The 2015 World Congress on
Advances in Structural Engineering and Mechanics (ASEM15)
Incheon, Korea, August 25-29, 2015

Self-intersection Avoidance for 3-D Triangular Mesh Model

Habtamu Minassie Aycheh[1)] and Min Ho Kyung[2)]

[1)] *Department of Computer Engineering, Ajou University, Korea,*
[2)] *Department of Digital Media, Ajou University, Korea,*
[1)] *hab2012@ajou.ac.kr*
[2)] *kyung@ajou.ac.kr*

ABSTRACT

Self-intersection is a fundamental problem in computational geometry. Model repair is a necessity in many applications like CAD because it destroys the integrity of a mesh surface that may leads to crashes. Geometric intersection occur due to operations on mesh models having narrow clearance. In accordance, we propose a self-intersection avoidance algorithm for 3-D triangular mesh models. The purpose of our work is to control unsafe 3-D modes that are liable to self-intersections. Our approach is based on finding an optimal perturbed values needed to update the vertex positions of a triangular mesh model without changing the basic shape of the model so that the operation on the model will not create self-intersection. We deployed numerical root finding methods with distance metrics. Evaluation of our experimentation shows promising results.

Keywords: computational geometry, self-intersection, triangular mesh

1. INTRODUCTION

Geometric modeling is the main building block of computer graphics. In relation to the growth of computer graphics applications, 3-D geometric models are commonly represented by polygons (Shirley 2009). The main essence of this pipeline is due to easy of computational description, rendering simplicity and hardware implementation of graphics algorithms. Triangular mesh is a widely accepted and simple polygonal representation of 3-D modeling (Botsch 2006). It is composed of a set of vertices, edges and triangular faces that defines the shape of an object in 3-D space.

Accordingly, polygonal models must satisfy some correctness criteria demanded by a target application (Botsch 2006, Ju 2009). A mesh model should be closed, manifold and free of self-intersection. For instance, geometric correctness is essential in engineering and manufacturing where numerical computations needed for solid objects, like finite element analysis and for real production like quick prototyping.

However, geometric errors are inevitable due to practical activities (Ju 2009). There are different ways of creating polygonal models. Currently, most 3-D objects designed by using interactive 3-D modeling software. Some reconstructed from imaging

---

[1)] Graduate Student
[2)] Professor

devices such as scattered points in 3-D scanning and from gray scale volume in bio-medical imaging. Other 3-D modes are from NURBS, subdivision surfaces and constructive solid geometry that often need to transform into polygonal forms for visualization and computation. Most computer graphics applications use large mesh sizes. Usually, 3-D polygonal mesh models are composed of huge number of connected vertices and associated polygonal faces that enable detailed representation of the model so as to increase the visual quality of the model. However, huge size 3-D models requires compression in networking environments (Abderrahim 2013, Payan 2005). The reconstruction of the compressed model may create artifacts.

Among well-identified geometric errors, a triangular mesh self-intersection is one of a fundamental problem in computational geometry (Botsch 2006). In connection to this, we propose self-intersection avoidance for triangular mesh models. Our algorithm does not require collision detection procedure. We used distance metrics and optimal non-linear root finding numerical methods. Evaluation of our system experimentation shows promising results.

The remaining part of the paper is organized as follows. Section 2 presents previous works. Section 3 describes the general research problem. Section 4 describes methodologies and proposed solution. The implementation of the proposed solution and experimental results will be presented in Section 5 and Section 6 respectively. The conclusion of the work is presented in section 7.

2. PREVIOUS WORK

In 3-D geometric models, detecting existence of self-intersection and removing them is a usual operation in different application domains such as solid modeling and computer aided design (CAD). Geometric self-intersection destroys the integrity of a mesh surface that may leads to crashes in applications. Based on this, many research works have been conducted on mesh self-intersection (Campen 2010, Ju 2009, Laidlaw 1986). However, the majority of works focused on detecting and removing self-intersections (Jung 2004, Yamakawa 2009). Moreover, collision detection and self-intersection computation are computationally very complex. In this line, progressive improvements has been shown in (Gottschalk 1998, Möller 1997, Park 2004, Sabharwal 2013) but still there are challenges in relation to the need of efficient and robust self-intersection control systems in different applications.

The proposed system attempts to avoid self-intersection without going through the collision detection process. That is, the input is mesh model which is free from self-intersection. Then, operations on the model that causes possible self-intersections are controlled by keeping the model safe from mesh elements colliding each other.

