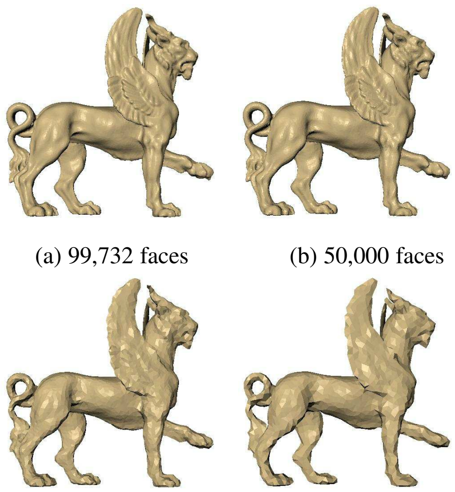
```
CPMesh    pMesh(m);
CHalfEdge hEdge(m);
for each i-th refinement step do
    CEdge e_i = pMesh.vSplit(v_i);
    hEdge.insert(e_i);
    perform the Algorithm 3;
end for
```

where CHalfEdge means an abstract half edge data structure for silhouette rendering, CEdge denotes an edge of triangle mesh and v_i denotes a vertex for performing vSplit operation in the i-th refinement step.

4 RESULTS

We have implemented our algorithm in C/C++, tested our non-optimized implementation on several datasets which is obtained from California Institute of Technology(feline, horse model) and CMU graphics lab(cow, bunny model). All experiments were conducted on a 850 MHz Pentium III running Microsoft Windows 2000. Our system had 512 MB of RAM and a graphics accelerator based on the NVIDIA GeForce 256 chip.

We simplified all models using our simplification tools (*VBPSlim*); the results of simplified feline models are shown in figure 4. The most important aspects of the progressive silhouette rendering process to an-



(a) 99,732 faces (b) 50,000 faces

(c) 10,000 faces (d) 5,000 faces

Figure 5: The result of our proposed simplification tool (*VBPSlim*).

alyze are the *simplification processing time* and *geometric errors* .

SIMPLIFICATION SPEED By optimizing our code, and by taking advantage of the fact that we only need to store base meshes in memory, we were able to achieve significant speed improvements in our simplification algorithm. We compare simplification speed with the *QSlim*[Heckbert97]. The simplification time of the bunny model is shown in figure 6. As shown in figure 6, we improve the simplification speed rather than the *QSlim*.

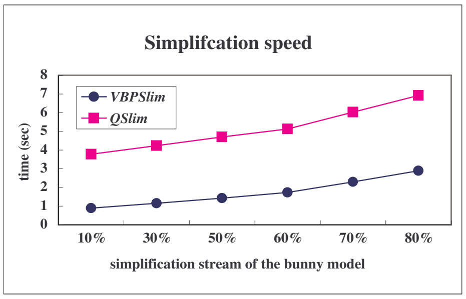


Figure 6: Comparison of the simplification speed.

GEOMETRIC ERRORS Our faster simplification algorithm would be of little value if it produced poor results. Thus it is important to verify the equality of the simplification produced. We use the *Metro* [Cignoni96] that can compute geometric errors of a simplified model in order to compare visual quality. We also compare the geometric errors of the simplified model between the *QSlim* and *VBPSlim* by using the Metro tool. In geometric errors, the *QSlim* is better than our proposed algorithm. Figure 8 shows the silhouette rendering results of Raskar's proposed

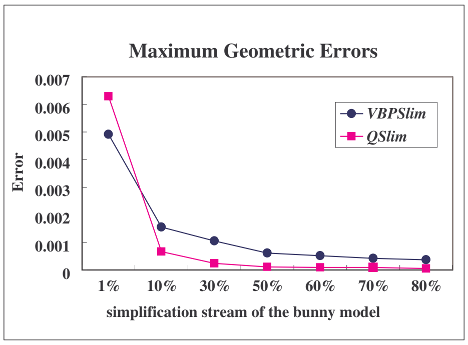
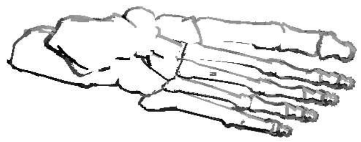
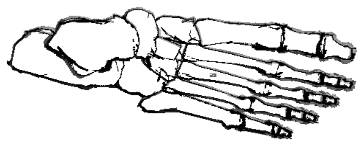


Figure 7: Maximum geometric errors

method and our proposed algorithm.



(a) A result of Raskar[Raskar99]



(b) A result of proposed method (using Charcoal brush function)

Figure 8: Stylized silhouette rendering

Figure 9 shows original horse model which uses in our experiment.

