Journal of Computer Science

Original Research Paper

Mesh Simplification by Curvature-Enhanced Quadratic Error Metrics

Paolo Pellizzoni and Gianpaolo Savio

University of Padua, Padua, Italy

Article history
Received: 08-05-2020
Revised: 24-08-2020
Accepted: 03-09-2020

Corresponding Authors:
Paolo Pellizzoni
University of Padua, Padua, Italy
Email: paolo.pellizzoni@studenti.unipd.it

Abstract: Polygonal meshes have a significant role in computer graphics, design and manufacturing technology for surface representation and it is often required to reduce their complexity to save memory. An efficient algorithm for detail retaining mesh simplification is proposed; in particular, the method presented is an iterative edge contraction algorithm based on the work of Garland and Heckberts. The original algorithm is improved by enhancing the quadratic error metrics with a penalizing factor based on discrete Gaussian curvature, which is estimated efficiently through the Gauss-Bonnet theorem, to account for the presence of fine details during the edge decimation process. Experimental results show that this new algorithm helps preserve the visually salient features of the model without compromising performance.

Keywords: Mesh Simplification, Discrete Curvature, Computational Geometry

Introduction

Nowadays, due to ever-improving computer-aided modeling in both the film industry and manufacturing, it's easy to find ultra-detailed 3D models with millions of faces. Moreover, 3D scanning techniques often produce even larger files due to their redundancy in collecting points. While in some cases this enormous resolution is necessary, in other scenarios, such as mobile applications or in additive manufacturing, it may be beneficial to have a reduced number of triangles in the mesh while maintaining even the finer details.

One of the most common approaches in polygonal mesh decimation is the iterative edge contraction; in order to simplify the mesh, this algorithm iteratively selects an edge and collapses it into a single vertex; this process is repeated until the number of faces of the model is reduced down to the desired quantity. Other well-known algorithms are vertex decimation, presented for the first time by Schroeder *et al*. (1997), which iteratively picks a vertex for removal, removes its adjacent faces and re-triangulates the resulting hole, or vertex clustering, which joins vertices that are within some threshold distance as presented in Rossignac and Borrel (1993) and Hua *et al*. (2015). Edge contraction algorithms often use an error function to decide the way edges are collapsed, so the choice of this function leads to different results. Among the choices of the error metric used in the literature, one of the most well known and widely used ones is the quadric error metrics, proposed by Garland and Heckbert (1997), which is fast and reliable, but cannot account for the presence of key features. As later shown by Heckbert and Garland (1999), this method implicitly relies on the principal curvatures of the surface. Michaud *et al*. (2017) used an error metric based solely on the curvature of the surface, computed through the use of algebraic spheres. While most of those approaches guarantee a small error in terms of distance from the original mesh, they often cannot preserve the most visually salient features of the model. Zhang *et al*. (2012) presented a way of taking into account relevant features by using the Laplacian operator as a measure of saliency, penalizing contractions in the most feature-rich areas. Moreover, Li *et al*. (2010) proposed a feature-preserving simplification algorithm based on an absolute curvature weighted quadric error metric, although the choice of using half-edge collapses could lead to suboptimal vertex positions (Garland and Heckbert, 1997). Another simpler approach can be found in Yao *et al*. (2015) where the quadratic error metrics is modified through an additive term based on a custom-made curvature estimate. One of the main drawbacks of this approach, other than using a suboptimal curvature estimation method (Surazhsky *et al*., 2003), is that the additive term becomes quickly negligible as the simplification progresses.

Among other attempts at providing a reliable method to detect perceptually salient features of the

mesh, Watanabe and Belyaev (2001) proposed a curvature-based ridge finding algorithm and Lee *et al*. (2005) proposed a center-surround mechanism to detect important features based on curvature variation on different scales; in both paper the estimated importance is then used as a weighting on the standard quadric error metrics. More recently Song *et al*. (2014) proposed a saliency extraction technique by making use of spectral analysis.

