Computing and Informatics, Vol. 32, 2013, 527–547

EFFICIENT DECIMATION OF POLYGONAL MODELS USING NORMAL FIELD DEVIATION

Muhammad Hussain

Department of Software Engineering
College of Computer and Information Sciences, King Saud University
Riyadh 11543, Saudi Arabia
e-mail: mhussain@ksu.edu.sa

Communicated by Horst Bischof

Abstract. A simple and robust greedy algorithm has been proposed for efficient and quality decimation of polygonal models. The performance of a simplification algorithm depends on how the local geometric deviation caused by a local decimation operation is measured. As normal field of a surface plays key role in its visual appearance, exploiting the local normal field deviation in a novel way, a new measure of geometric fidelity has been introduced. This measure has the potential to identify and preserve the salient features of a surface model automatically. The resulting algorithm is simple to implement, produces approximations of better quality and is efficient in running time. Subjective and objective comparisons validate the assertion. It is suitable for applications where the focus is better speed-quality trade-off, and simplification is used as a processing step in other algorithms.

Keywords: Polygonal models, polygonal simplification, LOD modeling

Mathematics Subject Classification 2010: 68U01, 68U05, 68U07

1 INTRODUCTION

Polygonal meshes, in particular triangular meshes, have now become a de facto standard for encoding 3D spatial information because of their mathematical simplicity. But because of growing complexity of polygonal meshes, it is hard to process, render and transmit them using mid-level systems. Though there is a substantial enhancement in the graphics acceleration techniques, complexity of polygonal meshes is

increasing at a faster rate. The solution of this problem is simplification. In the literature, there exist a large number of simplification algorithms, which focus (depending on the application scenario) on

- constructing approximations of the best quality, time of simplification is of minor concern [17, 12, 5, 25, 40],
- generating approximations with the best speed-quality tradeoff [36, 32, 22, 11, 14, 15, 18, 19, 30], or
- building approximations instantly, quality being of little concern [35, 27, 9, 3].

The proposed algorithm SESIMP (Simple and Efficient SIMPlification) falls in the second category where both the quality of approximations and the simplification time are crucial. The measure of local geometric distortion caused by a decimation operation drives an iterative simplification algorithm and is the major differentiating factor among such algorithms. Most of the methods, which are known for their good speed-quality trade-off, employ distance, volume and area based measures of local geometric distortion. The decimation operation distorts not only the geometry but also the normal field of a polygonal surface. It is a fact that the visual appearance of a 3D object depends on its normal field. Minimization of the normal field deviation during the simplification process not only reduces the visual artifacts but also ensures minimization of the geometric error [10]. As such, it is important for better visual fidelity to focus on the local normal field deviation and to minimize it during the process of decimation. Though the idea of using normal field deviation has been around for a while, the way it has been used in previous work does not result in good time-accuracy trade-off. In this paper, the local normal field deviation is exploited in a novel way that results in better quality approximations.

The evaluation and comparison reveals that SESIMP has improvement in execution time and the visual quality of approximations over similar state-of-the-art simplification methods. It preserves well the important surface features even at very low levels of detail, consumes less memory and automatically prevents mesh fold-overs. The implementation of SESIMP is quite simple and it is robust in the sense that models at various levels of complexity can be decimated with the same level of fidelity and efficiency.

The rest of the paper is organized as follows. Section 2 is devoted to the related work. In Section 3, the new measure of geometric distortion is elaborated. Section 4 gives the description of SESIMP. Detailed discussion on the performance of SESIMP, and its comparison with similar sate-of-the-art algorithms are provided in Section 5. Section 6 concludes the paper.

2 RELATED WORK

In the graphics literature, there exist many good simplification algorithms, which focus on different application domains. In the following discussion, our focus will be

only on the algorithms dealing with applications where visual quality of approximations as well as speed are equally important. For a thorough survey, an interested reader is referred to [28, 8, 15, 26]. In the following paragraphs, the review of some recent related simplification techniques is provided; this review is not exhaustive at all.

Though quadric error metric (QEM) [14] was proposed in 1997, it is still a benchmark for new proposals. QEM computes the importance of a vertex as a weighted sum of its squared distances from the planes of its incident faces and stores it as a 4×4 symmetric matrix. Though QSlim [14], the greedy algorithm based on QEM, provides the best speed-quality tradeoff so far, it can not automatically preserve surface discontinuities and visually important features, in particular, at very low levels of detail. Also, it is not memory efficient; for each vertex it adds a memory overhead of at least 44 bytes. A memoryless version of QEM was introduced in [25]. Though this version of QEM improves the quality of approximations, its running time complexity is at least three times more than that of QSlim. The simplification algorithms by Kim et al. [20], Yoshizawa et al. [41], Lee et al. [24] and Yan et al. [40] employ QEM with additional heuristics to overcome its drawbacks. Although these algorithms improve the quality of approximations, there is drastic increase in execution time. SESIMP does this job with reduced memory overhead and computational cost.

The algorithms proposed in [1, 30] use volume and area for measuring the geometric distortion. The error measures proposed by Tang et al. [38] generalize the error metrics introduced in [1, 30]. These techniques may be interesting from theoretical viewpoint, but practically their impact is minor on the simplification problem because despite significant increase in time and space complexity, there is no significant improvement in the quality of approximations.

Normal field has also been considered for simplification; it has been employed for two different purposes: to define an error measure for driving the greedy procedure [19, 18, 31, 37] and to perform clustering/sampling [3, 4, 10]. The greedy approach proposed in [19] uses normal field deviation for defining the measure of geometric fidelity; it overestimates geometric error because the normal field deviation is computed by comparing the current normal vectors of the local neighborhood of a vertex with their counterparts on the original mesh. Similarly, the greedy algorithm presented in [18] overestimates the distortion error because the error is accumulated every time an edge collapse takes place. Ramsey et al. [31] used normal field variation along with a threshold value for selecting edges for collapse; it is not a sophisticated way of exploiting the normal field deviation. The approximation generated by this algorithm is not of good quality because of significant volume loss. Southern et al. [37] define an error metric as a multiple of maximum normal deviation and local volume loss over the local neighborhood of the edge to be collapsed. All these methods are based on half-edge collapse and normal field deviation. SESIMP exploits normal field in a more sophisticated way; a measure of geometric fidelity is defined using the normal field deviation (computed considering normal field before and after half-edge collapse) over the local neighborhood of

an edge and the normal field variation over the one-ring neighborhood with flaps of the vertex to be eliminated.

Brodsky et al. [4] used normal field variation in their simplification algorithm for clustering faces. This approach is fairly efficient in running time but the constructed approximations are of poor quality. Cohen-Steiner et al. [10] introduced an error metric that is based on normal variation, and employed it in their global non-linear optimization technique for finding the best n polygon subsampling of the input detailed mesh. TopStoc [3] is based on stochastic sampling and topological clustering; it uses normal field variation for defining the probability of survival of a vertex. This method is computationally very efficient but constructs approximations of poor quality; it is suitable only for applications where speed is of major concern.

