Proximity-Aware Multiple Meshes Decimation using Quadric Error Metric

Anahid Ghazanfarpour, Nicolas Mellado, Chems-Eddine Himeur, Loic Barthe, Jean Pierre Jessel

Proximity-Aware Multiple Meshes Decimation using Quadric Error Metric

Anahid Ghazanfarpour, Nicolas Mellado, Chems Himeur, Loïc Barthe, Jean-Pierre Jessel

Proximity-Aware Multiple Meshes Decimation using Quadric Error Metric

Anahid Ghazanfarpour, Nicolas Mellado, Chems E. Himeur, Loïc Barthe, Jean-Pierre Jessel

ARTICLE INFO

Article history:
Received 21 June 2019
Received in revised form
30 October 2019

Keywords: Mesh decimation, Quadric error metric, Geometry processing, Virtual disassembly

ABSTRACT

Progressive mesh decimation by successive edge collapses is a standard tool in geometry processing. A key element of such algorithms is the error metric, which prioritizes the edge collapses to greedily minimize the simplification error. Most previous works focus on preserving local shape properties. However, meshes describing complex systems often require significant decimation for fast transmission and visualization on low-end terminals, and preserving the arrangement of objects is required to maintain the overall system readability for applications such as on-site repair, inspection, training, serious games, etc.

We present a novel approach for the joint decimation of multiple triangular meshes. We combine local edge error (e.g. Quadric Error Metric) with a proximity-aware penalty function, which increases the error of edge collapses modifying the geometry in proximity areas. We propose an automatic detection of proximity areas and we demonstrate the performances of our approach on several models generated from CAD scenes.

1. Introduction

It is today a common practice to gather large and detailed 3D meshes to represent geometrical information. Visualizing and interacting with such data remains very challenging, as the data complexity and scale stress both hardware and software components in real-life applications. This becomes even more critical with mobile platforms and web-embedded 3D visualization, which require fast 3D data transfer and rendering even on low end terminals. Despite their limited performances, mobile platforms are very attractive as they bring access to rich information on-site, and help users performing tasks in complex environments (e.g. machinery maintenance). The relatively low cost and ubiquity of mobile devices also make them very attractive as support for teaching and training.

To mitigate the increasing complexity of 3D datasets for low-performance hardware, a common solution is to reduce the polygon count of large 3D meshes using decimation algorithms [1]. In order to keep the decimation algorithms tractable, the preservation of the mesh properties (e.g. parameterization and shape) is generally ensured by a local prior, evaluated at each decimation step, using a point-to-plane distance minimization for collapsing edges [2]. Several distance measures have been proposed for this framework and the most popular is the Quadric Error Metric (QEM) [3].

Meshes representing man-made or Computer Aided Design (CAD) scenes are often massive models composed of many parts encoding both the geometry and the functional meaning of the scene. Preserving this meaning (or semantics) in addition to the geometry is mandatory to still understand the functioning of a system (for instance mechanical), even though meshes have been highly simplified to be manipulated on low-end remote terminals. If provided as input, this information can be used to simplify an object by feature importance [4]. But without any prior, the mesh decimation of multi-part scenes relies only on the geometry, and preserving both the individual parts properties and their relations is very challenging.

In this work, we focus on the decimation of 3D scenes composed of multiple objects (as illustrated in Figure 1). We target applications where 3D models are transferred to low-end mobile devices for visualization and manipulation. Simplify-

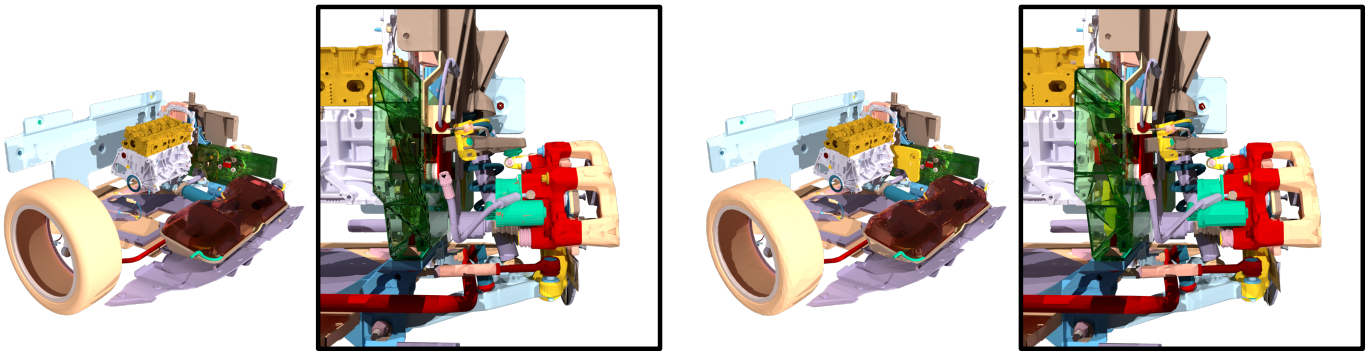


Fig. 1: Left: `Car` scene with 425 meshes and 3M faces in total. Right: Result of our proximity-aware decimation to 150k faces in total. Some meshes are rendered with transparent material to better observe the scene complexity.

ing meshes to low polygon counts is then mandatory to reduce the time required to transfer and render the 3D models. Our approach also targets applications where the 3D scene requires virtual disassembly or/and manipulation with a mechanical/functional meaning to preserve, e.g. for on-site repair, inspection, training, serious games, etc.

Our solution is based on the observation that in mechanical systems, the shape of an object is designed according to its functionality and its interactions with the surroundings (e.g. contacts and arrangement). As such, we propose to take into account the neighboring meshes in the decimation of an object. We formulate the neighboring information as a proximity error mitigating the importance of geometric structures according to an object surrounding. This proximity error is then used to penalize the error introduced by decimation operations in proximity areas, and thus reduce their priority, which yields to delay their simplification and better preserve the object shape (see Figure 2).

Our contribution is twofold. First, we introduce a proximity-aware error metric by inserting a proximity penalty function into the Quadric Error Metric (Section 5). Second, we parameterize this new error metric with a proximity analysis of the input scene (Section 6). In Section 7, we illustrate the benefits of our approach and how it better preserves the input shapes in nearby parts of CAD scenes, in comparison with standard decimation using QEM.

2. Previous Work

Over the years, several families of approaches have been proposed for simplifying meshes. We first review simplification techniques for single meshes, and then for scenes composed of multiple meshes.

2.1. Mesh Simplification

Mesh simplification is the process of *modifying* the tessellation of an input mesh in order to reduce its polygon count, by clustering vertices, collapsing edges or faces.