3. PROBLEM FORMULATION

There should exist a specified clearance between layered 3-D mesh models in order to safeguard geometric objects from possible self-intersections. Thus, we have to keep apart a surface of a geometric model so that they will not create self-intersection after some operations of a mesh model. We can describe the problem analytically using the elements of 3-D triangular mesh primitives such as vertices, edges and triangular faces as follows.

Assume that we have the following parameters from a given 3-D mesh model.

N = number of triangles, N ≤ 2M;
M = number of triangle pairs;
K = number of vertices, K ≤3N and
X = A variable vector formed from a set of vertex positions; i.e.

$$X = \left(V_{1x}, V_{1y}, V_{1z}, V_{2x}, V_{2y}, V_{2z}, \cdots, V_{Kx}, V_{Ky}, V_{Kz}\right);$$

$\delta_X =$ Minimum update value of vertex positions; i.e.

$$\delta_X = \left(\delta_{1X}, \delta_{1Y}, \delta_{1Z},\ \delta_{2X}, \delta_{2Y}, \delta_{2Z}, \cdots, \delta_{KX}, \delta_{KY}, \delta_{KZ}\right)$$

Where, (x, y, z) is a vertex position of the model in a 3-D space.

Accordingly, we can understand that X has a dimension of 3K. Moreover, we can form at most M distance equations, say f_i as shown in eq. (1) where d_i is the distance between a pair of 3-D triangles.

$$\begin{aligned} f_1(\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_K) &= d_1 \\ f_2(\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_K) &= d_2 \\ &\vdots \\ f_M(\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_K) &= d_M \end{aligned} \tag{1}$$

Based on eq. (1), let we specified a threshold value, τ , which is a minimum distance between the two layers of mesh model that keeps the model safe or free of possible self-intersection. Then, when $d_i \leq \tau$, we have to deduce an algorithm that gives an optimal updating values of X say vector δ_X ; Therefore, our major goal is to find a minimum δ_x such that $f_i\left(X + \delta_X\right) \geq \tau$ for all $i = 1, 2, \ldots, M$.

4. PROPOSED SOLUTION

The analytical description of our research problem shown in section 3 encompasses two core parts. The primary part is 3-D distance computation between triangles pair. The second is updating vertex positions of a model by small δ_X in accordance with a predefined threshold value, τ. In relation to these, our proposed algorithm is described as follows.

Input: 3-D triangular mesh free of self-intersection;

1. Define a minimum threshold value between triangles pair: τ ;
2. Compute the distances between 3-D triangles pair: $f_i\left(x_1, x_2, \ldots, x_K\right) = \mathrm{f}_i(X)$;
3. Identify triangle pairs having distance value less than the threshold value: $f_i\left(X\right) < \tau$;

4. Update vertex positions, X, for each $f_i(X) < \tau$ by small vector value δ_X ;
5. **Repeat** steps 2-4 until all $f_i(X) \geq \tau$.

The detail description of our proposed algorithm will be presented in the following subsections.

4.1 Shortest Distance between Two 3-D Triangles

In order to identify triangles pairs having a distance below the specified threshold value, a 3-D triangle-to-triangle distance computation is required. So, our preliminary step is computing distances between triangles.

As described in (Eberly 2006), the shortest distance between two 3-D triangles falls in at most four cases. They are:

1) The minimum distance between vertices of the two triangles;
2) The minimum distance between a vertex of a triangle to the edge of other triangle;
3) The minimum distance between edges of the triangles and
4) The minimum distance from a vertex of a triangle to the face of other triangle.

However, case 1 and 2 can be obtained in either of case 3 or case 4. In case 3, by treating each edge and vertex pairs as a line segment and enumerating the distance between the two lines segments; in case 4, by a projection of a vertex to a plane, if it lies on a perimeter i.e. on edge or a vertex, case 1 or case 2 can be obtained.

4.1.1 Edge to Edge Distance

The edge-to-edge distance of triangle pair can be computed in relation to the concept of shortest distance between two line segments. As illustrated in Fig 2, assume that we have two line segments L_1 and L_2 as given in eq. (2).

$$\begin{aligned} L_1 &: p_0 + s\,\vec{u} \\ L_2 &: q_0 + t\vec{v} \end{aligned} \tag{2}$$

Where, $0 \leq s \leq 1$, $0 \leq t \leq 1$, p_0 is a point on L_1 and $\vec{u}$ is normal direction of L_1 ; q_0 is a point on L_2 and $\vec{v}$ is normal direction of L_2 ;

