Hussain M. Efficient simplification methods for generating high quality LODs of 3D meshes. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 24(3): 604–inside back cover May 2009

Efficient Simplification Methods for Generating High Quality LODs of 3D Meshes

Muhammad Hussain

Department of Computer Science, King Saud University, Riyadh, Saudi Arabia

E-mail: mhussain@ksu.edu.sa; mhussain@ccis.edu.sa

Received June 27, 2008; revised March 18, 2009.

Abstract Two simplification algorithms are proposed for automatic decimation of polygonal models, and for generating their LODs. Each algorithm orders vertices according to their priority values and then removes them iteratively. For setting the priority value of each vertex, exploiting normal field of its one-ring neighborhood, we introduce a new measure of geometric fidelity that reflects well the local geometric features of the vertex. After a vertex is selected, using other measures of geometric distortion that are based on normal field deviation and distance measure, it is decided which of the edges incident on the vertex is to be collapsed for removing it. The collapsed edge is substituted with a new vertex whose position is found by minimizing the local quadric error measure. A comparison with the state-of-the-art algorithms reveals that the proposed algorithms are simple to implement, are computationally more efficient, generate LODs with better quality, and preserve salient features even after drastic simplification. The methods are useful for applications such as 3D computer games, virtual reality, where focus is on fast running time, reduced memory overhead, and high quality LODs.

Keywords polygonal models, simplification, LOD modeling, multi-resolution modeling

1 Introduction

Several application areas, which benefit from Computer Graphics, involve manipulation, visualization, storage and transmission of large amount of 3D spatial information. Nowadays, polygonal meshes, in particular triangular meshes, have become a defacto standard for encoding 3D spatial information because of their mathematical simplicity. Pursuit of realism and the recent advances in scanning devices and modeling systems have motivated the acquisition of highly complex polygonal models with convincing levels of realism; now polygonal models having millions of faces are commonplace and it is hard to process, transmit and render 3D information on current mid-level graphics systems. Although, there is substantial enhancement in graphics acceleration techniques and network bandwidth, the complexity of polygonal meshes is increasing more rapidly. Because of excessive detail and redundant information, polygonal models are far less suitable for interactive handling in applications like computer games, virtual reality, scientific visualization etc.

The solution of the problem is to decimate a mesh to get rid of the excessive detail that is really not needed by a particular application, and to create mesh instances at different levels of detail (LODs) to be adopted in the context of a specific application and rendering environment. This problem is NP-hard optimization problem, and none of the exiting solutions is optimal. Despite a sizeable body of literature on simplification methods, no algorithm exists that has the best trade off between the quality of LODs, running time and memory occupancy. To tackle this issue, two new algorithms are proposed, which create high quality LODs preserving their salient features in less running time and with less memory overhead. The underlying idea was first proposed in [1]; unlike the algorithm in [1] which uses only normal field deviation, the new algorithms employ normal field deviation as well as distance measure for computing geometric distortion. It is altogether an interesting idea to focus on the deviation of the normal field and distance measure, and to minimize them during the process of decimation. This heuristic ensures not only the minimization of normal field distortion but also geometric distortion[2]. Though the idea of using normal field is not new, in this paper it is exploited in a novel way along with distance measure to develop greedy simplification algorithms.

1.1 Related Work

A host of researchers have been attracted towards

Regular Paper

this problem during the last decade and a large number of simplification algorithms exist in the literature. For thorough survey, an interested reader is referred to consult [3–7]. We discuss here only some most related iterative algorithms. Different heuristics based on various local surface characteristics and measures like distance, volume, area, and normal field deviation, have been employed for constraining the geometric distortion during the process of decimation.

QSlim, the QEM based algorithm[8], uses distance measure and is the only algorithm so far that has good compromise between running time and the quality of generated LODs[7]. The main drawbacks of this algorithm are memory overhead and the preservation of high-frequency features. Memoryless simplification[9] is a memoryless version of QSlim and generates high quality LODs, but its running time complexity is high. Many modifications[10−12] have been proposed that incorporate feature preserving potential into QSlim; these proposals increase the running time drastically. Kim *et al.*[10] define an error metric that combines discrete curvature norm with QEM[8]. The position of the new vertex to which the edge is collapsed is found using the butterfly subdivision mask so as to minimize geometric error. Though this algorithm overcomes some of the drawbacks of QSlim (e.g., preservation of sharp features), it is not more efficient, especially, in terms of memory consumption and execution time. Time complexity of the algorithm has not been reported in the paper, but it takes much longer than QSlim because it computes Gaussian and mean curvatures at each vertex, which is time consuming process. Also, the proposed discrete curvature norm does not seem to be stable; at some vertices the square of mean curvature may be smaller than the Gaussian curvature and principal curvature becomes a complex number.

Normal field has also been employed to guide the simplification algorithms. Brodsky *et al.*[13] employ normal field variation for clustering faces in their simplification algorithm; each cluster is replaced by its representative vertex and the model is re-triangulated. This approach is fairly efficient in running time but it creates poor quality LODs. Ramsey *et al.*[14] use normal field variation and a threshold value to select edges for collapse; it is very crude treatment of normal field deviation. Though the quality of generated LODs is not reported in the paper, it is clear that it does not create good quality approximations because it results in a high degree of volume loss. Southern *et al.*[15] use normal field and volume to define an error metric that is used to order the edges for collapse. They compute maximum normal deviation over the local neighborhood of the collapsing edge by measuring the deviation of the normal vectors of the surviving faces in the neighborhood. When this measure is used to order the vertices, it results in significant loss of volume. To overcome this issue they multiply it with the local volume deviation to define the error metric. The decimation approach reported in [16] computes local distortion due to an instance of half-edge collapse by comparing current normal field of the domain submesh of the half-edge collapse transformation after the collapse with its original normal field. Though it creates LODs of acceptable quality and preserves salient features, it overestimates the distortion error because the comparison is limited to triangles in the domain of half-edge collapse and their counterparts in the original mesh, which do not constitute contiguous submesh. The method presented in [17] also uses a form of normal field distortion. It overestimates the distortion error because of accumulation of geometric error. The algorithms discussed so far exploit normal field to define a measure of geometric deviation. Erikson and Manocha[18] used a measure based on normal field to simplify normal field of the surface model but not the surface geometry; they use quadric error metric for measuring geometric deviation and combine it with other measures to preserve surface attributes like normals, colors and texture coordinates.