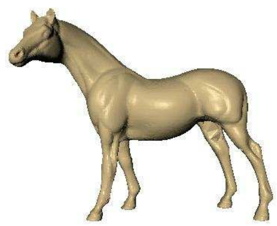


Figure 9: horse model : 48,485 vertices, 96,966 faces

Figure 10 shows the result of progressive silhouette rendering by using Charcoal brush function. A more artistic image might be achieved by varying the silhouette line density according to the light source.

5 CONCLUSIONS

We have described a new method for progressive rendering silhouette edges using level-of-detail meshes. We have used *stylistic rendering* with stylized brush

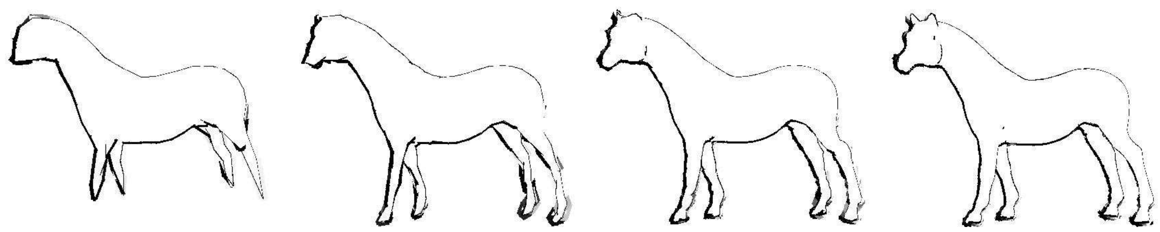


Figure 10: The result of progressive silhouette rendering. (500 faces, 1000 faces, 5000 faces, 50000 faces)

functions and *progressive nonphotorealistic rendering* techniques interchangeably.

In mesh simplification algorithm, we have presented a mesh simplification algorithm which which is based on local changes in volume over the surface and changes in area near surface boundaries. We can progressively render silhouette edges by using a preprocessed progressive mesh that is obtained by our proposed simplification algorithm.

In mesh simplification step, we can progressively render silhouette edges through applying iterative *vertex splits* to progressive mesh which had reconstruction history for original mesh. Results demonstrate that progressive silhouette rendering which use the progressive mesh and brush functions can cause effect that a person sketches an arbitrary object gradually.

REFERENCES

[Appel67] A. Appel, The Notion Of Quantitative Invisibility And The Machine Rendering Of Solids, *ACM National Conference '67 Proc.*, pages 387-393, 1967.

[Cignoni96] Cignoni P., Rocchini C. and Scopigno R., Metro: Measuring Error on Simplified Surfaces, *Technical report, Istituto I.E. I. C. N. R*, Jan. 1996.

[Gooch98] Amy Gooch, Bruce Gooch, Pter Shirley, and Elaine Cohen. A Non-photorealistic Lighting Model for Automatic Technical Illustration. *In Computer Graphics (SIGGRAPH ' 90 Proceedings)*, Aug. 1998.

[Gooch99] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Rich Riesenfeld. Interactive Technical Illustration. *In ACM Symposium on Interactive 3D Graphics ' 99*, pages 31-38, April. 1999.

[Heckbert97] Paul S. Heckbert and Michael Garland. Surface Simplification Using Quadric Error Metrics. *In SIGGRAPH ' 97 Proc.*, Aug. 1997.

[Hoppe96] Hugues Hoppe. Progressive meshes. *In SIGGRAPH ' 96 Proc.*, pages 99-108, Aug. 1996.

[Jünger98] Berend Jüger and Jack Snoeyink, Selecting Independent Vertices For Terrain Simplification, *In WSCG'98 Proceedings* , pages 157-164, 1998.

[Kim98] HyungSeok Kim, SoonKi Jung and KwangYun Wohn, A Multiresolution Control Method Using View Directional Feature, *In Proceedings of ACM VRST '98, Nov.*, pages 163-169, 1998.

[Markosian97] Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, Lubomir D. Bourdev, Daniel Goldstein and John F. Hughes. Real-time Nonphotorealistic Rendering. *In Computer Graphics(SIGGRAPH ' 97 Proceedings)*, Aug. 1997.

[Lindstrom98] Peter Lindstrom and Greg Turk, Fast and memory efficient polygonal simplification, *In IEEE Visualization '98 Proc.*, pages 279-286, 1998.

[Raskar99] Ramesh Raskar and Michael Cohen. Image Precision Silhouette Edges. *In ACM Symposium on Interactive 3D Graphics ' 99*, April. 1999.

[Rossignac92] Jareck Rossignac and Maarten van Emmerik. Hidden Contours On A Framebuffer. *In Workshop on Computer Graphics Hardware, Eurographics ' 92 Proceedings*, pages 31-38, Sept. 1992.

[Salisbury94] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel and David H. Salesin. Interactive Pen-and-Ink Illustration. *In Computer Graphics (SIGGRAPH ' 94 Proceedings)*, pages 101-108, July. 1994.