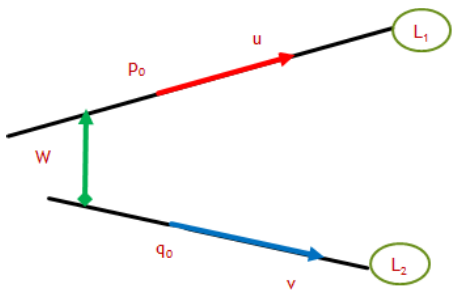


Fig. 1 An orthogonal Line to Two Lines

The shortest distance between L_1 and L_2 is W which is an orthogonal line both to L_1 and L_2. Thus, the shortest distance is the magnitude of vector W as given in eq. (3).

$$W : ?w_0 + s\vec{u} \; ?\; t\vec{v} \qquad (3)$$

Where, $w_0 = p_0 - q_0$; s and t can be found as given in eq. (4):

$$\begin{aligned} s &= (be - cd)/(ac - b^2) \\ t &= (ae - bd)/(ac - b^2) \end{aligned} \qquad (4)$$

Where, $a = \vec{u}.\vec{u},\; b = \vec{u}.?\; c = \vec{v}?\vec{v},\; d = \vec{u}.?w_0 \;\; and \;\; e = \vec{v}?\vec{w}_0$

In connection to this, the signed distance between two line segments L_1 *and* L_2 can be computed by using vector projection approach. As illustrated in Fig. 2, the procedure of this scenario can be summarized as follows.

1. Take a vector parallel to L_1 and point p_0 on L_1;
2. Take a vector parallel to L_2 and point q_0 on L_2 ;
3. Create a vector $\vec{n} = \vec{u} \times \vec{v}$, so that $\vec{n}$ is perpendicular to both lines
4. Create two parallel planes: one plane from p_0 and $\vec{n}$; the other plane from q_0 and $\vec{n}$.

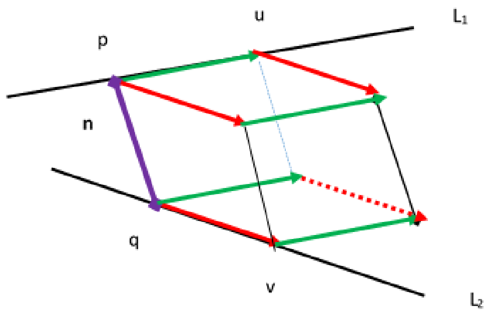


Fig. 2 Parallel Planes from Two Lines

Thus, the shortest distance between the two line segments becomes the same as the distance between the two parallel planes. Therefore, it can be computed as a point to plane distance as given in eq. (5).

$$d = \frac{w_0.\vec{n}}{\|\vec{n}\|} \qquad (5)$$

Note that the condition in eq. (2) must hold so that the projected point should lie on the line segments. In general, in edge-to-edge shortest distance computation, there are nine possible edge-to-edge combinations.

4.1.2 Vertex to triangle Distance

As illustrated in Fig. 3, given $\vec{n}$, a normal direction of the triangle $\Delta V_0 V_1 V_2$ such that $\vec{n} = \vec{u} \times \vec{v}$, the signed distance between a triangle and a point, P , which is the vertex of the other triangle, can be computed by using the scalar projection of (P-V_0) on to $\vec{n}$ as given in eq. (6).

$$d = \frac{\vec{w}.\vec{n}}{\|\vec{n}\|} \tag{6}$$

Where, $w = P - V_0$

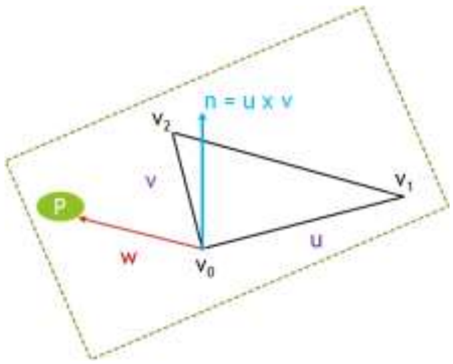


Fig. 3 Distance from a Point to a Triangle

We have to note that the projected vector should lie inside the triangle. Barycentric coordinate system can be used to verify whether a point lies inside a triangle or not (Skala 2013). A barycentric coordinate point, $P' = (x, y, z)$, in a triangle is described in eq. (7).

$$\mathbf{P}' = (\mathbf{1} - \mathbf{s} - \mathbf{t})\ \mathbf{V_0} + \mathbf{s}\mathbf{V_1} + \mathbf{t}\mathbf{V_2} \tag{7}$$

Where, $s \geq 0,\ t \geq 0$ *and* $(s+t) \leq 1$. s and t can obtained from barycentric coordinate computation as shown in eq. (8).

$$\begin{aligned} s &= (u.v)(w.v) - (v.v)(w.u) / (u.v)^2 - (u.u)(v.v) \\ t &= (u.v)(w.u) - (u.u)(w.v) / (u.v)^2 - (u.u)(v.v) \end{aligned} \tag{8}$$

Where, $u = V_1 - V_0$, $v = V_2 - V_0$, and $w = P_c$. Note that the point P' lies on an edge if s=0, t=0 or (s+t) =1. P' lies on vertex if V_0(0,0),V_1(0,1) or V_2(1,0) where the numbers in the brace indicates the value of (s , t). In vertex to triangle distance computation, there are six possible vertex to triangle combinations.