The proposals in [19, 20] employ volume and area based measures for minimizing the geometric distortion; Tang *et al.*[21] propose error measures that generalize the error metrics of [19, 20]. These techniques may be attractive from theoretical viewpoint, but practically they have no significant impact on the simplification problem.

1.2 Contributions

The proposed algorithms have the best trade-off between quality, speed and memory consumption.

• *Quality of Approximations*: the created LODs keep better fidelity to the original surface model in terms of Hausdorff distance and RMS error, see Fig.2. High level meaning of a surface model is preserved even after drastic reduction, see Chinese model in Fig.8.

• *Efficiency*: highly complex models can be simplified efficiently with less memory overhead, see Tables 1 and 2.

• *Simplicity and Robustness*: the implementation of the algorithm is quite simple and it is robust in the sense that models at different levels of complexity can be decimated with the same level of fidelity and efficiency.

The subsequent arrangement of the paper is as follows. In Section 2, we introduce new measures of geometric distortion. Two new greedy iterative algorithms are presented in Section 3. Performance and efficiency of the algorithms are discussed in Section 4. Section 5 concludes the paper.

2 Measures of Geometric Fidelity

Normal field of a surface model has fundamental role to play in its appearance. It is a proven fact that human visual system is more sensitive to changes in normal vectors than those in positions. Guskov *et al.*[22] have shown that normal field based approach better preserves high frequency detail. Optimization based on a distance metric does not guarantee that the normal field will also get optimized[23]. The Poincaré-Wertinger-Sobolev inequality implies that minimizing the normal field distortion ensures the minimization of the geometric deviation[2]. With this convincing evidence in support of normal field, we use it along with distance measure to define vertex and edge costs.

Note that the greedy algorithms that we introduced for simplification are different from the classical greedy simplification algorithms. In the proposed greedy frameworks, first vertices are globally ordered, then vertices are chosen one by one for removal, the vertex with minimum cost is selected first. Once a vertex is selected, the edges incident on this vertex are locally ordered; the vertex is eliminated by collapsing the edge with minimum cost. This approach speeds up the simplification process because after collapsing an edge, only the costs of the vertices in its one-ring neighborhood are updated instead of the costs of edges whose neighborhoods are affected by this collapse; obviously the number of such edges is bigger than that of the vertices in the one-ring neighborhood of the collapsed edge. The error plots shown in Fig.2 demonstrate that this technique generates LODs of better quality; it is simply because it ensures that the vertices are chosen for removal based on their local geometry, the vertices with flat or close to flat local neighborhood has negligible impact on the geometry of a model and are chosen first.

For clarity and compactness, we use the following notations:

a vector is a 3×1 matrix;

$\boldsymbol{v}$: the embedding of vertex v in R^3;

N_v: one-ring neighborhood of the vertex v;

NF_v: the set of triangular faces in N_v;

NE_v: the set of edges incident on v;

NV_v: the set of vertices in N_v;

NF_e: the set of faces incident on the ends of the edge e;

EF_e: the set of two faces incident on e.

2.1 Vertex Cost

In this subsection, we present a measure of geometric fidelity that is used to order the vertices globally. The vertex v is flat if all faces in NF_v are coplanar. Intuition dictates that a vertex is less important if it is flat, and its importance increases with increasing the degree of departure from being flat. Coplanarity of NF_v can be tested using its normal field. This observation leads to defining the priority of each vertex as follows.

The normal field deviation across the edge e_i can be defined as follows (see Fig.1(a)):

$$\begin{aligned} NFD(e_i) &= \int_{\frac{1}{2}\Delta_{t_i}} \|\boldsymbol{n}_{t_i}(s) - \boldsymbol{n}_{t_{i+1}}\|^2 \mathrm{d}s \\ &\quad + \int_{\frac{1}{2}\Delta_{t_{i+1}}} \|\boldsymbol{n}_{t_i} - \boldsymbol{n}_{t_{i+1}}(s)\|^2 \mathrm{d}s \\ &= \frac{1}{2}(\Delta_{t_i} + \Delta_{t_{i+1}})\|\boldsymbol{n}_{t_i} - \boldsymbol{n}_{t_{i+1}}\|^2 \end{aligned}$$

where t_i and t_{i+1} are triangular faces incident on e_i, and Δ_{t_i} and $\boldsymbol{n}_{t_i}$ are, respectively, the area and the unit normal vector to the face t_i. Note that $\|\boldsymbol{n}_{t_i} - \boldsymbol{n}_{t_{i+1}}\|^2 = (\boldsymbol{n}_{t_i} - \boldsymbol{n}_{t_{i+1}})^{\mathrm{T}}(\boldsymbol{n}_{t_i} - \boldsymbol{n}_{t_{i+1}})$ is equal to $2(1 - \boldsymbol{n}_{t_i}^{\mathrm{T}}\boldsymbol{n}_{t_{i+1}})$, which is computationally more efficient because it involves less number of arithmetic operations. This leads to the following expression:

$$NFD(e_i) = (\Delta_{t_i} + \Delta_{t_{i+1}})(1 - \boldsymbol{n}_{t_i}^{\mathrm{T}}\boldsymbol{n}_{t_{i+1}}).$$

This expression defines the normal deviation across the edge e_i. Considering the normal field deviation across each edge in NE_v, the cost of the vertex v is defined as follows:

$$C_1(v) = \sum_{e_i} NFD(e_i), \tag{1}$$

where summation is over all edges in NE_v. In certain situations, this measure may not reflect the local geometry, for example, see Figs. 1(b) and 1(c). To deal with this drawback, it is augmented with another measure that also exploits normal field and captures this kind of situations more accurately and forces the greedy approach to choosing the right vertex for removal. This measure is defined as follows:

$$C_2(v) = \sum_{t_i} \Delta_{t_i} - \Big\| \sum_{t_i} \Delta_{t_i}\boldsymbol{n}_{t_i} \Big\| \tag{2}$$

where the summation is over all triangular faces $t_i \in NF_v$. Incorporating it into (1), the cost of vertex v is given by

$$Cost(v) = C_1(v) + C_2(v). \tag{3}$$

It is obvious that if a vertex v is flat then $Cost(v) = 0$; $Cost(v)$ assumes values greater than 0 depending on how much the vertex v departs from being flat.