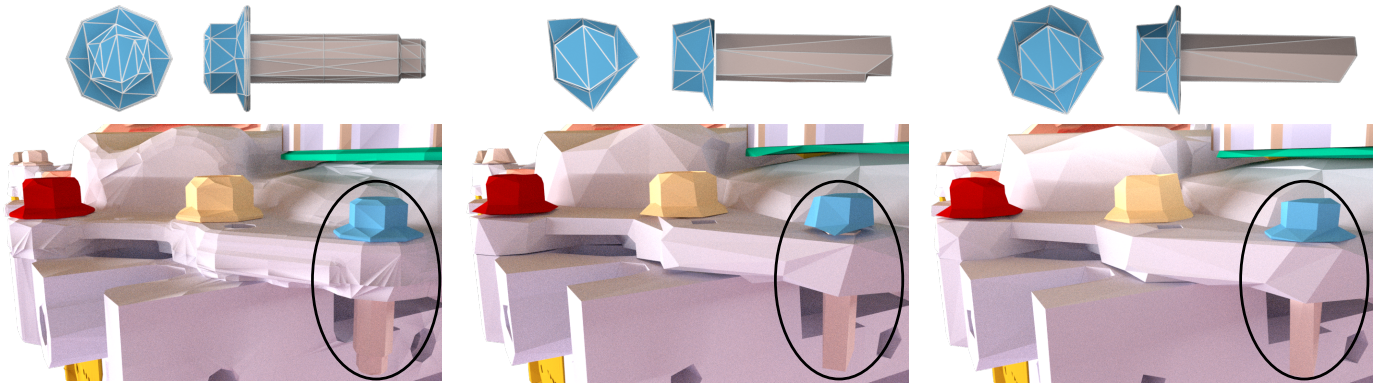
Clustering techniques group vertices and replace them by smaller sets. Rossignac and Borrel [5] group vertices according to the regular subdivision of the bounding volume of the mesh. Low and Tan [6] extend this idea to arbitrary shapes using voxel grids. Following these ideas, Boubekeur and Alexa [7] use stochastic vertex selection based on a local feature estimator, combined with triangle re-indexing to better preserve areas of high curvature. Overall, it remains difficult to localize decimation and control the granularity of the simplification with this family of approaches.

Mesh decimation has been introduced by Schroeder et al. [8]. In this approach, mesh vertices are removed sequentially according to a given decimation criterion, and the resulting hole is filled with new faces. Turk [9] proposes a similar technique where new vertices sampling the input surface are introduced to generate the local tesselation. It is then extended by Ciampalini et al. [10] with a global error criterion to avoid error accumulation during the consecutive decimation steps.

Hoppe [2] introduces *progressive meshes*, an iterative method that progressively simplifies the mesh using the local edge collapse operator. A standard implementation of edge collapse simplification consists in sorting the mesh edges by increasing error in a priority queue. At each step, the edge with the smallest error value is collapsed, and the errors of the edges impacted by the simplification are recomputed. In order to avoid error accumulation during this process, several approaches have been proposed to restrict the set of collapses to a prescribed tolerance volume [11, 12, 13, 14, 15].

Several error metrics can be used to decimate a mesh, such as the Hausdorff distance between the simplified and the original mesh [16]. Lindstrom and Turk [17] focus on mesh volume preservation and define their error metric as the sum of squared tetrahedral volumes formed by the vertex and its neighbor faces. The most popular and widespread method is the Quadric Error Metric (QEM) [3], which computes the sum of the squared distances from the newly inserted vertex to the set of planes spanning the collapsed edge neighbor faces. The strength of the QEM is to model the sum of squared distances as a quadric per edge, which is minimized to find the optimal vertex position for each collapse.

The QEM can be extended to preserve mesh attributes or properties, such as curvature [18, 19], geometric features [20] or local topology [21]. Alternatively, user-defined weighting functions might be applied on the mesh to delay collapses in specific regions of the mesh [22, 23, 24]. Note that Kho and Garland [22] also propose to combine local quadrics with user-defined quadrics modelling contour lines on the mesh. More recently, Salinas et al. [25] extended this idea with quadrics



(a) Input scene (b) Standard QEM (c) Proximity-Aware Error Metric

Fig. 2: (a) Close-up on the `Car` scene shown in Fig. 1 (425 meshes and a total of 3M faces). (b) Its edge collapse-based simplification with Quadric Error Metric. (c) Our approach better preserves the shape of 3D meshes that are close to other meshes in the scene, as for instance illustrated by the hex washer bolts head and the preservation of the contact surface with its support. Both simplified scenes have 150k faces each.

computed from proxy planes and line boundaries, in order to describe the high-level structure of the mesh.

A shared limitation of the aforementioned algorithms is to rely on sequential processing, which is greedy both in memory and computational power. Very large to huge meshes may thus be processed in parallel [26], out-of-core [27] or by combining the two approaches [28].

While different mesh properties and attributes may be preserved by the simplification, none of the previous methods takes particular care of the proximity between different parts.

2.2. Scene Simplification

In most CAD representations, geometrical primitives and their arrangement define high-level information that can be used to support the simplification process [4]. Based on the scene graph, Erikson et al. [29] propose to build hierarchical levels of detail by grouping nodes w.r.t. the scene graph and the meshes spatial arrangement, while the geometry is optimized using standard mesh decimation algorithms. In practice, however, complex scenes are often provided as an unstructured set of meshes and we are instead seeking a geometric-only multiple meshes simplification.

The joint decimation of multiple meshes has also been studied. Gumhold et al. [30] focus on avoiding collisions between close-by meshes during edge collapses. Each time an edge contraction generates a collision, a new intersection-free position is computed, and the collapse operation is re-inserted in the queue with the associated new error value. The main limitation of this approach is to only consider collisions, and ignore collapses that might affect the geometry of nearby meshes, e.g. by generating holes, cracks and removing small geometrical features. González et al. [31] propose a user-assisted method to simplify sub-objects with different levels of detail while preserving boundaries. To do so, vertices nearby other meshes in the scene are marked as boundary and preserved by performing halfedge collapses. The limitations of this approach are twofold: firstly, the approximation error in boundary areas is checked only along the edges, and not on the faces. Secondly, the edges that have both end-points in boundary areas are treated as standard edges, which prevents from penalizing shape variations of nearby surfaces inside boundary areas.

3. Technical Background

In this section, we remind the standard computation of Quadric Error Metrics (QEM) [3] and the main principles of incremental edge collapse decimation [2].

Quadric Error Metric. For each face f of an input mesh, a quadric Q_f is defined as a 4x4 matrix $Q_f = pp^T$, where $p = (n_x, n_y, n_z, d)^T$ defines the plane spanning the face f, and $\Delta_{qem}(\boldsymbol{x}) = \boldsymbol{x}^T Q_f \boldsymbol{x}$ is the point $\boldsymbol{x}$ to plane p squared distance. The distance between $\boldsymbol{x}$ and several planes is computed using the sum of the planes quadrics instead of Q_f in Δ_{qem}. Garland and Heckbert [3] showed that finding the point minimizing this point-to-planes distance boils down to the minimization of a quadratic polynomial function, which can be done efficiently in closed form.

In order to compute the optimal point collapsing an edge, i.e. the point minimizing Δ_{qem} for the set of planes in the collapsed edge neighborhood, we use in Δ_{qem} the quadric associated with the edge. This quadric is defined as $Q_e = \frac{1}{2}(Q_{\boldsymbol{v}_1} + Q_{\boldsymbol{v}_2})$, where $Q_{\boldsymbol{v}_1}$ and $Q_{\boldsymbol{v}_2}$ are the edge vertex extremities quadrics defined as:

$$Q_v = \frac{\sum_{f_i \in \tau(\boldsymbol{v})} w(f_i)\, Q_{f_i}}{\sum_{f_i \in \tau(\boldsymbol{v})} w(f_i)},$$

where $\tau(\boldsymbol{v})$ is the set of faces in the one-ring neighborhood of $\boldsymbol{v}$. Any weighting scheme w might be used for the faces, e.g. cotangent weights.

Edge collapse simplification. For each edge e of an input mesh, the error value is computed using QEM and sorted in a priority queue by increasing error. Then, the decimation is performed by iteratively collapsing the edge with the smallest error, i.e. the one located on the top of the priority queue. After each collapse operation, the error value of the edges modified by the collapse operation and their position in the priority queue are

updated. Hence, for each collapse, the complexity of updating the priority queue is up to $O(2k\log(E))$, where k is the number of edges to update (remove and insert) in a queue of E elements.