In this study, we present a mesh simplification algorithm which makes use of the quadric error metric for optimal vertex positioning and that accounts for discrete curvature in the model, computed through the Gauss-Bonnet scheme, to simplify more aggressively flat and rounded regions while maintaining a high triangle count in the zones where there are details and key features (spikes, sharp edges). The method proposed for determining the most visually salient areas has the goal to be, other than simple to implement, extremely fast to execute in comparison to the core simplification routine, which could be a competitive advantage over more complex methods such as the one proposed by Lee *et al*. (2005).

The section Algorithm presentation firstly summarizes the edge contraction technique as well as the quadric error metric utilization. Secondly, a simple mesh saliency estimation method is described, as well as its integration into the simplification process. Finally pseudocode and a complexity analysis outline are given. In the section Results and discussion the practical results of this method are exposed through a series of examples on benchmark models.

Algorithm Presentation

The STL Format

This paper is focused on the STL file format, which is one of the simplest and yet more widespread ones to represent meshes in 3D modelling and computer graphics. The surface of an object is represented, with obvious limits in resolution, as a list of triangular facets each written in the following form (Szilvśi-Nagy and Matyasi, 2003):

$$\begin{pmatrix} n_x & n_y & n_z \\ v_x^1 & v_y^1 & v_z^1 \\ v_x^2 & v_y^2 & v_z^2 \\ v_x^3 & v_y^3 & v_z^3 \end{pmatrix}$$

where, n is the normal of the facet represented as the position of its endpoint and v_i are the vertices. The STL file does not store any topological information about the way the triangles form the actual object. There are thus no data structures about adjacency relation nor any way to understand on-the-go whether a file represents a valid mesh or not. A big drawback of this implementation is that each vertex is stored one time for every triangle it is part of, causing a severe redundancy in the information stored in the file. Due to this organization of the file, it is necessary to traverse the entire mesh at the beginning of the simplification process and store information about the topology of the model. The algorithm keeps track of the triangles and the edges adjacent to each vertex in dedicated data structures.

Edge Contraction

An edge contraction, which can be written as $(v_1, v_2) \rightarrow \hat{v}$, is the process of joining v_1 and v_2 by removing them, connecting all their incident edges to a new vertex $\hat{v}$ and then removing all degenerate faces created in the process, as shown in Fig. 1. If the mesh is manifold and closed, which means that the infinitesimal neighborhood of every point is topologically equivalent to a disk, the faces removed at each step are two. Note that in the original paper of Garland and Heckbert (1997) the approach used is slightly different, in fact, they contract pairs of vertices even if there is no edge between them. While this method is effective in simplifying small disjoint models, it can produce non-manifold meshes which, especially for additive manufacturing, can lead to serious problems during the printing phase; for this reason, we decided to enable only edge contractions. The quality of the decimation algorithm lies in the solution adopted to solve two main problems: How to place the new vertex $\hat{v}$ and in which order to process the edges.

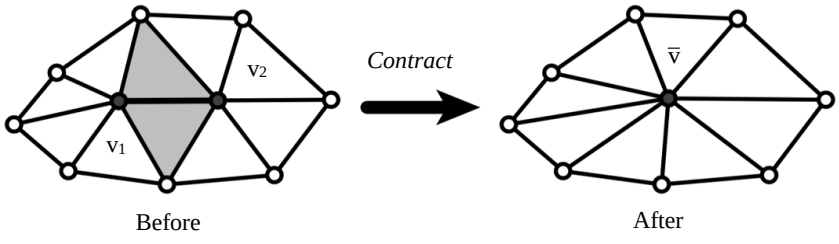


Fig. 1: The contraction of an edge

DOI: 10.3844/jcssp.2020.1195.1202

Error Minimization

To solve the problems mentioned above, as presented in (Garland and Heckbert, 1997), the error at a vertex $v = (x_v, y_v, z_v, 1)^\top$ has to be properly defined. Let Γ_v be a set of planes, each defined by the equation $ax + by + cz\text{-}d = 0$, with $a^2 + b^2 + c^2 = 1$. Each plane is then represented as $p = (a, b, c, d)^\top$. Recall that the distance from the point v to the plane p is $p^\top v$. The error at vertex v is then defined as the sum of squared distances of the vertex from the planes:

$$\Delta(v) = \sum_{p \in \Gamma_v} \left(p^\top v\right)^2 \tag{1}$$

Rearranging the expression above, the following is obtained:

$$\begin{aligned}\Delta(v) &= \sum_{p \in \Gamma_v} \left(v^\top p\right)\left(p^\top v\right) \\ &= \sum_{p \in \Gamma_v} v^\top \left(pp^\top\right) v \\ &= v^\top \left(\sum_{p \in \Gamma_v} F_p\right) v\end{aligned} \tag{2}$$

with, $F_p = \begin{pmatrix} a^2 & ab & ac & ad \\ ba & b^2 & bc & bd \\ ca & cb & c^2 & cd \\ da & db & dc & d^2 \end{pmatrix}$. Let then be:

$$Q_v = \sum_{p \in \Gamma_v} F_p = \begin{pmatrix} q_{00} & q_{01} & q_{02} & q_{03} \\ q_{10} & q_{11} & q_{12} & q_{13} \\ q_{20} & q_{21} & q_{22} & q_{23} \\ q_{30} & q_{31} & q_{32} & q_{33} \end{pmatrix}$$

The error at vertex v is thus the quadratic form $v^\top Q_v$. In the starting mesh, for each vertex, Γ_p is the set of planes adjacent to it. Note that the initial error at each vertex is 0 for it lies on its adjacent planes. In the contraction $(v_1, v_2) \rightarrow \hat{v}$ the set of planes of the resulting vertex is the union of the two starting ones. This can be approximated by using the rule that $Q_{\hat{v}} = Q_{v_1} + Q_{v_2}$, even if this means double-counting some of the planes. During each contraction, the algorithm places $\hat{v}$ so that it minimizes the error of the resulting vertex, ans the resulting mesh will be as close as possible to the initial one. Being the error a quadratic form, the optimal position for $\hat{v}$ can be easily found by solving a system of partial derivatives of Δ with respect to x, y and z; as stated in Garland and Heckbert (1997) solving this system is equivalent to solving:

$$\begin{pmatrix} q_{00} & q_{01} & q_{02} & q_{03} \\ q_{10} & q_{11} & q_{12} & q_{13} \\ q_{20} & q_{21} & q_{22} & q_{23} \\ 0 & 0 & 0 & 1 \end{pmatrix} v = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{3}$$

$$v = \begin{pmatrix} q_{00} & q_{01} & q_{02} & q_{03} \\ q_{10} & q_{11} & q_{12} & q_{13} \\ q_{20} & q_{21} & q_{22} & q_{23} \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{4}$$

If the matrix isn't invertible one can simply search for the minimum along the edge. This is thus a reliable method for choosing the position of the new vertex. Note that the error of the new vertex is also a good measure of the cost of an edge contraction, the algorithm will then process edges in increasing cost order.

Mesh Saliency Estimation

While the error minimization technique presented in Garland and Heckbert (1997) yields extremely good results in uniform surfaces, it has no way of detecting the presence of detail in the mesh, thus some of the features of the model could be washed away in the process. By taking into account the Gaussian curvature of the surface, the algorithm can penalize edges with larger absolute values of Gaussian curvature, increasing their cost so that they are processed later than edges in flatter and less detailed regions. The resulting mesh will have larger and coarser triangles where the surface is smooth, while finer triangles where more detail is needed.