4.2 Multivariable Non-linear Root Finding

The second major component of our proposed algorithm is updating vertex positions by small vector values so that the predefined minimum clearance distance, τ , is maintained. In relation to this, we computed small updating vertex values by using multivariable non-linear root finding method. Mostly, solving system of nonlinear equations is complex. It needs insight investigation to understand the constraints of the target problem. There is no standard way of solving nonlinear systems. The good way of solving nonlinear equations is by using numerical methods based on iterative methods. Iterative computation requires initial value approximation and convergence which makes challenging a usability of existing numerical algorithms for a high dimensional root finding problems. The most practical and robust multivariate nonlinear root finding method is Newton-Raphson method (Burden 2010, Ortega 1970, Press 2007). It is fast and powerful method with quadratic convergence. However, convergence is affected by initial value estimation that needs to be handled.

$$F(X) = 0 \qquad (9)$$

For a vector function given in eq. (9), assume F is an m x n dimensional vector valued function. In other words, we will have m non-linear equations with n unknowns. Thus, a numerical estimate of Newton Raphson iteration is given in eq. (10)

$$\mathrm{X}_{new} = \mathrm{X}_{old} + \mathrm{J}^{-1}\mathrm{F}(\mathrm{X}_{old}) \qquad (10)$$

Where, $F(\mathrm{X}) = (\mathrm{f}_1(x_1, x_2, ..., x_n), ..., f_m(x_1, x_2, ..., x_n))^T$. The Newton Raphson method procedure requires the computation of the inverse of the Jacobian matrix. For a bigger dimension of a Jacobian matrix, the inverse computation is not feasible due to the associated computational complexity or the nature of matrices in practical situations such as rectangular or singular matrices. Hence, we need to use other mechanisms like a linear algebraic equation solving method instead of a direct inverse computation. However, the selection of a proper linear solver depends on the nature of the problem. In our case, mostly we have a under constrained equation in which the number of equations are less than number of unknowns. Our Jacobian matrix is a rectangular sparse matrix. Hence, we used a Singular Value Decomposition (SVD) method as described in (Press 2007) though it has a performance trade-off. A linear algebraic estimate version of Newton method is given in eq. (11).

$$-F(X_i) = J(X_i)\Delta \mathrm{x} \qquad (11)$$

Where, $\square X = X_{i+1} \; ? \; X_i$ and J is a Jacobian matrix. After estimating the value of Δx by linear solver, we can complete the iteration of Newton step as in eq. (12).

$$X_{new} = X_{old} + \Delta x \qquad (12)$$

As depicted in (Burden 2010), the multivariable Newton Raphson root finding method goes through the following steps.

1. Initial vector value estimation, X0, and setting convergence criteria, tolerance
2. Jacobian matrix and residual function values computation, i.e. $J\left(X_i\right)$ and $F\,(X_i)$.

3. Applying linear solver on:
4. Compute the current iteration value,
5. Based on convergence criteria, if it converges STOP
6. If not converged, set $X_{new} = X_{old}$ and loop from step 2.

4.3 Convergence of Newton Method

The convergence of Newton Raphson method is affected by the initial value approximation because of its local quadratic convergence constraint. In some practical situations, the initial value approximation is challenging. Hence, in relation to a problem domain insight analysis is needed to devise a globally converging Newton Rapshon method. As described in (Press 2007), one of the best approaches is based on line searches and backtracking. The key point is a decision when to accept or reject the value of Δx in a given Newton step shown in eq. (13).

$$X_{new} = X_{old} + \lambda \Delta x;\ 0 \le \lambda < 1 \tag{13}$$

In the newton step of eq. (13), $f_i(X_{new}) \ge f_i(X_{old})$ should be satisfied for the acceptance of the step so that each distance function values are progressively increasing towards the defined threshold value.