4. Overview

Our approach takes as input a set of N triangular meshes $M_i, i \in [0\,..\,N[$, and outputs a set of decimated meshes M_i^d. We denote $\boldsymbol{v}_{i,j}$, $e_{i,j}$ and $f_{i,j}$ the j^{th} vertex, edge and face of a mesh M_i respectively. The decimation is performed by edge collapse operations that are interleaved by increasing simplification error in a single priority queue for all meshes.

Proximity-Aware Error Metric. We first introduce a new *collapse error computation*, evaluated at the collapsing vertex obtained by standard QEM (see Section 5). The key idea of our approach is to combine the edge quadric with the quadrics of the faces surrounding the edge on nearby meshes. With our approach, we increase the collapse error that changes the geometry where multiple meshes are close. These collapses are pushed back in the priority queue, so that the face budget is automatically rebalanced between the different meshes. We emphasize that, by affecting only the error computation but not the placement of the collapsing vertex, we take benefit of the robustness of QEM for placing the collapsing vertices.

Proximity Analysis. We finally present in Section 6 a generic approach to detect proximity areas on meshes. We analyze the spatial arrangement of the input meshes and their distances, from which we derive parameter values for the automatic configuration of the proximity-aware error metric.

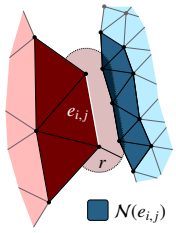
5. Proximity-Aware Decimation

In its initial formulation, the QEM represents the point-to-plane distance for all the faces surrounding a vertex or an edge. When decimating a scene with multiple meshes, the information stored in the quadrics is incomplete, as it is restricted to vertex or edge neighborhood in the topological sense, i.e. restricted to a single mesh. Several approaches may be considered to account for non-local properties of the mesh:

- Blocking collapses in proximity areas, which would lead to an undesired oversimplification outside the proximity areas for a given face budget.
- Increasing the weights of collapse operations in proximity areas, however, that would prevent some collapse operations from being performed even though they do not change the geometry, e.g. two nearly parallel and close planes with dense tessellation.
- Modifying the quadric minimized during the decimation operation by incorporating the face quadrics from the meshes that are surrounding the collapsed edge. This approach has already been used to decimate a single mesh with small holes [3] by creating virtual edges between close vertices. In our case, this would tend to move collapsing points toward the surrounding meshes, and thus to change the geometry.

In this work, we propose a new weighting strategy to delay the collapse operations in proximity areas. We emphasize that we keep the collapsing point given by QEM since we do not aim at changing the collapses but rather at reordering them considering proximity. To do so, we evaluate a proximity quadric computed from the surrounding meshes at the location of the collapsing point, and incorporate it into a proximity-aware error metric used to sort collapses in the priority queue.

Proximity quadric. In order to penalize collapses affecting edges that are close to other meshes, we use a proximity penalty function to increase their cost. For a given edge $e_{i,j}$, faces of other meshes located within a distance threshold r in the scene (computed as explained in Section 6) are considered in proximity.



Let $\mathcal{N}(e_{i,j})$ be the set of faces (in blue in the right inset) from close-by meshes of mesh M_i that fall in the euclidean neighborhood of radius r from the edge $e_{i,j}$. Note that we always consider the faces from the original meshes (i.e. not the decimated meshes) to compute $\mathcal{N}(e_{i,j})$, in order to avoid accumulating the decimation errors into the proximity quadrics. We denote $\hat{Q}_{i,j}$ the quadric combining the quadrics of the faces of $\mathcal{N}(e_{i,j})$ and define it as :

$$\hat{Q}_{i,j} = \frac{1}{\mathbf{card}(\mathcal{N}(e_{i,j}))} \sum_{f_{k,l} \in \mathcal{N}(e_{i,j})} w(e_{i,j}, f_{k,l}) * Q_{k,l} \qquad (1)$$

where $Q_{k,l}$ is the quadric associated with the face $f_{k,l}$ and w weights the contribution of each surrounding face w.r.t. the proximity threshold r, such that

$$w(e_{i,j}, f_{k,l}) = \phi(d(e_{i,j}, f_{k,l})),$$

where $d(e_{i,j}, f_{k,l})$ is the distance of the edge $e_{i,j}$ to the face $f_{k,l}$. In order to give more importance to close faces, we use the smooth polynomial kernel:

$$\phi(x) = \begin{cases} (1 - (\frac{x}{r})^3)^2 & \text{if } x \leq r, \\ 0 & \text{otherwise.} \end{cases} \qquad (2)$$

Proximity-aware error metric. We define our penalty function to be proportional to the deviation of the collapsing point $\boldsymbol{x}_{i,j}$ (obtained by minimizing the quadric error Δ_{qem}) from the edge neighborhood:

$$\Delta_{prox}(\boldsymbol{x}_{i,j}) = \boldsymbol{x}_{i,j}^T\, \hat{Q}_{i,j}\, \boldsymbol{x}_{i,j}.$$

Then, we define a proximity-aware error metric by combining this penalty function Δ_{prox} with Δ_{qem}. A naive solution is to add Δ_{prox} to Δ_{qem}, but this strategy is inefficient in our case as it prevents the simplification of flat areas with very low geometric error and high proximity error. The simplification of flat areas does not degrade the shape representation and should thus be simplified early in the decimation process, i.e. still be associated with a low error. Hence we propose to use Δ_{prox} as a penalty factor modulating the error computed with the QEM using Equation 3. We remind that, in this equation, $\boldsymbol{x}_{i,j}$ is the point

minimizing the second order equation of the standard QEM. The proximity-aware error $\Delta(\boldsymbol{x}_{i,j})$ is then computed with this position to sort the corresponding edge in the priority queue as follows:

$$\Delta(\boldsymbol{x}_{i,j}) = \Delta_{qem}(\boldsymbol{x}_{i,j})\ (1 + \alpha\Delta_{prox}(\boldsymbol{x}_{i,j})), \tag{3}$$

where α scales the proximity error so that penalized edges are adequately re-sorted in the priority queue. We estimate α so that the proximity-aware error Δ is in the same order of magnitude as a given large error e_{high} when (1) the QEM error is relatively low and (2) the penalty function Δ_{prox} is maximal. To do so, we set e_{high} as the error at 90% of the QEM error histogram, $\Delta_{qem} = e_{\frac{1}{4}}$ where $e_{\frac{1}{4}}$ is the QEM error of the first quartile, and $\Delta_{prox} = r^2$ as it is the square value of a distance bounded by r.

Using Equation 3, we directly obtain:

$$\alpha = \frac{1}{r^2}(\frac{e_{high}}{e_{\frac{1}{4}}} - 1).$$

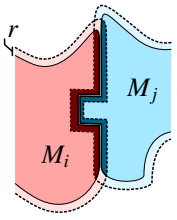
The values of e_{high} and $e_{\frac{1}{4}}$ have been first chosen intuitively to respectively represent a high error and a relatively low error. It is meant to avoid both a too high penalty that would prevent the simplification of edges even with a moderate proximity, and a too low penalty that would have an insignificant impact on the simplified meshes. These values have then been validated experimentally.

6. Proximity Analysis

In this section, we present the proximity distribution between two meshes (Section 6.1), how it is filtered to automatically compute a proximity threshold r (Section 6.2), and how it extends to N meshes (Section 6.3).

6.1. Proximity distribution between two meshes

Intuitively, two meshes M_i and M_j have a proximity relation when their faces are *close enough*. The goal of the proximity detection step is to analyze the scene and find r such that we can tag mesh faces that are considered in proximity, as illustrated in the right inset. To this mean, we build the distribution of the face-to-mesh distances between two input meshes, from which we extract robust distance thresholds using persistent homology. Our approach is based on the observation that nearby surfaces might generate peaks in the distance distribution function, as illustrated in Figure 3.