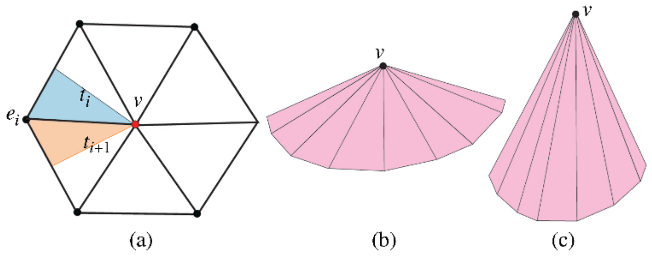

Fig.1. (a) Cost of v is calculated by considering the normal field deviation across each edge e_i. (b) and (c) Two vertices with different local geometry. Assume that the total area of N_v for each vertex is the same. If dihedral angle between each adjacent pair of faces in each case is the same then the measure (1) will associate the same cost with each vertex in spite of being different in their local geometry.

According to our knowledge, no one else has used only normal field or surface curvatures to define vertex cost like ours'. Note that a surface point is flat if $\kappa_{\min} = 0 = \kappa_{\max}$, where $\kappa_{\min}$ and $\kappa_{\max}$ are minimum and maximum normal curvatures at this point. A flat vertex is not visually important, and must be removed first of all. The vertex cost defined by (3) assigns zero value to such vertices and ensures their removal first of all. A surface point is elliptic, hyperbolic, or parabolic if $H \neq 0$ and $K > 0$, $K < 0$, $K = 0$, respectively, where $H = \frac{1}{2}(\kappa_{\min} + \kappa_{\max})$ and $K = (\kappa_{\min}\kappa_{\max})$ are mean and Gaussian curvatures at this point. Points of these kinds are equally important and contribute to the semantic meaning of a surface according to the magnitudes of surface curvatures at these points. The points where the magnitudes of surface curvatures are small are less important and those where the magnitudes are bigger carry higher level of visual importance. By the very construction of the vertex cost defined by (3), it assigns numerical value to the vertices according to the magnitudes of the surface curvatures and guides the simplification algorithm to remove the vertices according to their curvature values. It is to be noted that the vertex cost does not represent the exact curvature values, and in the perspective of the issue under hand it is not a problem because the purpose is to order the vertices according to their importance, which is served quite well and is corroborated by objective and subjective comparisons in Section 4.

2.2 Edge Collapse Cost

Once the vertex v is chosen based on its cost elaborated in Subsection 2.1, the edges in NE_v are ordered according to the expected geometric distortion caused by the collapse of each edge and the edge e_i that causes minimum geometric distortion is collapsed to eliminate v; this edge is referred to as *optimal edge*. This subsection presents the details of two measures of geometric fidelity for finding the optimal edge.

The optimal edge $e_i = \{v, v_i\}$ that is collapsed to the new vertex u is found in two different ways. The first approach is to determine the vertex u for each edge $e_i \in NE_v$ using the technique described in Subsection 2.3, and then to compute the cost of e_i as follows to decide whether it is an optimal edge. The distance of u from the faces in NF_{e_i} is given by

$$D(e_i) = \sum_{t_j} d_j(u), \tag{4}$$

where summation is over all $t_j \in NF_{e_i}$ and

$$d_j(u) = (\boldsymbol{n}_{t_j}^{\mathrm{T}}(\boldsymbol{u} - \boldsymbol{v}_j))^2$$

is the square of the distance of u from the plane of the triangle t_j. Note Garland *et al.*[8] use this measure to define QEM as a globally computed measure, but we use it as a local measure. Normal field of each $t_j \in NF_{e_i} \backslash EF_{e_i}$ also undergoes change; the normal field deviation due to t_j is defined as follows:

$$\begin{aligned} NFD(t_j) &= \int_{\Delta_{cj}} \|\boldsymbol{n}_{cj}(s) - \boldsymbol{n}_{pj}\|^2 \mathrm{d}s \\ &= \Delta_{cj}\|\boldsymbol{n}_{cj} - \boldsymbol{n}_{pj}\|^2. \end{aligned}$$

Alternatively, this can be written as

$$NFD(t_j) = \Delta_{cj}(1 - \boldsymbol{n}_{cj}^{\mathrm{T}}\boldsymbol{n}_{pj})$$

where Δ_{cj}, $\boldsymbol{n}_{cj}$, and $\boldsymbol{n}_{pj}$ are, respectively, the current area, current unit normal and previous unit normal (before edge collapse) of t_j. The normal field deviation due to the collapse of the edge $e_i = \{v, v_i\}$ is determined as follows:

$$NFD(e_i) = \sum_{t_j} NFD(t_j) \tag{5}$$

where the summation is taken over each $t_j \in NF_{e_i} \backslash EF_{e_i}$. By combining the distance and the normal field error measures, the cost of collapse of the edge $e_i = \{v, v_i\}$ is defined as follows:

$$Cost(e_i) = D(e_i) + NFD(e_i). \tag{6}$$

Note that this error measure constrains not only distance error but also normal field deviation.

The second approach is to find the optimal edge using the following measure

$$Cost(e_i) = NFD(e_i) \tag{7}$$

and then to compute u using the technique of Subsection 2.3.

2.3 Computation of Optimal Point

Using a technique similar to that by Garland *et al.*[8], the optimal position of the new vertex u corresponding to the edge $e_i = \{v, v_i\}$ is determined by minimizing the sum of the squared distances of u from the planes of the faces in NF_{e_i}. The square of the distance of u from the plane of $t_j \in NF_{e_i}$ is

$$\begin{aligned} d_j(u) &= (\boldsymbol{n}_{t_j}^{\mathrm{T}}(\boldsymbol{u} - \boldsymbol{v}_j))^2 \\ &= \boldsymbol{u}^{\mathrm{T}}\boldsymbol{A}_j\boldsymbol{u} - \boldsymbol{u}^{\mathrm{T}}\boldsymbol{A}_j\boldsymbol{v}_j - \boldsymbol{v}_j^{\mathrm{T}}\boldsymbol{A}_j\boldsymbol{u} + \boldsymbol{v}_j^{\mathrm{T}}\boldsymbol{A}_j\boldsymbol{v}_j, \end{aligned}$$

where v_j is a vertex on the plane of t_j and $\boldsymbol{A}_j = \boldsymbol{n}_{t_j}\boldsymbol{n}_{t_j}^{\mathrm{T}}$. The sum of the squared distances is

$$d(u) = \sum_{t_j}(\boldsymbol{u}^{\mathrm{T}}\boldsymbol{A}_j\boldsymbol{u} - \boldsymbol{u}^{\mathrm{T}}\boldsymbol{A}_j\boldsymbol{v}_j - \boldsymbol{v}_j^{\mathrm{T}}\boldsymbol{A}_j\boldsymbol{u} + \boldsymbol{v}_j^{\mathrm{T}}\boldsymbol{A}_j\boldsymbol{v}_j),$$

where the summation is over all faces in NF_{e_i}. Its optimization leads to the following linear system of equations:

$$\boldsymbol{A}\boldsymbol{u} - \boldsymbol{b} = 0 \tag{8}$$

where

$$\boldsymbol{A} = \sum_{t_j}\boldsymbol{A}_j, \quad \boldsymbol{b} = \sum_{t_j}\boldsymbol{A}_j\boldsymbol{v}_j.$$