5. IMPLEMENTATION

The implementation of our proposed algorithm has two main parts. The first is a 3-D triangle-to-triangle distance computation and the second one is finding roots of multivariable non-linear system of equations. The non-linear equations are formed from distance functions. As described in section 3, if the distance d_i is less than the predefined threshold value, τ , the vertex positions should be updated by small vertex values δ_X so that $f_i\left(X\right) > \tau$. That is if $f_i\left(X\right) < \tau$, then

$$\mathrm{f}_i(\mathrm{X} + \delta_X) = \tau \tag{14}$$

Based on eq. (14), in-order to solve for the unknown vector δ_X , we setup system of non-linear equations formed from distance functions. Then, we apply Newton Raphson method to find the roots of these non-linear functions.

Let our mesh model has K vertices and the number of triangle pairs having distance less than τ is m. Thus we can have m distance functions with n unknowns $\left(n = 3K\right)$. The system of non-linear equations has the form indicated in eq. (1). The Jacobian matrix is computed as the first order partial derivatives of vector valued distance functions as indicated in eq. (15).

$$J(X)=\begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \cdots & \frac{\partial f_n}{\partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \cdots & \frac{\partial f_2}{\partial x_n}(x) \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f_m}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \cdots & \frac{\partial f_m}{\partial x_n}(x) \end{bmatrix} \tag{15}$$

The actual analytical description of vertex positions update briefly described in the following subsections.

5.1 Edge to Edge distance

Recall that when $f_i(X) < \tau$, we need to update the vertex positions by small vector value δ_X so that $f_i(X + \delta_X) \geq \tau$. In line with this, the updated edge-to-edge distance between two 3-D triangles is given in eq. (16).

$$d_{new} = \frac{(\vec{w}_0)_{new}.\vec{n}_{new}}{\|\vec{n}\|} \tag{16}$$

Where,

$$\begin{aligned} (\mathrm{w}_0)_{\mathrm{new}} &= (\mathrm{u}_0 - \mathrm{v}_0) + (\delta_{\mathrm{u}0} - \delta_{\mathrm{v}0}) \\ (\mathrm{n}_1)_{\mathrm{new}} &= (\mathrm{u}_1 - \mathrm{u}_0) + (\delta_{\mathrm{u}1} - \delta_{\mathrm{u}0}) \\ (\mathrm{n}_2)_{\mathrm{new}} &= (\mathrm{v}_1 - \mathrm{v}_0) + (\delta_{\mathrm{v}1} - \delta_{\mathrm{v}0}) \\ \mathrm{n}_{\mathrm{new}} &= (\mathrm{n}_1)_{\mathrm{new}} \times (\mathrm{n}_2)_{\mathrm{new}} \end{aligned}$$

In eq. (16), there are four unknown vectors, $(\delta_{u0}, \delta_{u1}, \delta_{v0}, \delta_{v1})$, that updates the vector positions based on the edge-to -edge distance condition as defined in eq. (17).

$$f(\delta_{u0}, \delta_{v0}, \delta_{u1}, \delta_{v1}) = \frac{(\vec{w}_0)_{new}.\vec{n}_{new}}{\|\vec{n}\|} \tag{17}$$

Accordingly, the Jacobian matrix of eq. (17) is shown in eq. 18 – eq. 21 with respect to the four unknown vertex positions updating variables.

$$\frac{\partial f(.)}{\partial \delta_{u0}} = \frac{1}{\|n\|}(\mathrm{n} + (\mathrm{w}_0 \times n_2)) - \frac{w_0.n}{\|n\|^3}(n \times n_2) \tag{18}$$

$$\frac{\partial f(.)}{\partial \delta_{u1}} = \frac{1}{\|n\|}(n_2 \times w_0) - \frac{w_0.n}{\|n\|^3}(\mathrm{n}_2 \times n) \tag{19}$$

$$\frac{\partial f(.)}{\partial \delta_{v0}} = \frac{1}{\|n\|}(-\mathrm{n} + (\mathrm{n}_1 \times w_0)) - \frac{w_0.n}{\|n\|^3}(\mathrm{n}_1 \times n) \tag{20}$$

$$\frac{\partial f(.)}{\partial \delta_{v1}} = \frac{1}{\|n\|}(\mathrm{w}_0 \times n_1) - \frac{w_0.n}{\|n\|^3}(\mathrm{n} \times n_1) \qquad (21)$$

5.2 Vertex to triangle Distance

Similar to the edge-to-edge distance condition, when $f_i(X) < \tau$, we need to update the vertex positions by small vector value δ_X so that $f_i(X + \delta_X) \geq \tau$. Thus, the updated vertex -to - triangle distance is given in eq. (22).