3 MEASURE OF GEOMETRIC DEVIATION

SESIMP employs half-edge collapse as a decimation operator because it does not leave unset degrees of freedom like edge-collapse and vertex-removal, and is simple to implement, keeps original geometry and is suitable for more efficient progressive transmission and integrated level of detail extraction [22].

3.1 Normal Field Based Error Measure

Normal field of a surface model plays an important role in its visual appearance. Cohen-Steiner et al. [10] showed that normal field deviation is a better measure of visual fidelity than distance. The Poincaré-Wertinger-Sobolev inequality indicates that minimizing the normal field distortion ensures minimization of the geometric deviation. This evidence in support of normal field motivated us in exploiting it in a novel way to introduce a new error metric for deriving a greedy algorithm.

A typical half-edge collapse $\vec{e}_{st}(v_s, v_t) \mapsto v_t$ re-triangulates the submesh R_{v_s} consisting of faces incident on v_s, see Figure 1 (a), and causes the deviation in the normal field over R_{v_s}. Each face $f \in R_{e_{st}}$ (the set faces incident on edge e_{st}, see Figure 1 (a)) is eliminated and each surviving face $f \in R_{v_s} \backslash R_{e_{st}}$ is updated to have v_t instead of v_s. One way of computing the geometric distortion caused by this half-edge collapse is to consider the normal field deviation due to each face $f \in R_{v_s} \backslash R_{e_{st}}$, which is defined as follows:

$$\mathcal{ND}_f = \int_{\Delta_c} \|\hat{n}_c - \hat{n}_p\|^2 \, d\Delta = \Delta_c \|\hat{n}_c - \hat{n}_p\|^2 \tag{1}$$

where $\hat{n}_p$ and $\hat{n}_c$ are unit normal vectors of f before and after the collapse, respectively, and Δ_c the current area of f. Since $\|\hat{n}_c - \hat{n}_p\|^2 = (\hat{n}_c - \hat{n}_p) \cdot (\hat{n}_c - \hat{n}_p) = 2(1 - \hat{n}_c \cdot \hat{n}_p)$, Equation (1) can be written as

$$\mathcal{ND}_f = 2\Delta_c(1 - \hat{n}_c \cdot \hat{n}_p). \tag{2}$$

As $\vec{n}_c = \|\vec{n}_c\|\,\hat{n}_c = 2\Delta_c\hat{n}_c$, where $\vec{n}_c$ is the current normal vector of f, Equation (2) becomes

$$\mathcal{ND}_f = (2\Delta_c - \vec{n}_c \cdot \hat{n}_p). \tag{3}$$

This expression is computationally more efficient and results in significant improvement in execution time. The sum of the deviations due to each $f \in R_{v_s}$ leads to the following measure of the local normal deviation introduced by the half edge collapse $\vec{e}_{st}(v_s, v_t) \mapsto v_t$:

$$\mathcal{C}(v_s, v_t) = \sum_{f \in R_{v_s} \setminus R_{e_{st}}} \mathcal{ND}_f. \tag{4}$$

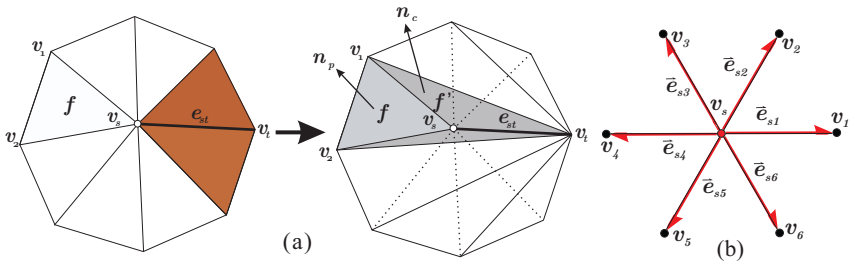

Fig. 1. (a) Half-edge collapse: $\vec{e}_{st}(v_s, v_t) \mapsto v_t$ with domain R_{v_s}, the set of faces incident on v_s ; it removes the two faces incident on edge e_{st} – the set $R_{e_{st}}$. (b) Half-edges having v_s as their origin.

This error metric can be used to prioritize the half-edges for collapse, but ordering half-edges results in excessive increase in memory overhead because it requires a heap of size $\approx 6n$, where n is the number of vertices in the input mesh. Instead, it is used to prioritize vertices. Using Equation (4), out of the half-edges whose origin is v_s (see Figure 1 (b)), the one with minimum cost is found. Let $\vec{e}_{st}(v_s, v_{t_m})$ and call it *optimal half-edge*. The cost of $\vec{e}_{st}(v_s, v_{t_m})$ is used as a priority value for v_s, which is removed by collapsing $\vec{e}_{st}(v_s, v_{t_m})$ to the *target vertex* v_{t_m}. In this way, instead of half-edges, vertices are ordered for collapse that requires a heap of size n, where n is the number of vertices.

Note that the metric defined by the Equation (4) does not measure the exact normal deviation over the region affected by a half-edge collapse. For efficiency reasons the normal field deviation defined by this metric is an approximation computed over the surviving triangles. In some situations it may not help to select the right edge to be collapsed. One such situation in 2D setting (edges e_1 and e_{01} represent triangles before and after decimation) is shown in Figure 3 (a)–(b); in this case the two vertices u, and v will have the same cost according to Equation (4) because in both cases the edge length $\|e_{01}\|$ and normal deviations are same and both are equally likely to be selected for decimation, but u is a low curvature vertex and must be decimated before v. To overcome this weakness of the metric, we scale it with a quantity that is reflective of the local geometry of a vertex and discriminates it

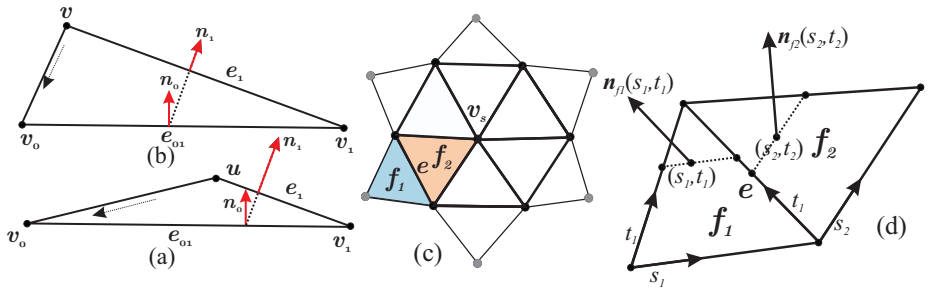

Fig. 2. (a)–(b) An explanation in 2D setting of the situation when two vertices u, and v with different local geometry have same cost according to Equation (4). (c) 1-ring neighborhood of v_s with flaps. (d) Parametrization of two adjacent faces f_1 and f_2 in (c), which is used to compute total normal field deviation over these faces.