The solution of this linear system yields the optimal position of $\boldsymbol{u}$. To deal with numerical instabilities, we impose the constraint that if $\|\boldsymbol{A}\| < 0.001$, then v_i is used instead. Note the difference between our approach and that by Garland *et al.*[8], in our case the constraints are computed locally on the fly and there is no need to store a 4×4 symmetric matrix as Garland *et al.*[8] do.

3 Mesh Decimation Algorithms

Two new algorithms, which employ the well-known greedy design technique, are proposed in this section. Other frameworks such as *local RS-heap*[24,25] and *multiple-choice approach*[26] can also be adopted easily and will result in even more efficiency.

The input is the original mesh M consisting of the set of vertices V, and the set of triangular faces F, which specify the geometry and topology, respectively, of M. For encoding vertex and face elements, two very simple data structures *Face* and *Vertex* are used. An instance of *Face* object is representative of a triangular face, and stores pointers to its three vertices and its current normal. An instance of *Vertex* object stands for a vertex v and includes its geometric position $p(x, y, z)$, its cost $Cost(v)$, the list — *adj_faces* — of pointers to faces in NF_v, and heap backlink. *Vertex* and *Face* objects corresponding to the vertices and faces of M are created and stored in two lists VL and FL. Each algorithm follows the framework given in the following pseudo code.

```
FSIMP(Mesh $M(V, F)$)
    Input: original triangular mesh $M = (V, F)$ and the
           target number of faces $numf$
    Output: an LOD with the given budget of faces
    For each vertex $v \in V$
        - Create Vertex object and put it in $VL$
    EndFor
    For each face $f \in F$
        - Create Face object and put it in $FL$
        - Add pointer of $f$ to $v_i.adj\_faces$ corresponding
          to its each vertex $v_i$, $i = 1, 2, 3$
    EndFor
    For each vertex $v \in V$
        - Compute $Cost(v)$ using (3)
        - Push $v$ into vertex heap $VH$
    EndFor
    While size of $VF$ is greater than $numf$
        - Pop off $v$ from $VH$
        - Find_Optimal_edge_and_point($v$)
        - Collapse the optimal edge $e_o$
        - Recompute the cost of each $v_i \in NV_v$ using (3)
          and update $VH$
        - Remove $v$ from $V$
    EndWhile
```

The two algorithms differ in the implementation of the method *find_optimal_edge_and_point*(v). This method greedily finds the optimal edge e_o and the optimal position of the new vertex u following one of the two approaches elaborated in Subsection 2.2.

```
M1 — find_optimal_edge_and_point($v$)
    - Set $C$ to MaxFloat
    For each edge $e_i \in NE_v$
        - Compute cost $C_i$ of $e_i$ using (7)
        - IF $C_i < C$
              Set $C$ to $C_i$
              Set $e_o$ to $e_i$
        - EndIF
    EndFor
    - Compute optimal point $u$ for $e_o$ using the method
      explained in Subsection 2.3
```

We refer to the algorithm that uses this method as FSIMP-1.

```
M2 — find_optimal_edge_and_point($v$)
    - Set $C$ to MaxFloat
    For each edge $e_i \in NE_v$
        - Compute optimal point $u$ for $e_i$ using the method
          explained in Subsection 2.3
        - Compute cost $C_i$ of $e_i$ using (6)
        - IF $C_i < C$
              Set $C$ to $C_i$
              Set $e_o$ to $e_i$
        - EndIF
    EndFor
```

The algorithm that employs this method is referred to as FSIMP-2. An edge collapse is considered valid if it does not create foldovers. For validity check, a method similar to the one presented in [8] is employed.

4 Results and Discussion

In this section, the performance of FSIMP-1 and FSIMP-2 is evaluated by comparing them with the state-of-the-art decimation algorithms. For comparison, QSlim[8] and GeMS[15] are selected because the former is considered standard simplification technique and a recent comparison by Oliver and Helio[7] reveals that QSlim performs better in terms of execution time and the quality of generated LODs, and the latter employs a measure of geometric error that exploits normal field and volume. For comparison, we scale the algorithms using four parameters: running time, memory consumption, quality of the generated LODs, and preservation of visually important features at low levels of detail, and employ four models of varying complexities: armadillo, dinosaur, dragon and Chinese-dragon.

4.1 Running Time Efficiency

Table 1 lists the execution times of the four algorithms to reduce each benchmark model to 100 faces on a system equipped with Intel Centrino Duo 2.1GHz CPU and 2GB of main memory. It is obvious that FSIMP-1 and FSIMP-2 outperform QSlim and GeMS in terms of running time. The reason is that FSIMP uses vertex heap and to compute/update the priority of a vertex v. It considers only N_v, and there is no need to make a single pass through the edges in NE_v to compute the priority. Though QSlim and GeMS can also be implemented using vertex heap, a single pass through the edges in NE_v will have to be made every time the cost of a vertex is computed/updated.