Let us denote $f_{i,k}$ a face of the mesh M_i, $c_j(f_{i,k})$ its closest face on the mesh M_j, and d the face-to-mesh distance, i.e. the euclidean distance between $f_{i,k}$ and $c_j(f_{i,k})$. We consider that $f_{i,k}$ is in a proximity area if the distance $d(f_{i,k}, c_j(f_{i,k}))$ is small enough and symmetric, i.e. $d(f_{i,k}, c_j(f_{i,k})) = d(c_j(f_{i,k}), c_i(c_j(f_{i,k})))$. In practice, strict symmetry is not always desirable as face-to-mesh distances might be affected by variation of tessellation across meshes. We thus rather measure $a(f_{i,k})$, the asymmetry between corresponding face-to-mesh distances as:

$$a(f_{i,k}) = \left|\ d(f_{i,k}, c_j(f_{i,k})) - d(c_j(f_{i,k}), c_i(c_j(f_{i,k})))\ \right|$$

with $|\cdot|$ the absolute value.

As we want to focus our analysis on mesh parts that are the most likely to be in proximity, we only consider faces with small asymmetry value. We thus start by computing $\hat{\mathcal{A}}(f_{i,k})$ as a function of the area $\mathcal{A}(f_{i,k})$ of mesh faces, weighted w.r.t. their asymmetry so that $\hat{\mathcal{A}}$ decreases when the asymmetry increases:

$$\hat{\mathcal{A}}(f_{i,k}) = \begin{cases} \mathcal{A}(f_{i,k}) * (1 - (\frac{a(f_{i,k})}{a_{\frac{1}{4}}})^2)^2 & \text{if } a \leq a_{\frac{1}{4}}, \\ 0 & \text{otherwise,} \end{cases}$$

where $a_{\frac{1}{4}}$ is the first quartile of asymmetry measured on the face-to-mesh distances between the two meshes. Intuitively, this value allows us to discard pairs of faces that will not contribute to the proximity computations. We have validated this value experimentally, and we notice that small changes in this value have a negligible effect on the simplification process.

We define $\mathcal{D}$ as the weighted distribution of the face distances between two meshes, where each distance sample is weighted by $\hat{\mathcal{A}}(f_{i,k})$. As the distance d is asymmetric by construction, we populate $\mathcal{D}$ with distances from M_i to M_j and from M_j to M_i.

6.2. Proximity distribution filtering

In the ideal case (Figure 3-a), all the faces in proximity areas are at the same distance d_{ideal}. This generates a clear peak in the distribution $\mathcal{D}$ and the proximity distance r can be easily set as $r = d_{ideal}$. In practice (Figure 3-b,c), geometric configurations are more ambiguous and the proximity threshold r is more complicated to set. We propose an automatic computation of r on which all our examples are based. This computation can also be easily edited if required by the user.

The proximity threshold r aims at classifying faces that are in proximity areas, such that their distance to other meshes is less or equal to r, as illustrated in Figure 3. When $\mathcal{D}$ is noisy, several proximity threshold candidates r_i might be considered. Intuitively, the candidates r_i split the input distribution $\mathcal{D}$ into groups of faces with consistent distances. In other words, we seek at grouping faces that share the same distance, e.g. local maxima in $\mathcal{D}$. Hence, a natural solution consists in extracting proximity threshold candidates as local minima of the input distribution.

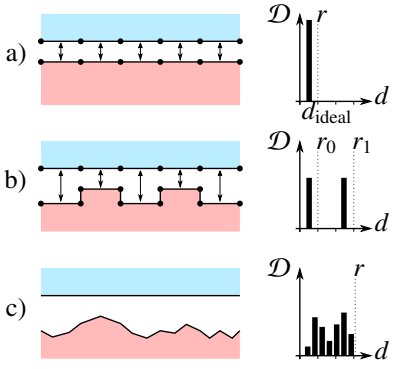


Fig. 3: Examples of distance distributions between two surfaces, ranging from simple (top) to more realistic (bottom) cases. r_i are examples of proximity threshold candidates.

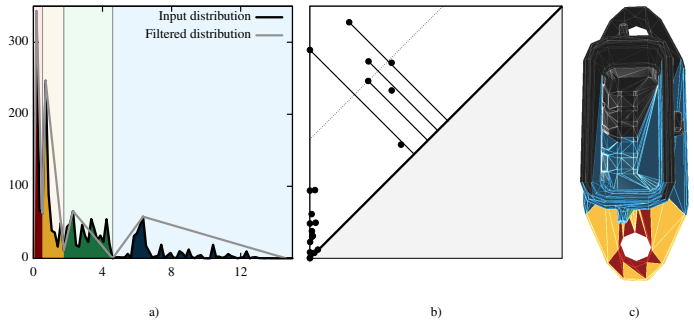


Fig. 4: The input distribution $\mathcal{D}$ (a-black curve) and its persistence diagram (b). The filtered distribution (a-gray curve) is constructed from the first 4 most persistent intervals. c) Input model from `Connector` scene, with faces colored w.r.t. candidate thresholds.

As visible in Figure 4-a), $\mathcal{D}$ is in practice too noisy for the direct extraction of its minima, and needs to be filtered. Smoothing the distribution might help to reduce the noise, however, it is not clear how much smoothing needs to be applied to not filter out the optimal, yet unknown, candidate r_i. Among all the existing approaches for signal analysis, we seek an approach that considers explicitly the shape of the distribution, and allows to select the number of groups required to explain the data. In this work, we considered Topological Data Analysis (TDA), and more specifically *persistent homology* (PH) [32], for its ability to consistently cluster a graph in a preset number of groups.

For the sake of completeness, we now briefly introduce the basics of PH for 1D curve processing. In a nutshell, PH sees the analyzed curve as a height field. For a given height, let us draw an imaginary horizontal line in the curve domain, and observe the segments of the line that are *above* the signal. When increasing the height of the line, some segments might appear (at local minimum), or existing segments might be merged (at local maximum) depending on the curve's shape. These events are *topological events*, that are tracked and represented in so-called *persistence diagrams*. As shown in Figure 4, a persistence diagram is a 2D graph containing a set of points. Each point represents one of the aforementioned line segments, with its x coordinate corresponding to the height where the segment appears, and y the height where it is merged with another one. The *persistence* of a segment is measured as the distance between its (x, y) coordinates and the line $f(x) = y$, as illustrated in Figure 4-b).

To simplify the curve representing the distribution $\mathcal{D}$, we select the m most prominent segments, i.e. those with the highest persistence value. For each segment, we know the local minimum and maximum associated respectively with its appearance and merge events. We construct the simplified distribution by linearly connecting the successive extrema corresponding to the m most prominent segments, and extract the proximity threshold candidates r_i as the local minima of this simplified curve. By construction, our approach is guaranteed to interpolate the local extrema of $\mathcal{D}$, and to generate m proximity threshold candidates r_i. As seen in Equation 2, r is used to weight faces using a smooth kernel. For a very narrow interval (e.g. in red in Figure 4-a), this smoothing might over smooth the influence of faces inside the interval. In order to get a conservative smoothing, we extend the interval so that the weight of any face inside the interval bounded by r_i is close to 1 to preserve the face influence, i.e. we compute r used in Equation 2 so that $\phi(r_i) = w_{\text{ref}}$ with w_{ref} equals a value close to 1, experimentally set to 0.9 in all our tests. Hence, for a chosen candidate r_i we compute the threshold r as:

$$r = \frac{r_i}{\sqrt[3]{1 - \sqrt[2]{w_{ref}}}}.$$

For all our experiments, we used the first most prominent local minima out of four ($m = 4$) extracted as proximity threshold candidates. This value of m is dependent on the objects shapes and their organization in the scene. While the choice of $m = 4$ is effective for all our experiments, it may not be optimal for all scenes. Thus, our system can output multiple suggestions visualized by a user (e.g. by coloring associated faces as seen in Figure 4-c). The user can then easily choose the appropriate number of prominent segments m and the local minima r_i setting the desired significant proximity.