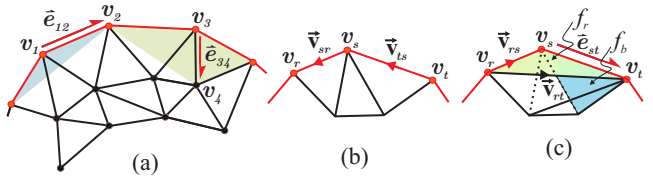

Fig. 3. (a) Half-edge $\vec{e}_{12}$ is legal whereas $\vec{e}_{34}$ is illegal for collapse. (b) The difference between $\vec{v}_{ts}$ and $\vec{v}_{sr}$ is reflective of the curvature of the boundary at vertex v_s. (c) The difference between the unit normal vectors to the faces f_r and f_b is a measure of the normal field deviation due to the displacement of the boundary faces resulted from the collapse of $\vec{e}_{st}$.

well from other vertices. One such quantity that is simple and easy to compute is the total normal field variation around a vertex; considering one-ring neighborhood

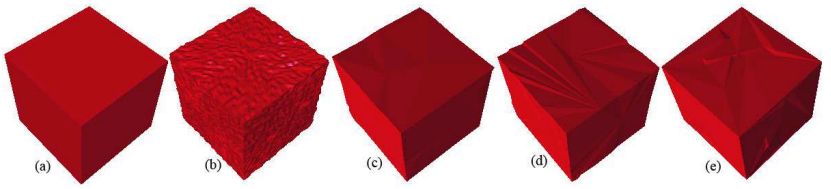

Fig. 4. (a) Cube model with #faces: 12 288. (b) Noisy cube with random noise, the level of noise is 3 % of B.B.D. (c) Simplified cube when 1-ring neighborhood with flaps is used for $\kappa(v_s)$, #faces:100. (d) Simplified cube when only 1-ring neighborhood is used for $\kappa(v_s)$, #faces:100. (e) Simplified model without $\kappa(v_s)$, #faces:100

with flaps FR_{v_s} of v_s (see Figure 2 (c)), is defined as follows

$$\kappa(v_s) = \sum_{e \in E_{v_s}} \kappa(e) \tag{5}$$

where E_{v_s} is the set of interior edges of FR_{v_s} (see Figure 2 (c)) and the total normal field variation across edge e is defined as follows ((see Figure 2 (d))):

$$\begin{aligned} \kappa(e) &= \int_{\Delta_{f_1}} \int_{\Delta_{f_2}} \|\hat{n}_{f_1}(s_1, t_1) - \hat{n}_{f_2}(s_2, t_2)\|^2 \, d\Delta_1 d\Delta_2 \\ &= \Delta_{f_1}\Delta_{f_2} \|\hat{n}_{f_1} - \hat{n}_{f_2}\|^2 \\ &= 2\Delta_{f_1}\Delta_{f_2}(1 - \hat{n}_{f_1} \cdot \hat{n}_{f_2}), \end{aligned}$$

where $d\Delta_1 = (ds_1, dt_1)$ and $d\Delta_2 = (ds_2, dt_2)$ are area elements of f_1 and f_2, respectively. For a curved surface, normal vectors corresponding to different points on the parameter space of each face are different and can be approximated by interpolation, but for efficiency reasons, we assume that the normal field over each triangle is constant.

The reason for using 1-ring neighborhood with flaps is that it results in visually smoother approximations [16], whereas only 1-ring is more sensitive to noise and a larger neighborhood adds to the time complexity. Figure 4 shows a cube model with random noise, and its simplified versions; it is obvious that when only 1-ring neighborhood is used, faces of the simplified cube are not smooth whereas 1-ring neighborhood with flaps generates smooth approximation. Note that $\kappa(v_s)$ depends on the neighborhood of a vertex irrespective of to which neighboring vertex it is collapsed unlike $\mathcal{C}(v_s, v_t)$ and measures the local geometry of a vertex quite well; it assigns negligible values to nearly flat vertices (those with nearly planar neighborhoods), and higher values to higher curvature vertices because in that case the variation in normal field is high. In this way, it causes to remove insignificant vertices and preserves salient features because important geometric features involve high curvature values. In addition, it results in visually smoother surfaces, see Figure 4. Also, note that the faces directly connected to a vertex contribute more to the local normal field variation around that vertex and these faces must be given more weight. This observation is incorporated in the computation of $\kappa(v_s)$ by counting direct faces twice; it has Gaussian-like smoothing effect.

Finally the cost of v_s is expressed as follows:

$$Cost(v_s) = \kappa(v_s) C(v_s, v_t). \tag{6}$$

Note that Cohen-Steiner et al. [10] compute a normal based error metric by comparing the normal vectors of a set of original triangles and its proxy, and employ it in their k-proxy clustering algorithm for creating mesh approximations. In contrast, SESIMP computes a normal based error metric by comparing local normal field before and after the decimation operation and considering the local normal

field variation around a vertex; this metric is used to select the next vertex to be eliminated in the iterative greedy procedure.

3.2 Preserving Boundary

A vertex on the boundary of an open mesh can be removed by collapsing either a half-edge whose only tail is on the boundary, e.g., $\vec{e}_{34}$ in Figure 3 (a) or a half-edge whose both tail and head are on the boundary, e.g., $\vec{e}_{12}$ in Figure 3 (d). Collapsing a half edge of the first kind severely distort the boundary, so collapse of this kind of edges is treated as illegal. Boundary is simplified by collapsing only edges of the second kind.

The cost of a boundary vertex is normally less than that of an interior vertex because 1-ring neighborhood of a boundary vertex is homeomorphic to a half-disk. The greedy algorithm may be trapped in the local minima along the boundary and will erode the surface indiscriminately. To tackle this problem a boundary constraint is needed that brings the boundary vertices in line with interior vertices in respect of their importance. For this purpose, we introduce the following constraint:

$$\mathcal{C}_b(v_s, v_t) = \Delta_{f_r}(\mathcal{D}_1 + \mathcal{D}_2) \tag{7}$$

where $\mathcal{D}_1 = 1 - \hat{n}_{f_r} \cdot \hat{n}_{f_b}$ with $\hat{n}_{f_r}$ and $\hat{n}_{f_b}$ being the unit normal vectors to the faces f_r and f_b (see Figure 3 (c)), and $\mathcal{D}_2 = 1 - \hat{v}_{rs} \cdot \hat{v}_{rt}$ with $\hat{v}_{rs}$ and $\hat{v}_{rt}$ being unit vectors along $\vec{v}_{rs}$ and $\vec{v}_{rt}$, respectively, see Figure 3 (c). Note that $\mathcal{D}_1$ ensures that the normal deviation along the boundary is minimum, and $\mathcal{D}_2$ guarantees that the removal of a boundary vertex does not distort the boundary; the closer the edges e_{rs} and e_{st} are to be collinear, the more likely the vertex v_s is to be removed by the collapse of either $\vec{e}_{st}$ or $\vec{e}_{sr}$.