Table 1. Simplification Times (in seconds to be simplified to 100 faces)

Model	M. Size (No. faces)	FSIMP-1	FSIMP-2	QSlim	GeMS
Dinosaur	112 384	1.375	1.516	1.718	2.313
Armadillo	345 944	4.891	5.125	6.400	7.375
Dragon	871 412	11.746	12.844	15.024	17.906
C. Dragon	7 219 045	112.765	115.593	125.547	154.844

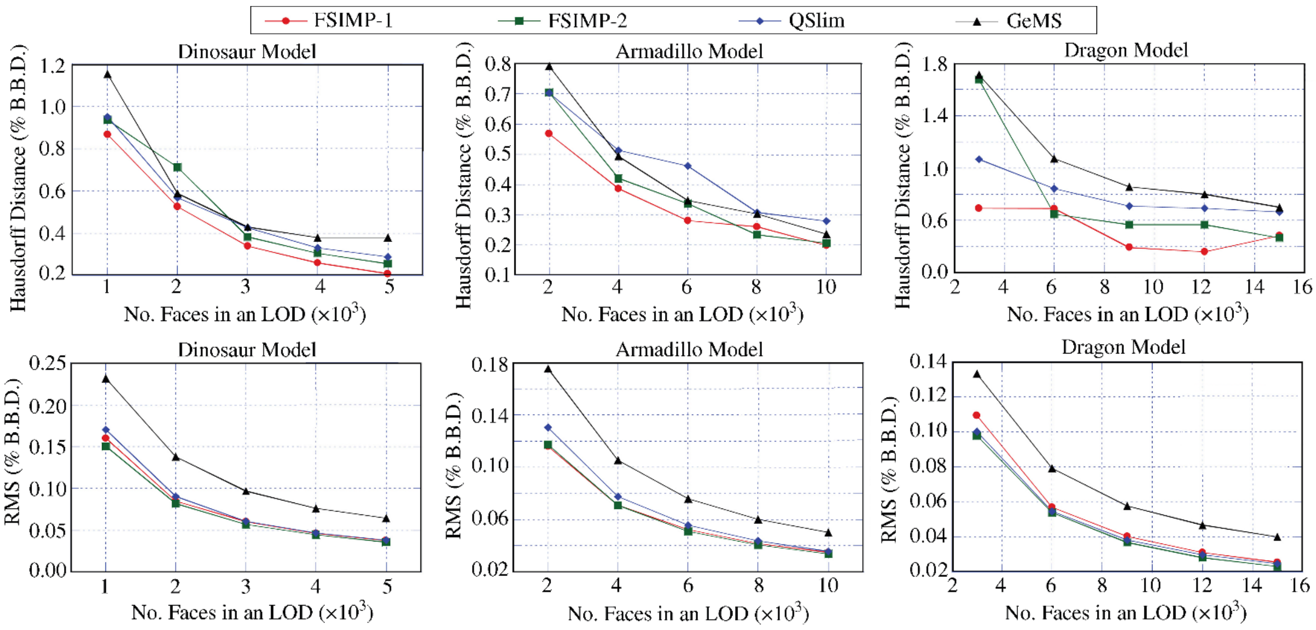

Fig.2. Plots of Hausdorff distance and RMS error.

4.2 Objective Comparison

The approximations generated by the four methods are compared by employing Symmetric Hausdorff distance and RMS error that are widely used for thorough comparison of polygonal models among graphics community. To avoid any kind of bias in objective comparison, we use well-known public domain Metro tool[27,28], which is widely used for mesh quality evaluation. Plots of the Hausdorff distances and RMS errors for 5 low resolution LODs of each benchmark model created by the four algorithms are shown in Fig.2.

4.3 Qualitative Comparison

For visual comparison, some LODs, generated with the four algorithms, of benchmark models are presented in Figs. 3, 5, 6, 7 and 8. It is quite apparent from the

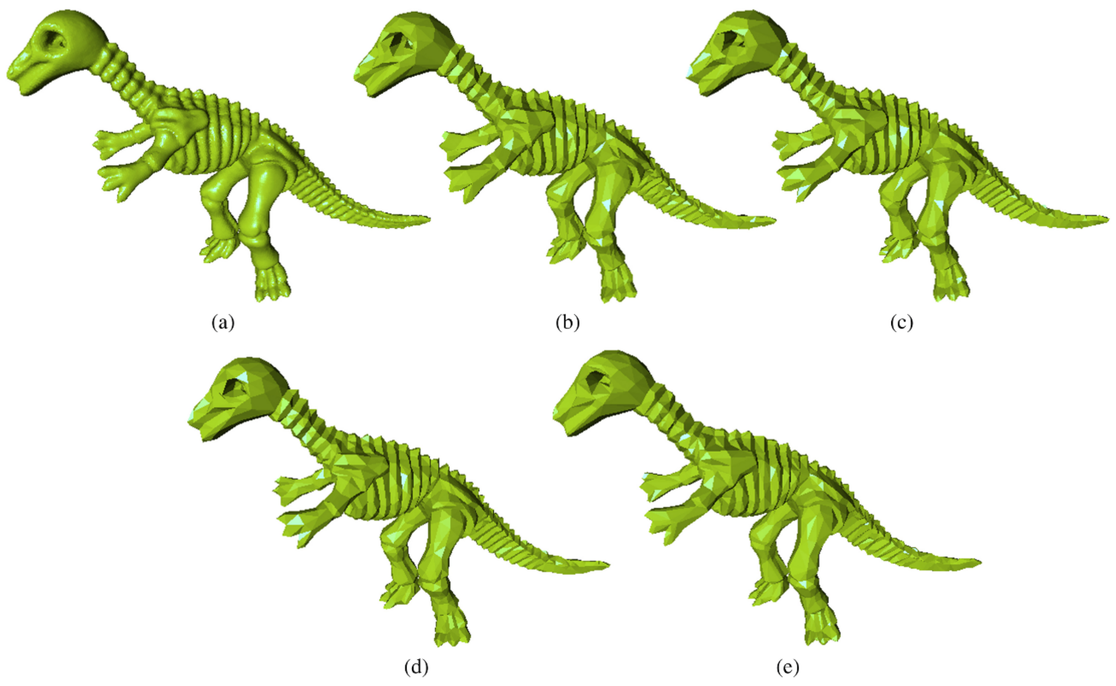

Fig.3. (a) Original dinosaur model (No. faces: 112 384). (b) LODs created by FSIMP-1. (c) LODs created by FSIMP-2. (d) LODs created by QSlim. (e) LODs created by GeMS. Each consists of 2000 faces.