6.3. Proximity between N meshes

Our approach naturally extends to N meshes by aggregating the pairwise distance distribution for the meshes in the scene. However, considering all pairs of meshes, especially those that are far away from each other, may lead to unwanted clusters and a relatively high distance might be computed as proximity threshold for the scene. To prevent this, we consider only pairs of meshes that have intersecting or close-by axis-aligned bounding boxes. We used a fixed epsilon value for all our experiments, set as 0.1% of the scene axis-aligned bounding box.

7. Results

We have evaluated our approach on four different scenes (see Table 1), by running our decimation algorithm on an Intel Xeon E5-2609 1.90GHz, 16BG of RAM. We implemented standard QEM [3], boundary-aware decimation [31] and our approach in C++ without optimization. All our timings are based on these homogeneously non-optimal implementations to provide fairer comparisons. We however point out that optimized QEM implementations are orders of magnitude faster as there is room for significantly optimizing our approach implementation. The incremental decimation is performed on a single thread, while the distance queries for the collapse error computation is parallelized using one thread per neighboring mesh. For each test scene, we estimated the proximity threshold r as described in Section 6 (with the four most prominent intervals) using the first cluster in all cases.

We compare our approach with standard QEM by deactivating our proximity error computation during the priority queue pre-computation and update routines. As QEM is not aware of proximity relations, we also implemented a naive weighting scheme where the collapse error is multiplied by a fixed value v in proximity areas ($\Delta(\boldsymbol{x}_{i,j}) = v * \Delta_{qem}(\boldsymbol{x}_{i,j})$). As finding a good weight is a non-obvious problem, we chose two extreme values: $v = \{2; 1000\}$. The former preserves the order of magnitude of the collapse errors, while the latter is very likely to

Scene	# objects	# input faces	# output faces	Prox. threshold (%)	Dec. time (s)	Method
`Tube`	2	5508	500	5.29	8	Proximity-Aware EM
(Fig. 5)					2	González et al. [31]
					2	Standard QEM
`Tube*`	2	11526	500	4.68	19	Proximity-Aware EM
(Fig. 6)					4	González et al. [31]
					3	Standard QEM
`Connector`	2	5820	580	1.98	5	Proximity-Aware EM
(Fig. 7)					2	González et al. [31]
					2	Standard QEM
`Engine`	17	42286	3800	8.34	811	Proximity-Aware EM
(Fig. 10)					9	González et al. [31]
					8	Standard QEM
`Car`	425	3075108	150000	0.00829	67680	Proximity-Aware EM
(Fig. 11)					25980	González et al. [31]
					23460	Standard QEM

Table 1: Scenes details. The proximity threshold is computed automatically and given as a percentage of the scene axis-aligned bounding box diagonal.

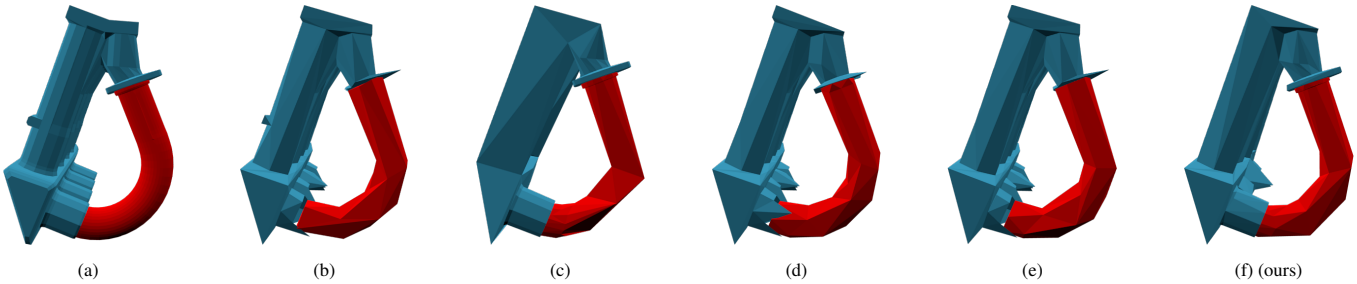
Scene	Proximity mean	Non-proximity mean	Proximity max	Non-proximity max	Method
`Tube`	**0.0897**	0.446	**0.594**	5.23	Proximity-Aware EM
(Fig. 5)	0.336	0.395	4.82	**5.01**	González et al. [31]
	0.384	**0.348**	4.81	5.02	Standard QEM
`Tube*`	**0.0774**	0.388	**2.45**	5.63	Proximity-Aware EM
(Fig. 6)	0.455	0.273	5.00	5.01	González et al. [31]
	0.462	**0.248**	4.98	**4.98**	Standard QEM
`Connector`	**0.0540**	0.305	**0.514**	**2.33**	Proximity-Aware EM
(Fig. 7)	0.254	0.313	1.15	2.44	González et al. [31]
	0.240	**0.301**	0.787	3.83	Standard QEM
`Engine`	**0.0475**	0.115	**1.08**	2.39	Proximity-Aware EM
(Fig. 10)	0.0527	0.205	1.47	2.91	González et al. [31]
	0.0487	**0.0488**	1.47	**0.389**	Standard QEM
`Car`	**0.00591**	0.0124	**0.659**	**1.10**	Proximity-Aware EM
(Fig. 11)	0.0143	0.0116	1.20	1.88	González et al. [31]
	0.0138	**0.0105**	1.75	2.50	Standard QEM

Table 2: Distances between meshes are given as a percentage of the scene axis-aligned bounding box diagonal (the less the better), and computed separately for proximity and non-proximity areas. Distances are computed by resampling the simplified scene and using point-to-plane distance with the closest face.

block collapses in proximity areas. For all the approaches, we set the stopping criterion as a target number of faces (see Table 1 for numerical values). As this number of faces is set for the whole scene, each approach is expected to balance the face budget differently on each object.

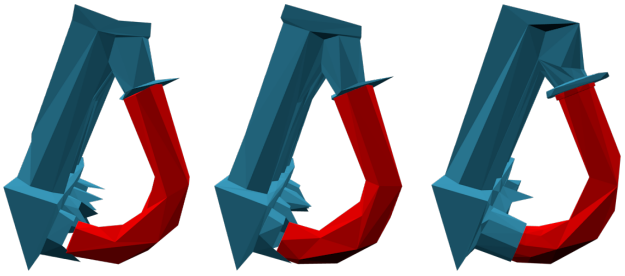
We also compare our approach to González et al. [31], who use a standard priority queue in which edges are sorted by geometric error, e.g. QEM.. When collapsed, edges having proximity, i.e. edges having one vertex nearby another object face, are collapsed with halfedge collapse rather than edge collapse. This strategy prevents the displacement of vertices marked as "boundary" in proximity areas but the geometry is still modified. We rather delay the simplification of edges having proximity while still collapsing with a standard edge collapse. As illustrated in Figures 5,6,7 and 10, the approach of González et al. [31] is less efficient in preserving the geometry in proximity areas and thus the inter-object relation is less readable.