Note that being the mesh a non-C^2-differentiable surface, the curvature is not defined. The Gaussian curvature K in a vertex can be approximated using the Gauss-Bonnet theorem as described in Surazhsky *et al.* (2003) and Szilvśi-Nagy and Matyasi (2003). Let θ_i be the angle of the i-th triangle in the immediate neighborhood of the vertex of interest. Then the Gauss-Bonnet theorem states that:

$$\iint_A K dA = 2\pi - \sum_{i=0}^{n-1} \theta_i \tag{5}$$

Assuming that K is constant in the local neighborhood of the vertex, which can be approximated as the Voronoi area or the barycentric area around it (Fig. 2), one can solve for K:

$$K = \frac{2\pi - \sum_{i=0}^{n-1} \theta_i}{\frac{1}{3}\sum_{i=0}^{n-1} A_i} \tag{6}$$

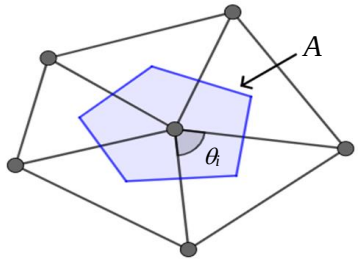


Fig. 2: The neighborhood of the vertex

As stated in Surazhsky *et al*. (2003), this estimation is one of the best among the various approximation algorithms. This approach offers at the same time a precise approximation of the Gaussian curvature of the mesh as well a very simple way to implement the curvature penalization, in fact, the curvature at each vertex can be calculated very efficiently through a series of sums of angles and areas.

Let $K_{(v_1,v_2)} = |K_{v_1}| + |K_{v_2}|$. Once the curvature of each vertex has been pre-calculated, one can scale the cost of each edge contraction through some function as so:

$$\Delta'_{(v_1,v_2)\to\hat{v}} = f\left(K_{(v_1,v_2)}, \Delta_{(v_1,v_2)\to\hat{v}}\right) \tag{7}$$

Note that K might vary very harshly, so it is reasonable to map it to [0, 1]. We found that good results can be obtained through the following function, where α is a parameter to tune the effect of curvature:

$$\Delta'_{(v_1,v_2)\to\hat{v}} = \Delta_{(v_1,v_2)\to\hat{v}}\left(1 - e^{-\alpha K}\right) \tag{8}$$

Figure 3 shows the plots of our estimation for the remapped curvature on two models often used in the literature in mesh processing.

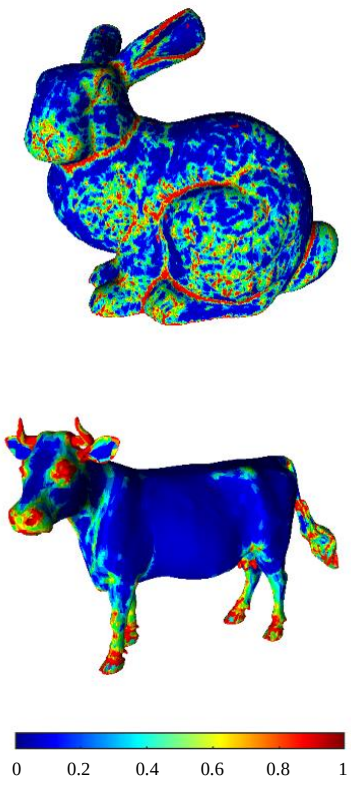


Fig. 3: Curvature estimation on two models

Pseudocode

The algorithm can be summarized as follows:

1. Compute Q and K for every vertex
2. Compute the best contraction vertex $\hat{v}$ for every edge and enqueue the pair using the resulting cost
3. Poll an edge from the queue and get its adjacent triangles and edges
4. Replace the edge with the best possible vertex, updating its matrix as well as the adjacent triangles and edges
5. Return to (3) until the desired size of the mesh is reached

```
Simplify-Mesh(m)
1   for TRIANGLE t in m
2       for VERTEX v in t
3           Q_v = Q_v + t.F_p
4           K_v = curvature at v
5   edgeQueue = PRIORITYQUEUE keyed on edge error
6   for TRIANGLE t in m
7       for EDGE e in t
8           v̂ = vertex that minimizes error
9           add e in edgeQueue keyed on the error of v̂
10  while m.size() > TARGET SIZE
11      e = edgeQueue.pollFirst()
12      SET ad jTrgs =
            {triangles adjacent to the vertices of the edge}
13      SET ad jEdges =
            {edges adjacent to the vertices of the edge}
14      v̂ = vertex that minimizes error
15      Q_v̂ = Q_e_v1 + Q_e_v2 // sum of the parents' matrices
16      for TRIANGLE t in ad jTrgs
17          update t according to the contraction
18          update the curvatures of t's vertices
19          if t isDegenerate()
20              remove t from the mesh
21          else
22              add the modified edges to edgeQueue
23      for EDGE edg in ad jEdges
24          remove edg from edgeQueue
25  return m
```