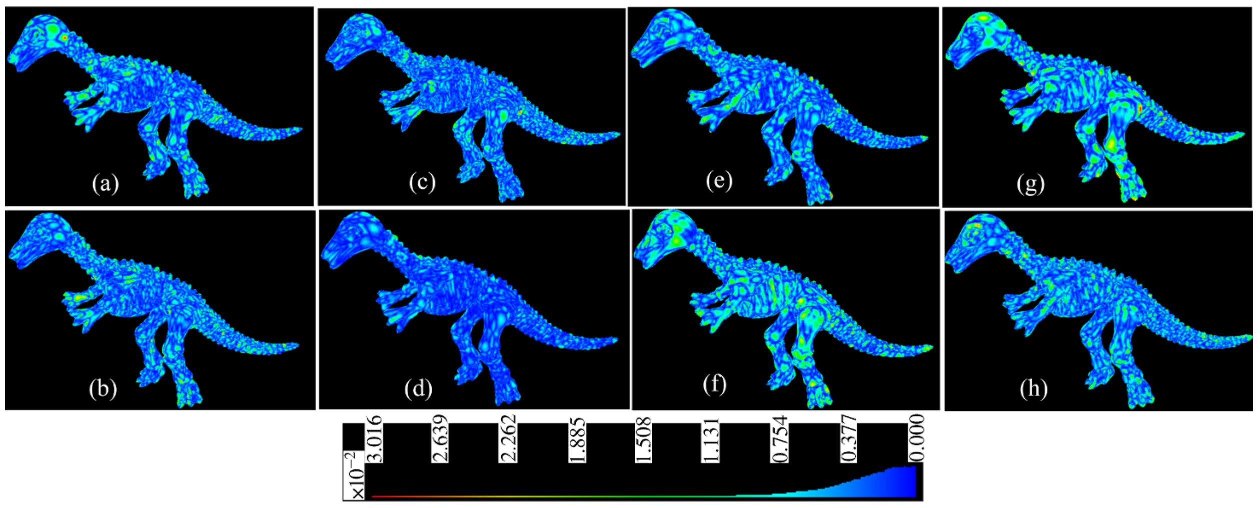

Fig.4. Error maps for two LODs (consisting of 2000 faces — left column — and 3000 faces — right column) of dinosaur model (No. faces: 112 384) created by each of the four methods. (a) and (b) FSIMP-1. (c) and (d) FSIMP-2. (e) and (f) QSlim. (g) and (h) GeMS.

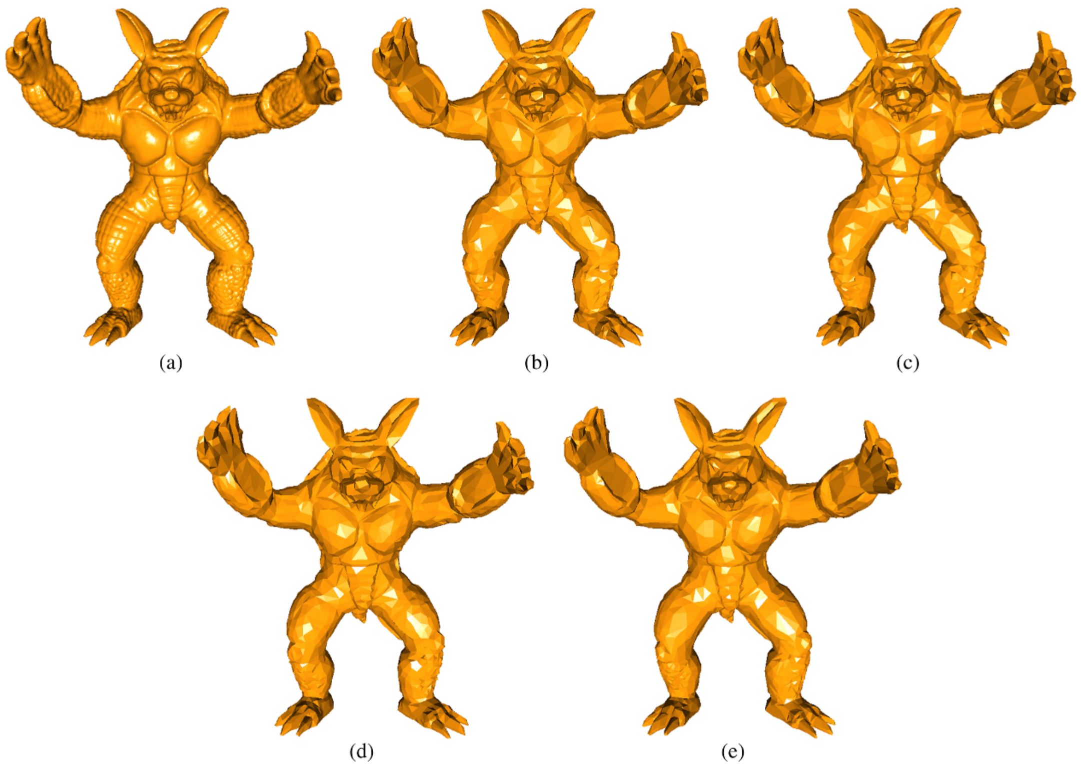

Fig.5. (a) Original armadillo model (No. faces: 345 944). (b) LODs created by FSIMP-1. (c) LODs created by FSIMP-2. (d) LODs created by QSlim. (e) LODs created by GeMS. Each consists of 5000 faces.

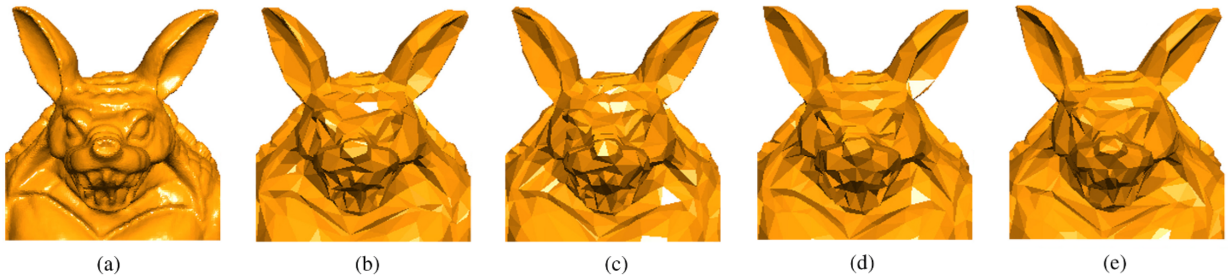

Fig.6. (a) Close-up of original armadillo model (No. faces: 345 944). (b) Close-up of its LOD created by FSIMP-1. (c) Close-up of its LOD created by FSIMP-2. (d) Close-up of its LOD created by QSlim. (e) Close-up of its LOD created by GeMS. (No. faces: 5000)

error map in Fig.4 that FSIMP-1 and FSIMP-2 outperform QSlim and GeMS, and also FSIMP-2 performs better than FSIMP-1. The close-up of armadillo model in Fig.6 shows that FSIMP-1 and FSIMP-2 have better potential to preserve visually important high frequency information and to keep the semantic meaning of the surface model. Each of FSIMP-1 and FSIMP-2 can faithfully and efficiently simplify models of sheer sizes, for example, see the Chinese dragon model shown in Fig.8 that consists of more than 7 million faces and its four LODs (with 10 000 faces each, 0.13% of the original size) created by FSIMP-1, FSIMP-2, QSlim and GeMS.