Similar to the scaling factor $\kappa(v_s)$ for a interior vertex, the scaling constraint $\kappa_b(v_s)$ for a boundary vertex is defined as follows:

$$\kappa_b(v_s) = (L_{sr}L_{st})^2 (\lambda + \mathcal{D}_3), \tag{8}$$

where $\mathcal{D}_3 = 1 - \hat{v}_{sr} \cdot \hat{v}_{ts}$, with $\hat{v}_{sr}$ and $\hat{v}_{ts}$ being the unit vectors along $\vec{v}_{sr}$ and $\vec{v}_{ts}$ respectively, see Figure 3 (b), $L_{sr} = \|\vec{v}_{sr}\|$ and $L_{st} = \|\vec{v}_{ts}\|$. It is to be noted that $\mathcal{D}_3$ is an indicator of the local curvature at the boundary vertex and it ensures that the low curvature boundary vertices are removed first. The scalar parameter $\lambda \geq 0$ is used to control the level of boundary preservation. The greater the value of λ, the more tightly the boundary is preserved. For results presented in this paper, $\lambda = 1.0$.

Together with boundary constraints, the cost of the boundary vertex v_s is expressed as follows:

$$Cost_b(v_s) = (\kappa(v_s) + \kappa_b(v_s)) (\mathcal{C}(v_s, v_t) + \mathcal{C}_b(v_s, v_t)) \tag{9}$$

4 DECIMATION ALGORITHM

SESIMP is based on the well-known greedy design technique. The input is a polygonal mesh $\mathcal{M}$ containing n vertices and m triangular faces. Simple data structures $Face$ and $Vertex$ corresponding to every face and vertex are created and stored in the lists VL and FL. $Face$ includes pointers to the three vertices and the normal vector of the corresponding face. $Vertex$ represents a vertex v_s and contains its geometric position $p_s(x, y, z)$, its cost $Cost(v_s)$, target vertex v_{t_m}, a dynamic list of pointers to faces incident on this vertex $adj\_faces(v_i)$, and a heap backlink. Flaps of the vertex v_s are found by traversing $adj\_faces(v_i)$ corresponding to its each neighbor v_i. Vertex heap VH of size n is maintained for ordering the vertices according to their cost.The pseudo code of the algorithm is as follows.

SESIMP(Mesh $\mathcal{M}(\mathcal{V}, \mathcal{F})$)
Input: Original triangular mesh $\mathcal{M} = (\mathcal{V}, \mathcal{F})$ and
the target number of faces $numf$
Output: An LOD with the given budget of faces
For each vertex $v_s \in V$
Create $Vertex$ and put in VL
EndFor
For each face $f \in \mathcal{F}$
Create $Face$ and put in FL
Add the pointer of $Face$ to $adj\_faces(v_i)$ for each v_i of f
EndFor
For each vertex $v_s \in \mathcal{V}$
Compute $Cost(v_s)$ (see Section 3) and find the target vertex v_{t_m}
Push v_s into vertex heap VH
EndFor
While size of VF is greater than $numf$
Pop v_s from VH
Remove each $f \in R_{e_{st_m}}$ from FL, and $adj\_faces(v_{t_m})$
Update each $f \in R_{v_s} \backslash R_{e_{st_m}}$
by replacing v_s with v_t
Recompute the cost of each $v_i \in N_{v_s}$ and update VH
Remove v_s from VL
EndWhile

5 EXPERIMENTS AND COMPARISONS

For validating the performance of SESIMP, it is compared with two algorithms: QSlim [14] and FMLOD [19]. QSlim still represents state-of-the-art algorithms having the best speed-quality tradeoff [29]. Like SESIMP, FMLOD also employs normal field deviation and competes well with QSlim in terms of quality and speed.

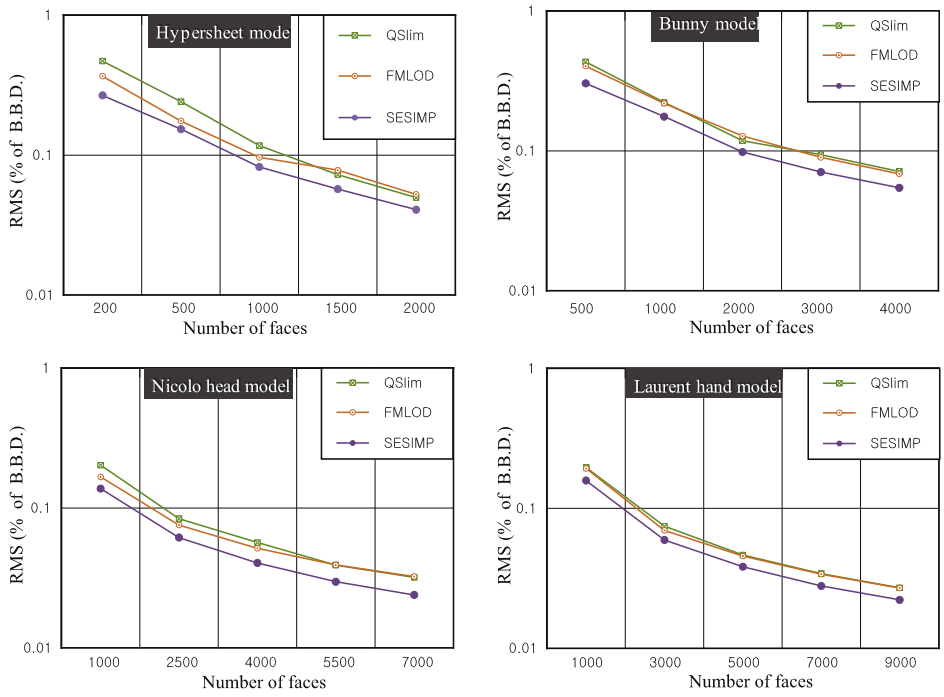

Fig. 5. Plots of root mean square (RMS) error for hypersheet, bunny, Nicolo head and Laurent hand models. In each of these plots RMS error (for five LODs of each model as the % of the bounding box diagonal of the original model) has been drawn using logarithmic y-axis.

As SESIMP is based on half-edge collapse, for fair comparison we used the variant of QSlim that also employs half-edge collapse.

Model	**#Faces**	**SESIMP**	**FMLOD**	**QSLIM-1**	**QSLIM-2**
Hypersheet	3 832	0.047	0.067	0.031	0.092
Bunny	69 451	1.149	1.325	0.906	1.875
Nicolo Head	355 886	6.789	7.359	5.64	10.578
Laurent Hand	701 543	12.891	14.485	11.234	21.672
Satva	3 631 628	72.339	82.531	68.203	156.297
David	7 227 031	145.476	167.953	153.297	–

Table 1. Running times (in seconds to simplify to one face)