Algorithm Analysis Outline

To analyze the running time complexity of the above algorithm, the following assumptions are needed. Being all matrices 4×4, operations such as sums and inversions can be considered as $O(1)$; indeed many of such operations can be precomputed for actual constant-time performance. Moreover, the sets used to

DOI: 10.3844/jcssp.2020.1195.1202

store information about the vertices can be implemented as hash sets with custom hash code functions to get $O(1)$ operations.

Let T be the number of triangles in the starting model. Being the mesh manifold, it's easy to see that the number of edges and vertices are $O(T)$ Lines 1-4 contribute for $O(T)$ operations, while lines 5-9, having $O(T)$ insertions in the priority queue, which take $O(logT)$ time each, contribute for a total of $O(TlogT)$ operations. There are n iterations of the while cycle. Considering that at each pass two triangles are removed, it holds that $n \in O(T)$. Lines 11-24 contribute for $O(logT)$ amortized operations due to the extraction and the insertions into the queue, as the operations on the elements of the sets ad *jTrg* and ad *jEdges* add up to $O(T)$ over the n iterations.

This leads to an overall complexity of $O(TlogT)$.

Results and Discussion

Various tests were performed on a wide range of parameters, executing our mesh simplification routine on models widely used in the literature. A comparison between the models simplified with and without the curvature-based penalization follows.

In Fig. 4a the results of a compression from 32 to 6.4 k triangles with α set to 150 on the left and to +inf (standard QEM error function) on the right are shown. Note in Fig. 5 that while in the smoother zones of the model the triangles are coarser, in the more detailed ones such as the eyes and nostrils there are finer triangles that lead to much greater detail in comparison to the standard method. A similar result can be obtained in the compression down to 1.6 k triangles (Fig. 4b).

Similar results are obtained on other meshes, such as the models in Fig. 6 and 7. Note that the back of the bunny presents larger triangles while the ears and the face present densification of vertices, increasing the level of detail maintained after the simplification; similarly, the smoother zones of the heart model present coarser triangles while the veins maintain a higher triangle count.