Interlocking objects. The `Tube` scene is composed of two objects that mechanically fit into one another (see Figure 5(a)). Preserving the geometrical features associated with this relation (e.g. mechanical shoulder) is necessary to preserve the scene understanding. As shown in Figures 5(b) and 5(d), QEM fails at preserving the tube (red model) outer walls on the base (blue model), which retracts during decimation. Using large weights to penalize collapses in proximity areas (see Figure 5(c)) preserves the geometry of the outer and inner tubes, however at the cost of a relatively dense tessellation. As a result, other parts of the meshes are undersampled and display an altered shape. In comparison, our approach (see Figure 5(f)) nicely preserves the top shoulder on the tube and the outer tube slot on the base. Outside of proximity areas, the results look similar to standard QEM. The only noticeable difference is on the small structure at the back of the base, which is discarded by our approach and provides a larger face budget in favour of proximity areas.



(a) (b) (c) (d) (e) (f) (ours)

Fig. 5: (a) `Tube` scene. (b) Standard QEM. (c) QEM with proximity weighted by a too large weight prevents the simplification in proximity areas. (d) QEM with proximity weighted by a too low weight and (e) González et al. [31] separate the tube and the base meshes like standard QEM. (f) Proximity-Aware EM keeps the two objects as a whole.



(a) (b) (c) (ours)

Fig. 6: Simplification of the `Tube*` scene, obtained by tessellating `Tube` before simplification with (a) standard QEM, (b) González et al. [31] and (c) Proximity-Aware EM. The three methods produce similar results to those obtained by simplifying the original scene.

Impact of tessellation. We evaluated the stability of our approach w.r.t. variations of tessellation. The `Tube*` scene has been generated by subdividing `Tube` using midpoint subdivision (i.e. bilinear dyadic primal subdivision on triangles). As shown in Figure 6, our approach shows similar stability as QEM, while still preserving the contact zone. In the proximity analysis, the faces contribution is normalized by their area in order to minimize the dependence on the density of the meshes in a scene. Throughout the decimation process, collapse operations of all meshes are interleaved by error, so that variations in density are naturally handled by the edge collapse error. Hence, we do not perform any mesh regularization as size variations of mesh elements are accounted for by our algorithm.

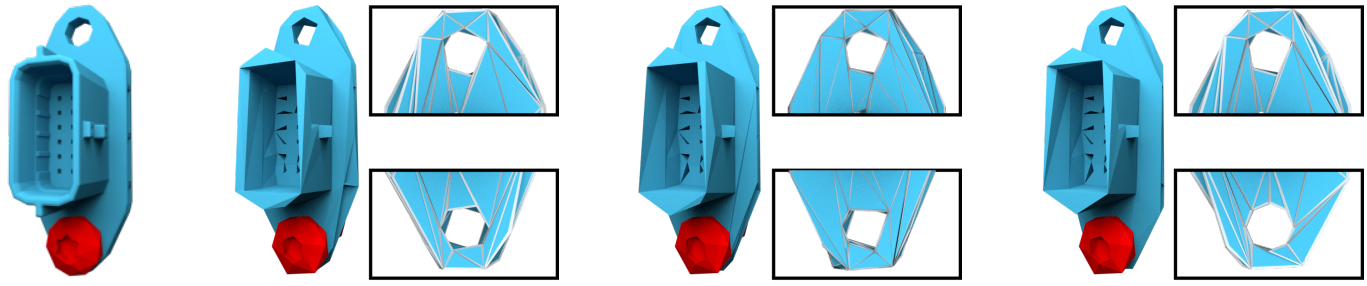
Cylindrical shapes. The `Connector` scene (see Figure 7) is composed of two objects, a connector and a screw passing through one of the connector holes. In this example, we want to preserve the shape of the connector hole crossed by the screw. As shown in Figure 7(d), our method displays a significant difference between the two holes of the connector, the one where the screw belongs is better preserved, as opposed to Figure 7(b) where both holes look similar and none of them looks circular anymore.

Proximity threshold evaluation. Our proximity-aware error metric is parameterized by a proximity threshold, which we define using automatic analysis of the scene. As illustrated in Figure 8, small variations of the proximity threshold have little impact on the results, while Figure 9 shows that our automatic procedure computes the appropriate order of magnitude.

Medium-complexity scene. The `Engine` scene is composed by 17 meshes with very different shapes (e.g. thin belts, cylinders and box-like shapes) and types of contacts (interlocked cylinders, partially colled belts, coplanar surfaces, etc), as illustrated in Figure 10(a). Even though the overall picture looks similar, our approach better preserves the geometry of meshes in proximity and improves the readability of the system functionality when the view is closer to the models. For instance, QEM generates intersections between the belt and its wheel, which are avoided with our approach (see the close-up in Figures 10(b)) and 10(d)). As for the `Tube` scene, our approach also better preserves the connection between the tubes and the base. On this scene, details are preserved in proximity areas by discarding some details in other parts of the scene and balancing the face budget. For instance, small geometrical components on top and at the bottom of the oil pan (in purple) are removed with our approach.

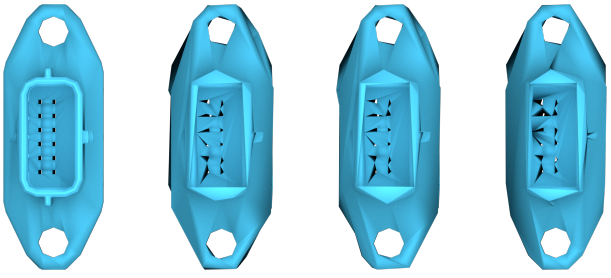
Realistic complex scene. The `Car` scene consists of 425 meshes modelling the front part of a real car, including systems and mechanical pieces (see Figure 1 for a global view, and Figure 11 for close-ups). As for the other scenes, by taking into account proximities in the simplification, functional information and shape relations are better preserved. As illustrated in both figures, screw heads are simplified differently with our approach, either to preserve contact with the screwed surface (Fig. 2) or to preserve the shape and alignment of the screw head with its axis (Fig. 11-top). Alignment is also better preserved between two connected axes (Fig. 11-middle), without creating intersections in contrast to QEM simplification. Functional readability may be severely deteriorated on very thin pipes (Fig. 11-bottom), which are shortened by QEM while they are well preserved with our approach.

Quantitative analysis. We numerically compared our results with standard QEM using the distances between the simplified and the original meshes. As our method aims at preserving the geometry in proximity areas, we measure separately the error introduced inside and outside the proximity areas. To this mean, the vertices of the original meshes are divided into two categories: those considered in proximity with other meshes, and those that are not. We report in Table 2 the mean and maximum (i.e. Haussdorff) distances for each scene, for both QEM



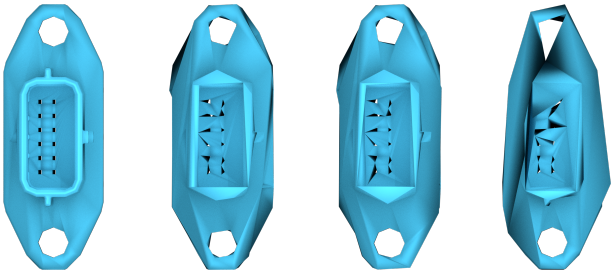
(a) Input scene (b) Standard QEM (c) González et al. [31] (d) Proximity-Aware EM

Fig. 7: `Connector` scene. Compared to previous approaches, our method better preserves the hole of the screw in the connector.



(a) (b) (c) (d)

Fig. 8: (a) The input `Connector` scene is simplified using (b) a proximity threshold twice lower than our automatically computed proximity threshold, (c) our proximity threshold, and (d) a proximity threshold twice higher. All these proximity thresholds are in the same order of magnitude and they produce similar results.