4.4 Memory Consumption

FSIMP-1, FSIMP-2 and GeMS use the same amount of memory, so we focus on the memory comparison of QSlim and FSIMP-1. Table 2 gives the statistics about memory consumption of the two algorithms. Note that $n = |V|$ is the number of vertices in the mesh and the statistics in Table 2 are based on the assumption that $|F| \approx 2n$ and $|E| \approx 3n$. It is evident that FSIMP-1 consumes about 66% less memory than QSlim.

5 Conclusions

Two new automatic greedy simplification algorithms

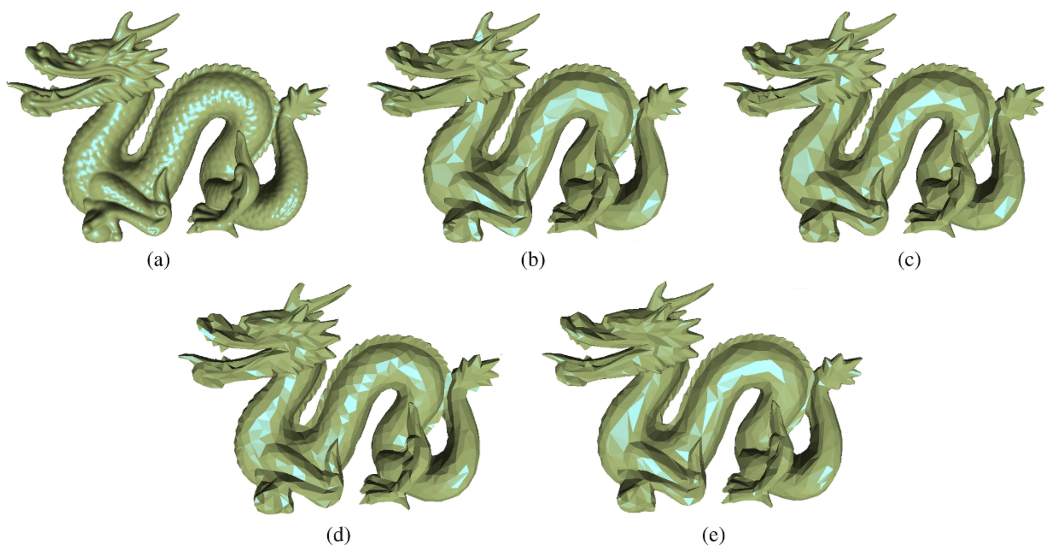
Fig.7. (a) Original dragon model (No. faces: 871 412). (b) LODs created by FSIMP-1. (c) LODs created by FSIMP-2. (d) LODs created by QSlim. (e) LODs created by GeMS. Each consists of 5000 faces.

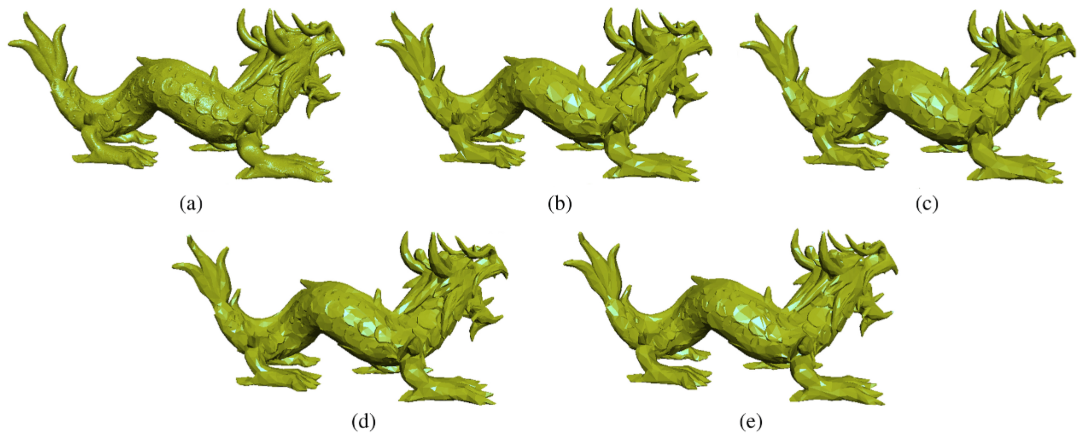
Fig.8. (a) Original Chinese dragon model (No. faces: 7 219 045). (b) LODs created by FSIMP-1. (c) LODs created by FSIMP-2. (d) LODs created by QSlim. (e) LODs created by GeMS. Each consists of 10000 faces.

Table 2. Statistics about Memory Consumption

Simplex	Data Item	FSIMP-1	QSlim
Vertices	Position	$12n$	$12n$
	Face Links	$24n$	$24n$
	Quadrics	–	$88n$
	Cost	$8n$	–
	Heap Backlink	$4n$	–
Edges	Endpoints	–	$24n$
	Target Vertex	–	$12n$
	Cost	–	$12n$
	Heap Backlink	–	$12n$
Faces	Vertices	$24n$	$24n$
	Current Normal	$24n$	–
Total		$96n$	$208n$

are proposed. Normal field deviation is employed to define a new measure of geometric fidelity for ordering the vertices for removal. Each vertex is decimated by collapsing one of the incident edges whose cost and substituent vertex are found using a measure based on distance and normal field deviation. The proposed algorithms have the best trade-off between quality, speed and memory consumption. They outperform the state-of-the-art algorithms in terms of execution time, memory consumption, preservation of salient features, and the quality of LODs. FSIMP-1 is slightly faster than FSIMP-2 but the latter performs better than the former in terms of RMS error. These algorithms can faithfully and efficiently simplify highly complex polygonal models. They are useful for applications where primary focus is on interactivity, small memory overhead and visual appearance. They are applicable only to manifold surface models. They are not suitable for applications where tight error bounds on LODs are required. Also, they address only the geometric simplification of polygonal models, not the issues like topology and attributes (color, texture coordinates etc.) simplification.