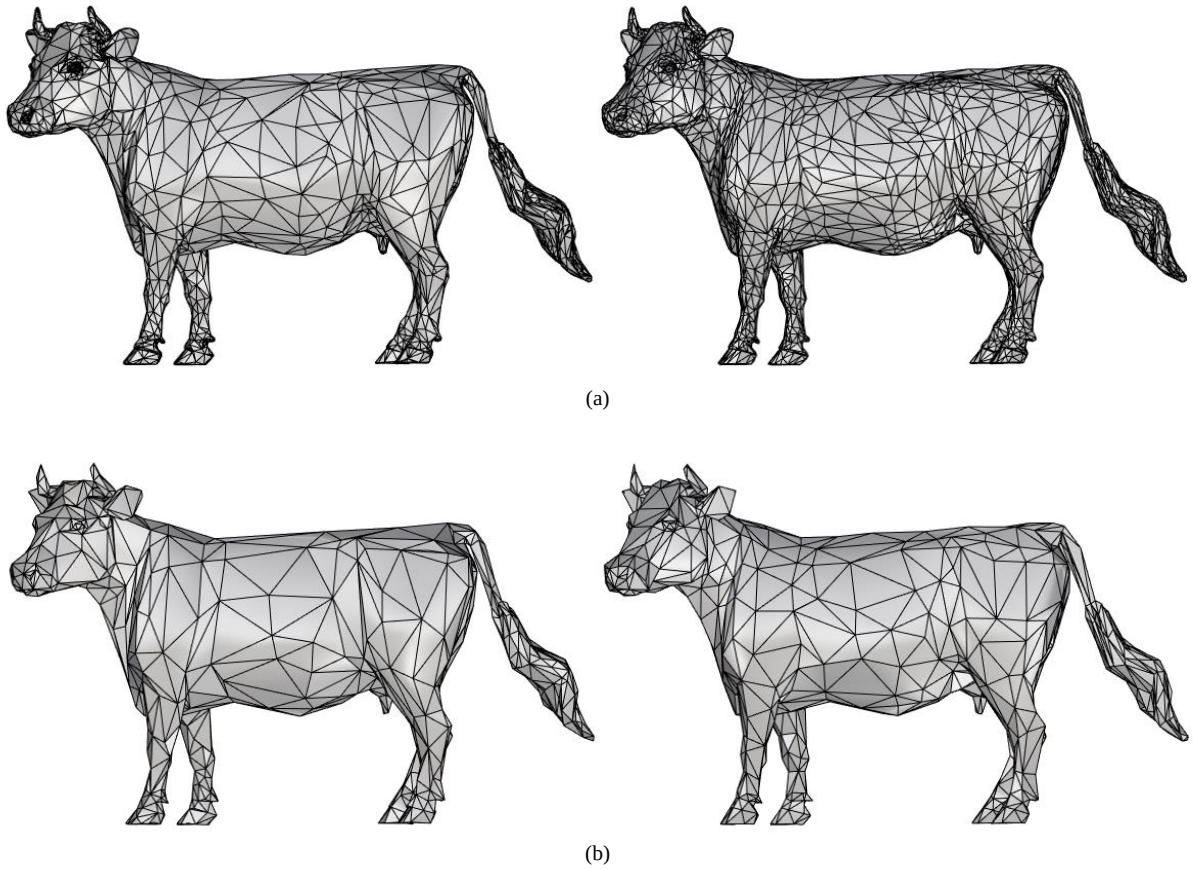
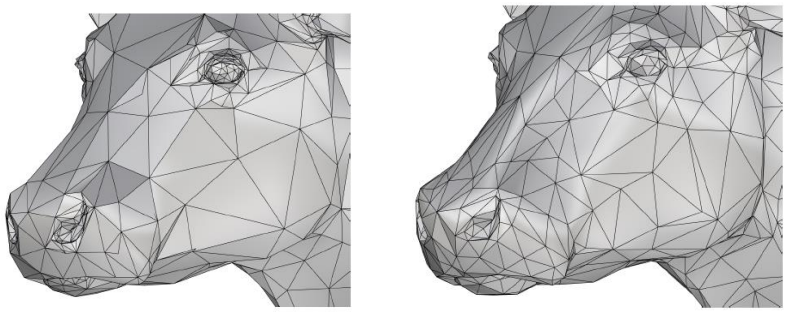


Fig. 4: Results with curvature penalization enabled (left) or disabled (right); (a) Model reduced down to 6.4 k triangles; (b) Model reduced down to 1.6 k triangles



(a) (b)

Fig. 5: A detail of the models with 6.4 k triangles; (a) Curvature penalization enabled; (b) Curvature penalization disabled

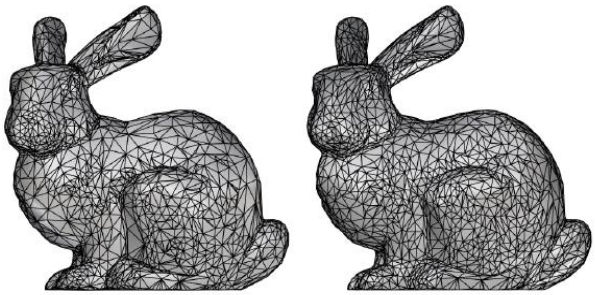


Fig. 6: Models reduced down to 5 k triangles. Curvature penalization enabled (left) or disabled (right)