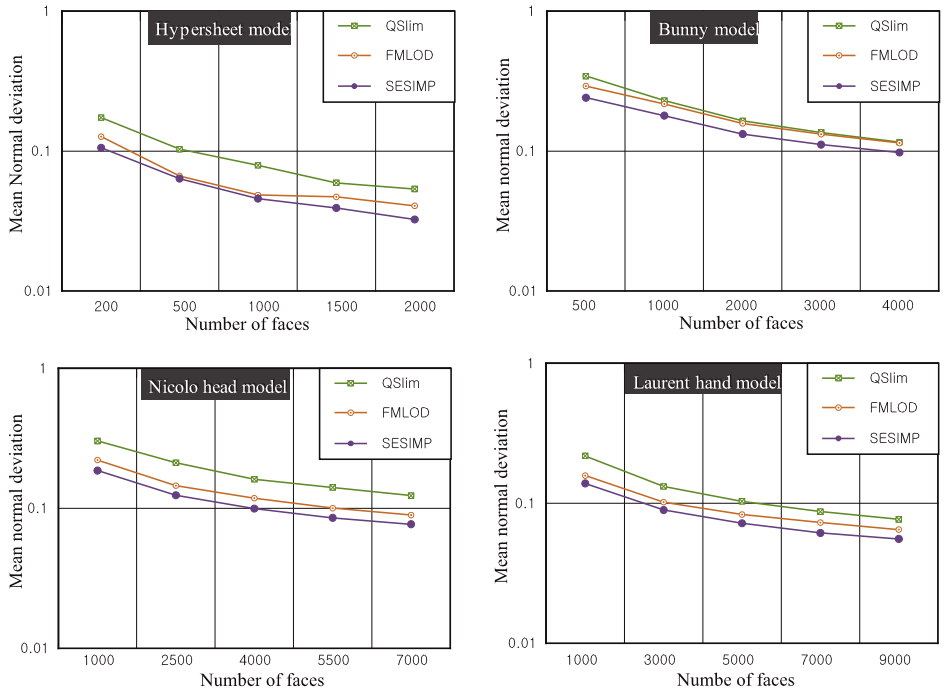

Fig. 6. Plots of mean normal deviation for hypersheet, bunny, Nicolo head and Laurent hand models. In each of these plots, mean normal deviation has been drawn using logarithmic y-axis.

5.1 Simplification Time

Table 1 summarizes the execution times for simplifying models of different complexities to one face on a system equipped with Intel Centrino Duo 2.1 GHz and 2 GB RAM. Note that QEM does not prevent fold-overs automatically [14]; QSlim-1 and QSlim-2 are the variants of QSlim without and with check for fold-overs. This table shows that SESIMP is comparable with FMLOD and performs better than QSlim in terms of speed. One reason for SESIMP to be more efficient than QSlim is that it does not need any check for fold-overs; in the case of a fold-over, a face is flipped causing drastic increase in the normal field deviation, and so the proposed error measure assigns a big cost to the corresponding vertex and prevents its elimination automatically. Also, note that QSlim-2 could not simplify David model and QSlim-1 takes relatively more time, perhaps, because of paging.

# Faces	QSlim	FMLOD	SESIMP	% Improvement over QSlim	% Improvement over FMLOD
Hypersheet Model					
200	0.469787	0.366432	0.267049	43	27
500	0.240903	0.175351	0.153235	36	13
1 000	0.116816	0.0962	0.082259	30	14
1 500	0.072448	0.077987	0.057186	21	27
2 000	0.049709	0.052434	0.040739	18	22
Bunny Model					
500	0.43209	0.403112	0.303148	30	25
1 000	0.221372	0.217826	0.175455	21	19
2 000	0.117966	0.127485	0.097973	17	23
3 000	0.093825	0.08975	0.070483	25	21
4 000	0.071224	0.068523	0.054399	24	21
Nicolo Head Model					
1 000	0.202267	0.166563	0.137615	32	17
2 500	0.083874	0.075487	0.061485	27	19
4 000	0.056629	0.051531	0.040477	29	21
5 500	0.039129	0.039148	0.029716	24	24
7 000	0.031941	0.032382	0.023918	25	26
Laurent Hand Model					
1 000	0.195942	0.192913	0.158171	19	18
3 000	0.074502	0.069616	0.059462	20	15
5 000	0.046437	0.045634	0.038272	18	16
7 000	0.034189	0.033912	0.027894	18	18
9 000	0.027051	0.026985	0.022165	18	18

Table 2. Root Mean Square (RMS) error

5.2 Objective Comparison

For objective comparison, we selected four models with different levels of complexity: *hypersheet*, *bunny*, *Nicolo head*, and *Laurent hand*, and used two quality measures (RMS – root mean square, and normal field deviation). RMS is less sensitive to outliers and is true predictor of overall geometric quality of approximations. Normal

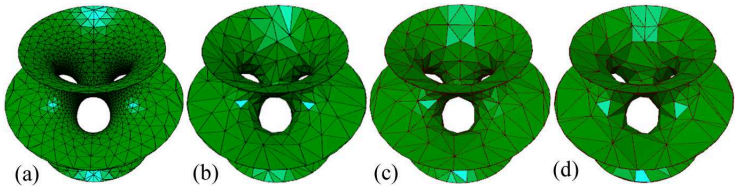

Fig. 7. Hypersheet model simplified from 3 832 to 500 faces (13 %) by (b) SESIMP, (c) FMLOD and (d) QSlim

# Faces	QSLIM	FMLOD	SESIMP	% Improvement over QSLIM	% Improvement over FMLOD
			Hypersheet Model		
200	0.173424	0.126755	0.105459	39	17
500	0.103228	0.066423	0.063602	38	4
1 000	0.079078	0.048543	0.045727	42	6
1 500	0.059236	0.047083	0.039219	34	17
2 000	0.053569	0.040616	0.032454	39	20
			Bunny Model		
500	0.343592	0.291232	0.240481	30	17
1 000	0.229357	0.216944	0.179085	22	17
2 000	0.16432	0.157686	0.132325	19	16
3 000	0.135751	0.132026	0.111222	18	16
4 000	0.115814	0.114155	0.097664	16	14
			Nicolo Head Model		
1 000	0.302842	0.22129	0.186146	39	16
2 500	0.211346	0.145165	0.124049	41	15
4 000	0.161226	0.118144	0.099499	38	16
5 500	0.140916	0.100236	0.085477	39	15
7 000	0.123213	0.0895	0.076999	38	14
			Laurent Hand Model		
1 000	0.217581	0.157535	0.138561	36	12
3 000	0.131858	0.101766	0.089471	32	12
5 000	0.103285	0.083077	0.071956	30	13
7 000	0.087184	0.072776	0.061327	30	16
9 000	0.076748	0.064843	0.055614	28	14

Table 3. Mean Normal Deviation

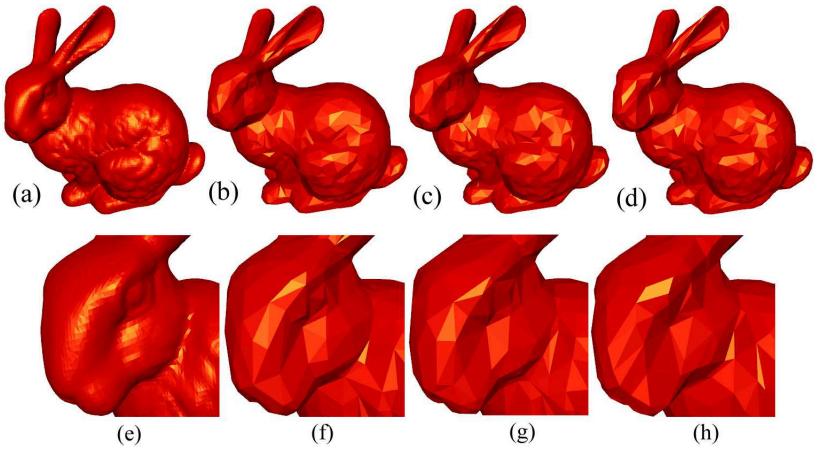

Fig. 8. (a) Original bunny model (#faces 69 451) and (e) close-up of its head simplified to 2 000 faces (2.87 %) by (b, f) SESIMP, (c, g) FMLOD, and (d, h) QSlim