$$d_{new} = \frac{(\vec{w})_{new}.\vec{n}_{new}}{\vec{n}_{new}} \qquad (22)$$

Where,

$$(\mathrm{w})_{\mathrm{new}} = (\mathrm{u}_0 - \mathrm{v}_0) + (\delta_{\mathrm{u}0} - \delta_{\mathrm{v}0})$$
$$(\mathrm{n}_1)_{\mathrm{new}} = (\mathrm{v}_1 - \mathrm{v}_0) + (\delta_{\mathrm{v}1} - \delta_{\mathrm{v}0})$$
$$(\mathrm{n}_2)_{\mathrm{new}} = (\mathrm{v}_2 - \mathrm{v}_0) + (\delta_{\mathrm{v}2} - \delta_{\mathrm{v}0})$$
$$\mathrm{n}_{\mathrm{new}} = (\mathrm{n}_1)_{\mathrm{new}} \times (\mathrm{n}_2)_{\mathrm{new}}$$

According to eq. (22), there are also four unknown vectors, $(\delta_{u0}, \delta_{v0}, \delta_{v1}, \delta_{v2})$, to update the vector positions. The Jacobian matrix of the vertex to triangle distance function is derived as shown in eq. 23 - eq.26.

$$\frac{\partial f(.)}{\partial \delta_{u0}} = \frac{n}{\|n\|} \qquad (23)$$

$$\frac{\partial f(.)}{\partial \delta_{v0}} = \frac{1}{\|\vec{n}\|}(-\mathrm{n} + (\mathrm{w} \times n_2) + (n_1 \times w_0)) - \frac{w.n}{\|n\|^3}((n \times n_2) + (\mathrm{n}_1 \times n)) \qquad (24)$$

$$\frac{\partial f(.)}{\partial \delta_{v1}} = \frac{1}{\|\vec{n}\|}(\mathrm{n}_2 \times w) - \frac{w.n}{\|n\|^3}(\mathrm{n}_2 \times n) \qquad (25)$$

$$\frac{\partial f(.)}{\partial \delta_{v2}} = \frac{1}{\|\vec{n}\|}(\mathrm{w} \times n_1) - \frac{w.n}{\|n\|^3}(\mathrm{n} \times n_1) \qquad (26)$$

6. EXPERIMENTS AND RESULTS

We experimented our system on Intel Corei7CPU 3.07GHz and 12GB RAM platform. We developed a C++ program for the implementation of our algorithm. Half-edge data structure and Wave front .obj file format are used to represent mesh model. We incorporated optimal best practices of Newton method and SVD presented in (Press 2007). Heuristically, we defined the clearance threshold value between mesh objects as $\tau = 0.001$. Models are created using Blender software. Accordingly, we evaluated our algorithm with respect to 3-D triangular mesh compression. We used Open 3D Graphics Compression (O3DGC) tool to encode/decode our experimental models. The three scenarios of our experimental results are presented as follows.

The box model experiment is shown in Fig. 4. It has 636 vertices and 1220 triangular faces. The input model to our system is indicated in Fig 4(a). As shown in Fig.

4(b), when the original model compressed and decoded, artifacts occurred due to a self-intersection. However, after the implementation of our algorithm, the artifacts are clearly removed as indicated in Fig. 4(c). We used different colors to differentiate the interior and exterior faces so that we can see easily the artifacts if any.