Table 1: Mesh-Mesh deviation comparison

Model	RMS error (std. method)	RMS error (curv. penal.)
Cow	0.0530	0.0508
Cow (Head)	0.0514	0.0473
Heart	0.0349	0.0339
Heart (Veins)	0.0325	0.0309

Table 1 shows, comparing our method to the standard one, the root mean squared error of the simplified models (20% of original triangles), which were computed in Rhinoceros 6 by "Mesh-Mesh Deviation" tool of the "Rhino Open Projects" (Savio *et al*., 2013), which computes the distance between the surface of the original mesh and the simplified one[1]. It must be noted that the metric itself has no way of detecting the presence of detail, thus heavily penalizing the coarser triangles that are created in the smoother zones of the model. In order to account for the quality of the simplification of salient areas, the tests were run, other than on the whole model, on selected parts of the mesh, namely on the head for the cow model and on the veins for the heart model, which are also shown in Fig. 5 and 7b. In general, using the curvature penalization method yields a smaller standard deviation between the meshes than the original method. The advantage, in terms of root mean squared error of the simplified model, of our algorithm over the standard QEM one becomes even larger when considering only the visually salient areas. Moreover, one of the strengths of our approach is that the user can decide to what extent the curvature will affect the overall result of the simplification; by tuning the parameter a, one can choose the result that best fits his needs.

From the tests it resulted that the overhead of estimating the curvature through Gauss-Bonnet, which is due mainly to the computation of the angles between the edges of the triangles and updating the values after each contraction, is minimal (Table 2). The tests were run on a Java implementation of the method proposed, using an AMD FX-8320 processor; it has to be noted that the code we produced is only for demonstration purposes and thus lacks any form of optimization.

Experimental results thus show that this algorithm is capable of preserving the smaller features of the mesh without impacting performance. This could be useful especially in areas of research such as medicine or archaeology, where the models returned by 3D scanning techniques must retain even the smaller features.

[1] Rhino Open Projects. https://www.food4rhino.com/app/rhino-openprojects. [Accessed: 07-21-2020]

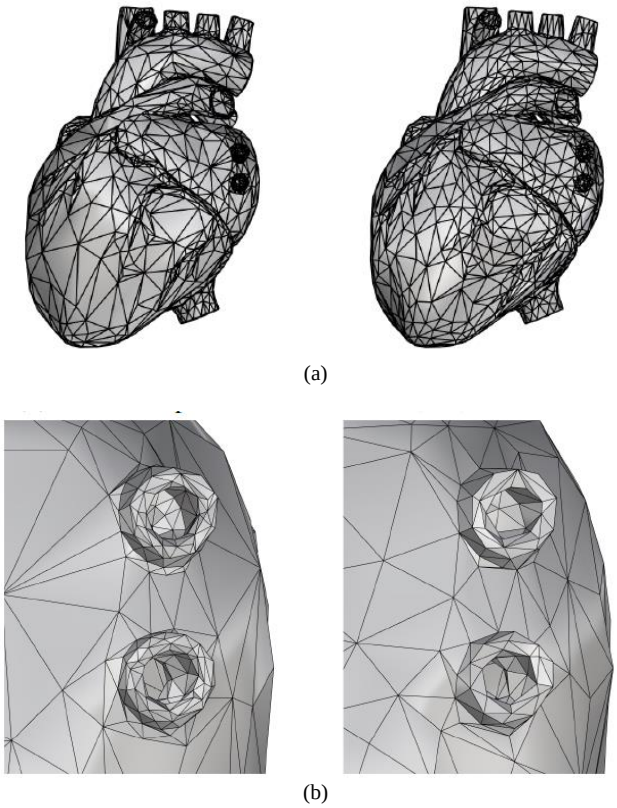


Fig. 7: Model of a human heart simplified from 85 to 17 k triangles; (a) Curvature penalization enabled (left) and disabled (right); (b) A detail of the veins