field deviation is an indicator of visual quality. Figure 5 shows the plots of RMS error (as percentage of the bounding box diagonal) measured with Metro [6, 7] and Figure 6 depicts the plots of the normal field deviation measured with MeshDev [33, 34]. These plots and the error statistics given in Tables 2 and 3 indicate that SESIMP performs better and has improvement of 17–43 % in terms of RMS and 16–39 % in terms of mean normal deviation. Also, note that average improvement of full edge collapse variant of QSlim is 30.6 % [15] and that of SESIMP is 24.4 % over half-edge collapse variant of QSlim. It means that SESIMP is a better choice if the limitation is to use only half-edge collapse. The reason why SESIMP performs better is that it employs normal based error metric which captures the anisotropy of the surface in a better way than distance based metric [10].

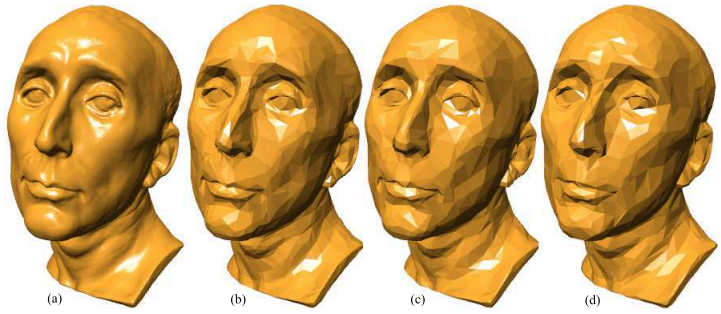

Fig. 9. (a) Original Nicolo head model (#faces 355 886) simplified to 3 000 faces (0.84 %) (b) SESIMP, (c) FMLOD, and (d) QSlim

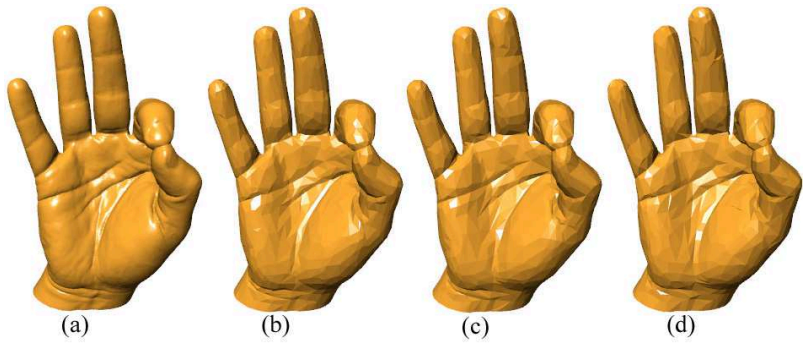

Fig. 10. (a) Original Laurent hand model (#faces 701 543) simplified to 3 000 faces (0.43 %) by (b) SESIMP, (c) FMLOD, and (d) QSlim

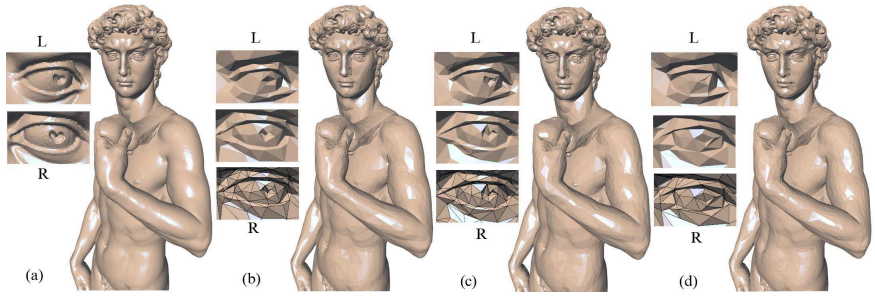

Fig. 11. (a) Original David model (#faces 7 227 031) along with close-up of eyes (L: left eye -upper, R: right eye) simplified to 40 000 faces (0.55 %) by (b) SESIMP, (c) FMLOD, and (d) QSlim

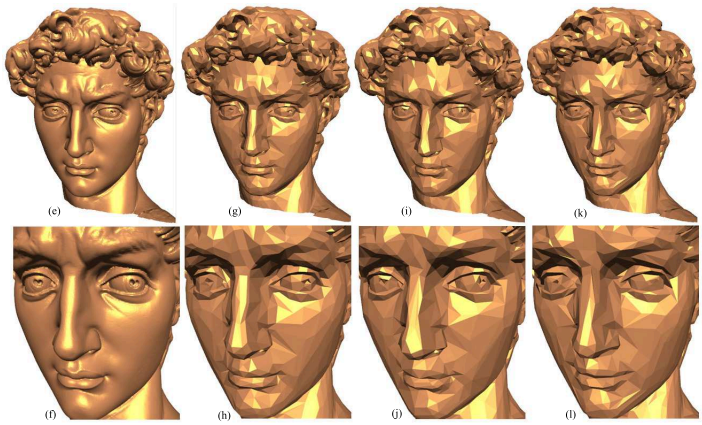

Fig. 12. (e), (f) An LOD of David head model with close-up generated by (g), (h) SESIMP, (i), (j) FMLOD and (k), (l) QSlim

5.3 Qualitative Comparison

For visual comparison, low resolution LODs, generated with the three algorithms, of some polygonal models having different degrees of complexity are presented in Figures 7, 8, 9, 10, 11, 12, 13; to keep the comparison transparent, these LODs are rendered using flat shading. In addition, error maps (computed with Metro) of some LODs are shown in Figure 14.

Figure 7 illustrates how well SESIMP preserves the original shape; note the hole. It is noteworthy that most of the triangular faces in the LOD created by SESIMP are well-shaped, which is important for numerical processing. Observe the low resolution LODs of the bunny model shown in Figure 8; the visual appearance of the model is kept by SESIMP in more better way, especially see the bunny eye in the close-up view; the error maps of the bunny LODs in Figure 14 further validate

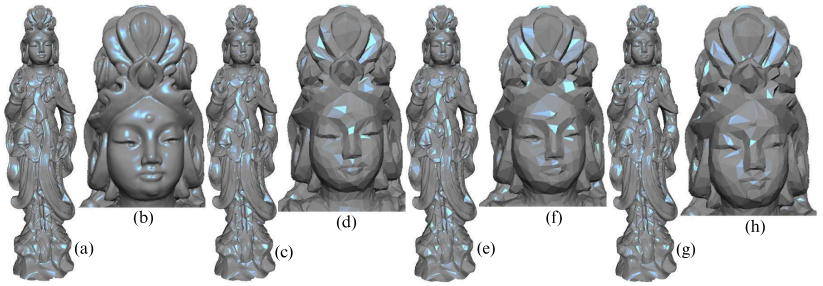

Fig. 13. (a) Original Satva model (#faces 3,631,628) and (b) its close-up simplified to 18 000 faces (0.49 %) by (c), (d) SESIMP, (e), (f) FMLOD, and (g), (h) QSlim

the assertion that the bunny LOD generated by SESIMP is of better quality. Also, observe an LOD (with 0.84 % faces) of Nicolo head model shown in Figure 9; visually important features like eyes and the dimple on the right cheek are well preserved by SESIMP.