(a) (b) (c) (d)

Fig. 9: (a) The input `Connector` scene is simplified using (b) a proximity threshold ten times lower than our automatically computed proximity threshold, (c) our proximity threshold, and (d) a proximity threshold ten times higher. The smaller proximity threshold is too low to preserve the bottom screw hole and the higher proximity threshold, while perfectly preserving the bottom hole, degrades the top hole.

and our approach. As expected, our approach yields to lower errors in proximity areas, at the cost of higher errors in other parts of the scene. In most cases, however, our method outputs comparable error to QEM, and in two cases (`Connector` and `Car`) introduces less error than QEM when measured using Haussdorff distance.

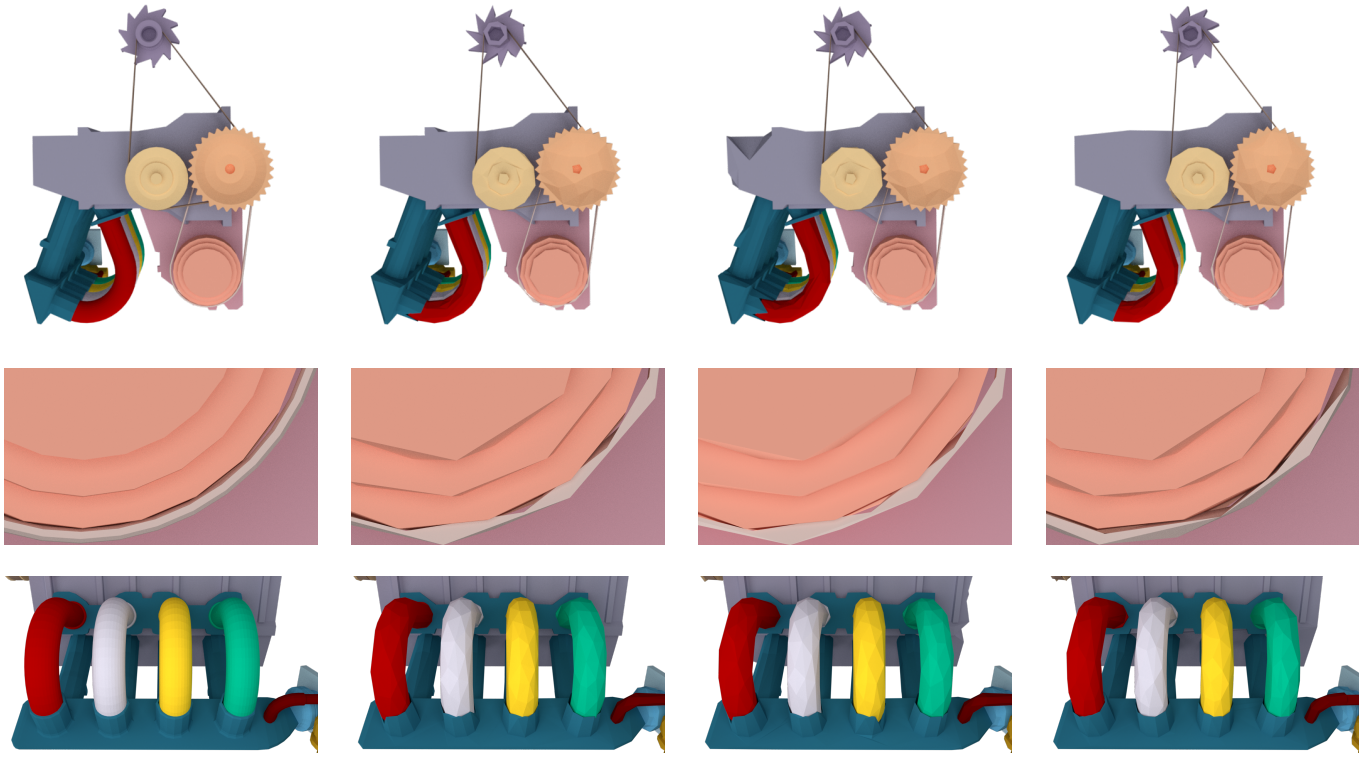
Error balance. Delaying collapses in proximity areas inevitably induces earlier simplifications in other areas. The result is that the geometric error increases more slowly in proximity areas than in other mesh parts where it grows faster. However, in practice, this does not lead to a very significant increase of the geometric error in the non-proximity areas and if other specific details are to be preserved, a dedicated penalty function may be defined and added in order to also delay the corresponding collapses.

Unwanted intersections. Our method naturally delays simplifications in proximity areas (especially for very close faces) and first collapses edges producing a low geometric error, i.e. edges in flat areas. Intersections are thus very unlikely to occur unless during extreme simplifications. If required, our method can be combined with existing methods that avoid intersections during the decimation, e.g. the intersection-free simplification [30].

8. Discussion and Conclusion

We propose an automatic pipeline for proximity-aware mesh decimation, extending standard QEM to account for spatial relationships when simplifying complex scenes composed of multiple objects. In pre-processing, we analyze the scene to specify which parts of the input meshes are in proximity, and we extend the QEM definition to increase the error of the collapse operations located in proximity areas. Our proximity analysis framework is robust to variations in tessellation, and can also be used to suggest multiple proximity configurations, enabling semi-automatic configuration for specific cases. As shown in our experiments, proximity-aware mesh decimation produces overall results comparable to standard decimation using QEM, while preserving details in proximity areas.

Our approach is based on the assumption that proximity relations imply semantic or functional relationships between objects, which is often true in CAD models, where shapes are designed to achieve a specific task. An interesting future investigation would be to extend our approach to other relations, such as alignment, symmetry, instances, or user-defined relations [22]. Our proximity detection algorithm might also be less accurate when the input objects contain large faces, as the distance anisotropy function is sampled for each face of the mesh. This is not a problem in most cases as we want to simplify detailed meshes, however, an interesting research direction would



(a) Input scene (b) Standard QEM (c) González et al. [31] (d) Proximity-Aware EM

Fig. 10: `Engine` scene. On the first row, the global view of the scene, where the simplification obtained with the previous approaches and our approach gives globally similar results. Details are however better preserved with our method, as visible on the two close-up views (middle and bottom rows).

be to integrate the spatial relationship over the mesh faces, in order to obtain a truly tessellation-independent method.

Our current implementation introduces a time overhead in comparison with QEM, which could be significantly reduced by optimizing distance queries and neighborhood relations. Our approach may be improved by caching faces in proximity areas and parallelizing the priority queue update procedures.

References

[1] Luebke, DP. A developer's survey of polygonal simplification algorithms. IEEE Computer Graphics and Applications 2001;21(3):24–35.

[2] Hoppe, H. Progressive meshes. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '96; ACM; 1996, p. 99–108. doi:10.1145/237170.237216.

[3] Garland, M, Heckbert, PS. Surface simplification using quadric error metrics. In: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '97; ACM; 1997, p. 209–216. doi:10.1145/258734.258849.

[4] Kwon, S, Kim, BC, Mun, D, Han, S. Simplification of feature-based 3d cad assembly data of ship and offshore equipment using quantitative evaluation metrics. Computer-Aided Design 2015;59:140–154.

[5] Rossignac, J, Borrel, P. Multi-resolution 3d approximations for rendering complex scenes. In: Modeling in computer graphics. Springer; 1993,.

[6] Low, KL, Tan, TS. Model simplification using vertex-clustering. In: Proceedings of the 1997 symposium on Interactive 3D graphics. ACM; 1997, p. 75–ff.