Table 2: Running time comparison

Model	Tot. CPU time	curv. Estimation CPU time
Bunny (250 k to 12k tr.)	15.97s	0.185s (1.2%)
Cow (32 k to 1.6k tr.)	1.51s	0.034s (2.2%)

Conclusion

In this study we presented a simple mesh saliency detection method based on discrete curvature estimation, which in turn is accomplished through the use of the Gauss-Bonnet theorem. This measure of visual importance is then embedded into the simplification routine to preserve the features of the mesh. The algorithm presented offers a good improvement over the original one, adding an alternative approach to the already existing ones. While the results obtained are encouraging, this algorithm has still room for improvement. Among the upgrades that are to be done in future work, we enumerate improving overall performance through the use of better data structures for keeping track of vertex neighboring triangles and edges, as well as not enqueuing edges that will be removed before being processed and implementing a better system to detect the formation of degenerate or non-manifold surfaces.

Author's Contributions

Paolo Pellizzoni: Conceptualization, Project administration, Investigation, Formal analysis, Writing, Review and Editing.

Gianpaolo Savio: Supervision, Investigation, Formal analysis, Writing, Review and Editing.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Garland, M., & Heckbert, P. S. (1997, August). Surface simplification using quadric error metrics. In Proceedings of the 24th annual conference on Computer graphics and interactive techniques (pp. 209-216).

Heckbert, P. S., & Garland, M. (1999). Optimal triangulation and quadric-based surface simplification. Computational Geometry, 14(1-3), 49-65.

Hua, Z., Huang, Z., & Li, J. (2015). Mesh simplification using vertex clustering based on principal curvature. International Journal of Multimedia and Ubiquitous Engineering, 10(9), 99-110.

Lee, C. H., Varshney, A., & Jacobs, D. W. (2005). Mesh saliency. In ACM SIGGRAPH 2005 Papers (pp. 659-666).

Li, L., He, M., & Wang, P. (2010, June). Mesh simplification algorithm based on absolute curvature-weighted quadric error metrics. In 2010 5th IEEE Conference on Industrial Electronics and Applications (pp. 399-403). IEEE.

Michaud, C., Mellado, N., & Paulin, M. (2017). Mesh simplification with curvature error metric.

Rossignac, J., & Borrel, P. (1993). Multi-resolution 3D approximations for rendering complex scenes. In Modeling in computer graphics (pp. 455-465). Springer, Berlin, Heidelberg.

Savio, G., Meneghello, R., & Concheri, G. (2013). Optical properties of spectacle lenses computed by surfaces differential quantities. Adv. Sci. Lett., 19(2), 595–600.

Schroeder, W., Zarge, J., & Lorensen, W. (1997). Decimation of triangle meshes. SIGGRAPH Comput. Graph., 26, 65–70.

Song, R., Liu, Y., Martin, R. R., & Rosin, P. L. (2014). Mesh saliency via spectral processing. ACM Transactions On Graphics (TOG), 33(1), 1-17.

Surazhsky, T., Magid, E., Soldea, O., Elber, G., & Rivlin, E. (2003, September). A comparison of gaussian and mean curvatures estimation methods on triangular meshes. In 2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422) (Vol. 1, pp. 1021-1026). IEEE.

Szilvśi-Nagy, M., & Matyasi, G. Y. (2003). Analysis of STL files. Mathematical and computer modelling, 38(7-9), 945-960.

Watanabe, K., & Belyaev, A. G. (2001, September). Detection of salient curvature features on polygonal surfaces. In Computer Graphics Forum (Vol. 20, No. 3, pp. 385-392). Oxford, UK and Boston, USA: Blackwell Publishers Ltd.

Yao, L., Huang, S., Xu, H., & Li, P. (2015). Quadratic error metric mesh simplification algorithm based on discrete curvature. Mathematical Problems in Engineering, 2015.

Zhang, L., Ma, Z., Zhou, Z., & Wu, W. (2012, December). Laplacian-Based feature preserving mesh simplification. In Pacific-Rim Conference on Multimedia (pp. 378-389). Springer, Berlin, Heidelberg.