Figure 10 shows the LODs of Laurent Hand model; the error maps of these LODs depicted in Figure 14 illustrate that SESIMP outperforms FMLOD and QSlim. Have a look at the LODs of huge David model (with more than 7 million faces) and the close-ups of its left and right eyes in Figure 11, SESIMP preserves eyes in a better way. Further, LODs in Figure 11 and their corresponding error maps in Figure 14 show how well salient features of David head model are preserved by SESIMP. Also, observe the LODs of Satva model (with more than 3.5 million faces) shown in Figure 13; it is apparent that face features are better preserved in the LOD created by SESIMP. Visual comparison and error maps of low resolution LODs of polygonal models with different levels of complexity reveal that SESIMP generates better quality LODs and preserves visually important features even at very low levels of detail.

5.4 Memory Usage

The statistics about memory used by the three methods are given in Table 4 in terms of the number of vertices n in a mesh. We assumed that the number of faces is about $2n$ and that of edges is $3n$. Note that QSlim uses 44 bytes for keeping quadric corresponding to each vertex. It is obvious that SESIMP has less memory overhead.

6 CONCLUSION

A simplification algorithm – SESIMP – has been proposed for in-core simplification. It provides an alternative to state-of-the-art algorithms known for their good time-accuracy trade-off like QSlim and FMLOD. It is based on a measure of geometric fidelity that exploits local normal field variation of a surface. A thorough comparison

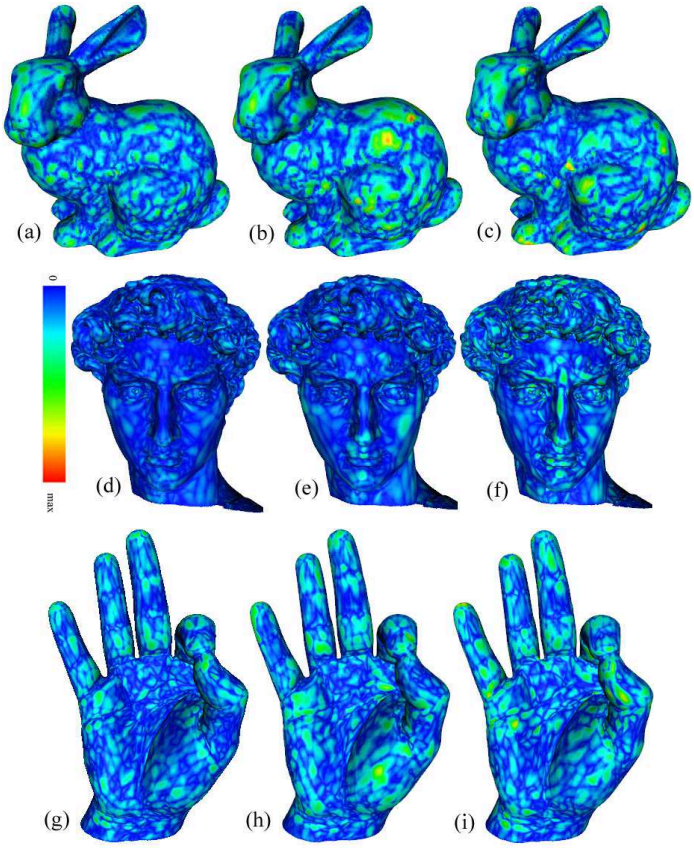

Fig. 14. Error map of LODs of bunny, David head, and Laurent hand models generated by (a), (d), (g) SESIMP, (b, e, h) FMLOD, and (c), (f), (i) QSlim. Color strip indicates the error from minimum (blue) to maximum (red)

with similar state-of-the-art methods reveals that SESIMP is well-versed with the potential of adaptively simplifying and preserving the important geometric features of a surface model even at very low levels of detail, prevents fold-overs automatically, involves less execution time and has less memory overhead. Moreover, it is simple to implement, and is robust in the sense that it is capable of constructing LODs of different kinds of polygonal surfaces with almost the same level of fidelity. It is suitable for applications where the focus is better tradeoff between quality and speed, and simplification is used as a processing step in other algorithms. The only drawback of SESIMP is that it is applicable only for the situation where approximation is needed to have vertices which form the subset of the original vertices. Also, this is not suitable for applications which need approximations with tight error bound. As SESIMP does not need to store any kind of geometric history, it can prove to be

Simplex	Data Item	SESIMP	FMLOD	QSlim
Vertices	Position	12n	12n	12n
	Face Links	24n	24n	24n
	Quadrics	–	–	88n
	Cost	8n	8n	–
	Target vertex	4n	4n	–
	Heap backlink	4n	4n	–
Edges	Endpoints	–	–	24n
	Target vertex	–	–	12n
	Cost	–	–	12n
	Heap backlink	–	–	12n
Faces	Vertices	24n	24n	24n
	Original normal	–	24n	–
	Current normal	24n	24n	–
Total		100n	124n	208n

Table 4. Statistics about memory usage in bytes

very useful for view-dependent refinement and out-of-core simplification. It is the subject of our future work.

Acknowledgment

This work was supported by the Research Center of College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia. The authors are grateful for this support.

REFERENCES

[1] Alliez, P.—Laurent, N.—Sanson, H.—Schmitt, F.: Mesh Approximation Using a Volume-Based Metric. Proc. Pacific Graphics 1999, pp. 292–301.

[2] Agarwal, P. K.—Suri, S.: Surface Approximation and Geometric Partitions. Proc. 5th ACM-SIAM Symp. on Discrete Algorithms 1994, pp. 24–33.

[3] Boubekeur, T.—Alexa, M.: Mesh Simplification by Stochastic Sampling and Topological Clustering. Computers and Graphics, DOI:10.1016/j.cag.2009.03.025, 2009.

[4] Brodsky, D.—Watson, B.: Model Simplification Through Refinement. Proceedings of Graphics Interface 2000, pp. 221–228.

[5] Ciampalini, A.—Cignoni, P.—Montani, C.—Scopigno, R.: Multiresolution Decimation Based on Global Error. The Visual Computer, Vol. 13, 1997, pp. 228–246.

[6] Cignoni, P.—Rocchini, C.—Scopigno, R.: Metro: Measuring Error on Simplified Surfaces. Computer Graphics Forum, Vol. 17, 1998, No. 2, pp. 167–174.

[7] Cignoni, P.: Metro Tool. Available at: `http://vcg.sourceforge.net/tiki-index.php?page=Metro`, 2007.

[8] Cignoni, P.—Montani, C.—Scopigno, R.: A Comparison of Mesh Simplification Algorithms. Computer and Graphics, Vol. 22, 1998, No. 1, pp. 37–54.