[7] Boubekeur, T, Alexa, M. Mesh simplification by stochastic sampling and topological clustering. Computers & Graphics 2009;33(3):241–249.

[8] Schroeder, WJ, Zarge, JA, Lorensen, WE. Decimation of triangle meshes. In: Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '92; ACM; 1992, p. 65–70. doi:10.1145/133994.134010.

[9] Turk, G. Re-tiling polygonal surfaces. In: Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '92; ACM; 1992, p. 55–64. doi:10.1145/133994.134008.

[10] Ciampalini, A, Cignoni, P, Montani, C, Scopigno, R. Multiresolution decimation based on global error. The Visual Computer 1997;13(5):228–246.

[11] Cohen, J, Varshney, A, Manocha, D, Turk, G, Weber, H, Agarwal, P, et al. Simplification envelopes. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '96; ACM; 1996, p. 119–128.

[12] Guéziec, A. Surface simplification inside a tolerance volume. IBM TJ Watson Research Center; 1996.

[13] Zelinka, S, Garland, M. Permission grids: Practical, error-bounded simplification. ACM Transactions on Graphics (TOG) 2002;21(2):207–229.

[14] Borouchaki, H, Frey, P. Simplification of surface mesh using hausdorff envelope. Computer methods in applied mechanics and engineering 2005;194(48-49):4864–4884.

[15] Mandad, M, Cohen-Steiner, D, Alliez, P. Isotopic approximation within a tolerance volume. ACM Transactions on Graphics (TOG) 2015;34(4):64.

[16] Klein, R, Liebich, G, Straßer, W. Mesh reduction with error control. In: Proceedings of Seventh Annual IEEE Visualization'96. IEEE; 1996, p. 311–318.

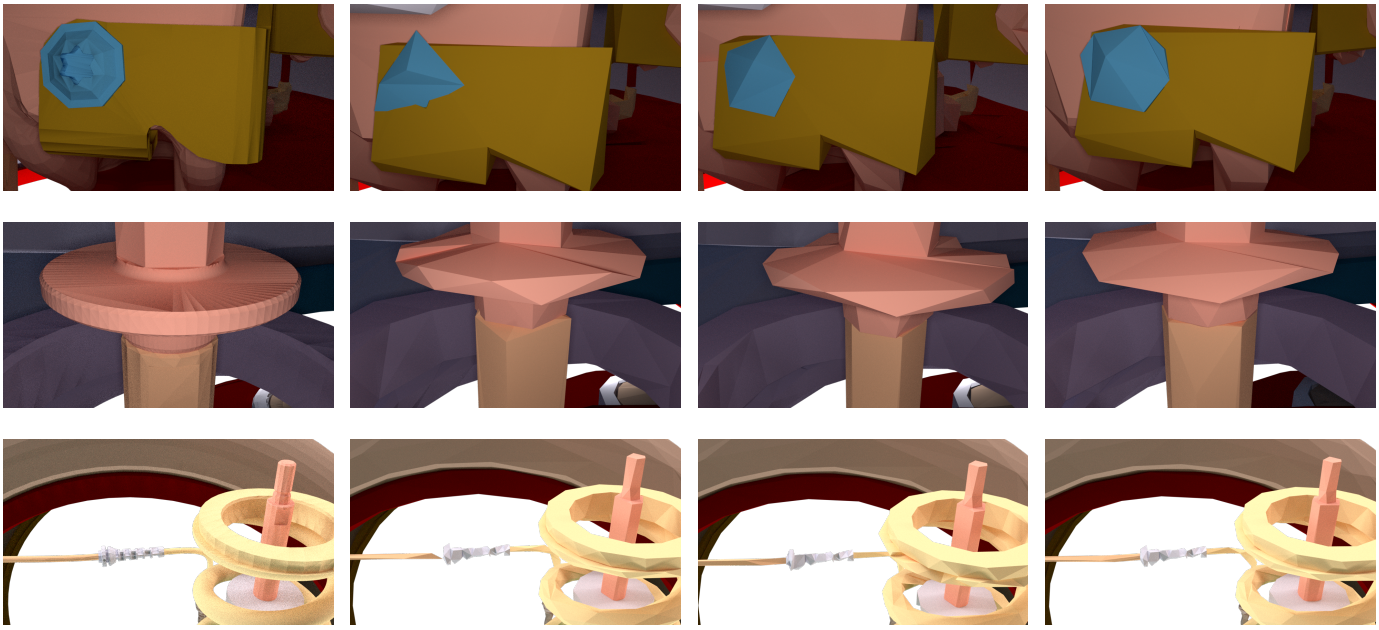
[17] Lindstrom, P, Turk, G. Fast and memory efficient polygonal simplification. In: Proceedings of the Conference on Visualization '98. IEEE; 1998, p. 279–286.

[18] Kim, SJ, Kim, CH, Levin, D. Surface simplification using a discrete curvature norm. Computers & Graphics 2002;26(5):657–663.

[19] Morigi, S, Rucci, M. Multilevel mesh simplification. The Visual Computer 2014;30(5):479–492.

[20] Marinov, M, Kobbelt, L. Automatic generation of structure preserving multiresolution models. In: Computer Graphics Forum; vol. 24. 2005, p. 479–486.

[21] Vivodtzev, F, Bonneau, GP, Le Texier, P. Topology-preserving simplification of 2d nonmanifold meshes with embedded structures. The Visual



(a) Input scene (b) Standard QEM (c) González et al. [31] (d) Proximity-Aware EM

Fig. 11: `Car` scene. Our method better preserves both the shape of parts having proximities and their relations.

Computer 2005;21(8-10):679–688.

[22] Kho, Y, Garland, M. User-guided simplification. In: Proceedings of the 2003 symposium on Interactive 3D graphics. ACM; 2003, p. 123–126.

[23] Pojar, E, Schmalstieg, D. User-controlled creation of multiresolution meshes. In: Proceedings of the 2003 symposium on Interactive 3D graphics. ACM; 2003, p. 127–130.

[24] Ho, TC, Lin, YC, Chuang, JH, Peng, CH, Cheng, YJ. User-assisted mesh simplification. In: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications. ACM; 2006, p. 59–66.

[25] Salinas, D, Lafarge, F, Alliez, P. Structure-aware mesh decimation. In: Computer Graphics Forum; vol. 34. Wiley Online Library; 2015, p. 211–227.

[26] Li, N, Gao, P, Lu, Y, Qiu, C, Wang, J, Yu, W. Parallel adaptive simplification of massive meshes. In: 2009 11th IEEE International Conference on Computer-Aided Design and Computer Graphics. IEEE; 2009, p. 632–635.

[27] Lindstrom, P. Out-of-core simplification of large polygonal models. In: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '00; ACM; 2000, p. 259–262. doi:10.1145/344779.344912.

[28] Cabiddu, D, Attene, M. Large mesh simplification for distributed environments. Computers & Graphics 2015;51:81–89.

[29] Erikson, C, Manocha, D, Baxter III, WV. Hlods for faster display of large static and dynamic environments. In: Proceedings of the 2001 symposium on Interactive 3D graphics. ACM; 2001, p. 111–120.

[30] Gumhold, S, Borodin, P, Klein, R. Intersection free simplification. International Journal of Shape Modeling 2003;9(02):155–176.

[31] González, C, Gumbau, J, Chover, M, Ramos, F, Quirós, R. User-assisted simplification method for triangle meshes preserving boundaries. Computer-Aided Design 2009;41(12):1095–1106.

[32] Edelsbrunner, H, Letscher, D, Zomorodian, A. Topological persistence and simplification. In: Proceedings 41st Annual Symposium on Foundations of Computer Science. IEEE; 2000, p. 454–463.