Acknowledgement We are thankful to the anonymous reviewers for their valuable comments to improve the quality of this document.

References

[1] Hussain M. Fast decimation of polygonal models. In *Proc. ISVC 2008*, Part I, *LNCS* 5358, Springer-Verlag, Las Vegas, USA, Dec. 2–4, 2008, pp.119–128.

[2] Cohen D S, Alliez P, Desbrun M. Variational shape approximation. *ACM Trans. Graphics*, 2004, 23(3): 905–914.

[3] Luebke D. A survey of polygonal simplification algorithms. Technical Report TR97-045, Department of Computer Science, University of North Carolina, December, 1997.

[4] Cignoni P, Montani C, Scopigno R. A comparison of mesh simplification algorithms. *Computer & Graphics*, 1998, 22(1): 37–54.

[5] Garland M. Quadric-based polygonal surface simplification [Ph.D. Dissertation]. Carnegie Mellon University, May 1999.

[6] Lindstrom P. Model simplification using image and geometry-based metrics [Ph.D. Dissertation]. Georgia Institute of Technology, November 2000.

[7] Oliver M K, Hélio P. A comparative evaluation of metrics for fast mesh simplification. *Computer Graphics Forum*, June 2006, 25(2): 197–210.

[8] Garland M, Heckbert P S. Surface simplification using quadric error metric. In *Proc. SIGGRAPH'97*, Los Angeles, USA, August 2–8, 1997, pp.209–216.

[9] Lindstrom P, Turk G. Fast and memory efficient polygonal simplification. In *Proc. IEEE Visualization'98*, NC, USA, Oct. 18–23, 1998, pp.279–286.

[10] Kim S J, Kim C H, Levin D. Surface simplification using a discrete curvature norm. *Computers & Graphics*, 2002, 26(5): 657–663.

[11] Yan J, Shi P, Zhang D. Mesh simplification with hierarchical shape analysis and iterative edge contraction. *Trans. Visualization and Computer Graphics*, 2004, 10(2): 142–151.

[12] Yoshizawa S, Belyaev A, Seidel H P. Fast and robust detection of crest lines on meshes. In *Proc. ACM Symp. Solid and Physical Modeling'05*, Wales, UK, June 2–4, 2005, 227–232.

[13] Brodsky D, Watson B. Model simplification through refinement. In *Proc. Conference Graphics Interface*, Quebec, Canada, May 15–17, 2000, pp.221–228.

[14] Ramsey S D, Bertram M, Hansen C. Simplification of arbitrary polyhedral meshes. In *Proc. Conference Computer Graphics and Imaging*, Hawaii, USA, August 13-15, 2003, pp.221–228.

[15] Southern R, Marais P, Blake E. Generic memoryless polygonal simplification. In *Proc. Int. Conf. Computer Graphics, Virtual Reality, Visualization and Interaction in Africa*, Cape Town, South Africa, Nov. 5–7, 2001, pp.7–15.

[16] Hussain M, Okada Y. LOD modelling of polygonal models. *Machine Graphics and Vision*, 2005, 14(3): 325–343.

[17] Hussain M, Okada Y, Niijima K. Efficient and feature-preserving triangular mesh decimation. *Journal of WSCG*, 12(1): 2004, pp.167–174.

[18] Erikson C, Manocha D. GAPS: General and automatic polygonal simplification. In *Proc. ACM Symposium on Interactive 3D Graphics'99*, Georgia, USA, April 26–28, 1999, pp.79–88.

[19] Alliez P, Laurent N, Sanson H, Schmitt F. Mesh approximation using a volume-based metric. In *Proc. Pacific Graphics Conference*, Seoul, Korea, Oct. 5–7, 1999, pp.292–301.

[20] Park I, Shirani S, Capson D W. Mesh simplification using an area-based distortion measure. *Journal of Mathematical Modelling and Algorithms*, 2006, 5(3): 309–329.

[21] Tang H, Shu H Z, Dillenseger J L, Bao X D, Luo L M. Moment-based metrics for mesh simplification. *Computers & Graphics*, 2007, 31(5): 710–718.

[22] Guskov I, Sweldens W, Schroeder P. Multiresolution signal processing for meshes. In *Proc. SIGGRAPH'99*, Los Angeles, USA, August 8–13, 1999, pp.325–334.

[23] Fu J H. Convergence of curvatures in secant approximations. *Jouranl of Differential Geometry*, 1993, 37: 177–190.

[24] Chen H K, Chen R M, Fahn C S, Tsai J J, Lin M B. A linear time algorithm for high quality mesh simplification. In *Proc. the 6th IEEE International Symp. Multimedia Software Engineering'04*, Miami, USA, Dec. 13–15, 2004, pp.169–176.

[25] Chen H K, Fahn C S, Tsai J J, Chen R M, Lin M B. Generating high-quality discrete LOD meshes for 3D computer games in linear time. *Multimedia Systems*, March 2006, 11(5): 480–494.

[26] Wu J, Kobbelt L. Fast mesh decimation by multiple-choice techniques. In *Proc. Int. Fall Workshop on Vision, Modeling, and Visualization'02*, Erlangen, Germany, Nov. 20–22, 2002, pp.241–248.

[27] Cignoni P, Rocchini C, Scopigno R. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 1998, 17(2): 167–174.

[28] Cignoni P. Metro tool. 2007, http://vcg.sourceforge.net/tiki-index.php?page=Metro.

Muhammad Hussain received the M.Sc. degree in applied mathematics and the M.Phil. degree in computational mathematics specializing in CAGD (computer aided geometric design), both from University of the Punjab, Lahore, Pakistan, in 1990 and 1993 respectively. In 2003, He received the Ph.D. degree in computer science from Kyushu University, Fukuoka, Japan. After completing the Ph.D. degree, he worked as a researcher for Japan Science and Technology Agency at Kyushu University from April 2003 to September 2005. In September 2005, he joined King Saud University as an assistant professor. Currently, he is working at King Saud University. His current research interests include multi-resolution modeling and analysis in computer graphics and image processing.