[9] Chen, H. K.—Fahn, C. S.—Tsai, J. P.—Chen, R. M.—Lin, M. B.: A Linear Time Algorithm for High Quality Mesh Simplification. Proc. IEEE Sixth International Symposium on Multimedia Software Engineering 2004, pp. 169–176.

[10] Cohen-Steiner, D.—Alliez, P.—Desbrun, M.: Variational Shape Approximation. ACM Transactions on Graphics. Special issue for SIGGRAPH Conference 2004, pp. 905–914.

[11] Erikson, C.—Manocha, D.: GAPS: General and Automatic Polygonal Simplification. Proc. 1999 ACM Symposium on Interactive 3D Graphics – ACM SIGGRAPH 1999, pp. 79–88.

[12] Cohen, J.—Varshney, A.—Manocha, D.—Turk, G.—Weber, H.—Agarwal, P.—Brooks, G.—Wright, W.: Simplification Envelopes. Proc. SIGGRAPH 96, pp. 119–128.

[13] Fu, J. H. G.: Convergence of Curvatures in Secant Approximations. Journal of Differential Geometry, Vol. 37, 1993, pp. 177–190.

[14] Garland, M.—Heckbert, P. S.: Surface Simplification Using Quadric Error Metric. Proc. SIGGRAPH '97, pp. 209–216.

[15] Garland, M.: Quadric-Based Polygonal Surface Simplification. Ph. D. thesis, Carnegie Mellon University, May 1999.

[16] Guskov, I.—Sweldens, W.—Schroeder, P.: Multiresolution Signal Processing for Meshes. Computer Graphics – Proc. SIGGRAPH 99, pp. 325–334.

[17] Hoppe, H.: Progressive Meshes. Proc. SIGGRAPH '96, pp. 99–108.

[18] Hussain, M.—Okada, Y.—Niijima, K.: Efficient and Feature-Preserving Triangular Mesh Decimation. Journal of WSCG, Vol. 12, 2004, No. 1, pp. 167–174.

[19] Hussain, M.—Okada, Y.: LOD Modelling of Polygonal Models. Machine Graphics and Vision, Vol. 14, 2005, No. 3, pp. 325–343.

[20] Kim, S. J.—Kim, C. H.—Levin, D.: Surface Simplification Using a Discrete Curvature Norm. Computers & Graphics, Vol. 26, 2002, pp. 657–663.

[21] Klein, R.—Liebich, G.—Strasser, V.: Mesh Reduction with Error Control. Proc. IEEE Visualization 1996, pp. 311–318.

[22] Kobbelt, L.—Campagna, S.—Seidel, H. P.: A General Framework for Mesh Decimation. Proc. Graphics Interface'98, pp. 311–318.

[23] Lee, A.—Moreton, H.—Hoppe, H.: Displaced Subdivision Surfaces. Computer Graphics (Proc. SIGGRAPH00) 2000, pp. 85–94.

[24] Lee, C. H.—Varshney, A.—Jacobs, D. W.: Mesh Saliency. ACM Transactions on Graphics, Vol. 24, 2005, No. 3, pp. 659–666.

[25] Lindstrom, P.—Turk, G.: Fast and Memory Efficient Polygonal Simplification. Proc. IEEE Visualization' 98, pp. 279–286.

[26] Lindstrom, P.: Model Simplification using Image and Geometry-Based Metrics. Ph. D. thesis, Georgia Institute of Technology 2000.

[27] Lindstrom, P.: Out-of-Core Simplification of Large Polygonal Models. ACM SIGGRAPH 2000, pp. 259–262.

[28] Luebke, D.: A Survey of Polygonal Simplification Algorithms. Technical Report TR97-045, Department of Computer Science, University of North Carolina 1997.

[29] Oliver, M. K.—Hélio, P.: A Comparative Evaluation of Metrics for Fast Mesh Simplification. Computer Graphics Forum, Vol. 25, 2006, No. 2, pp. 197–210.

[30] Park, I.—Shirani, S.—Capson, D. W.: Mesh Simplification Using an Area-Based Distortion Measure. Journal of Mathematical Modelling and Algorithms, Vol. 5, 2006, pp. 309–329.

[31] Ramsey, S. D.—Bertram, M.—Hansen, C.: Simplification of Arbitrary Polyhedral Meshes. Proceedings of Computer Graphics and Imaging 2003, pp. 221–228.

[32] Ronfard, R.—Rossignac, J.: Full-range Approximation of Triangulated Polyhedra. Computer Graphics Forum, Vol. 15, 1996, No. 3, pp. 67–76.

[33] Roy, M.—Nicolier, F.—Foufou, S.—Truchetet, F.—Koschan, A.—Abidi, M.: Assessment of Mesh Simplification Algorithm Quality. Proc. of SPIE Electronic Imaging, Vol. 4661, 2002, pp. 128–137.

[34] Roy, M.: MeshDev Tool. Available at: `http://meshdev.sourceforge.net`, 2002.

[35] Rossignac, J.—Borrel, P.: Multi-Resolution 3D Approximation for Rendering Complex Scenes. Modeling in Computer Graphics 1993, pp. 455–465.

[36] Schroeder, W.—Zarge, J. A.—Lorensen, W. E.: Decimation of Triangle Meshes. Computer Graphics – Proc. SIGGRAPH 92, Vol. 26, 1992, No. 2, pp. 65–70.

[37] Southern, R.—Marais, P.—Blake, E.: Generic Memoryless Polygonal Simplification. Proc. Computer Graphics, Virtual Reality, Visualization and Interaction in Africa 2001.

[38] Tang, H.—Shu, H. Z.—Dillenseger, J. L.—Bao, X. D.—Luo, L. M.: Moment-Based Metrics for Mesh Simplification. Computers and Graphics, DOI: 10.1016/j.cag.2007.05.001, 2007.

[39] Wu, J.—Kobbelt, L.: Fast Mesh Decimation by Multiple-choice Techniques. Proc. Vision, Modeling, and Visualization, Erlangen 2002.

[40] Yan, J.—Shi, P.—Zhang, D.: Mesh Simplification with Hierarchical Shape Analysis and Iterative Edge Contraction. IEEE Transactions on Visualization and Computer Graphics, Vol. 10, 2004, No. 2, pp. 142–151.

[41] Yoshizawa, S.—Belyaev, A.—Seidel, H. P.: Fast and Robust Detection of Crest Lines on Meshes. Proc. ACM Symposium on Solid and Physical Modeling 2005, pp. 227–232.

Muhammad Hussain is an Associate Professor in the Department of Software Engineering, King Saud University, Saudi Arabia. He received his M. Sc. and M. Phil., both from University of the Punjab, Lahore, Pakistan, in 1990 and 1993, respectively. In 2003, he received his Ph. D. in computer science from Kyushu University, Fukuoka, Japan. He worked as a researcher at Japan Science and Technology Agency from April 2003 to September 2005. In September 2005, he joined King Saud University as an Assistant Professor. He worked on a number of funded projects in Kyushu University, Japan and King Saud University, Saudi Arabia. His current research interests include multiresolution techniques in computer graphics and image processing.