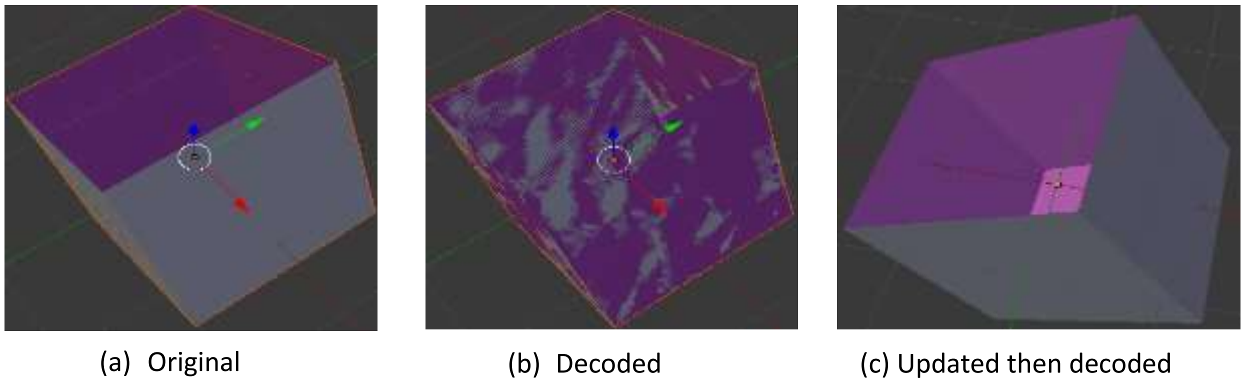
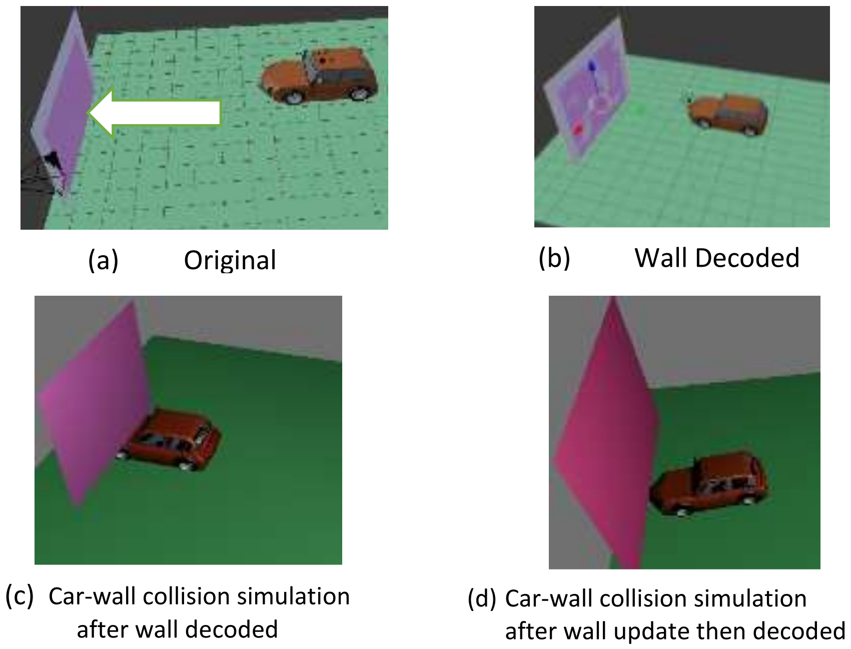

(a) Original (b) Decoded (c) Updated then decoded

Fig. 4 Box Model

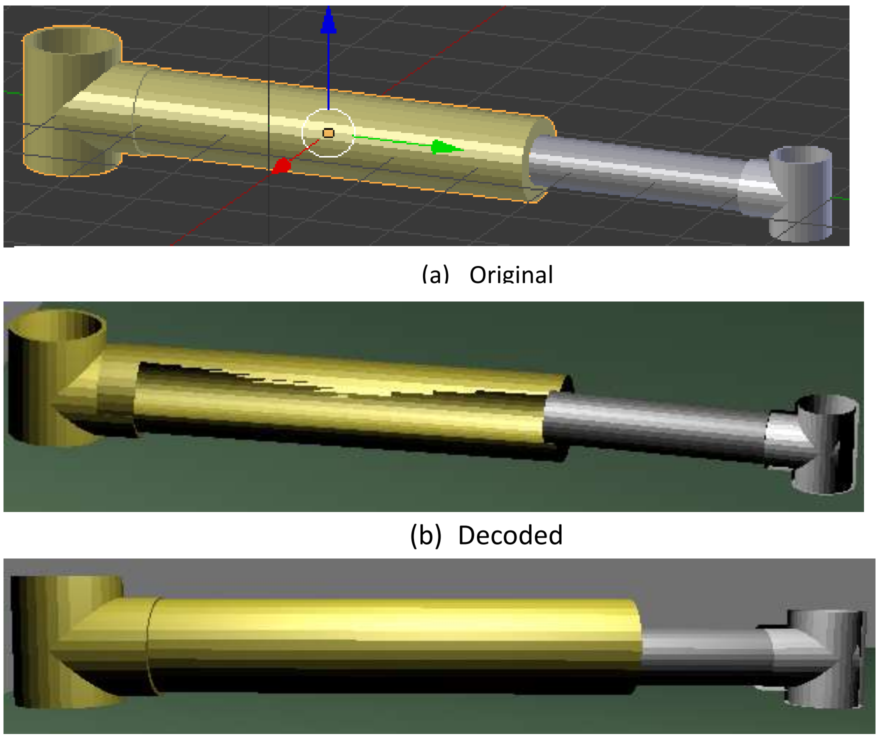
The car-wall collision experiment is indicated in Fig 5. The wall model has 288 vertices and 484 triangles. The car model has 37,074 vertices and 72,368 triangles. We implemented the 3-D compression on the wall model so that it will generate artifacts due to self-intersection. Then, we simulated a collision between a car and the wall using Bullet Physics to investigate the underling results. As shown in Fig 5(b), after compression and decode of the wall, artifacts were generated. This created a wrong collision result between the car and wall as shown in Fig. 5(c) due to the self-intersection error, which affected the normal direction of the underling faces of the wall. The implementation of our algorithm showed the correct simulation result of the car to wall collision as shown in Fig. 5(d).


(a) Original (b) Wall Decoded

(c) Car-wall collision simulation after wall decoded (d) Car-wall collision simulation after wall update then decoded

Fig. 5 Simulation of Car to wall Collision

The piston simulation experiment is presented in Fig. 6. The piston model has 960 vertices and 1856 triangular faces. As indicated in Fig 6(b), after the original piston was compressed and reconstructed, the cylinder of the piston created some artifacts. In addition, the piston was not well simulated, as the piston did not get into the cylinder because of a small size change created a collision between the piston and cylinder. After the update of the model by our algorithm, the piston is properly simulated as shown in Fig. 6(c).


(a) Original

(b) Decoded

(c) Decoded then Updated

Fig. 6 Piston Simulation

7. CONCLUSION

We presented an algorithm that avoids possible self-intersections of 3-D mesh models due to operations like compression. Our developed system is based on numerical methods particularly non-linear root finding system. We have found expected results from our experiments. The performance of the system depends more on the number of non-linear equations than the mesh size. In other words, the more unsafe modeling conditions or when we have more number of $(f_i(X) < \tau)$, the performance degrades. We adopted efficient data structure and optimized numerical methods for our experimentation. We have tested the system on 3-D triangular mesh model. For others geometric models, further investigation and analysis may be required.

ACKNOWLEDGMENT

This work is supported by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Science, ICT & Future Planning (NRF-2013R1A1A2011733).

REFERENCES

Abderrahim, Z., Techini, E., and Bouhlel, M. S. (2013), "Progressive compression of 3D objects with an adaptive quantization", Computational Geometry, Vol.10 (2-1), 504-511.

Botsch, M., Pauly, M., Rossl, C., Bischo, S., and Kobbelt, L. (2006), "Geometric modeling based on triangle meshes", Proceedings of ACM SIGGRAPH 2006, New York.

Burden, R. L. and Fairs, J. D. (2011), Numerical Analysis, 9ed, Brooks/Cole, MA.

Campen, M. and Kobbelt, L. (2010), "Exact and robust (self-) Intersections for polygonal meshes", Computer Graphics Forum, Vol. 29(2), 397-406.

Eberly, D.H. (2006), 3D Game Engine Design, 2ed, CRC Press, NW.

Gottschalk, S. and Lin, M. C. (1998), "Collision detection between geometric models: A survey", Proceeding of IMA Conference on the Mathematics of Surfaces.

Ju, T. (2009), "Fixing geometric errors on polygonal models: A survey", Journal of Computer Science and Technology, Vol 24(1), 19-29.

Jung, W., Shin, H., Choi, and Byoung, K. (2004), "Self-intersection removal in triangular mesh offsetting", Computer-Aided Design and Applications, Vol. 1(1-4),477-484.

Laidlaw,D. H., Trumbore, W. B., and Hughes, J. F. (1986), "Constructive solid geometry for polyhedral objects", Proceedings of SIGGRAPH 86, New York.

Möller, T. (1997), "A fast triangle-triangle intersection test", Journal of Graphics Tools, vol. 2, 25-30.

Ortega, J.M. and Rheinbolt, W.C. (1970), Iterative Solutions of Nonlinear Equations for Several Variables, Academic Press, FL.

Park, S. C. (2004), "Triangular mesh intersection", The Visual Computer, Vol. 20(7), 448-456.

Payan, F. and Antonini, M. (2005), "An efficient bit allocation for compressing normal mesh with an error-driven quantization", Computer Aided Geometric Design, Vol. 22, 466-486.

Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P (2007), Numerical Recipes: The Art of Scientific Computing, 3ed, Cambridge University Press.

Sabharwal, C.L., Leopold, J.L. and McGeehan, D. (2013), "Triangle-triangle intersection determination and classification to support qualitative spatial reasoning", Polibits, Vol 48, 13-22.

Shirley, P., and Marschner, S. (2009), Fundamentals of Computer Graphics, 3ed, CRC, A.K Peters, MA.

Skala V. (2013), “Robust barycentric coordinates computation of the closest point to a hyperplane in E^n”, Proceeding of International Conference on Applied Mathematics and Computational Methods in Engineering.

Yamakawa, S. and Shimada, K. (2009), “Removing self-intersections of a triangular mesh by edge swapping, edge hammering, and face lifting”, Proceedings of the 18th International Meshing Roundtable, Salt Lake City, UT.